

1. LISA User Documentation	10
1.1 iTKO & LISA Introduction	10
1.1.1 iTKO Overview	11
1.1.1.1 a) Service-Oriented Architecture	11
1.1.1.2 b) How iTKO LISA Drives Value	12
1.1.2 LISA Overview	13
1.1.2.1 a) LISA SOA Lifecycle quality solutions	13
1.1.2.2 b) LISA Four SOA Testing Suite - Products	14
1.1.2.3 c) LISA Enterprise	15
1.1.2.4 d) LISA Extension Kit (LEK)	16
1.1.3 Technical Support & Contact Info	16
1.2 LISA Agent Guide	17
1.2.1 1. LISA Agent Repository	17
1.2.1.1 1.1 Install	17
1.2.1.2 1.2 Alternate Install	18
1.2.1.3 1.3 Downloads	18
1.2.2 2. LISA Agent User Guide	19
1.2.2.1 2.1 LISA Agent Overview	19
1.2.2.2 2.2 Getting Started	19
1.2.2.3 2.3 Configuring the Agent	20
1.2.2.4 2.4 Using the Dev Console	21
1.2.2.5 2.5 Developing against the Agent	23
1.2.2.6 2.6 Custom Extensions	39
1.2.2.7 2.7 Troubleshooting	41
1.2.2.8 2.8 Videos	43
1.2.3 3. LISA Agent Architecture	43
1.2.3.1 3.1 Architecture Overview	43
1.2.3.2 3.2 Deployment	49
1.2.3.3 3.3 Algorithms	50
1.2.4 4. LISA Agent Platforms	50
1.2.4.1 4.1 Tomcat _ JBoss _ Geronimo _ Resin	50
1.2.4.2 4.2 IBM WebSphere	51
1.2.4.3 4.3 BEA_Oracle WebLogic	53
1.2.4.4 4.4 WebMethods IS	53
1.2.4.5 4.5 Sun Glassfish	54
1.2.4.6 4.6 Tibco BusinessWorks	54
1.3 LISA Developer's Guide (LEK)	55
1.4 LISA Installation and Configuration Guide	55
1.4.1 1. Installing and Running LISA Workstation	56
1.4.1.1 1.1 System Requirements and Prerequisites	56
1.4.1.2 1.2 Installation Instructions	58
1.4.1.2.1 1.2.1 Installation Login	59
1.4.1.2.2 1.2.2 Download Page	59
1.4.1.2.3 1.2.3 Downloading the Installer	62
1.4.1.2.4 1.2.4 Installing LISA Workstation	64
1.4.1.2.5 1.2.5 Providing a Valid LISA License	69
1.4.1.3 1.3 Starting LISA Workstation	71
1.4.1.4 1.4 LISA Registry	72
1.4.1.4.1 1.4.1 Creating LISA Registry	74
1.4.1.4.2 1.4.2 Switching Registries	74
1.4.1.5 1.5 Post Installation Directories & Files	75
1.4.1.6 1.6 LISA Installation Notes	76
1.4.2 2. Installing and Running LISA Demo Server	77
1.4.2.1 2.1 Installing the JBOSS Demo Server	78
1.4.2.2 2.2 Starting the Demo Server	79
1.4.2.3 2.3 Example Test Case using Demo Server	80
1.4.3 3. Installing and Running LISA Server	84
1.4.3.1 3.1 Introduction to LISA Server	84
1.4.3.2 3.2 LISA Server Hardware Requirements	85
1.4.3.3 3.3 LISA Server Components	85
1.4.3.4 3.4 LISA Server Architecture	87
1.4.3.4.1 3.4.1 Data Flow within LISA Server	87
1.4.3.5 3.5 Installing and Configuring the LISA Server	88
1.4.3.5.1 3.4.1 Starting LISA Server	89
1.4.3.5.2 3.4.2 Getting Started	92
1.4.3.5.3 3.4.3 LISA Server Custom Configurations	92
1.4.3.5.4 3.4.4 Creating a Test Registry	92
1.4.3.5.5 3.4.5 Creating and Running Coordinator Servers	94
1.4.3.5.6 3.4.6 Creating and Running Simulator Servers	95
1.4.3.5.7 3.4.7 Running Detached Simulator Servers	96
1.4.3.5.8 3.4.8 Starting and Stopping Service	97
1.4.3.6 3.6 Miscellaneous Configuration Notes	98
1.4.3.6.1 3.6.1 Project Directory Structure	98
1.4.3.6.2 3.6.2 Load and Performance Server Sizing	99
1.4.3.6.3 3.6.3 Calculating Simulator Instances	99

1.4.3.6.4 3.6.4 Configuring LISA to a Different Database	99
1.4.3.6.5 3.6.5 Running Tests in Server Environment	101
1.4.3.7 3.7 LISA Server Environment	102
1.4.3.8 3.8 LISA Registry Monitor	103
1.4.3.8.1 3.8.1 Using the Load Test Optimizer	104
1.4.4 4. Configuring Java Tools	105
1.4.4.1 4.1 Installing Perfmon	105
1.4.4.2 4.2 Installing and Configuring SNMP	105
1.4.4.2.1 4.2.1 Installing SNMP Agent	106
1.4.4.2.2 4.2.2 Configuring SNMP Agent	107
1.4.4.3 4.3 Running TCPMon	109
1.4.4.3.1 4.3.1 Using TCPMon as Explicit Intermediate	109
1.4.4.3.2 4.3.2 Using TCPMon as a Request Sender for Web Services	111
1.4.4.4 4.4 Installing Mercury Test Director Plugin	111
1.5 LISA Pathfinder Pro	119
1.5.1 1. Creating a Data Set	119
1.5.2 2. Creating a Baseline Test	120
1.5.3 3. Creating a Virtual Model & Image	122
1.5.4 4. Creating a Defect Case	123
1.5.5 5. Editing a Defect Case	128
1.5.6 6. Filtering Defect Cases	130
1.6 LISA Reference Guide	131
1.6.1 PART 1 - Introduction	132
1.6.2 PART 2 - Test Steps	132
1.6.2.1 1. Web_Web Services Steps	134
1.6.2.1.1 1.1 HTTP_HTML Request	135
1.6.2.1.2 1.2 REST Step	140
1.6.2.1.3 1.3 Web Service Execution (XML)	142
1.6.2.1.4 1.4 WSDL Validation	170
1.6.2.1.5 1.5 Raw SOAP Request	171
1.6.2.1.6 1.6 Base64 Encoder	173
1.6.2.1.7 1.7 Multipart MIME (Multipurpose Internet Mail Extensions) Step	174
1.6.2.1.8 1.8 SAML Assertion Query	175
1.6.2.1.9 1.9 Web Service Execution (Legacy)	177
1.6.2.1.10 1.10 Start or Stop Web Server	200
1.6.2.2 2. Java_J2EE Steps	201
1.6.2.2.1 2.1 Dynamic Java Execution	201
1.6.2.2.2 2.2 RMI Server Execution	204
1.6.2.2.3 2.3 Enterprise Java Bean Execution	206
1.6.2.3 3. Other Transaction Steps	209
1.6.2.3.1 3.1 SQL Database Execution (JDBC)	210
1.6.2.3.2 3.2 Corba Execution	212
1.6.2.4 4. Utilities Steps	214
1.6.2.4.1 4.1 Save Property as Last Response	215
1.6.2.4.2 4.2 Output Log Message	215
1.6.2.4.3 4.3 Write Properties to File	216
1.6.2.4.4 4.4 Read Properties from a File	217
1.6.2.4.5 4.5 Do-Nothing Step	217
1.6.2.4.6 4.6 Parse Text as Response	217
1.6.2.4.7 4.7 Base64 Encoder Step	218
1.6.2.4.8 4.8 Checksum Step	218
1.6.2.4.9 4.9 Convert XML to Element Object	219
1.6.2.4.10 4.10 Compare Strings for Response Lookup step	220
1.6.2.4.11 4.11 Compare Strings for Next Step Lookup step	221
1.6.2.5 5. External_Sub Process Steps	222
1.6.2.5.1 5.1 Execute External Command	223
1.6.2.5.2 5.2 File System Snapshot	225
1.6.2.5.3 5.3 Execute Sub Test Case	226
1.6.2.5.4 5.4 Execute Sub Process	227
1.6.2.5.5 5.5 Execute JUnit Test Case_Suite	229
1.6.2.5.6 5.6 Read a File (Disk, URL or Class Path)	230
1.6.2.5.7 5.7 FTP Step	231
1.6.2.6 6. JMS Messaging Steps	232
1.6.2.6.1 6.1 JMS Messaging (JNDI)	232
1.6.2.6.2 6.2 Message Consumer	237
1.6.2.7 7. Bea Steps	239
1.6.2.7.1 7.2 Weblogic JMS (JNDI)	240
1.6.2.7.2 7.3 Message Consumer	244
1.6.2.7.3 7.4 Read a File (Disk, URL, or Classpath)	244
1.6.2.7.4 7.5 Next Generation Web Service Execution (alpha)	244
1.6.2.7.5 7.6 Web Service Execution	244
1.6.2.7.6 7.7 Raw Soap Request	244
1.6.2.7.7 7.8 FTP Step	244
1.6.2.8 8. Sun JCAPS Steps	245
1.6.2.8.1 8.1 JCaps Messaging (Native)	245

1.6.2.8.2 8.2 JCaps Messaging (JNDI)	249
1.6.2.8.3 8.3 Message Consumer	253
1.6.2.8.4 8.4 Read a File (Disk, URL, or Classpath)	254
1.6.2.8.5 8.5 Next Generation Web Service Execution (alpha)	254
1.6.2.8.6 8.6 Web Service Execution	254
1.6.2.8.7 8.7 Raw Soap Request	254
1.6.2.8.8 8.8 SQL Database Execution	254
1.6.2.8.9 8.9 FTP Step	254
1.6.2.9 9. Oracle Steps	254
1.6.2.9.1 9.1 Oracle OC4J (JNDI)	254
1.6.2.9.2 9.2 Oracle AQ (JMS)	259
1.6.2.9.3 9.3 Oracle AQ (JPUB)	259
1.6.2.9.4 9.4 Message Consumer	260
1.6.2.9.5 9.5 Read a File (Disk, URL, or Classpath)	260
1.6.2.9.6 9.6 Next Generation Web Service Execution (alpha)	260
1.6.2.9.7 9.7 Web Service Execution	261
1.6.2.9.8 9.8 Raw Soap Request	261
1.6.2.9.9 9.9 SQL Database Execution	261
1.6.2.9.10 9.10 FTP Step	261
1.6.2.10 10. TIBCO Steps	261
1.6.2.10.1 10.1 TIBCO Rendezvous Messaging	261
1.6.2.10.2 10.2 TIBCO Direct JMS	268
1.6.2.10.3 10.3 Message Consumer	272
1.6.2.10.4 10.4 Read a File (Disk, URL, or Classpath)	273
1.6.2.10.5 10.5 Next Generation Web Service Execution (alpha)	273
1.6.2.10.6 10.6 Web Service Execution	273
1.6.2.10.7 10.7 Raw Soap Request	273
1.6.2.10.8 10.8 SQL Database Execution	273
1.6.2.10.9 10.9 FTP Step	273
1.6.2.11 11. Sonic Steps	273
1.6.2.11.1 11.1 SonicMQ Messaging (Native)	273
1.6.2.11.2 11.2 SonicMQ Messaging (JNDI)	277
1.6.2.11.3 11.3 Message Consumer	281
1.6.2.11.4 11.4 Read a File (Disk, URL, or Classpath)	282
1.6.2.11.5 11.5 Next Generation Web Service Execution (alpha)	282
1.6.2.11.6 11.6 Web Service Execution	282
1.6.2.11.7 11.7 Raw Soap Request	282
1.6.2.11.8 11.8 SQL Database Execution	282
1.6.2.11.9 11.9 FTP Step	282
1.6.2.12 12. WebMethods Steps	282
1.6.2.12.1 12.1 WebMethods Broker	282
1.6.2.12.2 12.2 webMethods Integration Server Services	286
1.6.2.12.3 12.3 Message Consumer	287
1.6.2.12.4 12.4 Read a File (Disk, URL, or Classpath)	287
1.6.2.12.5 12.5 Next Generation Web Service Execution (alpha)	287
1.6.2.12.6 12.6 Web Service Execution	287
1.6.2.12.7 12.7 Raw Soap Request	287
1.6.2.12.8 12.8 SQL Database Execution	287
1.6.2.12.9 12.9 FTP Step	287
1.6.2.13 13. IBM Steps	287
1.6.2.13.1 13.1 IBM Websphere MQ	288
1.6.2.13.2 13.2 Message Consumer	292
1.6.2.14 14. Virtual Service Environment Steps	292
1.6.2.14.1 14.1 Virtual Service Router	292
1.6.2.14.2 14.2 Virtual Service Tracker	293
1.6.2.14.3 14.3 Virtual Conversational_Stateless Response Selector	294
1.6.2.14.4 14.4 Virtual HTTP_S Listener	295
1.6.2.14.5 14.5 Virtual HTTP_S Live Invocation	295
1.6.2.14.6 14.6 Virtual HTTP_S Responder	296
1.6.2.14.7 14.7 Virtual JDBC Listener	297
1.6.2.14.8 14.8 Virtual JDBC Responder	297
1.6.2.14.9 14.9 Socket Server Emulator	298
1.6.2.14.10 14.10 Messaging Virtualization Marker	301
1.6.2.14.11 14.11 Socket Server Emulator	301
1.6.2.14.12 14.12 Compare Strings for Response Lookup	303
1.6.2.14.13 14.13 Compare Strings for Next Step Lookup	304
1.6.2.15 15. Custom Extension Steps	305
1.6.2.15.1 15.1 Custom Test Step Execution	305
1.6.2.15.2 15.2 Java Script Test	305
1.6.2.15.3 15.3 Swing Test Step	306
1.6.2.15.4 15.4 Java Protocol Request Listener	308
1.6.2.15.5 15.5 JavaProtocol.Responder	309
1.6.2.15.6 15.6 Java Pass Through	309
1.6.2.15.7 15.7 JMS Pass through	310
1.6.2.15.8 15.8 MQ Pass through	310

1.6.2.16	16. Standard Test Steps	310
1.6.2.16.1	16.1 Abort the Test	310
1.6.2.16.2	16.2 End the Test	311
1.6.2.16.3	16.3 Fail the Test	312
1.6.3	PART 3 - Filters	312
1.6.3.1	17. Utility Filters	313
1.6.3.1.1	17.1 Create Property Based on Surrounding Values	314
1.6.3.1.2	17.2 Store Step Response	316
1.6.3.1.3	17.3 Override "Last Response" Property	318
1.6.3.1.4	17.4 Save Property Value to File	318
1.6.3.1.5	17.5 Parse Property Value as Argument String	319
1.6.3.1.6	17.6 Save Property from one key to another	320
1.6.3.1.7	17.7 Time Stamp Filter	321
1.6.3.2	18. Database Filters	322
1.6.3.2.1	18.1 Extract Value from JDBC Result Set	324
1.6.3.2.2	18.2 Simple Result Set Filter	324
1.6.3.2.3	18.3 Size of JDBC Result Set	325
1.6.3.2.4	18.4 Set Size of a Result Set to a Property	325
1.6.3.2.5	18.5 Get Value For Another Value in a ResultSet Row	327
1.6.3.3	19. Messaging_ESBFilters	327
1.6.3.3.1	19.1 Extract payload and Properties from Messages	328
1.6.3.3.2	19.2 Convert a MQ Message to a VSE Request	329
1.6.3.3.3	19.3 Convert a JMS Message to a VSE Request	329
1.6.3.4	20. HTTP_HTML Filters	330
1.6.3.4.1	20.1 Create Resultset from HTML Table Rows	332
1.6.3.4.2	20.2 Parse Web Page for Properties	334
1.6.3.4.3	20.3 Parse HTML_XML Result for Specific Tag_Attributes Values	335
1.6.3.4.4	20.4 Parse HTML Result for Specific Tag_Attribute's Value and Parse It	336
1.6.3.4.5	20.5 Parse HTML Result for Tag's Child Text	337
1.6.3.4.6	20.6 Parse HTML Result for HTTP Header Value	338
1.6.3.4.7	20.7 Parse HTML Result for Attribute's Value	338
1.6.3.4.8	20.8 Parse HTML Result for LISA Tags	339
1.6.3.4.9	20.9 Parse HTML Result and Select Random Attribute Value	340
1.6.3.4.10	20.10 Parse HTML into List of Attributes	341
1.6.3.4.11	20.11 Parse HTTP Header Cookies	341
1.6.3.4.12	20.12 Dynamic Form Filter	342
1.6.3.4.13	20.13 Parse HTML Result by Searching Tag_Attribute Values	343
1.6.3.5	21. XML Filters	345
1.6.3.5.1	21.1 Parse text from XML	346
1.6.3.5.2	21.2 Read Attribute from XML Tag	347
1.6.3.5.3	21.3 Parse XML Result for LISA Tag	347
1.6.3.5.4	21.4 Choose Random XML Attribute	348
1.6.3.5.5	21.5 XML XPath Filter	349
1.6.3.6	22. Web 2.0 Filters	350
1.6.3.6.1	22.1 Web 2.0 Element Filter	351
1.6.3.6.2	22.2 Web 2.0 Text Filter	351
1.6.3.6.3	22.3 Web 2.0 Attribute Filter	351
1.6.3.6.4	22.4 Web 2.0 Java Script Filter	351
1.6.3.6.5	22.5 Web 2.0 Function Filter	351
1.6.3.6.6	22.6 Web 2.0 Composite Filter	351
1.6.3.7	23. Java Filters	352
1.6.3.7.1	23.1 Override "Last Response" Property	352
1.6.3.7.2	23.2 Store Step Response	353
1.6.3.7.3	23.3 Save Property Value to File	353
1.6.3.8	24. VSE Filters	354
1.6.3.8.1	24.1 Standard Data Protocol Filter	354
1.6.3.8.2	24.2 Dynamic Data Protocol Filter	354
1.6.3.9	25. Pathfinder Filters	355
1.6.3.9.1	24.1 LISA Integration Support for Pathfinder	356
1.6.3.9.2	24.2 LISA Integration Support for webMethods Integration Server	357
1.6.4	PART 4 - Assertions	358
1.6.4.1	25. HTTP Assertions	359
1.6.4.1.1	25.1 Highlight HTML Content for Comparison	360
1.6.4.1.2	25.2 Check HTML for Properties in page	361
1.6.4.1.3	25.3 Ensure HTTP Header Contains Expression	362
1.6.4.1.4	25.4 Check HTTP Response Code	362
1.6.4.1.5	25.5 Simple Web Assertion	362
1.6.4.1.6	25.6 Check Links on Web Responses	363
1.6.4.2	26. Database Assertions	363
1.6.4.2.1	26.1 Ensure Result Set Size	364
1.6.4.2.2	26.2 Ensure Result Set Contains Expression	364
1.6.4.3	27. Web 2.0 Assertions	365
1.6.4.3.1	27.1 Web 2.0 Basic Assertion	366
1.6.4.3.2	27.2 Web 2.0 Validation Assertion	366
1.6.4.3.3	27.3 Web 2.0 Branching Assertion	366

1.6.4.4 28. XML Assertions	366
1.6.4.4.1 28.1 Highlight Text Content for Comparison	367
1.6.4.4.2 28.2 Ensure Result Contains String	368
1.6.4.4.3 28.3 Ensure Step Response Time	369
1.6.4.4.4 28.4 Graphical XML Side-by-Side Comparison	369
1.6.4.4.5 28.5 XML Side-by-Side Comparison	370
1.6.4.4.6 28.6 XML XPath Assertion	372
1.6.4.4.7 28.7 Ensure XML Validation	374
1.6.4.5 29. Virtual Service Environment Assertions	375
1.6.4.5.1 29.1 Assert on Execution Mode	375
1.6.4.6 30. Other Assertions	375
1.6.4.6.1 30.1 Highlight Text Content for Comparison	376
1.6.4.6.2 30.2 Ensure Non-Empty Result	377
1.6.4.6.3 30.3 Ensure Result Contains String	378
1.6.4.6.4 30.4 Ensure Result Contains Expression	378
1.6.4.6.5 30.5 Ensure Property Matches Expression	378
1.6.4.6.6 30.6 Ensure Step Response Time	379
1.6.4.6.7 30.7 Scripted Assertion	379
1.6.4.6.8 30.8 Ensure Properties Equal	380
1.6.4.6.9 30.9 Assert on Invocation Exception	381
1.6.4.6.10 30.10 File Watcher Assertion	381
1.6.4.6.11 30.11 Check Content of Collection Object	382
1.6.4.6.12 30.12 WS-I Basic Profile 1.1 Assertion	382
1.6.4.6.13 30.13 Messaging VSE Workflow Assertion	383
1.6.5 PART 5 - Data Sets	384
1.6.5.1 32. Data Sets	384
1.6.5.1.1 32.1 Read Rows from a Delimited Data File	385
1.6.5.1.2 32.2 Create your own Data Sheet	386
1.6.5.1.3 32.3 Create your own Set of Large Data	387
1.6.5.1.4 32.4 Convert Data Sheet to Delimited File Data Set	388
1.6.5.1.5 32.5 Read Rows from JDBC Table	388
1.6.5.1.6 32.6 Create a Numeric Counting Data Set	389
1.6.5.1.7 32.7 Read Rows from Excel File	389
1.6.5.1.8 32.8 Read DTO's from Excel File	391
1.6.5.1.9 32.9 Read Rows from a JDBC ResultSet	394
1.6.5.1.10 32.10 Unique Code Generator	395
1.6.5.1.11 32.11 Random Code Generator	396
1.6.5.1.12 32.12 Load a Set of File Names	397
1.6.5.1.13 32.13 XML Data Set	397
1.6.5.1.14 32.14 CorrelationID data set	402
1.6.6 PART 6 - Companions	402
1.6.6.1 33. Companions	402
1.6.6.1.1 33.1 Common Companions	403
1.6.6.1.2 33.2 Web Browser Simulation	403
1.6.6.1.3 33.3 Browser Bandwidth Simulation	405
1.6.6.1.4 33.4 Configure LISA to use a Web Proxy	407
1.6.6.1.5 33.5 Set Up a Synchronization Point	407
1.6.6.1.6 33.6 Set up an Aggregate Step	408
1.6.6.1.7 33.7 Other Companions	409
1.6.6.1.8 33.8 Create a Sandbox Class Loader for Each Step	409
1.6.6.1.9 33.9 Set Final Step to Execute	411
1.6.6.1.10 33.10 Negative Testing Companion	411
1.6.6.1.11 33.11 Fail Test Case Companion	411
1.6.6.1.12 33.12 XML Diff Ignored Nodes Companion	412
1.6.7 PART 7 - Events	412
1.6.7.1 34. Events	412
1.6.7.1.1 34.1 Test Events	413
1.6.7.1.2 34.2 LISA Test Events Table	415
1.6.8 PART 8 - Appendix	417
1.6.8.1 35. Appendix_ Alternative Wizard Names	418
1.6.8.1.1 35.1 Alternative Names Table	418
1.7 LISA User Guide	419
1.7.1 PART 1 - Getting Started	420
1.7.1.1 1. LISA Basic Concepts	420
1.7.1.2 2. LISA Workstation Overview	423
1.7.1.2.1 2.1 Starting LISA Workstation	423
1.7.1.2.2 2.2 LISA Workstation Main Menu	427
1.7.1.2.3 2.3 LISA Workstation Toolbar	438
1.7.1.2.4 2.4 LISA Quick Start Menu	441
1.7.1.2.5 2.5 LISA Workstation Views	449
1.7.1.2.6 2.6 LISA Runtime Information	456
1.7.1.3 3. Anatomy of a LISA Test Case	459
1.7.1.3.1 3.1 Elements of a Test Case	460
1.7.1.3.2 3.2 Elements of a Test Step	462
1.7.1.3.3 3.3 Test Case Quick Start	464

1.7.1.3.4 3.4 Multi-tier-combo Testcase	468
1.7.1.4 4. LISA Tutorials	469
1.7.1.4.1 4.1 Tutorial 1 - Basic Concepts - LISA Project, Test Case & Properties	470
1.7.1.4.2 4.2 Tutorial 2 - Basic Concepts - LISA Data Sets	482
1.7.1.4.3 4.3 Tutorial 3 - Basic Concepts - Filters and Assertions	490
1.7.1.4.4 4.4 Tutorial 4 - Manipulating Java Objects (POJOs)	496
1.7.1.4.5 4.5 Tutorial 5 - Running a Demo Server Web Application	502
1.7.1.4.6 4.6 Tutorial 6 - Testing a Web Site	507
1.7.1.4.7 4.7 Tutorial 7 - Testing an Enterprise JavaBean (EJB)	524
1.7.1.4.8 4.8 Tutorial 8 - Testing a Web Service	536
1.7.1.4.9 4.9 Tutorial 9 - Examining and Testing a Database	550
1.7.1.4.10 4.10 Tutorial 10 - Staging a Quick Test	560
1.7.2 PART 2 - Building Test Cases	566
1.7.2.1 5. Using Properties	566
1.7.2.1.1 5.1 Specifying a Property	567
1.7.2.1.2 5.2 Property Expressions	567
1.7.2.1.3 5.3 String Patterns	567
1.7.2.1.4 5.4 LISA Property Sources	572
1.7.2.1.5 5.5.1 LISA Property File (lisa.property)	573
1.7.2.1.6 5.5.2 Custom Property Files (local.property, site.property)	573
1.7.2.1.7 5.5 Important Property Files	574
1.7.2.1.8 5.6 Common LISA Properties and Environment Variables	574
1.7.2.2 6. Using Configurations	574
1.7.2.2.1 6.1 Project Configuration	575
1.7.2.2.2 6.2 Adding a Configuration	576
1.7.2.2.3 6.3 Creating a Configuration File	577
1.7.2.2.4 6.4 Default Configuration	578
1.7.2.2.5 6.5 Active Configuration	579
1.7.2.2.6 6.6 Alternate Configuration	580
1.7.2.2.7 6.7 Editing a Configuration	580
1.7.2.2.8 6.8 Copying a Configuration	581
1.7.2.2.9 6.9 Deleting a Configuration	581
1.7.2.2.10 6.10 Renaming New Configuration	582
1.7.2.2.11 6.11 Importing a Configuration File	582
1.7.2.2.12 6.12 Applying Config when running Test Case	583
1.7.2.3 7. Adding Filters	584
1.7.2.3.1 7.1 Adding Filters	585
1.7.2.3.2 7.2 Deleting a Filter	595
1.7.2.3.3 7.3 Reordering a Filter	595
1.7.2.3.4 7.4 Drag and Drop Filter	595
1.7.2.3.5 7.5 Filter Toolbar	595
1.7.2.3.6 7.6 Types of Filters	596
1.7.2.4 8. Adding Assertions	597
1.7.2.4.1 8.1 Adding Assertions	598
1.7.2.4.2 8.2 Assertions Toolbar	607
1.7.2.4.3 8.3 Deleting Assertions	607
1.7.2.4.4 8.4 Reordering Assertions	608
1.7.2.4.5 8.5 Renaming Assertions	608
1.7.2.4.6 8.6 Drag & Drop Assertions	608
1.7.2.4.7 8.7 Configuring Next Step of an Assertion	608
1.7.2.4.8 8.8 Types of Assertions	609
1.7.2.5 9. Using Data Sets	610
1.7.2.5.1 9.1 Global and Local Data Set	611
1.7.2.5.2 9.2 Random Data Set	613
1.7.2.5.3 9.3 Example Scenarios	614
1.7.2.5.4 9.4 Adding a Data Set	614
1.7.2.5.5 9.5 Deleting a Data Set	616
1.7.2.5.6 9.6 Reordering a Data Set	617
1.7.2.5.7 9.7 Renaming a Data Set	617
1.7.2.5.8 9.8 Drag and Drop Data Set	617
1.7.2.5.9 9.9 Data Set Next Step Selection	617
1.7.2.5.10 9.10 Data Set and Properties	618
1.7.2.5.11 9.11 Data Set Types	618
1.7.2.6 10. Using Companions	619
1.7.2.6.1 10.1 Adding a Companion	619
1.7.2.6.2 10.2 Companion Toolbar	620
1.7.2.6.3 10.3 Deleting a Companion	620
1.7.2.6.4 10.4 Reordering Companions	620
1.7.2.6.5 10.5 Companion Types	621
1.7.2.6.6 10.6 LISA Hooks	621
1.7.2.7 11. Using Complex Object Editor (COE)	622
1.7.2.7.1 11.1 COE User Interface	622
1.7.2.7.2 11.2 Object Call Tree Panel	625
1.7.2.7.3 11.3 Data Sheet & Call Sheet Panels	626
1.7.2.7.4 11.4 Object Interaction Panels	628

1.7.2.7.5	11.5 Using Data Sets in the COE	632
1.7.2.7.6	11.6 Usage Scenarios - Simple Objects	634
1.7.2.7.7	11.7 Usage Scenarios - Complex Objects	637
1.7.3	PART 3 - Building Documents	643
1.7.3.1	12. Building Test Cases	644
1.7.3.1.1	12.1 Creating a LISA Project	644
1.7.3.1.2	12.2 Building Test Steps	653
1.7.3.1.3	12.3 Creating Test Cases	669
1.7.3.1.4	12.4 Building Sub Processes	677
1.7.3.2	13. Building Staging Documents	684
1.7.3.2.1	13.1 Creating a Staging Document	685
1.7.3.2.2	28.2 Staging Document Editor	686
1.7.3.3	14. Building Test Suites	707
1.7.3.3.1	14.1 Running a Test Suite	708
1.7.3.3.2	14.2 Test Registry Monitor for LISA Server	712
1.7.3.3.3	Delete	716
1.7.3.4	15. Building Audit Documents	716
1.7.3.4.1	15.1 Creating an Audit Document	716
1.7.3.4.2	15.2 Sample Audit Document	718
1.7.3.5	16. Applying Metrics	719
1.7.3.5.1	16.1 Types of Metrics	719
1.7.3.5.2	16.2 Adding Metrics	731
1.7.3.5.3	16.3 Viewing Metrics in Quick Test	731
1.7.3.6	17. Understanding Events	736
1.7.3.6.1	17.1 Events	737
1.7.3.6.2	17.2 Filtering Events	737
1.7.3.6.3	17.3 Adding_ Viewing Events in LISA	738
1.7.3.6.4	17.4 LISA Test Events Table	742
1.7.4	PART 4 - Running Test Cases_Suites	745
1.7.4.1	18. Running a Single Test	745
1.7.4.1.1	18.1 Starting a Single test	746
1.7.4.1.2	18.2 Test Monitor Window	747
1.7.4.1.3	18.3 Quick Test Toolbar	752
1.7.4.1.4	18.4 Example Test Case	753
1.7.4.1.5	18.5 View Reports	755
1.7.4.2	19. Running a Quick Test	756
1.7.4.2.1	19.1 Running a Quick Test	756
1.7.4.2.2	19.2 Test Monitor window	758
1.7.4.2.3	19.3 Quick Test Toolbar	763
1.7.4.2.4	19.4 Example Test Case	764
1.7.4.2.5	19.5 View Report	766
1.7.4.3	20. Running a Test Suite	768
1.7.4.3.1	20.1 Creating a Test Suite	768
1.7.4.3.2	20.2 Test Suite Editor	769
1.7.4.4	21. Running a Test using ITR	777
1.7.4.4.1	21.1 Running the Interactive Test Run (ITR)	777
1.7.4.4.2	21.2 Interpreting the Results in an ITR Run	782
1.7.4.4.3	21.3 Graphical Diff Utility	785
1.7.5	PART 5 - Viewing LISA Portals	789
1.7.5.1	22. Pathfinder	789
1.7.5.1.1	22.1 Exploring Pathfinder	791
1.7.5.1.2	22.2 Pathfinder Console	792
1.7.5.1.3	22.3 Available Paths Tab	793
1.7.5.1.4	22.4 Component Details Tab	794
1.7.5.1.5	22.5 Statistics in Path Tab	798
1.7.5.1.6	22.6 Filtering Paths	800
1.7.5.1.7	22.7 Sorting & Selecting Columns	801
1.7.5.1.8	22.8 Integrating Pathfinder Broker and Portal Server	803
1.7.5.1.9	22.9 Pathfinder Summary	803
1.7.5.2	23. CVS Dashboard	804
1.7.5.2.1	23.1 Starting CVS	804
1.7.5.2.2	23.2 Understanding CVS Dashboard	805
1.7.5.2.3	23.3 Monitor Tab	811
1.7.5.2.4	23.4 Graphs Tab	814
1.7.5.2.5	23.5 Events Tab	815
1.7.5.2.6	23.6 Examples	816
1.7.5.2.7	23.7 CVS Notes	819
1.7.5.3	24. Reports	820
1.7.5.3.1	24.1 Reports 5.0	820
1.7.5.3.2	24.2 Legacy Reports 4.0	854
1.7.5.3.3	24.3 General Information	883
1.7.6	PART 6 - Recorders & Test Generators	884
1.7.6.1	25. Recording a Website	885
1.7.6.1.1	25.1 Recording a Web Site via HTTP Proxy	885
1.7.6.1.2	25.2 Recording a Web Site via DOM Events	891

1.7.6.2 26. Generating a Web Service	891
1.7.6.2.1 26.1 XML Web Service	891
1.7.6.2.2 26.2 Legacy Web Service	897
1.7.6.2.3 26.3 Web Service Customization	900
1.7.6.2.4 26.4 Virtual Web Service (VSE)	900
1.7.7 PART 7 - LISA Advanced Features	902
1.7.7.1 27. Using BeanShell in LISA	903
1.7.7.1.1 27.1 Using BeanShell Scripting Language	903
1.7.7.1.2 27.2 Using LISA Date Utilities	904
1.7.7.2 28. Running LISA with Ant and JUnit	905
1.7.7.2.1 28.1 Running LISA Tests as JUnit Tests	905
1.7.7.2.2 28.2 Usage Examples	906
1.7.7.2.3 28.3 Usage Examples II	907
1.7.7.2.4 28.4 Integration with Cruise Control	908
1.7.7.3 29. Class Loader Sandbox Example	909
1.7.7.4 30. In-Container Testing (ICT)	909
1.7.7.4.1 30.1 Access via EJB (J2EE Container Environments)	910
1.7.7.4.2 30.2 Access via RMI (Custom Java Server and Application Environments)	910
1.7.7.4.3 30.3 Testing your ICT Installation from LISA Workstation	911
1.7.7.5 31. LISA Command Line Utilities	912
1.7.7.5.1 31.1 Test Runner	913
1.7.7.5.2 31.2 CVS Manager	919
1.8 LISA VSE Guide	919
1.8.1 1. VSE Introduction	920
1.8.1.1 1.1 LISA Workstation Main Menu for VSE	920
1.8.1.1.1 1.1.1 File Menu	920
1.8.1.1.2 1.1.2 Edit Menu	920
1.8.1.1.3 1.1.3 View Menu	921
1.8.1.1.4 1.1.4 System Menu	921
1.8.1.1.5 1.1.5 Actions Menu	921
1.8.1.1.6 1.1.6 Help Menu	923
1.8.1.2 1.2 LISA workstation Toolbar for VSE	923
1.8.1.3 1.3 Service Virtualization	925
1.8.1.4 1.4 High-level Virtualization Steps	925
1.8.1.5 1.5 Benefits of Virtualization	927
1.8.2 2. VSE Installation and Configuration	928
1.8.2.1 2.1 System Requirements	928
1.8.2.2 2.2 Installing LISA Virtualize	929
1.8.2.3 2.3 Configuring LISA Virtualize	929
1.8.2.4 2.4 Installing the Database Simulator	931
1.8.2.5 2.5 Installing a Messaging Simulator	933
1.8.2.6 2.6 DDLs for Major DBs	933
1.8.3 3. Understanding LISA Virtualize	936
1.8.3.1 3.1 Following the LISA Virtualize Process	936
1.8.3.2 3.2 Concept Diagram	936
1.8.3.3 3.3 Virtual Service Model (VSM)	937
1.8.3.4 3.4 Services Image	937
1.8.3.5 3.5 How Virtualization Works	938
1.8.3.6 3.6 Magic Strings and Dates	940
1.8.4 4. Understanding Transactions	941
1.8.4.1 4.1 Stateless and Conversational Transactions	942
1.8.4.2 4.2 Logical Transactions	942
1.8.4.3 4.3 Match Tolerance	943
1.8.5 5. Recording a Service Image	943
1.8.5.1 5.1 Virtual Service Model (.vsm)	944
1.8.5.2 5.2 Service Image	945
1.8.5.2.1 5.2.1 Virtualizing JDBC	978
1.8.5.3 5.3 Identification of Conversations and Transactions	980
1.8.5.4 5.4 Virtual Service From WSDL	984
1.8.6 6. Editing a Service Image	988
1.8.6.1 6.1 VSE Service Images Window	988
1.8.6.2 6.2 Service Image Editor	989
1.8.6.3 6.3 Service Image Editor Basic Info Tab	990
1.8.6.4 6.4 Service Image Editor Stateless Transactions Tab	991
1.8.6.5 6.5 Service Image Editor Conversations tab	995
1.8.6.6 6.6 Conversation Tree Editor	996
1.8.7 7. Editing a VSM	100
1.8.7.1 7.1 Virtual Conversational_Stateless Response Selector Step	100
1.8.7.2 7.2 Virtual HTTP_S Listener step	100
1.8.7.3 7.3 Virtual HTTP_S Responder Step	100
1.8.7.4 7.4 Virtual JDBC Listener step	100
1.8.7.5 7.5 Virtual JDBC Responder Step	100
1.8.7.6 7.6 Socket Server Emulator Step	100
1.8.7.7 7.7 Messaging Virtualization Marker Step	101
1.8.7.8 7.8 Compare Strings for Response Lookup Step	101

1.8.7.9 7.9 Compare Strings for Next Step Lookup Step	101
1.8.8 8. Doing the Virtualization	101
1.8.8.1 8.1 Preparing for Virtualization	101
1.8.8.2 8.2 Deploying and Running a VSM	101
1.8.8.3 8.3 Running Live Request Against LISA VSE	101
1.8.8.4 8.4 Execution Mode	101
1.8.8.5 8.5 Session Viewing and Model Healing	101
1.8.9 VSE Command Line Options	102
1.9 LISA Web 2.0 Guide	102
1.9.1 PART 1 - LISA Web 2.0 - User Guide	102
1.9.1.1 1. Introduction to Web 2.0	102
1.9.1.1.1 1.1 System Requirements	102
1.9.1.1.2 1.2 Technologies and Platforms	102
1.9.1.1.3 1.3 Getting Started with LISA Browser	102
1.9.1.2 2. Recording Mode	103
1.9.1.2.1 2.1 Recording Example	103
1.9.1.2.2 2.2 Recording a Swing Test	104
1.9.1.2.3 2.3 Recording an Applet Test	104
1.9.1.2.4 2.4 Different Views of a Web Page	104
1.9.1.2.5 2.5 Post Recording	105
1.9.1.3 3. Playback Mode	105
1.9.1.4 4. Edit Mode	106
1.9.1.4.1 4.1 Event Types	106
1.9.1.4.2 4.2 Logical Events	106
1.9.1.4.3 4.3 Object Details	106
1.9.1.4.4 4.4 Filters	107
1.9.1.4.5 4.5 Assertions	107
1.9.1.4.6 4.6 Datasets	107
1.9.1.4.7 4.7 Editing Steps in Workstation	107
1.9.1.5 5. Debugging	107
1.9.1.6 6. Setting up ADF Extensions	108
1.9.1.7 7. Running Browser Standalone	108
1.9.1.8 8. Troubleshooting	108
1.9.1.9 9. Known Limitations	108
1.9.2 PART 2 - LISA Web 2.0 - How Tos	108
1.9.2.1 1. Introduction	108
1.9.2.2 2. Web sites and frameworks	108
1.9.2.3 3. How To - Generate random data (4.5.1.x)	108
1.9.2.4 4. How To - Capture Dynamic HTML for later test editing	108
1.9.2.5 5. How To - Deal with time-sensitive events	108
1.9.2.6 6. How To - Parametrize dynamic data entry in loops	108
1.9.2.7 7. How To - Deal with dynamic elements	109
1.9.2.8 8. How To - Extract complex data from a page	109
1.9.2.9 9. How To - Ajax auto-complete fields	109
1.9.2.10 10. How To - Write custom Web 2.0 steps	109
1.9.2.11 11. How To - Write cross-browser tests	109
1.9.2.12 12. How To - Use Pathfinder integration	109
1.9.2.13 13. How To - Write Java Swing and WebStart tests	109
1.9.2.14 14. How To - Write .NET WinForms tests	110
1.9.2.15 15. How To - Debug a test	110
1.9.2.16 16. How To - Use global filters and global assertions	110
1.9.2.17 17. How To - Interact with external resources	110
1.9.2.18 18. How To - Run Load Tests	111
1.9.2.19 19. How To - Run in a non-privileged account or on 64 bit platforms	111
1.9.2.20 20. How To - Record and replay against non us-english websites	111
1.9.2.21 21. How To - Run in Crash Dump mode	111
1.9.3 PART 3 - LISA Web 2.0 - Reference	111
1.9.3.1 1. Recorder Reference	111
1.9.3.2 2. Debugger Reference	111
1.9.3.3 3. Settings Reference	111
1.9.3.4 4. XPath syntax Reference	112
1.9.3.5 5. Scripting Objects Reference	112
1.9.3.6 6. Command line Reference	112
1.9.4 PART 4 - LISA Web 2.0 - Videos	112
1.9.5 PART 5 - LISA Web 2.0 - Repository	112
1.9.5.1 1. Instructions	112
1.9.5.2 2. Always update	112
1.9.5.3 3. Update with major revisions changes	112
1.9.5.4 4. First time update (i.e. only if you're missing these files)	112
1.9.6 PART 6 - LISA Web 2.0 - FAQ	112

LISA User Documentation

LISA User Documentation

Welcome to LISA online documentation! The following documentation set gives a complete overview of LISA, its functionality and features. No prior knowledge of LISA is necessary to understand and utilize this online documentation. Although to work in LISA basic knowledge of Java is required.

NOTE: In addition to all of the online user documentation summarized below please note that the following resources are also available...

[Tech Notes](#)

[Forums](#)

The online user documentation is made up of the following documentation..

iTKO & LISA Introduction - This contains information on iTKO as well as an Introduction to LISA. It also contains LISA technical support information.

LISA Agent Guide - The LISA Agent is a piece of server-side technology that can get installed inside any java process (including J2EE containers) and allows LISA to control and monitor the server side of things.

LISA Developer's Guide (LEK) - This guide contains instructions for a more advanced user of LISA. LISA functionality can be extended with programmers as there is a high amount of customization available with the LEK (LISA Extension Kit). This guide supports that customization ability.

*NOTE: This document is **only available in .pdf format** at this time.*

Additional NOTE: The name is now being changed from LEK to SDK. Updates to this space will occur at a future date.

LISA Installation and Configuration Guide - This guide describes how to install and configure LISA™. It describes how to install and configure LISA Workstation as well as LISA Server.

LISA Pathfinder Pro - This guide contains detail on the features included in Pathfinder Pro, which requires an additional user license. *(This is currently under construction.)*

LISA Reference Guide - This guide contains detailed instructions on how to create and configure LISA's built in components. As such it assumes that you are familiar with the concepts, and the usage of these components after going through the User Guide.

LISA User Guide - This guide is the first complete guide that covers LISA and one that any basic user should use as a reference. It contains a [hands-on tutorial](#) section as well as detailed information covering the functionality and features of LISA. This is the **best place to start** once the introduction is reviewed and LISA is installed.

LISA VSE Guide - This guide contains detailed instructions on how to install, configure and use LISA Virtualize. LISA can be taught to mimic another service and later be used as a stand in for that service. To use LISA Virtualization one needs Virtualization to be enabled in the license, and this guide provides support for that functionality.

LISA Web 2.0 Guide - This guide contains covers Web 2.0 functionality. Web 2.0 testing works by letting LISA emulate a web browser. It allows you to record events at the Document Object Model level such as mouse clicks, mouse movements, keys being typed, etc and play those events back as a browser during text execution.

iTKO & LISA Introduction

iTKO & LISA Introduction

iTKO mitigates the business risk of IT quality issues at the rate of business change.

iTKO's industry-leading LISA™ software provides Complete, Collaborative and Continuous testing for SOA (Service-Oriented Architecture) applications. LISA reduces the business risk of migrating to SOA environments, while increasing trust across shared technology assets.

LISA tests all components of SOA workflows throughout their lifecycle, from web apps, to web services, databases, messaging layers, legacy objects and application Servers. Developers, QA teams and business analysts can rapidly automate functional and performance tests on a continuous basis through LISA's no-code interface.

Founded in 1999, iTKO's customers range from Fortune 500 financial and insurance leaders, to high-tech and software companies, to healthcare,

travel and the government sector. All of our customers, however, come to us because they have a commitment to achieving quality within their business processes. Our goal is to ensure success and high service levels for every company that selects iTKO as a quality partner.

The following chapters are available in this topic.

[iTKO Overview](#)
[LISA Overview](#)
[Technical Support & Contact Info](#)

iTKO Overview

iTKO Overview

Our mission is for everyone to own quality. Basically since our inception as a company, iTKO has preached the message that systems are evolving at a rapid rate from monolithic, centralized applications, to highly distributed composite applications consisting of multiple technologies and authority domains that control them. Indeed we have seen the market move in this direction, with analyst estimates from 80%-90% of Global 500 companies moving some or all of their development and integration efforts toward an SOA (Service-Oriented Architecture) strategy in 2007. So we can see that massive efforts (and budgets) have been assigned to the task of SOA migration in many large companies. The first market response to this coming budgetary shift was a move for the leading software Platform players to start offering the infrastructure components of SOA: the presentation/UI layer approach, messaging frameworks, Servers, and tools for delivering business logic.

Next the software industry as a whole, and several more players, started focusing on SOA Governance, Security and Virtualization, and the compliance rules needed for better interoperability among systems. But establishing trust in this Wild West of competing platforms can be extremely difficult, when the ability to deliver and integrate SOA applications exceeds the industry's ability to properly test these systems. The more complex your IT environment and integration becomes, the more relevant iTKO LISA becomes.

[a\) Service-Oriented Architecture](#)
[b\) How iTKO LISA Drives Value](#)

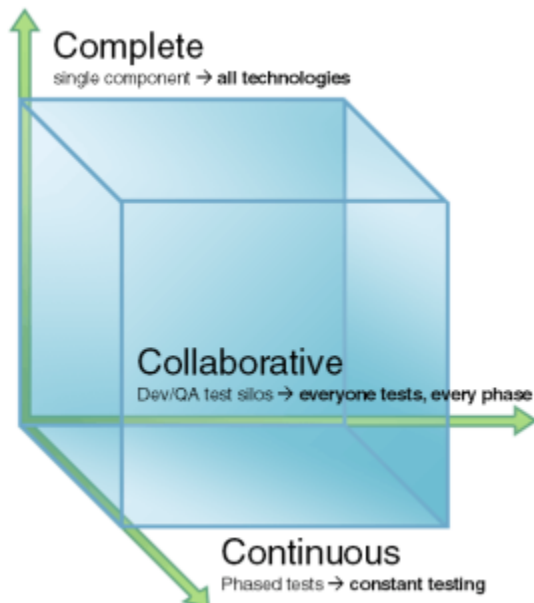
a) Service-Oriented Architecture

a) Service-Oriented Architecture

Service Oriented Architectures offer distinct business advantages.

SOA adoption has gone from experimentation to the mainstream within Fortune 1000 companies, with analyst estimates of IT budgetary allocations toward SOA-based technology (software and services) reaching US\$350B annually by 2010.

The growth of SOA is driven by:



- **Reduced integration cost** eased by loosely-coupled components and adoption of industry standards.
- **Increased asset reuse**, enabling new business workflows to be built from existing services to form composite applications with less effort.
- **Increased business agility**, with better control of business process definition and management to meet customer needs.
- **Reduced business risk**, with IT governance, compliance, and strategic development controlled through increased business visibility and agility.

But how will companies trust the quality of SOA for business?

To deliver SOA applications with confidence, enterprises must achieve Complete, Collaborative, and Continuous SOA quality:

- **Complete** testing of business workflows across every heterogeneous technology layer of the SOA, at both a system and component level.
- **Collaborative** involvement of the whole team in quality. Enable both developers and non-programmers to define and share Test Cases that prove how the SOA meets business requirements during the entire application lifecycle.
- **Continuous** validation of a changing SOA deployment as it is consumed at runtime, ensuring that business requirements will continue to be met as the system dynamically changes.

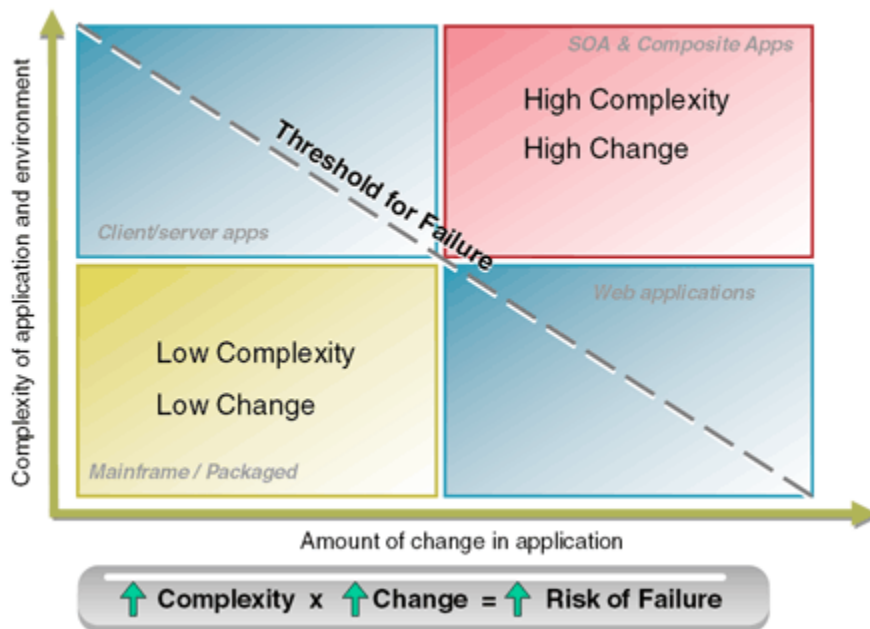
Leading companies with the most critical need for quality work with iTKO both for best practices and testing strategy, as well as LISA's ability to be used for unit and functional tests during development, then integration, load and performance testing in deployment without having to write code. In preproduction and production, LISA is used to provide continuous testing and monitoring of critical SOA applications.

b) How iTKO LISA Drives Value

b) How iTKO LISA Drives Value

SOA adoption is being driven by the promise of business agility and cost benefits, through loose coupling and reuse of technology assets. The tools to build and deploy SOA applications are available, but the means to properly ensure quality in these systems has lagged behind this delivery capability. In order to achieve the value expected from SOA, enterprises need to address the complexity of heterogeneous SOA environments, and the associated business risk of costly failures.

The Tipping Point of Quality?



The tipping point of SOA Testing Value

As shown above, the cost of testing, and the risk of failure, increase significantly when the rate of Change in IT (increased iterations and new connections) and the Complexity (increased number of components, services and standards) grow. Failure to properly test will inhibit the agility of the company to release as often as needed, as well as forcing the company to shy away from new, value added functionality.

SOA aligns technology around the business requirements of multiple stakeholders involved in the application – so no true SOA is built on a single technology standard. Continuous testing and validation must occur at every layer of the SOA architecture at every stage in the development, testing and deployment lifecycle. Without building Quality Lifecycle processes into the ongoing practice of creating and using services-based apps, the enterprise cannot accomplish the Trust necessary for successful SOA.

Value for Quality Professionals and Engineers

Software professionals know testing early and often is essential to having reliable software and reducing the risk of costly failures. However for complex Service-Oriented Architecture (SOA) applications, testing can no longer simply be a phase in the development process. How is testing SOA different from testing typical applications? To avoid the unintended consequences of change in SOA applications, continuous functional and performance monitoring of the business workflow must occur.

- SOA is by nature a heterogeneous environment, so tests must be able to invoke and verify functionality and performance at every layer of the architecture.
- In SOA, most of the business logic does not reside in the user interface, so tests must span multiple layers of the architecture and support dynamic test data.
- Management, Policy and Testing are the key supports for an SOA governance strategy. Good SOA testing practices contribute verifiable examples of Policy that are shared throughout the management process.
- Continuous testing must happen at both build and runtime, as each layer of SOA applications can be developed and introduced into the active deployment on its own lifecycle.
- Extensibility of SOA testing is important, since all enterprise environments contain legacy applications and custom components that contain key business functionality.

Value for Architects

Architecturally, your job is to set the organization on the right path toward SOA. – both for the ability to properly Govern your technology assets to meet business needs, but to prove that your interoperability with third parties and legacy applications meets those business goals. LISA enables Trust, both vertically within the chain of command, and horizontally across departments and business partners, can only happen with a shared vision for Lifecycle Quality, and a shared capability to automate and collaborate on continuous SOA testing.

- There are multiple valid approaches to constructing SOA applications. LISA supports WS-* approaches, enterprise Java, ESB, REST and other design patterns that may be used as part of an SOA strategy.
- Testing must enforce that Policies are met at a shared level across the extended set of stakeholders in SOA, while respecting the autonomy of multiple Federated authority domains.
- LISA allows Publishers and Consumers of services to share a common platform for certifying and validating that the SOA workflows are sustainable as conditions change.
- While Structural integrity and Performance are important and tested with LISA, Functional Monitoring is another key aspect of effective SOA that cannot be put off until deployment time.

Value for Managing and Governing IT

Companies are driven toward SOA by the promise of business agility and reuse. But SOA also creates constant change – which creates business risk due to software failures and misinterpreted requirements. To ensure the level of reliability and trust needed for mission-critical operations, the company must make sure Quality is embedded into the lifecycle of the SOA. Testing must happen from the time the architecture is designed, through development and integration, and continuously at runtime as conditions change.

- Provides automated risk management. You can't just test expect developers to find bugs at the code level or wait until the application is done at UI acceptance testing. LISA tests continuously, not only for structural or performance problems at runtime, but functional errors – business mistakes that can be particularly costly and hard to detect.
- LISA enforces a real lifecycle quality process alongside the development and delivery processes, enabling real SOA Governance via robust, test-enforceable Policies.
- A shift in processes change is often resisted within an organization. iTKO has extensive experience and best practices in guiding teams toward realizing the benefits of a SOA quality strategy.
- The implementation and training process for LISA is extremely rapid and low-impact, since LISA is easy for teams to learn and tests most technologies involved in SOA right out of the box.

LISA Overview

LISA Overview

The following topics are available...

- a) [LISA SOA Lifecycle quality solutions](#)
- b) [LISA Four SOA Testing Suite - Products](#)
- c) [LISA Enterprise](#)
- d) [LISA Extension Kit \(LEK\)](#)

a) LISA SOA Lifecycle quality solutions

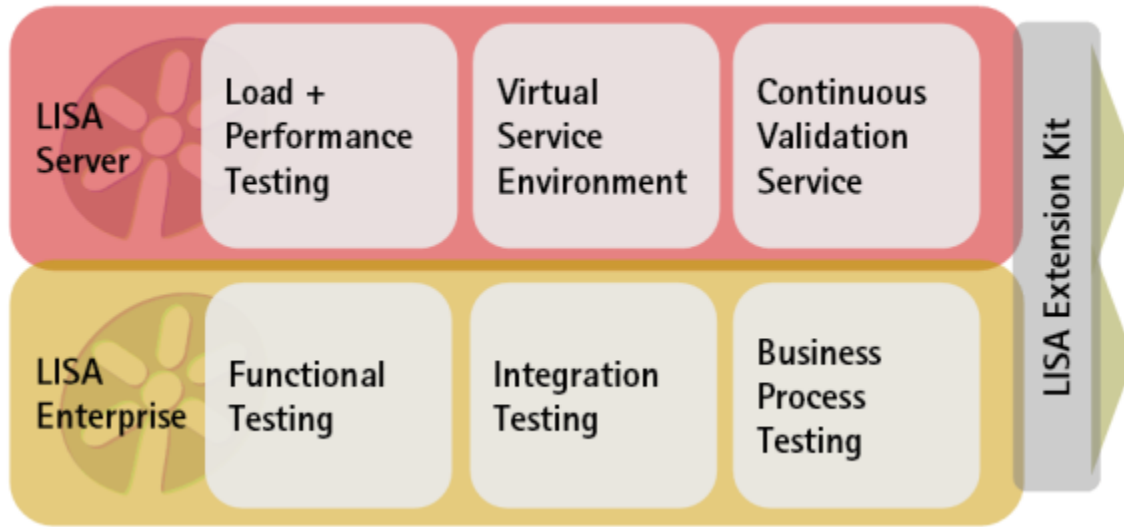
a) LISA SOA Lifecycle quality solutions

LISA provides Complete, Collaborative and Continuous Testing and Validation of SOA applications. Unlike conventional code testing and UI testing tools, LISA directly invokes and validates against every component of SOA application workflows.

iTKO LISA software helps companies reduce the risk of migrating to SOA environments, while increasing trust and reuse benefits. LISA invokes

and deeply tests every component of SOA applications throughout their lifecycle, from the web UI, to web services, databases, messaging layers and application Servers. With a simple double-click installation of LISA, developers, QA teams and business analysts can quickly learn to create, share and automate functional and performance tests on a continuous basis through LISA's no-code interface.

iTKO LISA 4 SOA Testing Suite



The iTKO LISA 4 SOA Testing Suite was developed from the ground up to test multi-tier SOA applications. All of LISA's test functionality is a singularly developed platform for ensuring quality and minimizing risk in complex, changing enterprise applications, so there is never a concern about sharing LISA tests and testing processes across technologies and phases of the application lifecycle.

LISA is divided into two primary groupings, with an extensibility API for future needs:

- **LISA Enterprise** is the test client that end users employ on their desktop to build and stage unit, functional, regression, and process tests against the system under test.
 - Functional Testing
 - Integration Testing
 - Business Process Testing
- **LISA Server** is the Server-side "engine" for LISA tests, managing the scheduling and orchestration of created LISA functional, load and performance tests on a continuous basis, and managing virtual service deployments and test beds, to ensure critical Service Levels are met at every stage of the lifecycle.
 - Load & Performance Testing
 - Virtual Service Environment
 - Continuous Validation Service
- **LISA Extension Kit** provides the essential flexibility toolset and API required for developers to test-instrument the many custom and legacy technologies that are inevitable in an enterprise IT environment.
 - LISA Extension Kit

b) LISA Four SOA Testing Suite - Products

b) LISA Four SOA Testing Suite - Products

LISA Server

LISA stands for "**Learn, Invoke, Simulate, Analyze,**" which describes the basic engine that allows the many types of tests LISA can build and run directly against the system under test. Sounds a little too scientific? Well, whatever your role in IT, LISA can test from design, to development, and throughout deployment to meet the full quality lifecycle needs of enterprise software and SOA applications.

iTKO's LISA Server automates and schedules LISA test suites, providing complex staging, user simulation and continuous test orchestration services for a constantly evolving SOA application environment.

Load and Performance Testing

Load testing has changed from an exercise in deployment to a vital step in the agile (iterative) development process. Waiting for a completed application to stress test at the user interface level is no longer an option. LISA allows for testing individual components, process, and workflows during design and development, during integration, and in deployment. Individual functional tests and system-wide business processes are load tested using the same environment, resulting in efficient test coverage, with rich functional and performance metrics and reports.

Virtual Service Environment

Coordinating and validating services produced and consumed in applications requires maximum flexibility of the platform. LISA provides virtualization of external systems providing consistent responses and reactions when testing SOA applications. Mock services are deployed when a system under test requires interaction with unavailable systems. Testing should not be limited to client calls into the system under test, validation of outbound service requests by virtual services is required to test the entire application.

- Fake services – LISA will create a web service Server and provide dummy SOAP response data to be used as a place holder
- Mock services – LISA will create a web service Server from WSDL and a LISA Test Case is used to validate (assert) inbound SOAP requests and respond with specific SOAP response (JDBC and .XLS data sources)
- Service Simulation – LISA invokes and gathers responses from a service, and creates a robust behavioral model of the service's behavior, including the underlying implementation and data layers underneath it, and creates a Virtual Service internal to the testing process that allows relevant testing to continue in absence of the implementation.

Virtual Test Beds support (at Fail-Over)

- System replay – new proxy level recording of web services and replay of Server SOAP responses.
- Deploy remote testing suites to simulate systems to create the test harness around an application. In many SOA test labs testing of 3rd party services may not be available, Ability to provide virtualization of the supporting infrastructure and provide read responses back.
- Automates virtual test environment staging at build time and change time

Continuous Validation Service

Catching system dependencies and the unintended consequences of changed service behavior is accomplished with LISA's Continuous Validation Service. Application and system validation starts with automated regression of the build lifecycle, through continuous validation of complex interwoven services and business processes. Automating validation through Continuous Build testing will capture regression errors before inter-service business processes are validated. Continuous Deployment validation of service usage and dependencies across multiple builds provides visibility into inter-component trust and governance.

c) LISA Enterprise

c) LISA Enterprise

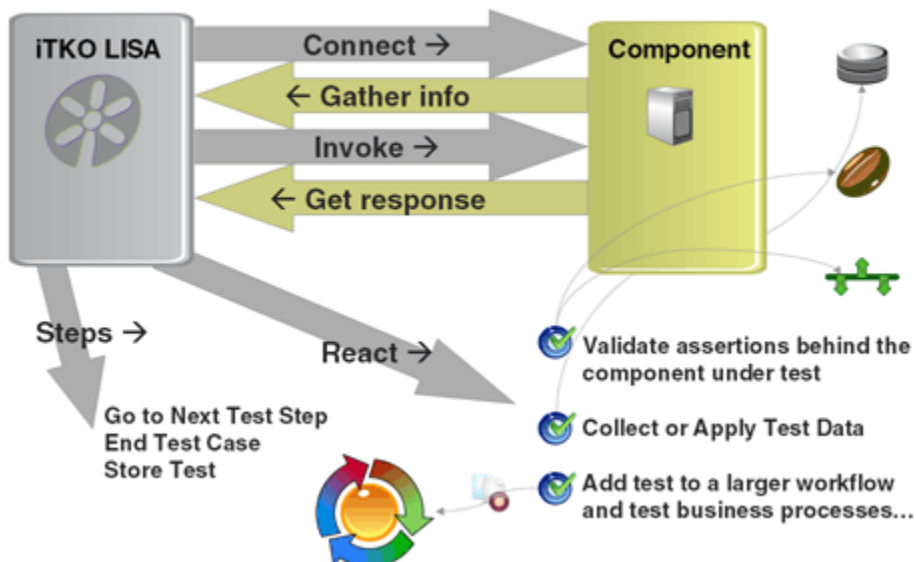
Functional Testing

LISA now allows QA, Development, and Business Analyst teams to test rich browser and web user interfaces, as well as the building blocks below the UI. With LISA, all of the components the team needs to functionally test can now be exercised with one tool in a codeless manner.

We have changed the way applications are written and function, and accordingly, we now have to change the way we functionally test the application. Functional testers are challenged to test new "headless" functions that have no interface, and more complex, heterogeneous integrations, while still validating the overall user experience.

The complete testing of an application can only be accomplished when every layer of the application is tested and validated by the business (and not just the developers who built it).

LISA Functional Testing



Integration Testing

Testing simple and complex system integrations has moved to the forefront of testing challenges. Out of the box, LISA provides a unique feature-rich environment to address testing of computer to computer integrations and customer to computer integrations. With native support for the major integration platforms like TIBCO, webMethods/SoftwareAG, BEA, Sonic, and IBM MQ/WebSphere, LISA provides the largest breadth of native integration testing capabilities, from unit and functional testing, through load and deployment testing.

Business Process Testing

Business Process Testing redefines the need for a strong collaborative testing platform. Business Process Testing is not only testing end to end processes and the complex systems that make up a business process, but it requires collaboration between the system designers and implementers. Business analysts who understand the flow of information have the ability to augment and tie Test Cases together gaining end to end testing of a process. LISA provides a rich feature set to allow collaboration of testing assets and data between developers and Analysts.

d) LISA Extension Kit (LEK)

d) LISA Extension Kit (LEK)

Every business has existing applications and business logic that needs to be leveraged. SOA and integration lifecycles involve heterogeneous technologies, so LISA gives you powerful extensibility features to bring testability to your legacy and custom-developed components. iTKO built LISA's core testing engines so new tests can be created to "learn" how to talk to your applications. In addition, the LISA Extension API allows developers to embed testability within custom or legacy code, providing rich test feedback and instructions from within the application under test. The LEK toolset has provided rapid testability for dozens of leading enterprises who previously thought there was no way to truly test all of the dependent layers of the architecture.

For additional documentation detailing the functionality available with the LEK, see the [LEK Developer's Guide](#)

For more information on LISA, visit the iTKO LISA official website at www.itko.com.

Technical Support & Contact Info

Technical Support & Contact Info

If you have a technical question or problem related to an iTKO product, check your service contract and/or support guidelines to find your regional support office location and contact information. Below is a summary of other [iTKO Customer Support options](#) as well as specific [iTKO Contact Information](#).

iTKO Customer Support options

LISA FAQ's

iTKO answers the most common test and configuration questions LISA customers face in our [FAQ's](#).

LISA Tech Notes

Deeper content on specific tips/tricks/pitfalls from the iTKO Support is available in our [Tech Notes](#).

Note: You must register and log in to see these topics.

LISA Forums

The iTKO LISA Forums at [Forums](#) is the one-stop-shop for LISA users to share and ask iTKO for answers on testing strategies and challenges encountered with LISA testing.

Note: You cannot log in to the Forums without a valid username and password. Please contact support@itko.com for a log in.

LISA Release Notes

iTKO LISA release specific information is available on <http://www.itko.com/downloads/ga>. You will need license information to view this page as it is password protected.

Contact iTKO

Direct support contact is available to current iTKO LISA Editions paid customers, through support@itko.com.

We encourage all customers to start at the above resources to ensure you get the fastest response possible.

iTKO World HQ Address :1505 LBJ Freeway Suite 250 Dallas, TX 75234 USA

Website : <http://www.itko.com>

Email : info@itko.com

Switchboard :877-BUY-ITKO (289-4856)

iTKO International :1-214-245-4361

Fax:(817) 281-2458

LISA Agent Guide

LISA Agent Guide

The LISA Agent is a piece of server-side technology that can get installed inside any java process (including J2EE containers) and allows LISA to control and monitor the server side of things.

In a nutshell, it can do what most profilers do (monitor loaded classes/objects, cpu usage, memory usage, threads, track method calls, etc...) but works across multiple JVMs and it is used in conjunction with LISA to bring unique features to the testing game.

The LISA Agent Guide is made up of the following chapters.

1. **LISA Agent Repository**
2. **LISA Agent User Guide**
3. **LISA Agent Architecture**
4. **LISA Agent Platforms**

1. LISA Agent Repository

LISA Agent Repository

The following chapters are available in this section.

- 1.1 Install
- 1.2 Alternate Install
- 1.3 Downloads

1.1 Install

1.1 Install

To turn on the LISA Agent on any java process, there are only 3 steps to follow:

Step 1. Download [LisaAgent.jar](#) and [\(lib\)JavaBinder\(.dll, .so, .jnilib\)](#) from the links below anywhere on your disk that has read permissions from your java process and is not automatically loaded in the application classpath (such as /WEB-INF/lib directories).

Step 2. Start your java process as you normally do after defining the following environment variable:

On Sun's jre 1.5+ or JRockit 1.5+:: `JAVA_TOOL_OPTIONS=-agentpath:<Path to JavaBinder.xxx>[url=<broker url>][name=<agent name>]`

On IBM's jre 1.5+:: `IBM_JAVA_OPTIONS=-agentpath:<Path to JavaBinder.xxx>[url=<broker broker>][name=<agent`


```
name>]
```

On Sun's jre 1.4:: `_JAVA_OPTIONS=-Xbootclasspath/a:<Path to LisaAgent14.jar> -Xdebug -Xnoagent -Djava.compiler=NONE -XrunJavaBinder:jar=file:<Path to LisaAgent14.jar>[,url=<broker url>][,name=<agent name>]`

On IBM's jre 1.4::

`IBM_JAVA_OPTIONS=-Xbootclasspath/a:<Path to LisaAgent14.jar> -Xdebug -Xnoagent -Djava.compiler=NONE -XrunJavaBinderIBM14:jar=file:<Path to LisaAgent14.jar>[,url=<broker url>][,name=<agent name>]`

Step 3. Start the broker: `java -jar LisaAgent.jar -broker [broker url]`

Note: setting `JAVA_TOOL_OPTIONS` in a shell will apply the agent to all java processes started in that shell. To avoid this, you can instead supply the arguments as a JVM option in the command-line. For examples, many J2EE containers support `JAVA_OPTS`, as in:
`JAVA_OPTS=-agentpath:<Path to JavaBinder.xxx>`

Important: if you specify `-agentpath:...` as a JVM argument instead of through an environment variable then the `jar` option is required.

Also note that with 1.4 JVMs the native library must be in the `PATH` (or `LD_LIBRARY_PATH`).

See the [LISA Agent User Guide](#) for more options and troubleshooting tips.

1.2 Alternate Install

1.2 Alternate Install

We now support a pure java agent for any jre 1.5 and above. It doesn't have the full functionality of the native-based agent but all of Pathfinder, Pathfinder Pro and Java Virtualize are available.

To install it follow the same steps as above but replace `(lib)JavaBinder(.dll, .so, .jnilib)` with `LisaAgen2.jar` and in **Step 2.** start your java process as you normally do after defining the following environment variable:

```
JAVA_TOOL_OPTIONS=-javaagent:<Path to LisaAgent2.jar>[,url=<broker url>][,name=<agent name>]
```

1.3 Downloads

1.3 Downloads

Cross-Platform Java Agent	LisaAgent.jar
Cross-Platform Pure Java Agent	LisaAgent2.jar
Cross-Platform Java Agent for java 1.4	LisaAgent14.jar
Windows (32 bit) Native Agent	JavaBinder.dll
Windows (32 bit) Native Agent (IBM java 1.4)	JavaBinderIBM14.dll
Mac OSX (x86) 32-bit (java 1.5) Native Agent	libJavaBinder.x86.jnilib
Mac OSX (x64) 64-bit (java 1.6) Native Agent	libJavaBinder.x64.jnilib
Linux (x86) 32-bit Native Agent	libJavaBinder.x86.so
Linux (x64) 64-bit Native Agent	libJavaBinder.x64.so
Solaris (x86) 32-bit Native Agent	libJavaBinder.solaris.x86.so
Solaris (x64) 64-bit Native Agent	libJavaBinder.solaris.x64.so
Solaris (Sparc) 32-bit Native Agent	libJavaBinder.solaris.sparc32.so
Solaris (Sparc) 64-bit Native Agent	libJavaBinder.solaris.sparc64.so
Sample configuration file	rules.properties

Agent client for LISA	lisa-agent.jar , _misc-agent.jar
Derby db and drivers	derby.jar , derbynet.jar , derbyclient.jar
mysql drivers	mysql.jar
postgres drivers	postgresql.jar

Note: Using this download site, you are responsible for backing up the files you overwrite in case you need to revert to a previous version later.

2. LISA Agent User Guide

2 LISA Agent User Guide

The following chapters are available in this section.

- [2.1 LISA Agent Overview](#)
- [2.2 Getting Started](#)
- [2.3 Configuring the Agent](#)
- [2.4 Using the Dev Console](#)
- [2.5 Developing against the Agent](#)
- [2.6 Custom Extensions](#)
- [2.7 Troubleshooting](#)

2.1 LISA Agent Overview

2.1 LISA Agent Overview

The LISA Agent is a piece of server-side technology that can get installed inside any java process (including J2EE containers) and allows LISA to control and monitor the server side of things.

In a nutshell, it can do what most profilers do (monitor loaded classes/objects, cpu usage, memory usage, threads, track method calls, etc...) but works across multiple JVMs and it is used in conjunction with LISA to bring unique features to the testing game.

In particular, it can give LISA users, a visibility into what each test or even test step causes the server(s) to do behind the scenes so as to identify bugs and bottlenecks. This is similar to what the current LISA Pathfinder does but works across all protocols used by java applications without the need to instrument any code or even any configuration files.

Another target area for the Agent is supporting VSE. By enabling record and replay of traffic and/or method calls across protocols, it gives VSE complete control over any areas of the target application that users may wish to virtualize, and provides a unified framework to accomplish this regardless of protocol.

Other feature areas have been identified but have not yet been publicized.

2.2 Getting Started

Getting Started

After starting an agent-enabled java process (see [Repository](#) we need to communicate with it. This is accomplished by using a JMS broker. If running standalone, the broker can be started with the command line: `java -jar LisaAgent.jar -broker [connection url]`

The only piece of configuration that is needed on the agent and console sides is to tell them where the broker is located, i.e. they need the broker connection string. If it is not specified, it will default to `tcp://localhost:2009`.

In addition, persistent information captured by the agents (such as transaction trees, statistics, etc...) is automatically flushed by the broker to a database that agent clients can later query if the data is too old to still be in memory.

What this database is configured through the [sink](#) property as detailed in the next section (see [Configuration](#)), but if no sink is configured then the broker will automatically start an embedded Derby database. For this to work, the Derby jars must be placed in the broker directory:

[derby.jar](#) and [derbynet.jar](#). Similarly the Derby client jars must be placed in the client jar directory: [derbynet.jar](#) and [derbyclient.jar](#).

Any other database that supports jdbc can be used to flush this information. The jdbc driver jars must be placed in the same directory as the Broker jar. The way that agents are told of where these databases reside is described in the next section.

See [Deployment](#) for a picture of these file requirements.

Startup Options:

url: if you want to specify a non default broker connection string you set the `url` parameter, as in: `JAVA_TOOL_OPTIONS=-agentpath:<Path to JavaBinder.xxx>=url=tcp://192.168.1.100:2009`

This combines with starting the broker on that connection string with: `java -jar LisaAgent.jar -broker tcp://192.168.1.100:2009`

name: the Agent can be given a name using the `name` parameter. This helps with readability (agents are displayed with that name in the UI) but also serves to identify processes across shutdowns and startups by keeping some persistent identifier:

`JAVA_TOOL_OPTIONS=-agentpath:<Path to JavaBinder.xxx>=name=myjboss`

token: if you want to make access to the Agent secure, you can use a password in the environment variable using the `token` parameter, as in:

`JAVA_TOOL_OPTIONS=-agentpath:<Path to JavaBinder.xxx>=token=password`

jar: if you want to put `LisaAgent.jar` and `JavaBinder(.dll, .lib, .so)` in different directories you can specify the location of the jar with:

`JAVA_TOOL_OPTIONS=-agentpath:<Path to JavaBinder.xxx>=jar=file:<Path to LisaAgent.jar>`

heap: can be set to true or false, is false by default. This option, available only with the native agent enables all the heap walking APIs (getting object instances, reference graphs, etc...) at a small performance cost (less than 5% for modern VMs).

ex: can be set to true or false, is false by default. This option, available only with the native agent enables the superstacks feature at a small performance cost (less than 5% for modern VMs).

All options can be combined in any way you want by delimiting them with commas, e.g.: `JAVA_TOOL_OPTIONS=-agentpath:<Path to JavaBinder.xxx>=url=tcp://orion:61616,name=myjboss,token=password`

2.3 Configuring the Agent

2.3 Configuring the Agent

The Agent makes the utmost effort to self-configure but you can tune or override parameters by placing a text file called **rules.properties** in the Agent jar directory. This file supports the following directives:

Setting a general property (see the sample rules.properties in the [Repository](#) for the full list):

`wsproperty key=<key> value=<value>`

e.g. `property key=TRANSACTION_BUFFER_SIZE value=10`

Setting a database sink for Agent persistent data. This will stay in use for the lifetime of the agent.

`sink driver=<Driver class> url=<jdbc url> user=<username> password=<password>`

e.g. `sink driver=org.postgresql.Driver url=jdbc:postgresql://localhost:5432/agtdb user=postgres password=postgres`

Adding a class for tracking (so instances can easily be retrieved from the heap later):

`track class=<class name>`

e.g. `track class=java.io.File`

Adding a method for interception (so its invocations will be tracked, either for pathfinder or VSE purposes):

`intercept class=<class name> method=<method name> signature=<signature>`

e.g. `intercept class=com.itko.lisa.test.Client method=method1 signature=*`

Excluding a method, class or package from interception and virtualization (note that exclusion takes precedence over interception and virtualization, but it works only for the classes specified and not their descendants):

`exclude class=<class name> method=<method name> signature=<signature>`

e.g. `exclude class=com.itko.lisa.test.Client method=* signature=*`

Adding a class for virtualization (both recording and playback modes):

`virtualize class=<class name>`

e.g. `virtualize class=javax.ejb.SessionBean`

Setting up a custom transform for virtualized method parameters and results before being XStream'ed ([this](#) and [that](#) are keywords with [this](#) referring to the object being transformed and [that](#) has a custom meaning depending on the object, usually its state):

`virtualize transform=<class name> code=[<code>]`

e.g. `virtualize transform=javax.servlet.http.HttpServletRequest code=[return this.getRequestURL().toString();]`

e.g. `virtualize transform=javax.servlet.http.HttpServletResponse code=[return that;]`

Telling the agent how to determine what constitutes a session for a given protocol by giving a code snippet that returns a session identifier:

```
virtualize track=<class name> method=<method name> signature=<signature> code=[<code>] push=<true|false>
e.g. virtualize track=javax.servlet.http.HttpServlet method=service
signature=(Ljavax/servlet/http/HttpServletRequest;Ljavax/servlet/http/HttpServletResponse;)V code=[return
$1.getSession().getId();] push=false
virtualize track=javax.ejb.SessionBean method=setSessionContext signature=(Ljavax/ejb/SessionContext)V
code=[return $1.getEJBObject().getHandle().toString();] push=true
virtualize track=javax.ejb.EntityBean method=setEntityContext signature=(Ljavax/ejb/EntityContext)V
code=[return $1.getPrimaryKey().toString();] push=true
```

The above 3 lines are already hardcoded in the agent thus unnecessary in your [rules.properties](#) but show how it works so it can be done for any number of protocols other than HTTP and EJB without recompiling the agent.

The value of the [track](#) parameter is the class from which we gain access to a session, the value of the [method](#) and [signature](#) parameters determine the method that, when invoked, will compute the session identifier for us, using the value of the [code](#) parameter (in which [\\$0](#) represents the source object and [\\$1](#), [\\$2](#), ... are the method arguments).

Finally, the push parameter decides how we store that session for later use by VSE frames: [push=false](#) means the session is basically thread-scoped and we store in a thread-local variable, whereas [push=true](#) means the session is set for us by the container and we keep a mapping of object to session. The innermost session (identifier) will be served back through VSE in the [com.itko.lisa.remote.vse.VSEFrame.getSessionId\(\)](#) method.

Extending the agent with custom code - custom interceptors:

```
interceptor class=<class name>
```

If you package java classes in jar files located in the agent directory, those can be made available to the agent. In particular, if a class implements [com.itko.lisa.remote.transactions.interceptors.IInterceptor](#) and is specified as above with an [interceptor](#) directive, its [block](#), [preProcess](#) and [postProcess](#) methods will be invoked as hooks.

Extending the agent with custom code - custom formatters:

```
formatter class=<class name> name=<formatter name>
```

Note - A simple JDBC invoke program can not be recorded until program attached to Agent starts the pathfinder path. This can be achieved by updating the [rules.properties](#) file to look for a specific method being invoked to start the path.

Limitation - The LISA Agent does not intercept methods in inner classes.

2.4 Using the Dev Console

2.4 Using the Dev Console

A full client console is being built as part of LISA Workstation. Until it is completed, you can use the **unsupported** Dev Console to see what the agent is doing. It is packaged inside of the [LisaAgent.jar](#) file and requires the database jars to be downloaded in the agent directory for db access operations (see the [Repository](#) page for downloads).

There are several command-line options available, all of which can be accessed by running:

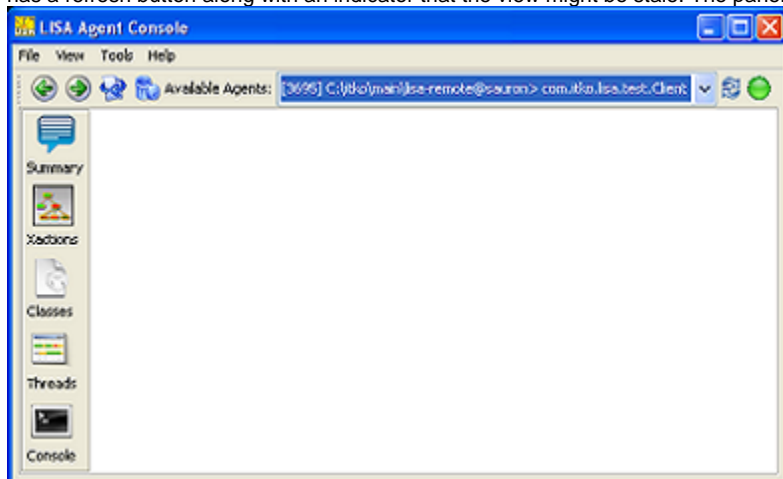
```
java -jar LisaAgent.jar -help
```

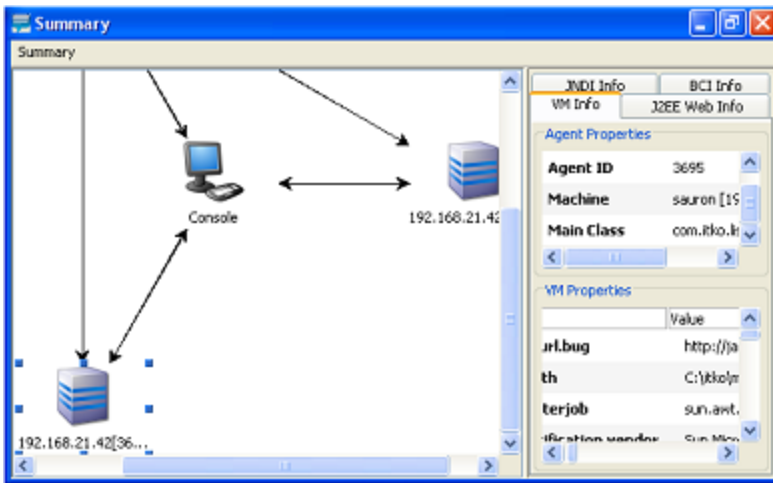
The options relevant to this section are:

```
java -jar LisaAgent.jar -console [connection url] to run the standalone console
```

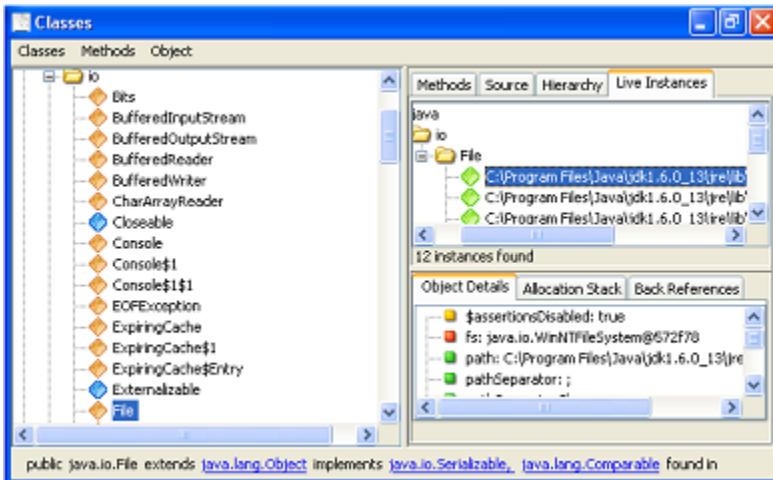
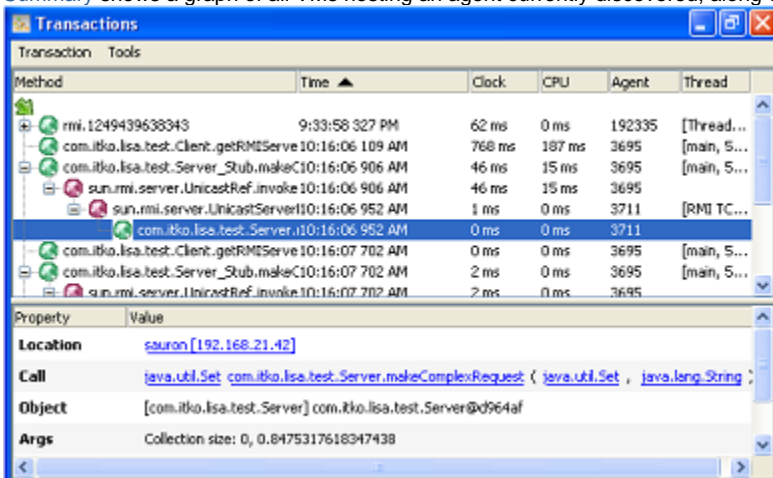
```
java -jar LisaAgent.jar -dev [connection url] to run the console with an embedded broker
```

The console has a toolbar and 5 panels. The toolbar (dropdown) lets you specify which agent the panels are currently showing information for and has a refresh button along with an indicator that the view might be stale. The panels are:

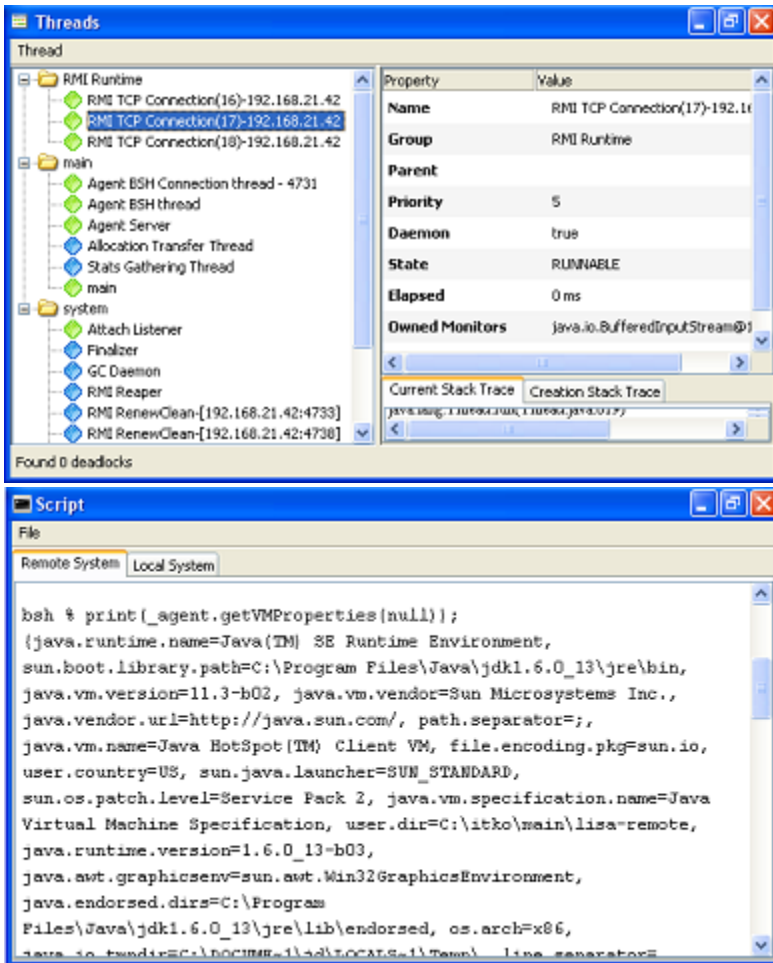




Summary shows a graph of all VMs hosting an agent currently discovered, along with some details about each (System properties, etc...)



- **Transactions** is the set of transaction trees retrieved from the specified agent sink
- **Classes** lets you browse the full class hierarchy loaded in the JVM (by package or by archive), including source code and live object instances



- **Threads** shows a current thread stack dump of JVM and a lot of information about each thread (including deadlocks).
- **Console** is a beanshell terminal for the local and remote JVMs. The variable `_agent` refers to the [com.itko.lisa.remote.IAgent](#) interface.

2.5 Developing against the Agent

2.5 Developing against the Agent

The main interface to the agent from the client side is [com.itko.lisa.remote.client.AgentClient](#). It has methods to invoke APIs on the Agents or the Broker, to discover Agents and to be notified of their status changes (online/offline).

It also gives access to the classes responsible for Agent interaction in the main areas of functionality:

[com.itko.lisa.remote.client.AgentClient](#)
[com.itko.lisa.remote.client.DiscoveryClient](#)
[com.itko.lisa.remote.client.TransactionsClient](#)
[com.itko.lisa.remote.client.VSEClient](#).

General APIs

Most of the client APIs need to specify an agent as their target. This is accomplished by passing a parameter of type [com.itko.lisa.remote.IAgentInfo](#). This represents an object that uniquely identifies an agent as well as some of its basic information:

```

/** A unique identifier for this agent or console. If it is named the guid is persistent across VM
lifespans */
public long getGuid();
public void setGuid(long guid);

/** A human-readable name to identify this agent, manually given or generated from system properties
*/
public String getName();
public void setName(String name);

/** The name of the machine this object was generated on (if available) */
public String getMachine();
public void setMachine(String machine);

/** The IP of the machine this object was generated on (if available) */
public String getIp();
public void setIp(String ip);

/** The working directory of the JVM this object runs in */
public String getWorkingDir();
public void setWorkingDir(String workingDir);

/** The classpath as it is returned by the java.class.path property */
public String getClasspath();
public void setClasspath(String classpath);

/** The library path as it is returned by the java.library.path property */
public String getLibpath();
public void setLibpath(String libpath);

/** For agents, returns the java class containing the main method that was invoked */
public String getMainClass();
public void setMainClass(String mainClass);

/** A short-hand version of the command-line (for representation purposes as it may not be accurate)
*/
public String getCommandLine();

/** Transient field to keep track of when this object is sent or received */
public Date getGenerationTime();
public void setGenerationTime(Date time);

/** Transient field to keep track of a required password to invoke APIs on this agent over JMS */
public String getToken();
public void setToken(String token);

```

We can now look at some of the APIs directly off [com.itko.lisa.remote.client.AgentClient](#). This list is not exhaustive but should cover most of the needs of most clients:

```

/** Gets the class responsible for all discovery and information for a given agent */
public DiscoveryClient getDiscoveryClient();

/** Gets the class responsible for all transaction related operations */
public TransactionsClient getTransactionClient();

/** Gets the class responsible for all vse related operations */
public VSEClient getVSEClient();

/**
 * Get all the discovered agents - this is the main API to get IAgentInfos used in all other API
calls
 * @param tokens a map of agent guids or names to tokens, null if no agent has token-enabled security
 * @return a set of IAgentInfo objects representing agents that are currently online
 */
public Set getRemoteAgentInfos(Map tokens);

/**
 * Forward agent online information to registered listeners
 * @param info the agent that was just detected to come online
 */
public void onAgentOnline(IAgentInfo info);

/**
 * Forward agent offline information to registered listeners
 * @param info the agent that was just detected to go offline
 */
public void onAgentOffline(IAgentInfo info);

/**
 * Evaluate arbitrary code on the specified agent
 * @param info the agent this code will be evaluated against
 * @param input the code to execute. The variable '_agent' represents the Agent instance.
 * @return the value returned by the code
 * @throws JMSInvocationException if the code throws on the agent
 */
public Object eval(IAgentInfo info, String input) throws JMSInvocationException

```

Discovery APIs

The [com.itko.lisa.remote.client.DiscoveryClient](#) class can be obtained with a call to `AgentClient.getInstance().getDiscoveryClient()`; and provides methods pertaining to discovering data on an agent:

```

/**
 * The system properties of the specified agent
 * @param info
 * @return
 * @throws JMSInvocationException
 */
public Map getVMPProperties(IAgentInfo info) throws JMSInvocationException;

/**
 * Returns a list of StatsFrames recorded for the specified agent between the from and the to dates.
 * @param agentInfo the agent for which to retrieve statistics
 * @param from how far back to filter
 * @param to how recently to filter
 * @return the desired StatsFrames list ordered by decreasing time (starting at the to date)
 */
public List getStatistics(IAgentInfo agentInfo, Date from, Date to);

/**
 * Returns a list of StatsFrames recorded for the specified agent between the from and the to dates.
 * @param agentInfo the agent for which to retrieve statistics
 * @param from how far back to filter
 * @param to how recently to filter
 * @param interval how to aggregate the results in seconds. 10 means average the results of every 10

```

```

secs, etc...
    * @param limit        the maximum number of results
    * @return              the desired StatsFrames list ordered by decreasing time (starting at the to date)
    */
public List getStatistics(IAgentInfo agentInfo, Date from, Date to, int interval, int limit);

/**
 * TODO: (re)implement - currently will throw
 * @param info
 * @return
 * @throws JMSInvocationException
 */
public Topology getTopology(IAgentInfo info) throws JMSInvocationException;

/**
 * Exit points are MethodInfo that capture classes/methods that make network calls down the stack
 * @param info
 * @return
 * @throws JMSInvocationException
 */
public Set getExitPoints(IAgentInfo info) throws JMSInvocationException;

/**
 * Returns the name of the J2EE container (or java if it's not a j2ee container)
 * @param info
 * @return
 * @throws JMSInvocationException
 */
public String getServerInfo(IAgentInfo info) throws JMSInvocationException;

/**
 * Returns the web applications deployed in the specified j2ee container
 * @param info
 * @return
 */
public WebApplication[] getWebApps(IAgentInfo info) throws JMSInvocationException;

/**
 * Returns the JNDI hierarchy on the specified agent represented by a ClassNode tree
 * @param info
 * @return
 * @throws JMSInvocationException
 */
public ClassNode getJNDIRoot(IAgentInfo info) throws JMSInvocationException;

/**
 * The current threads on the agent VM
 * @param info
 * @return
 * @throws JMSInvocationException
 */
public ThreadInfo[] getThreadInfos(IAgentInfo info) throws JMSInvocationException;

/**
 * The current threads stacks on the agent VM
 * @param info
 * @return
 */
public String[] dumpThreads(IAgentInfo info) throws JMSInvocationException;

/**
 * The set of all files in the classpath of the specified agent
 * @param info
 * @return
 * @throws JMSInvocationException
 */
public Set getClasspath(IAgentInfo info) throws JMSInvocationException;

/**
 * Returns the class hierarchy under the specified path

```



```

    * @param info
    * @param fromPath
    * @return
    */
    public ClassNode getClassNodes(IAgentInfo info, String fromPath) throws JMSInvocationException;

    /**
     * The class hierarchy found in the archive at the given url
     * @param info
     * @param url
     * @return
     * @throws JMSInvocationException
     */
    public ClassNode getArchiveNodes(IAgentInfo info, URL url) throws JMSInvocationException;

    /**
     * A set containing data about the class (fields/methods/src)
     * @param info
     * @param className
     * @return
     * @throws JMSInvocationException
     */
    public Set getClassInfo(IAgentInfo info, String className) throws JMSInvocationException;

    /**
     * Decompile and return the source to a class
     * @param info
     * @param clazz
     * @param loc decompile on the client or in the agent
     * @return
     * @throws JMSInvocationException
     */
    public String getClassSrc(IAgentInfo info, String clazz, boolean loc) throws JMSInvocationException,
    IOException;

    /**
     * Returns the hierarchy this class belong to, i.e. all ancestors but also all extenders/implementers
     * @param info
     * @param className
     * @return
     * @throws JMSInvocationException
     */
    public ClassNode[] getClassHierarchy(IAgentInfo info, String className) throws
    JMSInvocationException;

    /**
     * Returns (references to) all objects in the heap of the specified class - use with caution
     * @param info
     * @param className
     * @return
     * @throws JMSInvocationException
     */
    public ClassNode getInstancesView(IAgentInfo info, String className) throws JMSInvocationException;

    /**
     * Returns (references to) all objects on the heap tracked by the agent
     * @param info
     * @return
     * @throws JMSInvocationException
     */
    public ClassNode getTrackedObjects(IAgentInfo info) throws JMSInvocationException;

    /**
     * A crude graph representation of an object (recursively computed fields)
     * @param info
     * @param clazz
     * @param hashCode
     * @return
     * @throws JMSInvocationException
     */

```

```
public ClassNode getObjectGraph(IAgentInfo info, String clazz, int hashCode) throws
JMSInvocationException;

/**
 * Gets the path from an object to a GC root
 * @param info
 * @param clazz
 * @param hashCode
 * @return
 * @throws JMSInvocationException
 */
public ClassNode getRootPath(IAgentInfo info, String clazz, int hashCode) throws
JMSInvocationException;

/**
 * Gets a file on the agent filesystem, downloads it to the client in a temp location and return a
handle to it
 * @param info
 * @param file
 * @return
 * @throws JMSInvocationException
```

```

    */
    public File getFile(IAgentInfo info, String file) throws JMSInvocationException, IOException;

```

In the statistics related APIs above, the results are lists of [com.itko.lisa.remote.stats.StatsFrame](#) objects, which are POJOs used to hold various counter values (more platform-specific - ala vmstat/iostat/netstat/sar/... - may be added in the future):

```

/** The Agent Id this statistics frame was recorded for */
public long getAgentId();
public void setAgentId(long agentId);

/** The time at which this frame was recorded */
public long getTime();
public void setTime(long time);

/** The total CPU usage for this process during the last sampling interval */
public long getCpuUsage();
public void setCpuUsage(long cpuUsage);

/** The total memory consumed by the heap at the time this was recorded */
public long getHeapUsage();
public void setHeapUsage(long heapUsage);

/** The total memory consumed by non-heap resources at the time this was recorded */
public long getNonHeapUsage();
public void setNonHeapUsage(long nonHeapUsage);

/** The number of bytes received (over the network) during the last sampling interval */
public long getIoIn();
public void setIoIn(long ioIn);

/** The number of bytes emitted (over the network) during the last sampling interval */
public long getIoOut();
public void setIoOut(long ioOut);

/** The number of garbage collection events during the last sampling interval */
public long getGcCount();
public void setGcCount(long gcCount);

/** The time spent garbage collecting during the last sampling interval */
public long getGcTime();
public void setGcTime(long gcTime);

```

Note that in all APIs above, the sampling interval's default value is 1 second and can be modified in the [rules.properties](#) using the key [STATS_SAMPLING_INTERVAL](#). Other statistics configuration properties can be tuned as documented in the [rules.properties](#).

Transaction APIs

A transaction is a code path executed by one or more servers as the result of a client request. It is represented by a tree structure rooted at the client initiating the request, and nodes of that tree are [com.itko.lisa.remote.transactions.TransactionFrame](#) objects.

They encapsulate information about a class, method and arguments that were invoked as part of the server processing. They also contains ancillary information, such as duration, time of execution, thread in which it occurred, etc...

You can think of a transaction as a method call stack, the main differences being that transactions cross thread, process or even machine boundaries and stacks contains all methods involved in a thread's code execution whereas transactions skip some levels and have frames only for chosen method of interest (those are called intercepted methods).

The main way to obtain and work with those TransactionFrames is through a few overloads of the following APIs supplied by [com.itko.lisa.remote.client.TransactionsClient](#), which in turn can be obtained with the following call:

```
AgentClient.getInstance().getTransactionsClient();
```

```

/**
 * Start recording transactions
 * @param info the agent to start recording on
 */
public void startXRecording(IAgentInfo info) throws JMSInvocationException;

/**

```

```

    * Stop recording transactions
    * @param info the agent to stop recording on
    */
    public void stopXRecording(IAgentInfo info) throws JMSInvocationException;

    /**
     * Start tracking socket usage on the client to use in reconciling client and server transactions.
     * This must be called prior to the call we're adding in addClientTransaction.
     * @param global install on all sockets or only on this thread
     */
    public void installSocketTracker(boolean global);

    /**
     * Stop tracking socket usage on the client to use in reconciling client and server transactions.
     * This should be called after the call we're adding in addClientTransaction.
     * @param global uninstall on all sockets or only on this thread
     */
    public void uninstallSocketTracker(boolean global);

    /**
     * As a client, you can invoke this method to root a transaction tree at a new client transaction
     created using
     * the specified parameters.
     * Note: you must call installSocketTracker before you initiate the client side transaction you're
     adding here
     * and it is recommended you call uninstallSocketTracker after you're done with the network call.
     * @param stamp a unique string you can later use to identify the root transaction
     * (in calls to getTransactions for ex.)
     * @param stepInfo a nice human-readable string that tells what the transaction is doing (can be
     null)
     * @param args the parameters the client passes to the transaction (can be empty)
     * @param result the result of the transaction (can be null)
     * @param duration the client's view of the transaction duration
     */
    public void addClientTransaction(String stamp, String stepInfo, Object[] args, Object result, long
    duration);

    /** Delete all transactions and dependent data that originated from this client. */
    public void clearTransactions();

    /**
     * Gets a flat list of TransactionFrames received from all agents that satisfy the filters passed as
     arguments
     * @param offset offset
     * @param limit max number of results
     * @param category filter by transaction category (see TransactionFrame.CATEGORY_XXX - 0 for no
     filter)
     * @param clazz filter by class name (null or "" for no filter)
     * @param method filter by method name (null or "" for no filter)
     * @param minTime filter by frame duration greater than minTime
     * @return a list of TransactionFrames satisfying the supplied criteria ordered by
     decreasing time
     */
    public List getTransactions(int offset, int limit, int category, String clazz, String method, int
    minTime);

    /**
     * Get a list of transaction trees rooted at the specified transaction(s) and satisfying the
     specified criteria
     * @param offset offset
     * @param limit max number of results
     * @param stamps an array of root client transaction stamps - returns all if this is empty
     * @param minTime duration below which transactions are pruned out of the results
     * @return ret a list of transaction trees matching the criteria ordered by decreasing time

```

```

    */
    public List getTransactionsTree(int offset, int limit, String[] stamps, int minTime)

```

Note that the parameters to these APIs do not specify an Agent or AgentInfo because transactions can span multiple Agents. Those APIs returns list of [com.itko.lisa.remote.transactions.TransactionFrame](#) (or trees thereof) so let's look at what data they encapsulate:

```

/** A unique identifier for this frame */
public String getFrameId();
public void setFrameId(String frameId);

/** The frame id of this frame's parent frame
public String getParentId();
public void setParentId(String parentId);

/** An identifier shared by all frames belonging to the same transaction (same as global root frame
id) */
public String getTransactionId();
public void setTransactionId(String transactionId);

/** The parent TransactionFrame object */
public TransactionFrame getParent();
public void setParent(TransactionFrame parent);

/** The list of child TransactionFrame objects */
public List getChildren();
public void setChildren(List children);

/** The unique identifier of the Agent this frame was recorded in */
public long getAgentGuid();
public void setAgentGuid(long agentGuid);

/** Increasing counter that helps order the frames (time may not be precise enough) */
public long getOrdinal();
public void setOrdinal(long ordinal);

/** Network incoming or outgoing frames set this to tell us what IP they are talking from */
public String getLocalIP();
public void setLocalIP(String ip);

/** Network incoming or outgoing frames set this to tell us what port they are talking from */
public int getLocalPort();
public void setLocalPort(int port);

/** Network incoming or outgoing frames set this to tell us what IP they are talking to */
public String getRemoteIP();
public void setRemoteIP(String ip);

/** Network incoming or outgoing frames set this to tell us what port they are talking to */
public int getRemotePort();
public void setRemotePort(int port);

/** The name of the thread this frame was recorded in */
public String getThreadName();
public void setThreadName(String threadName);

/** Name of the class or interface this frame was recorded in (as per the interception spec) */
public String getClassName();
public void setClassName(String className);

/** Name of the class of the actual object this frame was recorded in */
public String getActualClassName();

/** Name of the method this frame was recorded in */
public String getMethod();
public void setMethod(String method);

/** Signature of the method this frame was recorded in */

```

```

public String getSignature();
public void setSignature(String signature);

/** Formatted representation of the object this frame was recorded in */
public String getSource();
public void setSource(Object source);

/** Formatted representation of the arguments to the method this frame was recorded in */
public String[] getArguments();
public void setArguments(Object[] arguments);

/** Formatted representation of the result of the method this frame was recorded in */
public String getResult();
public void setResult(Object result);

/** Number of times this frame was duplicated within its parent */
public long getHits();
public void setHits(long hits);

/** Server time at which the frame was recorded */
public long getTime();
public void setTime(long time);

/** Wall clock duration this frame took to execute */
public long getClockDuration();
public void setClockDuration(long clockDuration);

/** CPU duration this frame took to execute */
public long getCpuDuration();
public void setCpuDuration(long cpuDuration);

/** Custom representation of state associated with this frame */
public String getState();
public void setState(Object state);

/** Formatted LEK info as encoded/decoded by the LEKEncoder class */
public String getLekInfo();
public void setLekInfo(String lekInfo);

/** Pre-computed category this frame belongs to - see TransactionFrame.CATEGORY_XXX */
public int getCategory();
public void setCategory(int category);

/** Bitwise or'ed combination of various internal pieces of information - see
TransactionFrame.FLAG_XXX */

```

```
public long getFlags();  
public void setFlags(long flags);
```

VSE APIs

VSE stands for Virtualized Service Environment. It enables users to stub out processes, services, or parts of them along well-defined boundaries and replace their internals with a layer run by LISA according to custom user-defined rules - a.k.a a model. Usually a default starting point for those rules is obtained via a recording of live system interactions.

LISA already supports **VSE** for the HTTP protocol (allowing virtualization of web applications and web services), JMS, and JDBC to a certain extent.

The Agent provides APIs to enable virtualization directly from within server processes, thus making it protocol-agnostic. It can be enabled of course for HTTP, JMS and JDBC but also for RMI, EJB or any custom java objects.

LISA (or any other client of the Agent) can achieve this virtualization by using the following APIs defined in **com.itko.lisa.remote.client.VSEClient** as obtained by `AgentClient.getInstance().getVSEClient();`:

```

/**
 * Returns a list of all class/interface names whose name matches the supplied regular expression
 * or that extend/implement a class/interface whose name matches the supplied regular expression
 * if implementing is true. Searching for annotations is supported through the syntax:
 * class regex@annotation regex (e.g. ".*.Remote@.*.Stateless").
 * @param agentInfo
 * @param regex
 * @param impl
 * @return
 */
public String[] getMatchingClasses(IAgentInfo info, String regex, boolean impl) throws
JMSInvocationException

/**
 * Register a VSE callback with the specified agent whose onFrameRecord will be invoked
 * in recording mode for all virtualized methods and whose onFramePlayback method will be invoked
 * in playback mode for all virtualized methods.
 * @param info
 * @param callback
 */
public void registerVSECallback(IAgentInfo info, IVSECallback callback);

/**
 * Unregister a VSE callback with the specified agent.
 * @param info
 * @param callback
 */
public void unregisterVSECallback(IAgentInfo info, IVSECallback callback);

/**
 * Start calling our virtualization recording callback on the specified agent.
 * @param agentInfo
 * @throws RemoteException
 */
public void startVSERecording(IAgentInfo agentInfo) throws JMSInvocationException

/**
 * Start calling our virtualization playback callback on the specified agent.
 * @param agentInfo
 * @throws RemoteException
 */
public void startVSEPlayback(IAgentInfo agentInfo) throws JMSInvocationException

/**
 * Stop virtualizing on the specified agent.
 * @param agentInfo
 * @throws RemoteException
 */
public void stopVSE(IAgentInfo agentInfo) throws JMSInvocationException

/**
 * Virtualizes the specified class/interface and all its descendants on the specified agent.
 * @param agentInfo
 * @param className
 * @return
 */
public void virtualize(IAgentInfo agentInfo, String className) throws JMSInvocationException

```

The interface for the callback APIs is defined by [com.itko.lisa.remote.vse.IVSECallback](#) and defines the following methods:


```

/**
 * This is the method that gets invoked by agents that have VSE recording turned on
 * when a virtualize method gets called. The VSE frame has all the information needed
 * to later replay the method in playback mode.
 * @param frame
 * @throws RemoteException
 */
void onFrameRecord(VSEFrame frame) throws RemoteException;

/**
 * This is the method that gets invoked by agents that have VSE playback turned on
 * when a virtualize method gets called. The VSE frame has all the information needed
 * to match an existing recorded frame so its result (and by reference arguments)
 * can be set appropriately.
 * @param frame
 * @return
 * @throws RemoteException
 */
VSEFrame onFramePlayback(VSEFrame frame) throws RemoteException;

```

Finally, the [com.itko.lisa.remote.vse.VSEFrame](#) object is a POJO with getters and setters for the following properties:

```

/** Get/sets a unique identifier for this frame */
public String getFrameId();
public void setFrameId(String frameId);

/** The agent id this frame originates from */
public long getAgentGuid();
public void setAgentGuid(long agentId);

/** The thread name this frame method was invoked on */
public String getThreadName();
public void setThreadName(String threadName);

/** The name of the class this frame method was invoked on */
public String getClassName();
public void setClassName(String className);

/** A unique identifier that tracks objects for the span of the VM's life */
public String getSourceId();
public void setSourceId(String srcId);

/** The session id of the innermost session-scoped protocol enclosing this frame */
public String getSessionId();
public void setSessionId(String sessionId);

/** The name of the method that was invoked */
public String getMethod();
public void setMethod(String method);

/** The XStream'ed arguments array to the method that was invoked */
public String[] getArgumentsXML();
public void setArgumentsXML(String[] argumentsXML);

/** The XStream'ed result of the method that was invoked */
public String getResultXML();
public void setResultXML(String resultXML);

/** The (server) time the method was invoked */
public long getTime();
public void setTime(long time);

/** The time the method took to execute */
public long getClockDuration();
public void setClockDuration(long duration);

/** Whether to use getCode or the ResultXML to compute the desired result in playback mode */
public boolean isUseCode();
public void setUseCode(boolean useCode);

/**
 * The code to execute on the server if isUseCode is true. This can be arbitrary code
 * that has access to the object ($0) and method arguments ($1, $2,...)
 */
public String getCode();
public void setCode(String code);

```

LEK APIs

LEK stands for LISA Extension Kit. It enables client to write code in their application that will be automatically be picked up by the agent and be sent back to LISA for further processing.

This generation of the **LEK** is designed in such a way that it requires no dependencies whatsoever on LISA code or files, thus the application can still run without the agent with no impact.

The generic way to make a LEK call is by creating a `java.util.Properties` object and setting some well-known keys on it. This is better understood using an example:

```

Properties p = new Properties();

/** Tell LISA to issue a log message event with message "The quick brown fox" upon step completion */
p.put("lisa.log.msg", "The quick brown fox");

/** Tell LISA to generate a warning event, including the stack trace of the supplied exception */
p.put("lisa.log.warning", new Object[] { "jumped over the lazy dog", new RuntimeException("oops") });

/** Tell LISA to raise a property set event using the specified key-value pair */
p.put("lisa.set.prop", new Object[] { "lek", "cool" });

/** Sets the next node in the flow of the current LISA test (thus tests can be remote controlled
server-side) */
p.put("lisa.set.next", "abort");

/** Raises a LISA event using a TestEvent id, message, description and optionally a next node */
p.put("lisa.raise.event", new Object[] { 20, "short msg", "long msg" });

```

The last call is the general form of the other ones, which are just provided for convenience:

```

new Properties().put("lisa.raise.event", new Object[] { eventId, message, description, nextNode });

```

The message and description fields need not be strings. If they are throwables, their stack trace will be returned, if they are non-string objects, their XML representations (XStream'ed) will be returned.

The list of well-know keys is as follows (subject to change before release):

[lisa.log.msg](#) (Event ID = 21)

[lisa.log.warning](#) (Event ID = 41)

[lisa.set.prop](#) (Event ID = 14)

[lisa.set.next](#) (Event ID = 0)

[lisa.raise.event](#) (Event ID = *)

The [rules.properties](#) file contains a configurable boolean property called [LEK_LOG_CALLS](#) that controls whether LEK API calls will generate log messages on the server side.

As a stylistic note, the following code is equivalent to what was shown above (and some may prefer it):

```

System.getProperties().put("lisa.raise.event", new Object[] { eventId, message, description, nextNode
});

```

Finally, if client code makes extensive use of these LEK calls, it may be beneficial to create a tiny wrapper class that wraps these calls to help eliminate typos (the price of no jar dependencies):

```

public class LEK {
    public void log(Object msg, Object desc) {
        new Properties().put("lisa.log.msg", new Object[] { msg, desc });
    }

    public void warn(Object msg, Object desc) {
        new Properties().put("lisa.log.warning", new Object[] { msg, desc });
    }

    public void setProperty(String key, Object value) {
        new Properties().put("lisa.set.prop", new Object[] { key, value });
    }

    public void setNext(String next) {
        new Properties().put("lisa.set.next", next);
    }

    public void raiseEvent(int id, String msg, Object desc, String next) {
        new Properties().put("lisa.raise.event", new Object[] { id, msg, desc, next });
    }
}

```

On the LISA side, when a [com.itko.lisa.remote.transactions.TransactionFrame](#) tree is received, all the LEK information that was generated on the server can be retrieved from it using the [com.itko.lisa.remote.lek.LEKEncoder](#) class, which has the following relevant APIs:

```

/**
 * Extracts LEK data from a frame (if any) and deserializes it as a list of LEKEvent objects.
 * Note this extracts LEK data only for the supplied frame and not its hierarchy.
 * @param frame
 * @return
 */
public static List decode(TransactionFrame frame);

/**
 * Finds the next node as set in a transaction tree using total node ordering on the tree
 * defined by left to right and top to bottom
 * (essentially - but not necessarily exactly - following the code timeline that generated this
 * tree).
 * @param frame
 * @return
 */
public static String findNext(TransactionFrame frame);

/**
 * Returns a list of all LEKEvent objects generated and stored on the supplied transaction tree
 * (in timeline order - see findNext).
 * This is essentially the same as decode but extracts LEK data for the whole hierarchy.
 * @param frame
 * @return
 */
public static List decodeAll(TransactionFrame frame);

```

The results returned by those APIs are encapsulated in POJOs called [com.itko.lisa.remote.lek.LEKEvent](#). Those are a lightweight version of LISA's TestEvent objects:

```

/** The TestEvent id that was used when invoking the "lisa.raise.event" LEK call */
public int getEvtId();
public void setEvtId(int evtId);

/** The xml-ized (if need be) message
public String getMsg();
public void setMsg(String msg);

/** The xml-ized (if need be) description
public String getDesc();
public void setDesc(String desc);

/** Optionally a next next node */
public String getNext();
public void setNext(String next);

/** Key-value pairs used in "lisa.set.prop" calls */
public String getKey();
public String getValue();
public void setKeyValue(String key, String value);

```

Examples

Example 1: generating a transaction from client code and retrieving the transaction tree it generated

```

private static void testAddTransaction() throws Exception
{
    String request = "http://localhost:8080/examples/servlets/servlet/HelloWorldExample";
    TransactionsClient tc = AgentClient.getInstance().getTransactionClient();

    tc.installSocketTracker(false);
    long start = System.currentTimeMillis();

    String response = testMakeRequest(request);

    long end = System.currentTimeMillis();
    tc.uninstallSocketTracker(false);

    String frameId = tc.addClientTransaction("test", new Object[] { request }, response, end - start);
    TransactionFrame frame = tc.getTransactionTree(frameId);

    System.out.println(frame.isComplete() ? "Yes!" : "No!");
}

```

This sample makes use of the following utility function which has nothing to do with the agent but we list it for completeness sake.

```

/** Assuming Tomcat is running on localhost:8080 */
private static String testMakeRequest(String url) throws IOException
{
    StringBuffer response = new StringBuffer();
    HttpURLConnection con = (HttpURLConnection) new URL(url).openConnection();

    con.setRequestMethod("GET");
    con.setDoOutput(true);
    con.setUseCaches(false);

    BufferedReader br = new BufferedReader(new InputStreamReader(con.getInputStream()));
    for (String line = br.readLine(); line != null; line = br.readLine()) response.append(line);
    br.close();

    return response.toString();
}

```

Example 2: registering a logging VSE callback

```

AgentClient.getInstance().addListener(new IAgentEventListener()
{
    public void onAgentOffline(final IAgentInfo info) {}
    public void onAgentOnline(final IAgentInfo info)
    {
        AgentClient.getInstance().getVSEClient().registerVSECallback(info, new IVSECallback()
        {
            public void onFrameRecord(VSEFrame frame) { System.out.println("Recorded: " + frame); }
            public VSEFrame onFramePlayback(VSEFrame frame) { System.out.println("Played back: " + frame); }
        });
        return frame; }
    public int hashCode() { return 0; }
    public boolean equals(Object o) { return o instanceof IVSECallback; }
});

try { AgentClient.getInstance().getVSEClient().startVSERecording(info); } catch
(JMSInvocationException e) {}
};

```

More Details Coming...

2.6 Custom Extensions

2.6 Custom Extensions

It is possible (and easy) to customize the agent behavior by writing java classes that implement the [com.itko.lisa.remote.transactions.interceptors.IInterceptor](#) interface or extend the [com.itko.lisa.remote.transactions.interceptors.AbstractInterceptor](#) class:

```
public interface IInterceptor
\{
    /**
     * Whether this custom interceptor is currently disabled
     */
    public boolean isDisabled();

    /**
     * Returns whether the interception should return (true) or proceed (false)
     */
    public boolean block(boolean wayIn,
        Object src,
        String spec,
        String clazz,
        String method,
        String signature,
        Object args[],
        Object ret);

    /**
     * Called after method entry to possibly modify the current frame based on interceptor logic.
     */
    public boolean preProcess(TransactionFrame frame,
        Object src,
        String spec,
        String clazz,
        String method,
        String signature,
        Object[] args,
        Object ret);

    /**
     * Called before method exit to possibly modify the current frame based on interceptor logic
     */
    public boolean postProcess(TransactionFrame frame,
        Object src,
        String spec,
        String clazz,
        String method,
        String signature,
        Object[] args,
        Object ret);
\}
```

These methods are automatically invoked for all classes and methods that have been intercepted or tracked. To intercept a method or track a class you can specify it using the previously documented syntax in the [rules.properties](#) file, or alternatively you can programmatically specify the interception in the extension class's constructor. The advantage of the latter is that it makes the extension completely self-contained.

To deploy an extension, just compile it and package it in a jar file with a manifest containing the following entry:

Agent-Extension: [extension class name](#)

When you drop this jar in the LISA Agent jar directory, the agent will automatically pick it up. A hot load mechanism will ensure that if you add the file after the agent has been started, it will be applied. If you update the extension jar as the Agent is running, the jar classes will be reloaded dynamically, making it easy and fast to test your extension code without restarting the server.

Another thing worth mentioning is that the extension jars get loaded by a classloader that can see all the classes used in the extension class. This means you can compile your extension source against the `LisaAgent.jar` and all container jars that define classes you may want to use, so you don't have to use reflection in your extension.

As an illustration of all those points, here is a small sample that shows how to print the xml contents of a `WebMethods.com.wm.data.IData` objects as it gets invoked in a flow of Integration Server (note that the agent already supports WM and an extension is not necessary for it, this is just an example):

```

package com.itko.lisa.ext;

import java.io.ByteArrayOutputStream;
import com.itko.lisa.remote.transactions.interceptors.AbstractInterceptor;
import com.itko.lisa.remote.transactions.TransactionFrame;
import com.wm.app.b2b.server.ServiceManager;
import com.wm.app.b2b.server.BaseService;
import com.wm.data.IData;
import com.wm.util.coder.IDataXMLCoder;

public class IDataInterceptor extends AbstractInterceptor \{

    public IDataInterceptor() \{
        super.intercept("com.wm.app.b2b.server.ServiceManager",
            "invoke",
            "(Lcom/wm/app/b2b/server/BaseService;Lcom/wm/data/IData;Z)Lcom/wm/data/IData;");
    \}

    public boolean preProcess(TransactionFrame frame,
        Object src,
        String spec,
        String clazz,
        String method,
        String signature,
        Object\[\] args,
        Object ret) \{
        if (ServiceManager.class.getName().equals(clazz) && "invoke".equals(method)) \{
            doCustomLogic((BaseService) args\[0\], (IData) args\[1\], false);
        \}

        return super.preProcess(frame, src, spec, clazz, method, signature, args, ret);
    \}

    public boolean postProcess(TransactionFrame frame,
        Object src,
        String spec,
        String clazz,
        String method,
        String signature,
        Object\[\] args,
        Object ret) \{
        if (ServiceManager.class.getName().equals(clazz) && "invoke".equals(method)) \{
            doCustomLogic((BaseService) args\[0\], (IData) ret, true);
        \}

        return super.preProcess(frame, src, spec, clazz, method, signature, args, ret);
    \}

    private void doCustomLogic(BaseService flow, IData p, boolean output) \{
        ByteArrayOutputStream baos = new ByteArrayOutputStream(63);

        try \{
            new IDataXMLCoder().encode(baos, p);
        \} catch (IOException e) \{
            e.printStackTrace();
        \}

        System.out.println("Flow: " + flow.getNSName());
        System.out.println((output ? "Output" : "Input") + "Pipeline: " + baos);
    \}
\}

```

To build this extension yourself you will need to compile this code against [LisaAgent.jar](#), [wm-isclient.jar](#) and [wm-isserver.jar](#) then jar the class file with a manifest containing the line: `Agent-Extension: com.itko.lisa.ext.IDataInterceptor`

Alternatively you can download this sample extension here: [IDataExtension.jar](#) A demo video is also available [here](#).

2.7 Troubleshooting

2.7 Troubleshooting

The best advice here is: **find out about the target environment ahead of time and make sure it's been tested** or test it yourself (the vast majority of issues will be OS or JVM-dependent, not application dependent so you should be able to do that for every engagement). Then be sure to carefully follow the instructions supplied in this documentation. If that doesn't help, let's review the most common problems.

The majority of serious issues involving the agent will happen at start-up (agent not being found, process crashing or hanging, etc...). Generally once you get past this you're in good shape and it's a matter of tweaking the configuration and/or writing extensions.

Issue: You get the following error at startup: Error occurred during initialization of VM. Could not find agent library in absolute path...:

Make sure the library is indeed in that path and is using the same architecture as java (both 32 bit or 64 bit).

You can also verify the library is not missing any dependencies (using `depends.exe` on Win32, `otool -L` on Mac OSX and `ldd -d` on Linux and Unix). If libraries are missing, please make sure to set the `LD_LIBRARY_PATH` to include the directories where they reside.

If `ldd` doesn't return cleanly the agent will not run properly, so this is the first thing to get right. If you can't find an agent version for the desired platform, consider using the pure java agent.

Issue: The agent exits immediately (or shortly after starting the process):

If you see the message `LISA AGENT: VM terminated` then it's likely the process terminated normally (several containers have launcher processes and it's normal for them to exit quickly).

If there is no such message, or a crash dump happens (after `ldd` returns cleanly), you may have run into a legitimate agent bug. Notify dev and try the pure java agent. If it still crashes or produces a dump you may have run into a jvm bug. One such well-known occurrence is for the IBM jvm 1.5 on some OSes (caused by trying to instrument threads). In that case, try supplying the command-line jvm argument `-Dlisa.debug=true` and it should start properly.

Issue: The agent hangs or throws a lot of exceptions at startup (LinkageErrors, CircularityErrors, etc...)

A side effect of instrumenting java bytecode using java is that it can subtly alter some of the class loading order for early classes (java.* and such) resulting in a deadlock or bytecode verification errors.

We've made an effort to find and eliminate those for all combinations of JVMs and OSes but it's possible you have run into an untested combination. Notify dev, and if it's a hang please send them a thread dump (CTRL+Break on Windows, CTRL+\ or kill -3 <pid> on Unix/Linux). You may also try the Java agent as the class loading order is slightly different and you may get lucky.

Issue: The agent starts but the consoles or broker can't see the agent

This typically means there is a firewall or port problem between the agent and the broker. At the top of the agent log you should see a warning that says "Can't connect to broker at tcp://<ip>:<port>". Look at this to make sure the ip and port the agent is using are correct. Then make sure the broker is listening on the specified port on the specified ip (`netstat -ano | grep <port>` should show a port listening on the supplied ip or 0.0.0.0). Finally check for firewall issues by running `telnet <ip> <port>` from the agent machine to make sure it can see the broker.

Issue: Abnormal resource consumption (CPU, memory, file handles,...)

If the CPU usage is abnormally high or spikes periodically, note the period of the spikes since it will help us determine the faulty thread(s). Also try to turn off Pathfinder (if on) and VSE (if on) from the Dev console, to see if it helps. Dev will provide you with scripts to run from the Dev console that will enable and disable various components of the agent so we can identify the culprit.

If you get OutOfMemory errors, monitor the java heap usage (using the Performance tab in the Dev console for example). If it is above the -Xmx limit then increase that limit, unless it's already high and well in excess of normal app usage without the agent. In that case, generate a heap dump (you can do it from the Dev Console's script sub-console for jre 1.6 and above. You can use WAS's HeapDump utility in Websphere's case or the free [Eclipse MAT](#) tool for older versions). If the memory usage is below the -Xmx limit at the time of the error it's likely a leak in native code and you'll need to notify Dev.

If you get unexplained IOExceptions (like too many file handles), especially on Unix or Linux, check the ulimit on the box (`ulimit -n -H` and `ulimit -n -S`) and if it's low consider asking the administrator of the box to raise it (under 4,096 is considered low for modern j2ee apps). Don't forget to reboot the machine after doing that.

The agent tries to keep the number of file handles under that limit by triggering garbage collection when the limit is approached. If the limit couldn't be read by the agent at startup the agent will use a default value of 1,024 which may result in excessive garbage collection (and semi-random frequent CPU spikes). In that case raise the value of `lisa.agent.handles.max` in [rules.properties](#).

Issue: You can see the agent started but Pathfinder data is missing or incomplete

There are several stages in the data lifecycle process: transaction capture (in the agent), transfer to the broker, partial transaction assembly (in the broker), transfer to consoles, persistence to DB and retrieval from DB. Missing or incomplete data could be due to a problem in any of these stages.

First look in the agent log for capture exceptions, broker and console logs for transfer or persistence exceptions. Then look try to look at the data in

the Dev console (since it has one less layer than the Web console) and in particular at the Live Paths sub-console since it takes all persistence issues out of the equation.

If there are no exceptions and the data is still nowhere to be found, turn on debug or dev logging in the agents (the easiest way to do this is from the Log sub-console of the Dev console). You should see statements like "[Sent partial transaction...](#)". If you don't then Pathfinder is probably not turned on. If none of this is conclusive contact Dev for further instructions.

Issue: I turned on Java VSE recording or playback but VSE does not receive any requests from the agent(s)

First, check that the agent is in fact in VSE record or playback mode. You can do that either by looking at its log (and look for "[Starting VSE record/playback...](#)") or by looking at the Java VSE subconsole in the Dev console and seeing the Record or Playback button being disabled.

Then look for exceptions in the Agent logs and the VSE logs. If they are clean it's possible that the class you think is virtualized is not actually virtualized. Once again, look in the agent log for a statement like "[Virtualized com.xxx....](#)". If it is not there, it's possible that the class simply had not yet been loaded by the app. Another possibility is if you're using an early version of java 1.4, some flavors don't support HotSwapping (which is what is used by Java VSE by default). In that case you'll need to specify the virtualized classes in the rules.properties of the agent and restart it.

2.8 Videos

Using the Dev Console:

Part I: Turning Pathfinder against LISA Workstation and the Demo Server

PF Pro I

Part II: Rules and Extensions

PF Pro II

Part III: Superstacks, Java VSE Intro, Heap Access Intro

PF Pro III

3. LISA Agent Architecture

3. LISA Agent Architecture

The following chapters are available in this section.

[3.1 Architecture Overview](#)
[3.2 Deployment](#)
[3.3 Algorithms](#)

3.1 Architecture Overview

3.1 Architecture Overview

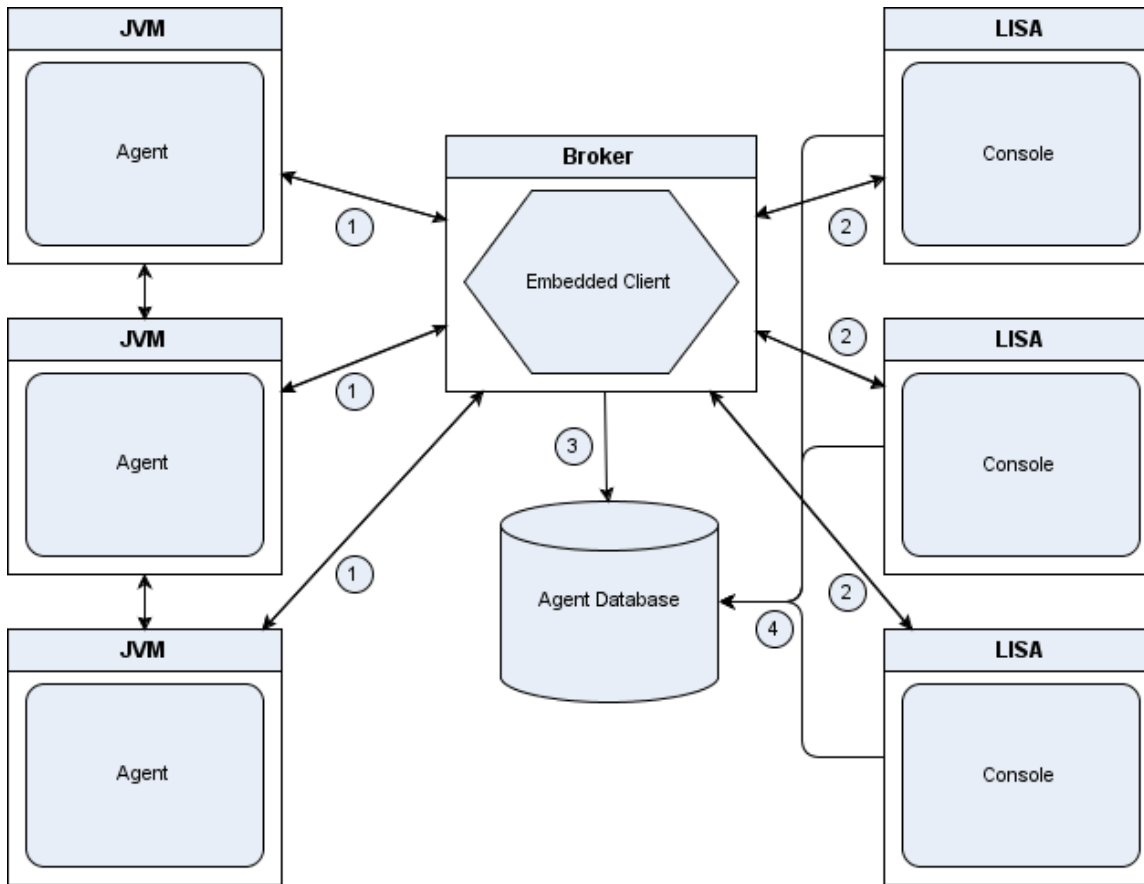
The LISA Agent is made up of 4 distinct components: the agent itself, which runs embedded in a java process, the JMS broker along with an embedded JMS client, the consoles (or clients) and the database.

The **Agent** captures data for Pathfinder ([TransactionFrame](#)), for VSE ([VSEFrame](#)) and for Knock-Out. It also exposes some APIs that can be invoked from the consoles.

The **Broker** is just a hub that dispatches JMS messages between agents, consoles and the embedded client. The embedded client keeps track of network wide/agent-spanning properties, such as the set of agents, their or current network connections.

As such it is able to assemble partial transaction data into full transactions for the benefit of consoles. Once such data is assembled it is pushed to the consoles (short term) and persisted to the database (long term archival). From this point on, we will ignore the distinction and refer to the embedded client as the broker.

Finally the **Consoles** get their finalized data from the broker (if recent) and the database (if older) for display and user interaction. These components and interactions are detailed below:



The flow of data is fully specified by the following JMS destinations:

lisa.agent.info: Topic carried over connections **1** and **2**, i.e. produced by the agents and consumed by the broker and the consoles. This gives the broker and the consoles a view of which agents are currently online and what their basic properties are.

lisa.agent.port: Topic carried over connection **1** i.e. produced by the agents and consumed by the broker. This gives the broker a view of the connections currently active between multiple agents.

lisa.agent.api: Topic carried over connection **1** and **2** i.e. produced by the consoles and consumed (and replied to) by the agents. This allows the consoles to invoke Agent apis over JMS.

lisa.broker.api: Topic carried over connection **2** i.e. produced by the consoles and consumed (and replied to) and consumed by the broker. This allows the consoles to invoke broker apis over JMS.

lisa.stats: Topic carried over connections **1** and **2**, i.e. produced by the agents and consumed by the broker and the consoles. This gives consoles an idea of what kind of load the agents are currently under and lets the broker persist those to the database.

lisa.vse: Topic carried over connections **1** and **2**, i.e. produced by the agents and consumed by the consoles. When VSE is turned on, consoles receive VSE frames (and reply to them in playback mode).

lisa.tx.partial: Queue carried over connection **1**, i.e. produced by the agents and consumed by the broker. When an agent has captured a partial transaction (i.e. all the frames that happen in its JVM), it sends it to the broker for assembly.

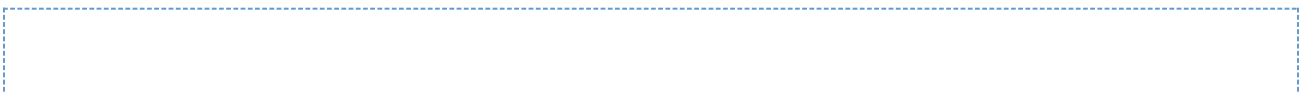
lisa.tx.full: Topic carried over connection **2**, i.e. produced by the broker and consumed by the consoles. When the broker is done assembling partial transactions received over **lisa.tx.partial**, it pushes the full ones to the consoles.

JDBC connection **3** is used when the broker saves **StatsFrames** or fully assembled **TransactionFrames**.

JDBC connection **4** is then used by the consoles to perform their queries for transactions or statistics that are no longer held in memory.

The Agent database schema is created automatically by the broker but if you need to create it manually (for security or process reasons, or even migration or documentation), you can obtain the DDL statements by running the command line: `java -jar LisaAgent.jar -ddl`.

Further options will be added for schema migrations and versioning, but currently the schema looks like this:



```

create table LISA_AGENT_VERSION (
  SCHEMA_VERSION int);

create table LISA_AGENT (
  AGENT_ID bigint not null,
  AGENT_NAME varchar(128),
  MACHINE varchar(256),
  IP varchar(32),
  WORKINGDIR varchar(256),
  CLASSPATH varchar(1024),
  LIBPATH varchar(1024),
  MAINCLASS varchar(128),
  VERSION varchar(32),
  FLAGS int,
  primary key (AGENT_ID));

create table LISA_STATISTICS (
  AGENT_ID bigint not null,
  TIME bigint not null,
  CPU int, HEAP bigint,
  NON_HEAP bigint,
  IO_IN bigint,
  IO_OUT bigint,
  GC_COUNT bigint,
  GC_TIME bigint,
  foreign key (AGENT_ID) references LISA_AGENT(AGENT_ID));

create table LISA_TRANSACTION_FRAME (
  FRAME_ID varchar(64) not null,
  PARENT_ID varchar(64),
  TRANSACTION_ID varchar(64) not null,
  AGENT_ID bigint not null,
  THREAD_NAME varchar(256),
  CLASSNAME varchar(128),
  METHOD varchar(128),
  SIGNATURE varchar(256),
  MODIFIERS bigint,
  SOURCE varchar(256),
  ARGS varchar(1024),
  RESULT varchar(256),
  SESSION_ID varchar(128),
  TIME bigint not null,
  ORDINAL bigint not null,
  CLOCK_DURATION bigint,
  CPU_DURATION bigint,
  LEK_INFO varchar(256),
  CATEGORY int,
  LOCAL_IP varchar(16),
  LOCAL_PORT int,
  REMOTE_IP varchar(16),
  REMOTE_PORT int,
  FLAGS bigint,
  COMPLEXITY bigint,
  primary key (FRAME_ID),
  foreign key (AGENT_ID) references LISA_AGENT(AGENT_ID));

create table LISA_FRAME_DATA (
  FRAME_ID varchar(64) not null,
  STATE text,
  REQUEST text,
  RESPONSE text,
  CDATA text,
  foreign key (FRAME_ID) references LISA_TRANSACTION_FRAME(FRAME_ID) on delete cascade);

create table LISA_VSE_FRAME (
  FRAME_ID varchar(64) not null,
  AGENT_ID bigint not null,
  SRC_ID varchar(64),
  SESSION_ID varchar(128),
  THREAD_NAME varchar(256),

```

```

CLASSNAME varchar(128),
METHOD varchar(128),
TIME bigint not null,
CLOCK_DURATION bigint,
FLAGS int,
primary key (FRAME_ID),
foreign key (AGENT_ID) references LISA_AGENT(AGENT_ID));

create table VSE_FRAME_DATA (
FRAME_ID varchar(64),
TYPE int,
ORDINAL int,
DATA text,
foreign key (FRAME_ID) references LISA_VSE_FRAME(FRAME_ID) on delete cascade)

create table LISA_TICKET(
TICKET_ID varchar(64) not null,
SESSION_ID varchar(128),
FRAME_ID varchar(64),
CAPTURE_TIME bigint not null,
DEFECT_ID varchar(64),
CUSTOMER_ID varchar(64),
REPORTER_ID varchar(64),
REPORTER_EMAIL varchar(128),
STATUS int,
SEVERITY int,
TITLE varchar(256),
COMPONENT varchar(128),
DESCRIPTION text,
SCREENSHOT text);

create index IDX_AGENT on LISA_TRANSACTION_FRAME (AGENT_ID);
create index IDX_TRANSACTION on LISA_TRANSACTION_FRAME (TRANSACTION_ID);
create index IDX_PARENT on LISA_TRANSACTION_FRAME (PARENT_ID);
create index IDX_TIME on LISA_TRANSACTION_FRAME (TIME, ORDINAL);
create index IDX_FLAGS on LISA_TRANSACTION_FRAME (FLAGS);
create index IDX_SESSION on LISA_TRANSACTION_FRAME (SESSION_ID);
create index IDX_ADDRESSES on LISA_TRANSACTION_FRAME(LOCAL_IP, LOCAL_PORT, REMOTE_IP, REMOTE_PORT);
create index IDX_CATEGORY on LISA_TRANSACTION_FRAME (CATEGORY);
create index IDX_TDATA on LISA_FRAME_DATA (FRAME_ID);
create index IDX_SAGENT on LISA_STATISTICS (AGENT_ID);
create index IDX_STIME on LISA_STATISTICS (TIME);
create index IDX_VTIME on LISA_VSE_FRAME (TIME);
create index IDX_VAGENT on LISA_VSE_FRAME (AGENT_ID);
create index IDX_VMETHOD on LISA_VSE_FRAME (CLASSNAME, METHOD)
create index IDX_VDATA on VSE_FRAME_DATA (FRAME_ID);

insert into LISA_AGENT_VERSION (SCHEMA_VERSION) values (1);

```

The open wire format used to communicate between agents and broker is [OpenWire](#) in combination with XML formats for all the objects mentioned above, as specified in the following schemas:

AGENT_INFO

```

<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified"
  elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- AgentInfo -->
  <xs:element name="ai">
    <xs:complexType>
      <!-- Agent Guid -->
      <xs:attribute name="id" type="xs:unsignedByte" use="required" />
      <!-- Agent Name -->
      <xs:attribute name="n" type="xs:string" use="required" />
      <!-- Agent Machine -->
      <xs:attribute name="m" type="xs:string" use="required" />
      <!-- Agent IP Address -->
      <xs:attribute name="ip" type="xs:string" use="required" />
      <!-- Agent Version -->
      <xs:attribute name="v" type="xs:string" use="required" />
      <!-- Agent Class Path -->
      <xs:attribute name="cp" type="xs:string" use="optional" />
      <!-- Agent Library Path -->
      <xs:attribute name="lp" type="xs:string" use="optional" />
      <!-- Agent Working Dir -->
      <xs:attribute name="wd" type="xs:string" use="optional" />
      <!-- Agent Main Class -->
      <xs:attribute name="mc" type="xs:string" use="optional" />
      <!-- Agent Flags -->
      <xs:attribute name="f" type="xs:unsignedByte" use="required" />
      <!-- Agent Runtime Flags -->
      <xs:attribute name="rf" type="xs:unsignedByte" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>

```

STATS_FRAME

```

<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified"
  elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- StatsFrame -->
  <xs:element name="stf">
    <xs:complexType>
      <!-- Frame Id -->
      <xs:attribute name="aid" type="xs:unsignedInt" use="required" />
      <!-- Time -->
      <xs:attribute name="t" type="xs:unsignedLong" use="required" />
      <!-- Cpu Usage -->
      <xs:attribute name="cpu" type="xs:unsignedByte" use="optional" />
      <!-- Heap Memory Usage -->
      <xs:attribute name="h" type="xs:unsignedInt" use="optional" />
      <!-- Non Heap Memory Usage -->
      <xs:attribute name="nh" type="xs:unsignedByte" use="optional" />
      <!-- IO Input -->
      <xs:attribute name="ioi" type="xs:unsignedByte" use="optional" />
      <!-- IO Output -->
      <xs:attribute name="ioo" type="xs:unsignedByte" use="optional" />
      <!-- GC count -->
      <xs:attribute name="gcc" type="xs:unsignedByte" use="optional" />
      <!-- GC Time -->
      <xs:attribute name="gct" type="xs:unsignedByte" use="optional" />
    </xs:complexType>
  </xs:element>
</xs:schema>

```

TRANSACTION_FRAME

```

<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified"
  elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- TransactionFrame -->
  <xs:element name="tf">
    <xs:complexType>
      <xs:sequence>
        <!-- Thread Name -->
        <xs:element name="tn" type="xs:string" use="optional" />
        <!-- Frame Object -->
        <xs:element name="src" type="xs:string" use="optional" />
        <!-- Method Arguments -->
        <xs:element name="args" type="xs:string" use="optional" />
        <!-- Method Return -->
        <xs:element name="ret" type="xs:string" use="optional" />
        <!-- Frame State -->
        <xs:element name="state" type="xs:string" use="optional" />
        <!-- Frame Request -->
        <xs:element name="req" type="xs:string" use="optional" />
        <!-- Frame Response -->
        <xs:element name="resp" type="xs:string" use="optional" />
        <!-- LEK Data -->
        <xs:element name="lek" type="xs:string" use="optional" />
        <!-- Children TransactionFrame -->
        <xs:element name="tf" type="tf" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <!-- Frame Id -->
      <xs:attribute name="id" type="xs:string" use="required" />
      <!-- Parent Frame Id -->
      <xs:attribute name="pid" type="xs:string" use="required" />
      <!-- Transaction Id -->
      <xs:attribute name="tid" type="xs:string" use="required" />
      <!-- Agent Id -->
      <xs:attribute name="aid" type="xs:unsignedInt" use="required" />
      <!-- Session Id -->
      <xs:attribute name="sid" type="xs:string" use="optional" />
      <!-- Frame Category -->
      <xs:attribute name="cat" type="xs:unsignedByte" use="required" />
      <!-- Frame Flags -->
      <xs:attribute name="f" type="xs:unsignedByte" use="required" />
      <!-- Frame Class -->
      <xs:attribute name="c" type="xs:string" use="required" />
      <!-- Frame Method -->
      <xs:attribute name="m" type="xs:string" use="required" />
      <!-- Frame Method Signature -->
      <xs:attribute name="sig" type="xs:string" use="required" />
      <!-- Frame Method Modifiers -->
      <xs:attribute name="mods" type="xs:unsignedByte" use="required" />
      <!-- Time -->
      <xs:attribute name="t" type="xs:unsignedLong" use="required" />
      <!-- Ordinal -->
      <xs:attribute name="ord" type="xs:unsignedByte" use="required" />
      <!-- Clock Duration -->
      <xs:attribute name="cd" type="xs:unsignedInt" use="required" />
      <!-- CPU Duration -->
      <xs:attribute name="cpu" type="xs:unsignedInt" use="required" />
      <!-- Local IP Address -->
      <xs:attribute name="lip" type="xs:string" use="optional" />
      <!-- Local Port -->
      <xs:attribute name="lp" type="xs:unsignedShort" use="optional" />
      <!-- Remote IP Address -->
      <xs:attribute name="rip" type="xs:string" use="optional" />
      <!-- Remote Port -->
      <xs:attribute name="rp" type="xs:unsignedShort" use="optional" />
      <!-- Complexity -->
      <xs:attribute name="cx" type="xs:unsignedLong" use="optional" />
    </xs:complexType>
  </xs:element>

```

```

    </xs:element>
</xs:schema>

```

TICKET

```

<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified"
  elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- Ticket -->
  <xs:element name="ti">
    <xs:complexType>
      <xs:sequence>
        <!-- Component -->
        <xs:element name="cmp" type="xs:string" use="optional" />
        <!-- Title -->
        <xs:element name="ttl" type="xs:string" use="required" />
        <!-- Description -->
        <xs:element name="desc" type="xs:string" use="optional" />
        <!-- Screenshot data (BASE64 encoded) -->
        <xs:element name="ss" type="xs:string" use="optional" />
      </xs:sequence>
      <!-- Ticket Id -->
      <xs:attribute name="id" type="xs:string" use="required" />
      <!-- Session Id -->
      <xs:attribute name="sid" type="xs:string" use="required" />
      <!-- Frame Id -->
      <xs:attribute name="fid" type="xs:string" use="required" />
      <!-- Time -->
      <xs:attribute name="t" type="xs:unsignedLong" use="required" />
      <!-- Defect Id -->
      <xs:attribute name="did" type="xs:unsignedShort" use="optional" />
      <!-- Customer Id -->
      <xs:attribute name="cid" type="xs:string" use="optional" />
      <!-- Reporter Id -->
      <xs:attribute name="rid" type="xs:string" use="optional" />
      <!-- Reporter Email -->
      <xs:attribute name="e" type="xs:string" use="required" />
      <!-- Status -->
      <xs:attribute name="st" type="xs:unsignedByte" use="required" />
      <!-- Severity -->
      <xs:attribute name="sev" type="xs:unsignedByte" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>

```

VSE_FRAME

```

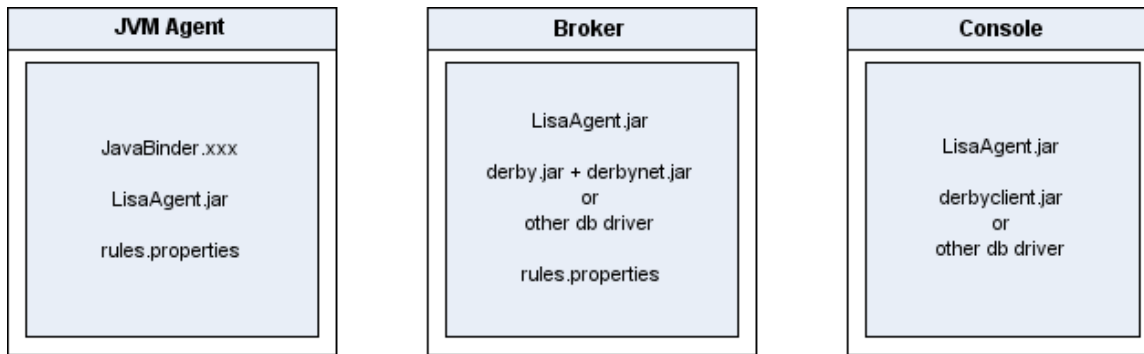
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified"
  elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- VSE Frame -->
  <xs:element name="vf">
    ...
  </xs:element>
</xs:schema>

```

3.2 Deployment

3.2 Deployment

The required files for an agent deployment are illustrated below.



Note: this is not final as [LisaAgent.jar](#) may get repackaged in Agent, Broker and Client flavors, possibly with Derby jars embedded to simplify deployment further.

3.3 Algorithms

3.3 Algorithms

The startup order of agents, broker and consoles has no relevance because all communications are asynchronous and can reestablish themselves automatically. This is especially important for agents to not have their performance suffer because the broker goes down for example.

When an agent comes online, it starts pulsing its information over **lisa.agent.info** at regular (short) intervals. If no broker is available then it won't try to notify any listeners of anything else until a connection is (re)established.

If the broker is available, all interested parties will become quickly notified of which agents are online. The broker and consoles keep a running list of those agents so that when they stop pulsing their info, they're expired and removed from the list after a while.

The main difficulty is to assemble partial transactions. Here is how it happens:

Coming...

4. LISA Agent Platforms

4. LISA Agent Platforms

The Platforms section is made up of the following chapters.

- 4.1 Tomcat / JBoss / Geronimo / Resin
- 4.2 IBM WebSphere
- 4.3 BEA/Oracle WebLogic
- 4.4 WebMethods IS
- 4.5 Sun Glassfish
- 4.6 Tibco BusinessWorks

4.1 Tomcat _ JBoss _ Geronimo _ Resin

4.1 Tomcat _ JBoss _ Geronimo _ Resin

The standard instructions apply well for those containers:

In **Tomcat's** `startup.bat(.sh)` define `set JAVA_OPTS=-agentpath:<path to JavaBinder>` before calling the `catalina.bat(.sh)`

In **JBoss's** `run.bat(.sh)` define `set JAVA_OPTS=-agentpath:<path to JavaBinder>` before calling java executable.

In **Geronimo's** `startup.bat(.sh)` define `set JAVA_TOOL_OPTIONS=-agentpath:<path to JavaBinder>` before calling `geronimo.bat(.sh)`.

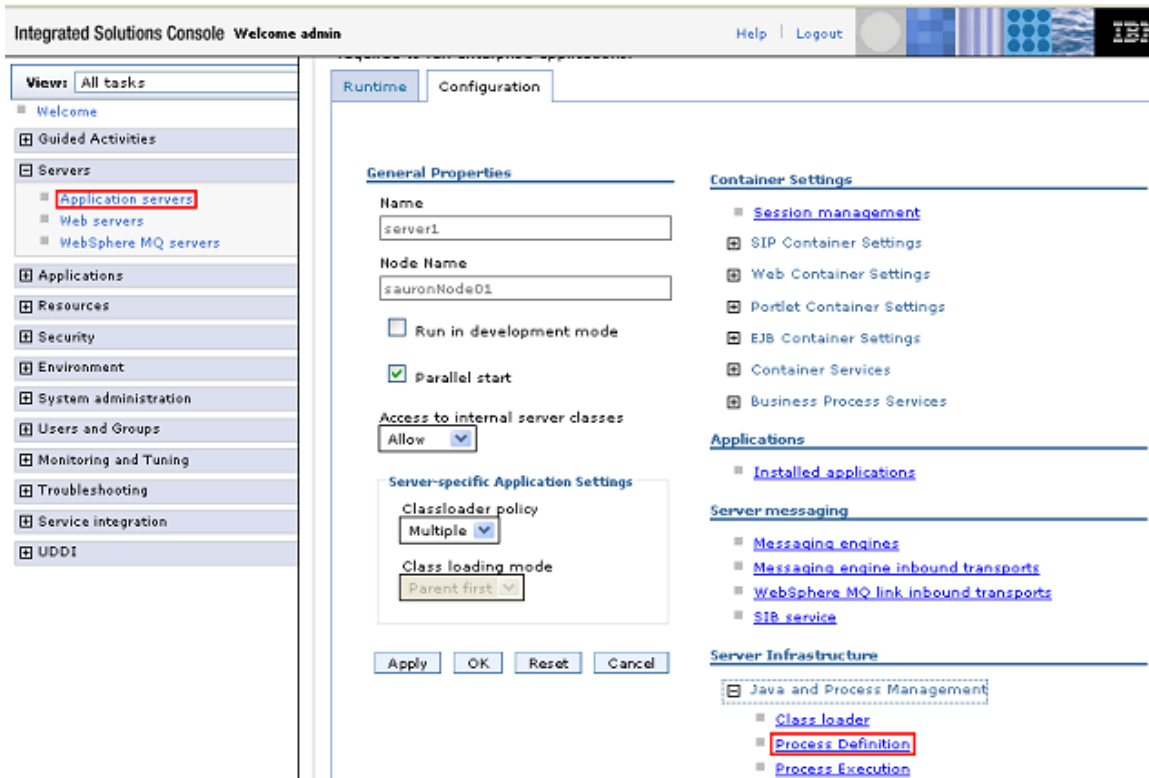
In **Resin** simply define `set JAVA_TOOL_OPTIONS=-agentpath:<path to JavaBinder>` in the launching shell of `resin.exe(.sh)` or pass `-J-agentpath:...` as an argument.

4.2 IBM WebSphere

4.2 IBM WebSphere

WebSphere is typically started as a service and thus makes it more difficult to set the variable used to turn on the agent. There are 2 ways to accomplish this:

- go to the WebSphere Admin Console (<https://localhost:9043/ibm/console/secure/securelogin.do> by default), then navigate to your WAS Application Server and click on Process Definition



Then select Java Virtual Machine

Integrated Solutions Console Welcome admin Help | Logout

Application servers [Close page](#)

Application servers

Application servers > server1 > Process Definition

Use this page to configure a process definition. A process definition defines the command line information necessary to start or initialize a process.

Configuration

General Properties	Additional Properties
<p>Executable name</p> <p>Executable arguments</p> <p>startCommand</p> <p>startCommandArgs</p>	<p>Java Virtual Machine</p> <p>Environment Entries</p> <p>Process Execution</p> <p>Process Logs</p> <p>Logging and Tracing</p>

and enter the agent JVM arguments as described in the [Install](#) section in the genericJvmArguments text box.

Integrated Solutions Console Welcome admin Help | Logout

Application servers

Application servers > server1 > Process Definition

Use this page to configure a process definition. A process definition defines the command line information necessary to start or initialize a process.

Configuration

<p><input type="checkbox"/> Verbose garbage collection</p> <p><input type="checkbox"/> Verbose JMI</p> <p>Initial Heap Size</p> <p>Maximum Heap Size</p> <p><input type="checkbox"/> Run HProf</p> <p>HProf Arguments</p> <p><input type="checkbox"/> Debug Mode</p> <p>Debug arguments</p> <p>-Djava.compiler=NONE -Xdebu</p> <p>Generic JVM arguments</p> <p>-agentpath:C:\itko\main\lisa-re</p> <p>Executable JAR file name</p> <p><input type="checkbox"/> Disable JIT</p>
--

- Or edit the <WebSphere Home>/AppServer/profiles/AppSrv01/config/cells/<machine name>Node01Cell/nodes/<machine name>Node01/servers/server1/server.xml manually. At the bottom there is an entry you can modify as follows:
 <jvmEntries ... genericJvmArguments="-agentpath:<Path to JavaBinder.dll>=jar=file:<Path to LisaAgent.jar>,name=was" .../>

Using wsadmin

C:\IBM\WebSphere61\AppServer\bin>hostname

cam-aa74651f617

C:\IBM\WebSphere61\AppServer\bin>wsadmin

WASX7209I: Connected to process "server1" on node cam-aa74651f617Node01 using SOAP connector; The type of process is: UnManagedProcess

WASX7029I: For help, enter: "\$Help help"

wsadmin>set server1 [\$AdminConfig getid /Cell:cam-aa74651f617Node01Cell/Node:cam-aa74651f617Node01/Server:server1/] server1(cells/cam-aa74651f617Node01Cell/nodes/cam-aa74651f617Node01/servers/server1|server.xml#Server_1255494205517)

```
wsadmin>set jvm [$AdminConfig list JavaVirtualMachine $server1]
(cells/cam-aa74651f617Node01Cell/nodes/cam-aa74651f617Node01/servers/server1|server.xml#JavaVirtualMachine_1255494205517)
wsadmin>$AdminConfig modify $jvm genericJvmArguments "-agentpath:<Path to JavaBinder.dll>=jar=file:<Path to
LisaAgent.jar>,name=was"
wsadmin>$AdminConfig save
wsadmin>quit
```

wsadmin jacl scripts

Zip of wsadmin jacl scripts

More complex IBM products that are built on top of Websphere Application Server have complex configurations. Not all 'servers' that are identified in wsadmin are truly 'application servers' that can have LISA Agent installed on them. The linked attachment above contains a zip file of wsadmin scripts that can be used to deploy LISA Agent into such environments. To use it:

- Download, unzip, and copy the contents into your LISA Agent directory
- Put a Java Binder library, LisaAgent.jar, and rules.properties into the LISA Agent directory as usual
- Edit agent.env per the comments in the file, changing the values appropriately
- With Websphere up and running, run deployAgent.sh to deploy the LISA Agent to all application servers in a WAS install. Use undeployAgent.sh to undeploy the Agent.

The JACL scripts have the "smarts" to not remove existing JVM arguments unrelated to Agent, and to not add / remove duplicate LISA Agent arguments.

4.3 BEA_Oracle WebLogic

4.3 BEA_Oracle WebLogic

The default install instructions apply for WebLogic but to follow the officially supported way to add arguments to the JVM, you can go to the WebLogic admin console and specify the agentpath in there:

The screenshot shows the Oracle WebLogic Server Administration Console. The 'Configuration' tab is selected, and the 'Server Start' sub-tab is active. The 'Arguments' field is highlighted with a red box, showing the following text: `-agentpath:c:\itko\main\lisa-remote\dist\agent\JavaBinder.dll=jar=file:c:\itko\main\lisa-remote\dist\agent\LisaAgent.jar,name=uls -Dlisa.log.level=debug`. Other fields like Java Home, Java Vendor, BEA Home, Root Directory, and Class Path are also visible.

Alternatively you can edit the `<WebLogic Home>/user_projects/domains/base_domain/config/config.xml` and add the `-agentpath` property to the `<server>/<server-start>/<arguments>` node of the desired server.

4.4 WebMethods IS

4.4 WebMethods IS

In WebMethods's `IntegrationServer/bin/server.bat(.sh)` define `set JAVA_TOOL_OPTIONS=-agentpath:<path to JavaBinder>` before the line

calling java. `server.exe` just calls into `server.bat` so there is no need for special instructions to use it.

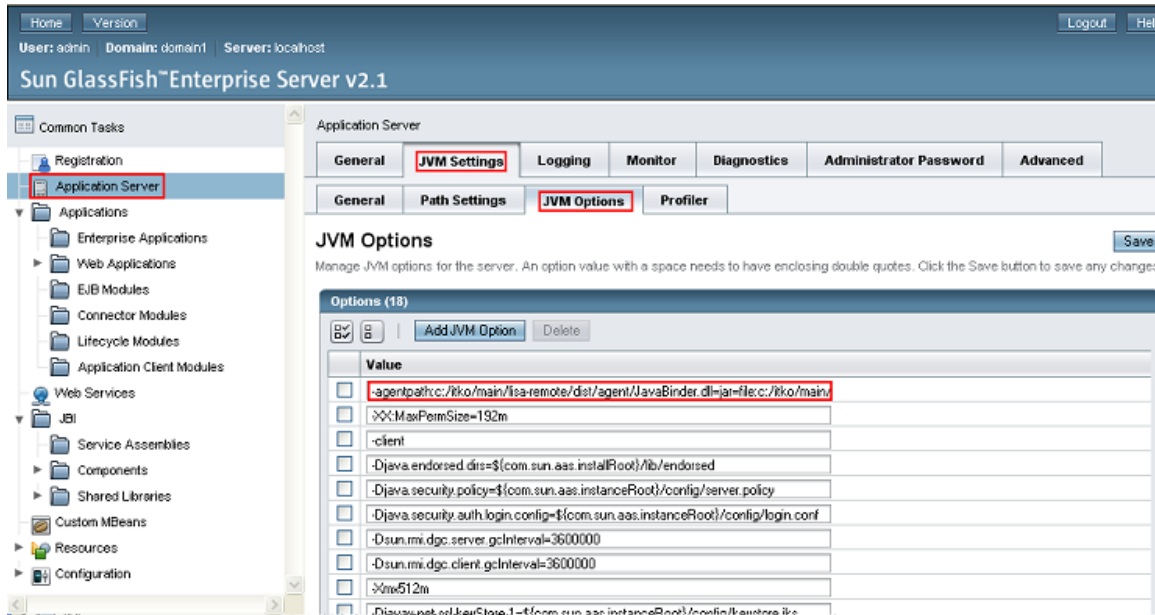
If IS is defined as a Windows Service, you can edit the `server.bat` by inserting the `-agentpath:...` argument in the `/jvmargs` value defined in the if `"%1"=="1-service"` switch, e.g.:

```
%S_DIR%\bin\SaveSvcParams.exe" /svcname %2 /jvm "%JAVA_DIR%\..\" /binpath "%PATH%" /classpath %CLASSPATH% /jvmargs "-agentpath:c:/Lisa/agent/JavaBinder.dll=jar=file:C:/Lisa/agent/LisaAgent.jar,name=MyIS %JAVA2_MEMSET%" /progargs "%S_DIR%\bin\ini.cnf" #-service %2"##PREPENDCLASSES_SWITCH%##PREPENDCLASSES%##APPENDCLASSES_SWITCH%##APPENDCLASSES%##ENV_CLASSP.
```

4.5 Sun Glassfish

4.5 Sun Glassfish

You can apply the usual instructions with Glassfish or you can use the administrative console which is the officially supported way to do it. Go to <http://localhost:4848/>, navigate to JVM Options and enter the `-agentpath:...` option before saving, as shown below:



Alternatively you can manually edit the configuration file that stores these entries at `<Glassfish home>/domains/domainXXX/domain.xml`. You need only add an entry with the `-agentpath:...` value under `/domain/configs/java-config/jvm-options`.

4.6 Tibco BusinessWorks

4.6 Tibco BusinessWorks

Tibco BW 5.X

Edit the following file to affect all applications:

```
... \tibco\bw\5.4\bin\bwengine.tra
```

Note that this will not affect applications that have already been created.

Edit the following file to affect a single application that has already been created:

```
... \tibco\tra\domain\<domain name>\application\<application name>\<application name>-<service name>.tra
```

Where:

- <domain name> is the name of your Tibco domain
- <application name> is the name of the deployed Tibco Application
- <service name> is the name of the deployed service

For example:

```
...\tibco\tra\domain\itko-tibcobw\application\MyBWProjectXml\MyBWProjectXml-itkoUserCRUD.tra
```

In either file, add the following line:

```
java.extended.properties=-agentpath:<lisa home>/agent/JavaBinder.dll=jar=file:<lisa home>/agent/LisaAgent.jar,name=<name>
```

Where:

- <lisa home> is the absolute path of the installation directory for LISA
- <name> is a descriptive name to give the LISA Agent. This can be the name of your service if editing the individual service file, or just something like 'bwengine' if editing the main bwengine.tra file.

LISA Developer's Guide (LEK)

LISA Developer's Guide (LEK) (.pdf format)

*

NOTE: This guide is only available for those customers who have the LISA Extension Kit (LEK), **which requires an additional license**. Now known as SDK vs. LEK. This document is stored on a separate space which also includes additional information.

The Developer's Guide is available in .pdf format only at this time.

The **LISA Developer's Guide (LEK)** available in another space contains instruction for the more technical user of LISA. LISA functionality can be extended with programmers as there is a high amount of customization available with the LEK (LISA Extension Kit)_Developer's Guide.

This guide provides some detailed information on that customization ability.

LISA Installation and Configuration Guide

LISA Installation and Configuration Guide

Welcome to the LISA 5.0 Installation and Configuration Guide!

This guide contains detailed instructions on how to setup, install, configure and get started with LISA. Each chapter in the Installation guide is devoted to a particular LISA installation or a third party tool.

LISA has three types of installations:

1. [LISA Workstation Installation](#)
2. [LISA Demo Server Installation](#)
3. [LISA Server Installation](#)

This guide also include details on setting up JAVA Tools (third party tool).

NOTE: While reading this guide, it is recommended that you have the other LISA guides available. The other available guides consist of [iTKO & LISA Introduction](#), [LISA Developer's Guide \(LEK\)](#), [LISA Reference Guide](#), [LISA User Guide](#), [LISA VSE Guide](#), and [LISA Web 2.0 Guide](#). For a summary of all these guides please see [LISA User Documentation](#). It is also helpful to be familiar with the concepts and usage of [LISA components](#).

The following chapters are available in the Installation Guide and Configuration Guide.

1. Installing and Running LISA Workstation

This chapter describes the installation of LISA workstation, its pre-requisites, post installation directories etc.

2. Installing and Running LISA Demo Server

This chapter describes the installation of the LISA Demo Server. This is optional, but is required if you plan to install the examples and work through them as detailed in the Getting Started section of the User Guide.

3. Installing and Running LISA Server

This chapter describes LISA Server and its installation.

4. Configuring Java Tools

This chapter gives guidelines to install and run certain Java Tools, which you may need during your testing.

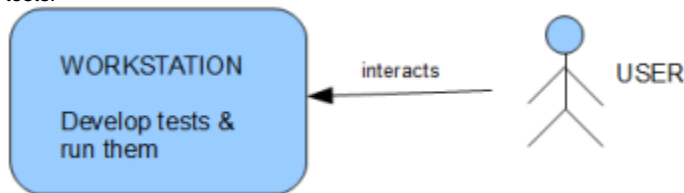
1. Installing and Running LISA Workstation

1. Installing and Running LISA Workstation

LISA Workstation is a single, easy to use, integrated environment, for developing, staging and monitoring tests, with almost no coding required. LISA Workstation is the basic version of LISA, with minimal setup.

The **LISA Workstation** not only creates the tests, but also manages and runs the tests within the LISA Workstation environment. LISA Workstation is the only application you will use.

The LISA Workstation is explained by the following diagram, where the user interacts with the LISA Workstation to develop, monitor, and run the tests.



This chapter describes how to install LISA Workstation with normal defaults, add license credentials, and start the LISA Workstation.

The following topics are available in this section.

1.1 System Requirements and Prerequisites

1.2 Installation Instructions

1.3 Starting LISA Workstation

1.5 Post Installation Directories & Files

1.6 LISA Installation Notes

1.1 System Requirements and Prerequisites

1.1 System Requirements and Prerequisites

System Requirements

The following tables list the requirements for the various LISA components. These general requirements may need to be customized depending on the scope of the project.

LISA Workstation

In prior versions, LISA Workstation was called Test Manager. LISA Workstation is required for a user desktop system for script creation/execution. This is where the tests and virtual services are created and maintained.

Below are baseline requirements.

Description	Requirement
CPU	1 dual core CPU
RAM	2GB or more
Disk Space	2GB free space
OS	Windows 2000 *, 2003, XP, Vista, Linux, Solaris, AIX 6.1 (Only with LISA 5.0) LISA; not yet supported on Windows 7 or 2008 Server.

LISA Server

LISA Server is required for maintaining a central reporting repository and Load and Performance testing. In a LISA Server configuration, at a minimum, there will be one Test Registry and one Coordinator Server, and one Simulator Server (but could have more than one Simulator Server).

Below are baseline requirements only.

Description	Requirement
CPU	2 dual core CPUs
RAM	4GB or more
Disk Space	5GB free space
OS	Recommended: 64 Bit OS. Also supported: Windows 2000 *, 2003, XP, Vista, Linux, Solaris, AIX 6.1 (Only with LISA 5.0)
DB	Not required but recommended to use MySQL, Oracle, DB2, or SQL Server for production usage. DB can reside on a different system and should have at least 10 GB of storage.

For Load and Performance (L&P) Testing the following requirements are recommended.

250 Virtual Users/Simulator
1 Processor Core and 2GB RAM / Simulator

Example: 4000 Concurrent Virtual Users
16 Simulators; 16 processor core; 32 GB RAM (for LISA)

Per Data Center

- *1 Test Registry & 1 Coordinator
- *1 Processor Core/process = 2 processor cores
- *2GB RAM/each = 4GB (for LISA)

LISA VSE

LISA VSE is required for maintaining a Service Oriented Virtualization environment. As LISA VSE is a server level service, it can co-exist with a TestRegistry that has a Coordinator and Simulator attached to it, but the Simulator and Coordinator are not mandatory to run VSE.

Below are baseline requirements only.

Description	Requirement
CPU	2GHz or faster
RAM	2GB or more
Disk Space	5GB free space
OS	Recommended: 64 Bit OS. Also supported: Windows 2000 *, 2003, XP, Vista, Linux, Solaris, AIX 6.1 (Only with LISA 5.0)
DB	Not required but recommended to use MySQL, Oracle, DB2, or SQL Server for production usage. DB can reside on a different system and should have atleast 10 GB of storage.

The following requirements are recommended for larger scale VSE deployments.

256 Virtual Service Threads per VSE instance
1 Processor Core and 2GB RAM per VSE instance

Example: 1,000,000 transactions per day
1 thread/service to support functional tests ~6 threads/service to support Virtualization for L&P Tests
8 cores = 2,048 concurrent virtual service threads
16 GB RAM (for LISA)

Below are some additional requirements:

Requirement	Description
JVM	iTKO Recommendation: 64 Bit Java 6 The following are also supported. <ol style="list-style-type: none">1. Java 5--Client side only. (You can use LISA running in a Java 5 JRE to test applications on application servers running in an older JRE.)2. Windows installer defaults to 1.6.0_05
Operating system	iTKO Recommendation: 64 Bit Operating System especially for LISA Server. The following are also supported. <ol style="list-style-type: none">1. Windows 2000 */XP/Vista, Windows Server 2003 (With latest service pack and all critical updates applied)2. Mac OS X 10.4 and above3. Red Hat Enterprise Linux 4 and above4. Fedora 8 and above5. AIX 6.1 (Only with LISA 5.0)6. Solaris 10
Software	LISA Server 4.7 and above

NOTE: iTKO does not fully support LISA running on Windows 7 (32 or 64 bit) and Windows 2008 Server; Both will be supported in a release in late 2010. At this time there are no known blocking issues with these operating systems.

NOTE: Microsoft Support for Windows 2000 is no longer supported by iTKO, this support ended on July 13, 2010.

LISA Readme File

The Readme file is located in the %LISA_HOME%/Doc directory and contains information about LISA Environment settings, known issues, LISA editions etc.
Please read the **readme.html** file provided by LISA before starting LISA.

Java JDK

LISA requires a Java Development Kit (JDK) installed.
The minimum supported Java version for LISA 5.0 is Java 1.5. This is a client side requirement only; LISA running in a Java 1.5 virtual machine (VM) can be utilized to test applications on application servers running with an older JRE.

For Windows Installer

LISA includes a suitable JDK as part of the Windows installer. If you wish to use an alternative JDK on Windows, you can replace the one included with LISA with one of your choice. The procedure to do this is explained in the later section.

For MAC and Unix Installer

For platforms other than Windows, you must make sure you have the correct JDK installed, since it is not included in the LISA Installer.
If you do not have an appropriate version of Java installed already, download it from <http://java.sun.com> and follow the installation instructions.

NOTE: Remember that LISA requires a JDK. A JRE is not sufficient because LISA requires the **tools.jar** file that is only included in the JDK.

Once you have installed Java,

- Set the environment variable **JAVA_HOME** to point to the Java installation directory.

Once you have installed LISA,

- Copy the **tools.jar** file that came with your Java installation into the LISA "**hotDeploy**" directory. The hotDeploy directory is in the LISA installation directory.

1.2 Installation Instructions

1.2 Installation Instructions

The following topics are available to complete the LISA Workstation installation.

- 1.2.1 Installation Login
- 1.2.3 Downloading the Installer
- 1.2.4 Installing LISA Workstation
- 1.2.5 Providing a Valid LISA License

1.2.1 Installation Login

1.2.1 Installation Login

The LISA Workstation can be downloaded from the iTKO LISA download site at:
<http://www.itko.com/downloads/ga/http://www.itko.com/downloads/ga/>

After clicking the link, you will need to enter your user name and password in the following manner:
Your user name will be: "The last node of your license domain / your license username"
Your password will be: the license password.

For example:

Your license domain is iTKO/LISA/ABC.

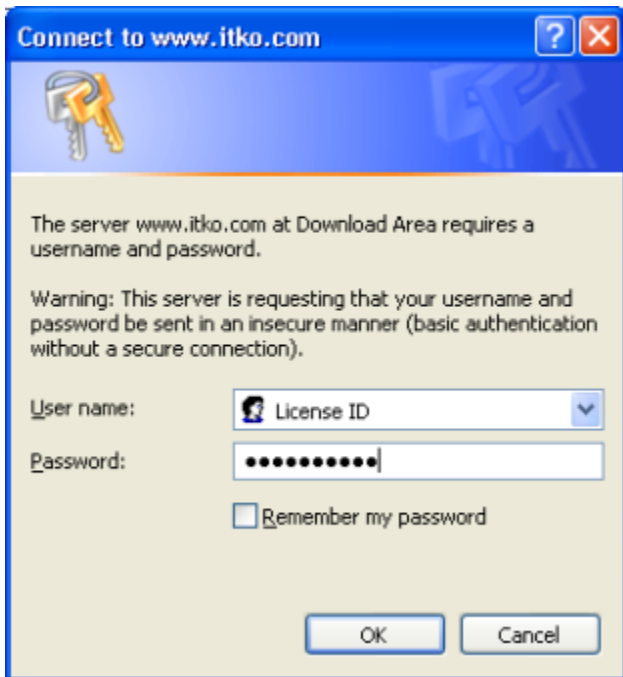
Your license username is EnterpriseWorkstation and your license password is z5RiW4NNr.

Then your download credentials would be:

username: ABC/EnterpriseWorkstation

password: z5RiW4NNr

NOTE: All of this information should be in your license email. If you have not received this email contact your iTKO sales representative.



Enter the **Username** and **Password**
Check **Remember my password** and click **OK**

1.2.2 Download Page

1.2.2 iTKO LISA Download Page

The Download Page includes the following sections.



LISA GA Download

STOP AND READ: The minimum supported Java version for LISA 4.6 is Java 5. Note this is a client side requirement only. LISA running in a Java 5 JRE can be utilized to test applications on application servers running in an older JRE. If you require running in an earlier version of the JDK, then please do not download this version. This is a new release and will require a license good for version 4. If you are not ready to do this, do not install this version. You can contact support@itko.com to inquire about a new license.

If you encounter any issues, please report them to iTKO at support@itko.com. Also, please be sure to inform us which release is not functioning properly.

Build Information

Version: 4.6.4
Build Number for Individual Components:
core: 4.6.4.1
esb: 4.6.4.1
httpweb: 4.6.3.2
install: 4.6.4.7
j2ee: 4.6.3.1
jdbridge: 4.6.3.8
jmx: 4.6.3.1
web20: 4.6.3.12
editor: 4.6.3.3
reporting: 4.6.3.4
virtualize: 4.6.4.0
ws-xml: 4.6.4.2
(Go To Release Notes)
Build Date (YYYY/MM/DD): 2009/08/11

[Stop and Read](#)
[Build Information](#)
[Education Materials](#)
[Installer Types](#)
[Deployable Example Server](#)
[Other \(Plugins\)](#)

STOP AND READ section

Please Read the STOP and READ content given in **RED**, on this page before you start the installation.

Build Information

The latest build information is provided along with a link for Release Notes. The link takes you to the appropriate section of the page where Release Notes may be available.

LISA Education Materials

Scroll down to get a list of **LISA Educational Material** and the **Most common Installers** list:

List of LISA Education Materials

The [Installation and Configuration Guide](#) describes how to install and configure LISA™ software.

The [Getting Started Guide](#) describes how to quickly understand how to use the LISA™ software by walking through scenarios.

The [LISA User Guide](#) provides detailed information for the LISA software, focused on the LISA Test Manager™ application. The LISA Test Manager application enables anyone to create, validate, and manage automated test cases.

The [LISA Reference Guide](#) contains detailed instructions on how to create and configure LISA's built in components. As such it assumes that you are familiar with the concepts, and the usage of these components. The Reference Guide is complementary to the LISA User Guide.

The [LISA VSE User Guide](#) contains detailed instructions on how to install, configure and use LISA Virtualize.

Installers

Most Common Installer * most users will download this version *****



Windows Workstation with samples

[Download >](#)

[View Instructions](#)

Other Available Installers

Platform	Workstation	Server	Instructions
 Windows	Download >	Download >	View
 Mac OS X	Download >	Download >	View
 Linux/Solaris/Unix	Download >	Download >	View

*** If none of the above options will work for you, we offer a [generic .tar file](#) [Download >](#) Once you expand the tar you must run `configure.sh`. This is only recommended as a solution for Unix users.

Installer Types

You can choose and download from the following type of Installers:

Installer with LISA Samples:

LISA also has an installer that bundles some LISA examples to be used with the LISA Demo Server. The most common LISA Workstation installation is the **"Windows Workstation with samples"**.

Click the **Download>** link as shown below to start the download:

This installer has LISA sample test cases, which can be referred to while you get started with LISA.



The screenshot shows the 'Installers' section of the LISA website. It features a heading 'Most Common Installer *** most users will download this version ***' followed by a list of installers. The 'Windows Workstation with samples' installer is highlighted with a red box around its 'Download >' link. Below this, there is a table titled 'Other Available Installers' with columns for Platform, Workstation, Server, and Instructions. The table lists three options: Windows, Mac OS X, and Linux/Solaris/Unix, each with a 'Download >' link and a 'View' link. At the bottom, there is a note: '*** If none of the above options will work for you, we offer a generic .tar file Download > Once you expand the tar you must run configure.sh. This is only recommended as a solution for Unix users.'

This installer installs the Windows version of **"LISA Workstation"** and deploys a **"JBoss Demo Server"** on your machine that is pre-configured and ready to run LISA examples.

NOTE: The other workstation installers (summarized below) do not include the Demo Server, and you will have to download it separately. See [Installing and Running the LISA Demo Server](#).

For Windows (64 bit also available):

Workstation - `Lisa_wrk_win.exe` – Choose for Lisa Workstation Installation (about 165 mb)

Server - `Lisa_svr_win.exe` – Choose for Lisa Server Installation (about 265 mb)

For MAC:

Workstation - `Lisa_wrk_osx.dmg` – Choose for Lisa Workstation Installation (about 140 mb)

Server - `Lisa_svr_osx.dmg` – Choose for Lisa Server Installation (about 230 mb)

For Linux/Solaris/Unix:

Workstation - `Lisa_wrk_unix.sh` – Choose for Lisa Workstation Installation (about 140 mb)

Server - `Lisa_svr_unix.sh` – Choose for Lisa Server Installation (about 235 mb)

To run command line version:

```
./lisa_svr_unix.sh -c
```

For Unix users, if the above doesn't work, LISA offers a [generic .tar file](#), [lisa_unix.tar.gz](#).

Once you expand this tar file, you must run **configure.sh**.

Once you decide on the **Installer Platform(From Windows/Linux/Mac)**, you can also click **View** to see detailed download instructions. Choose the appropriate "Installer type" and download the installer, by clicking the appropriate **Download>** link.

For more detailed instructions on the install see [1.2.3 Downloading the Installer](#).

Deployable Example Server

This section allows you to download the LISA Demo Server if you choose one of the installers that does not automatically include the LISA Demo Server.

Other (LISA Plugins)

LISA provides the following plugins which can be run with LISA tests.

- IBM Rational TestManager for Windows
- Mercury's Quality Center TestDirector for Windows

Other

IBM's Rational TestManager for Windows Plugin: iTKO provides a way to open/edit and run LISA tests and suites from IBM's Rational TestManager product, including returning data to RTM for summary and reporting.

[Download the plugin](#)

Mercury's Quality Center (TestDirector) for Windows Plugin: iTKO provides a way to import into and run LISA tests from TestDirector so you can take advantage of all TestDirector features while harnessing the power of LISA testing.

[Download the plugin](#)

By using or downloading any of the above software, you agree to abide by iTKO's Terms Of Use for the software and this site. LISA Automated Testing is a registered trademark of iTKO (Interactive TKO), Inc. Rational TestManager is a registered trademark of IBM Corporation. Windows is a registered trademark of Microsoft Corporation. All other marks are properties of their respective owners.

Click "**Download the plugin**" link to start the download.

1.2.3 Downloading the Installer

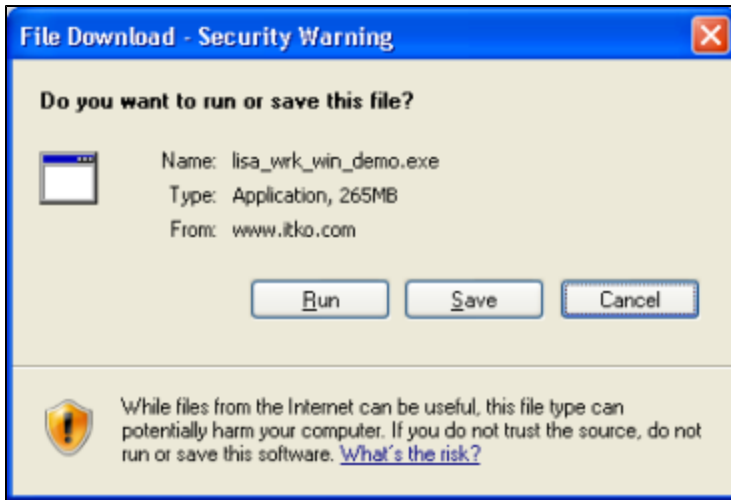
1.2.3 Downloading the Installer

The LISA Workstation can be downloaded from the iTKO LISA download site. For more information see [1.2.1 Installation Login](#).

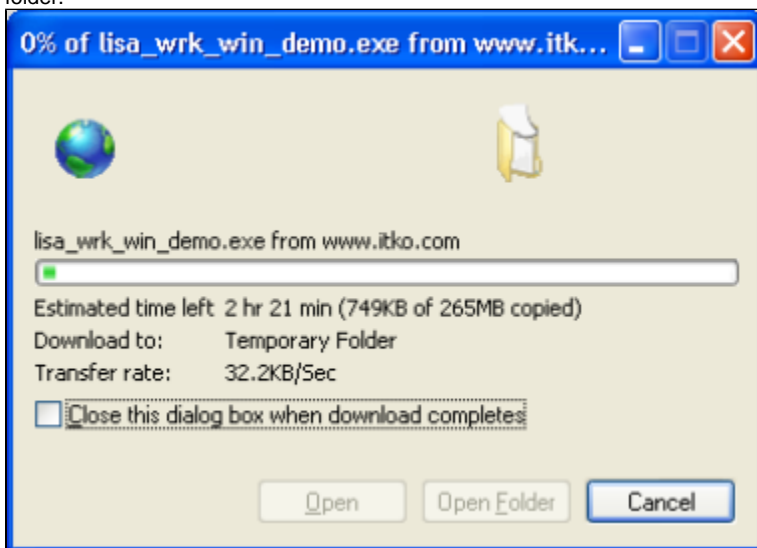
Once you click the **Download>** link available for the installer you wish to install based on platform, the **File Download - Security Warning** window opens.

Click **Save** to download and save the installer, or click **Run** to run the installer without saving it.

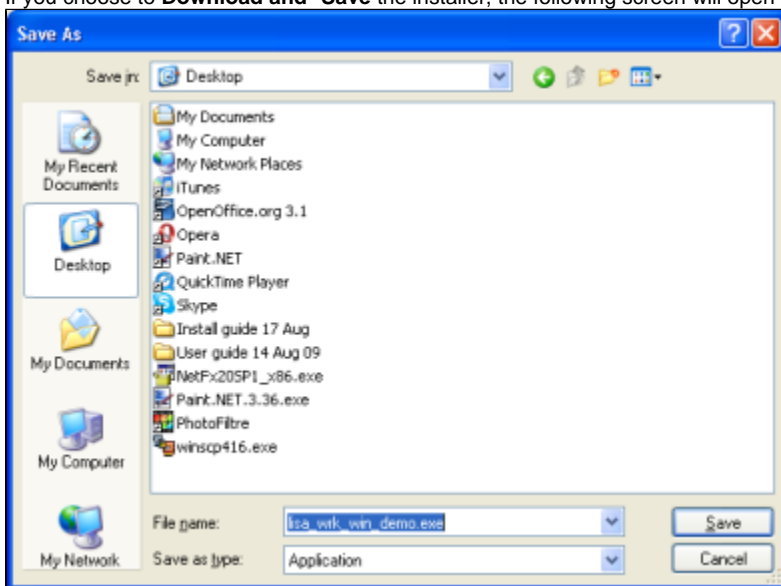
NOTE: The installer is about **270 mb** and will take several minutes to download (this is the same whether you choose to Run or Save). Before installation, please make sure that you have the disk space available.



If you choose to **Download and Run** the installer without saving, you will see the following screen. The installer file is downloaded to a temporary folder.



If you choose to **Download and *Save** the installer, the following screen will open so I download location can be entered.



Browse to the directory where you want to save the installer and click **Save**. The installer will now be downloaded to the specified location.

Once the Installer is downloaded to the specified location, double-click the **Installer name (.exe)** to start the installer. The installer name for the "

Windows Workstation with Example Samples" is: lisa_wrk_win_demo.exe

The Security Warning dialog opens, click **Run** to start the installer.



For detailed steps on using the Installer File and installing LISA Workstation, see [1.2.4 Installing LISA Workstation](#).

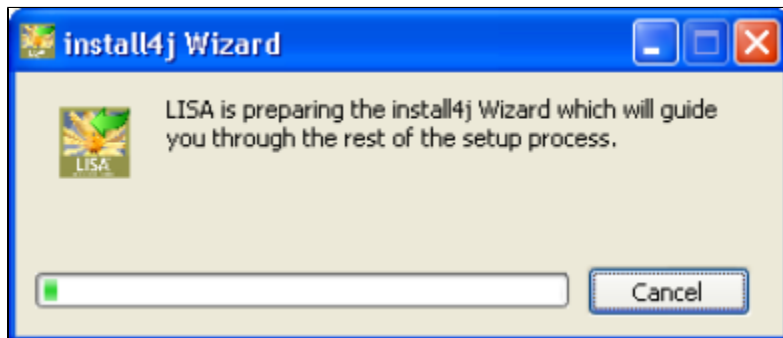
1.2.4 Installing LISA Workstation

1.2.4 Installing LISA Workstation

Once you download the installer, you can start the LISA Workstation installation.

- Click **Run** on the installer window.

The **install4j wizard** will start and guide you through the rest of the installation process.

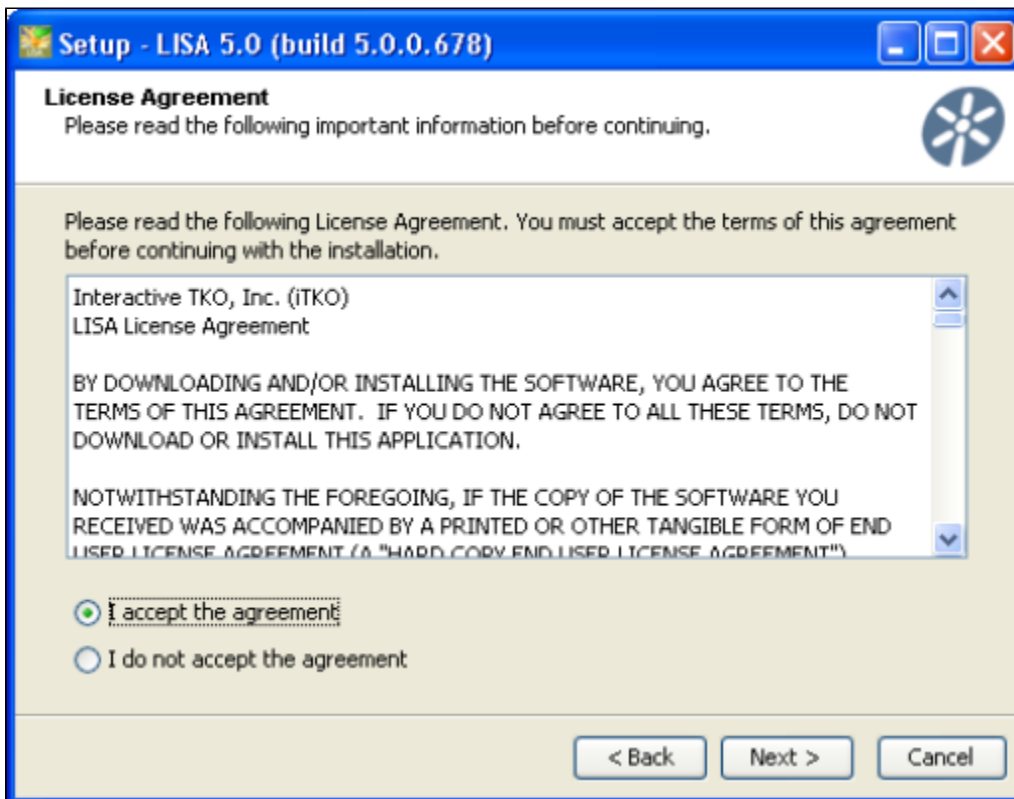


The **Setup - LISA** wizard opens.



The welcome section lists all of the LISA components along with the versions that will be installed.

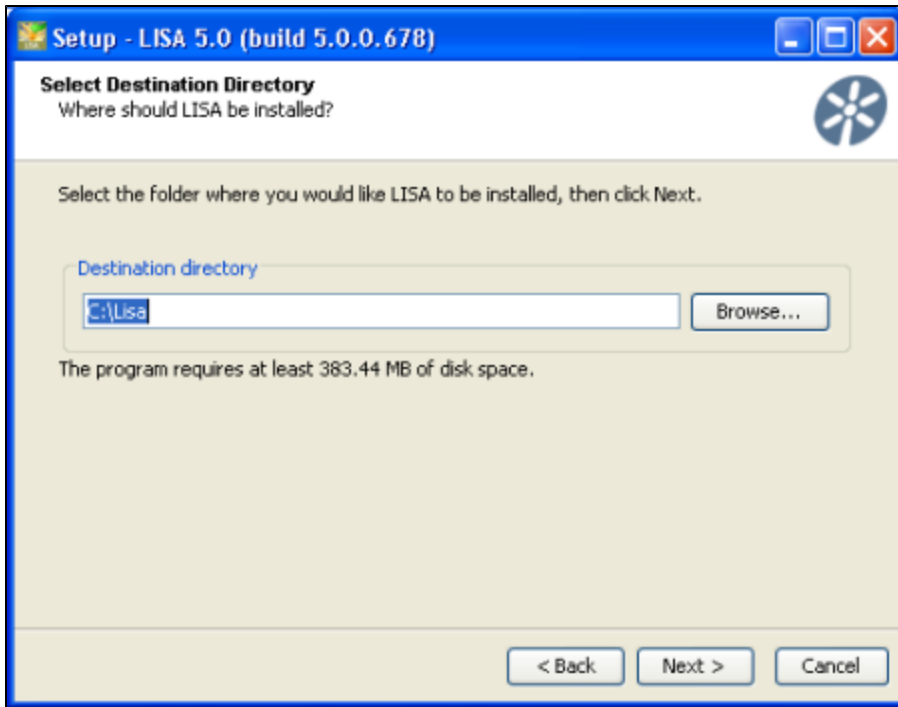
- Click **Next** to continue to the **License Agreement** information:



Read the license agreement carefully.

The license agreement contains terms and conditions you should be aware of before downloading LISA.

- Click **I Accept the Agreement** box.
- Click **Next** to continue to the **Select Destination Directory** information:



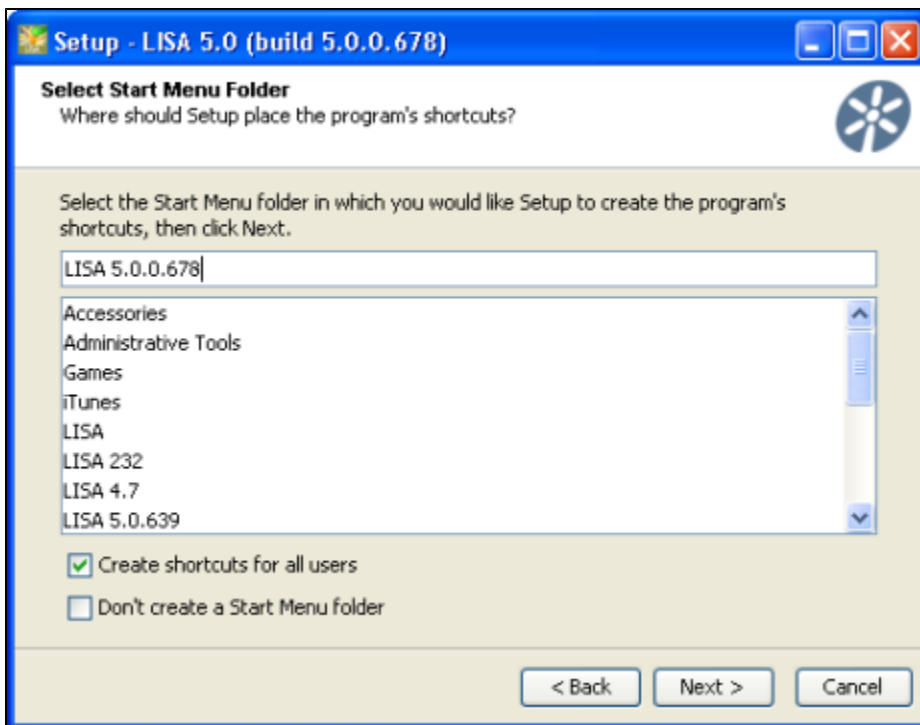
Here you need to specify the directory the LISA software will be installed.

The default location for LISA install is **C:/LISA**. Going forward the LISA home is specified as **%LISA_HOME%**, which means **C:/LISA** and will be known as the **LISA installation directory**.

NOTE: You can use spaces in the Installation directory for a Windows installer. (Ex: C:/Program Files).

Do not use spaces in directory names in a Unix Installer.

- Click **Next** to continue to the **Select Start Menu Folder** information:

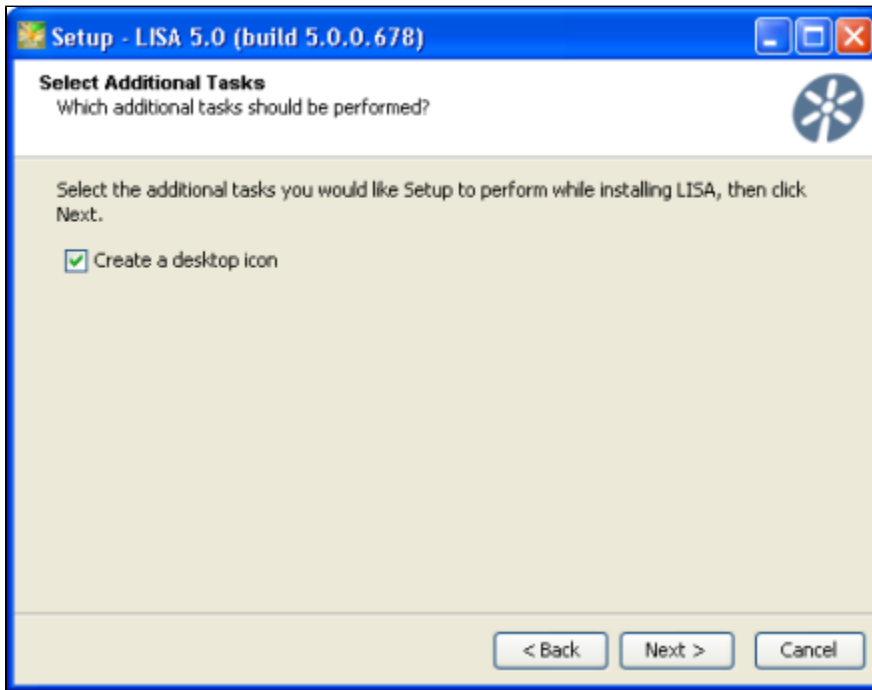


- Enter the name of the **Start Menu** folder. By default it will be **LISA (version number followed by the build number)**.

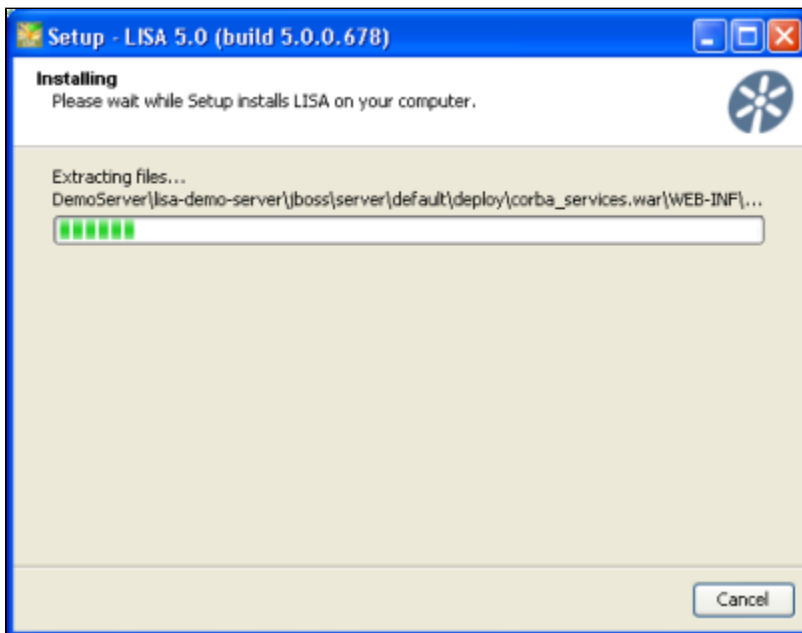
Create Shortcuts for all Users: Check to create a shortcut for all users using LISA.

Don't Create a Start Menu Folder: If checked, will not create a Start Menu Folder.

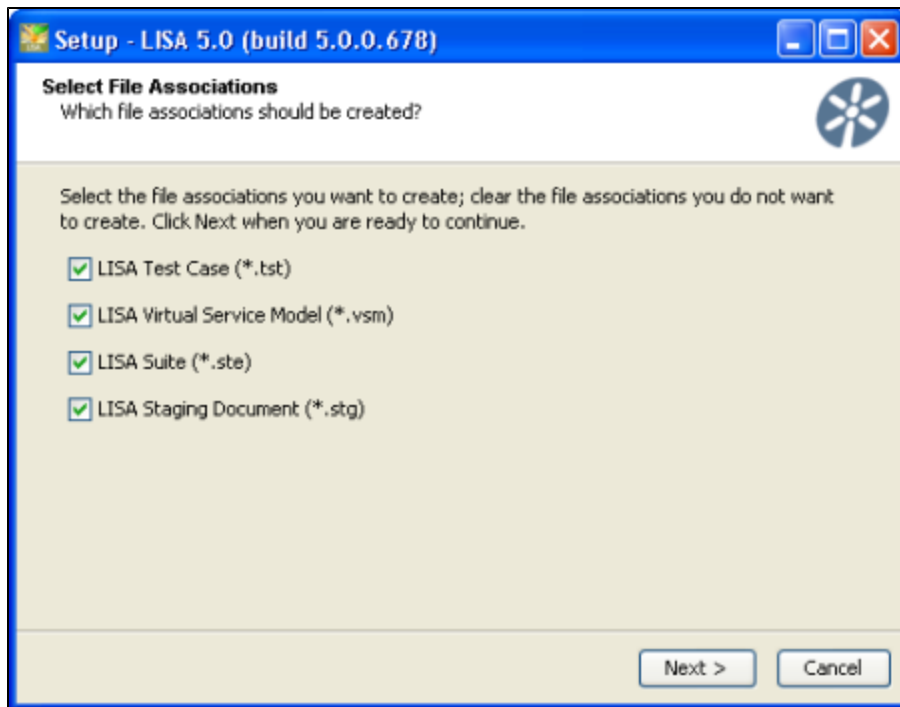
- Click **Next** to continue to the **Select Additional Tasks** information:



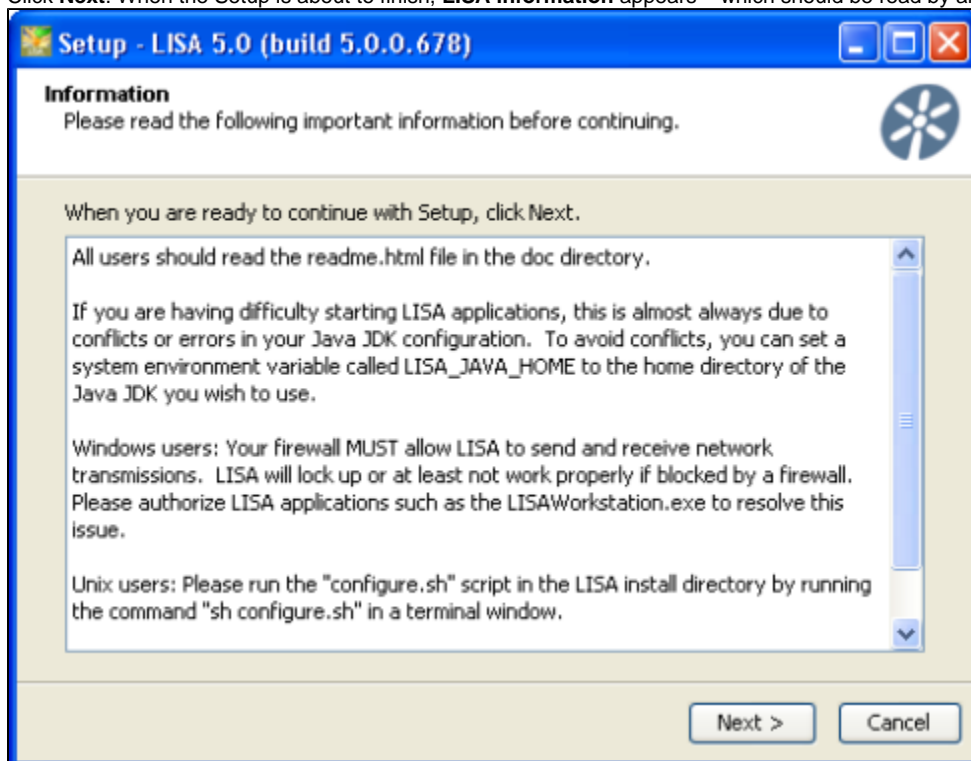
Create a Desktop Icon: Check to create a desktop icon. Click **Next** to continue and start the Installation procedure as shown below:



The installer will now extract and setup LISA in the specified installation directory (C:/LISA). Once the files are extracted, the **Select the File Associations** information appears as below. Check the file extensions you want to associate with LISA. By default all the associations (.tst, .vsm, .ste, .stg) will be selected.

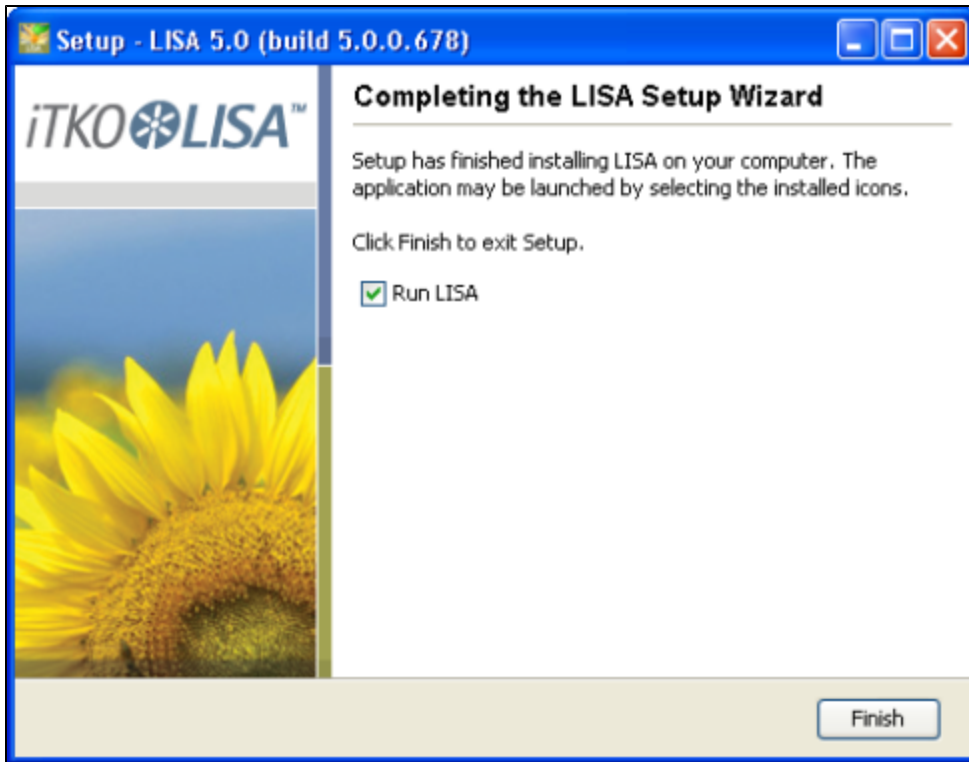


Click **Next**. When the Setup is about to finish, **LISA information** appears – which should be read by all users.



Please read the information displayed on the screen carefully especially the **Readme.html** and information about the **Firewall**.

Click **Next** to continue to the **Setup Complete** information.



Check **Run LISA** to open LISA workstation once the Installation is complete.

- Click **Finish** to complete LISA installation. Once the Setup is complete, LISA Workstation will open.

You are now ready to run LISA and need to enter your license credentials.

See 1.2.5 Providing a Valid LISA License.

1.2.5 Providing a Valid LISA License

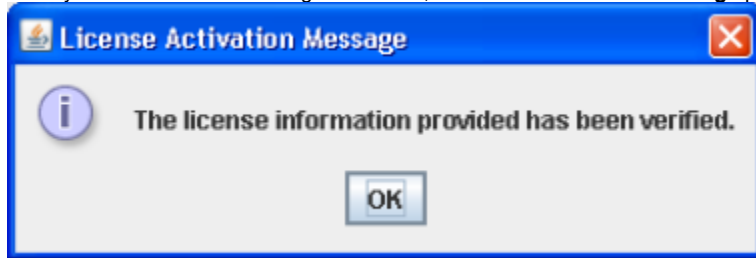
1.2.5 Providing a Valid LISA License

The first time you launch the LISA Workstation, you will be asked to enter your license information. The license information is supplied by iTKO and is available when you purchase LISA. If you do not have the license information, please contact sales@itko.com for help.

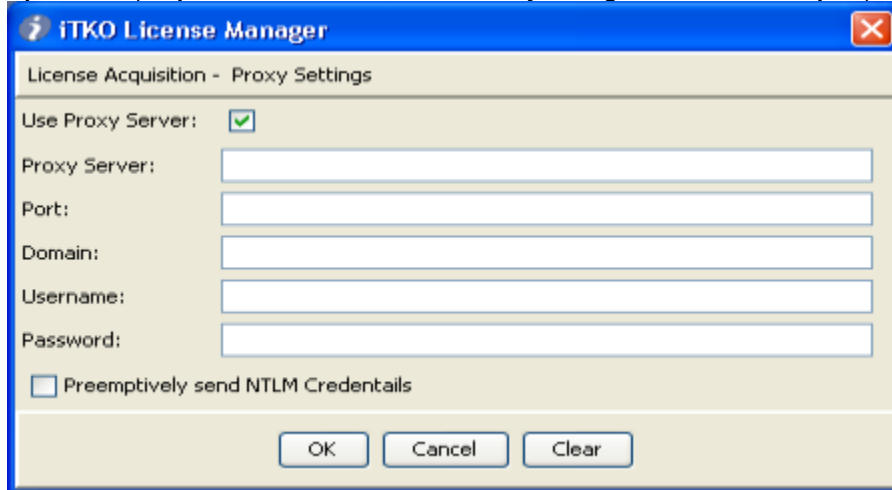
The LISA license dialog box is as shown below. There are four entries for the license settings.

The LISA license uses a **license Server**, which the LISA Workstation visits periodically to check for a valid license. The license Server is accessed over **HTTPS on port 443** and can support most proxies if your network requires proxy access to public Internet sites.

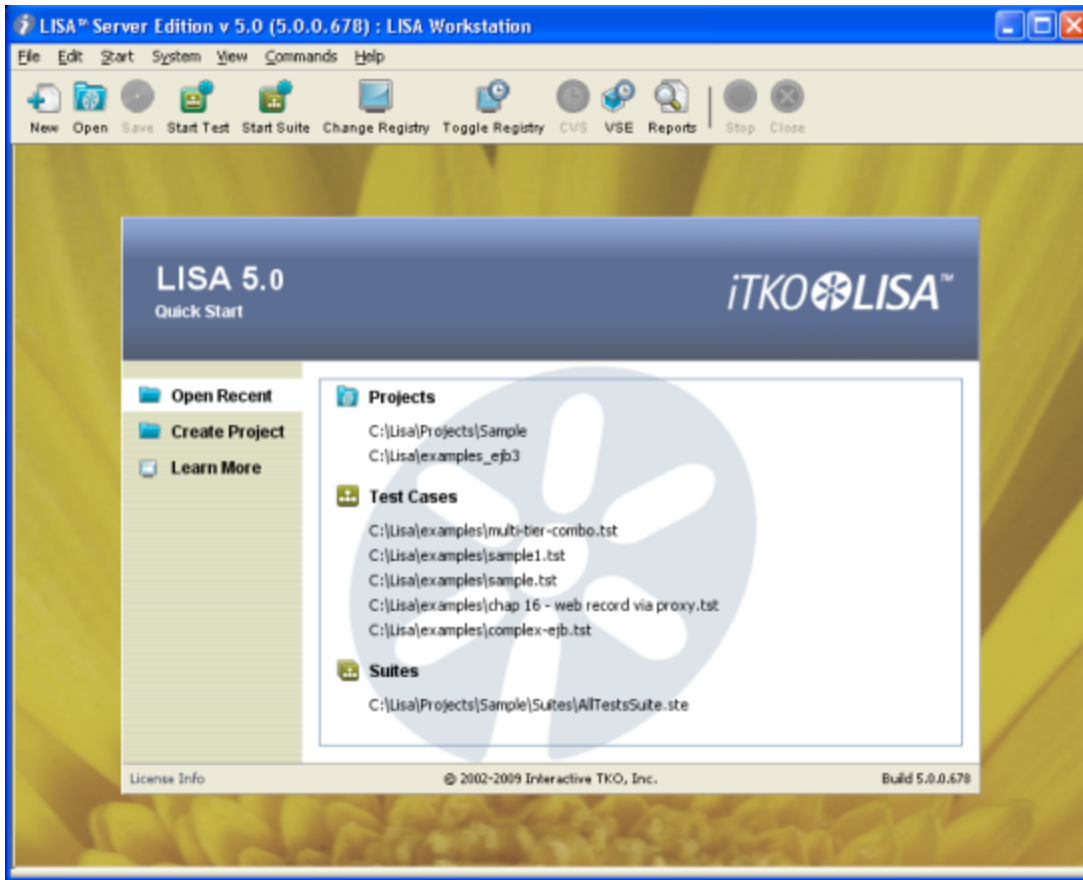
Once you enter correct licensing information, a **License Activation message** pops up after verification, as shown below:



If you need a proxy to reach the Internet, click the **Proxy Settings...** button and enter your proxy related information.



Once you enter all the information, click **OK** to start the LISA Workstation. When the LISA Workstation starts, you will see **LISA Quick Start** as the first screen as shown below:



You can either Open a **Recent** Project or **Create** a Project and then create or open Test Cases.

You can also click **Learn More** to open the entire LISA documentation set.

A good place to start is the [Tutorials section](#) of the [LISA User Guide](#). This guide will walk you through several useful tutorials.

You are now ready to use the LISA Workstation.

Note: The instructions in this section assume that you have selected the default values during the installation process. If you did not choose the defaults, you will need to modify all path information accordingly.

1.3 Starting LISA Workstation

1.3 Starting LISA Workstation

Once you are done with the Installation, you now are ready to start the LISA Workstation. All of LISA's components are managed and controlled through the LISA Workstation.

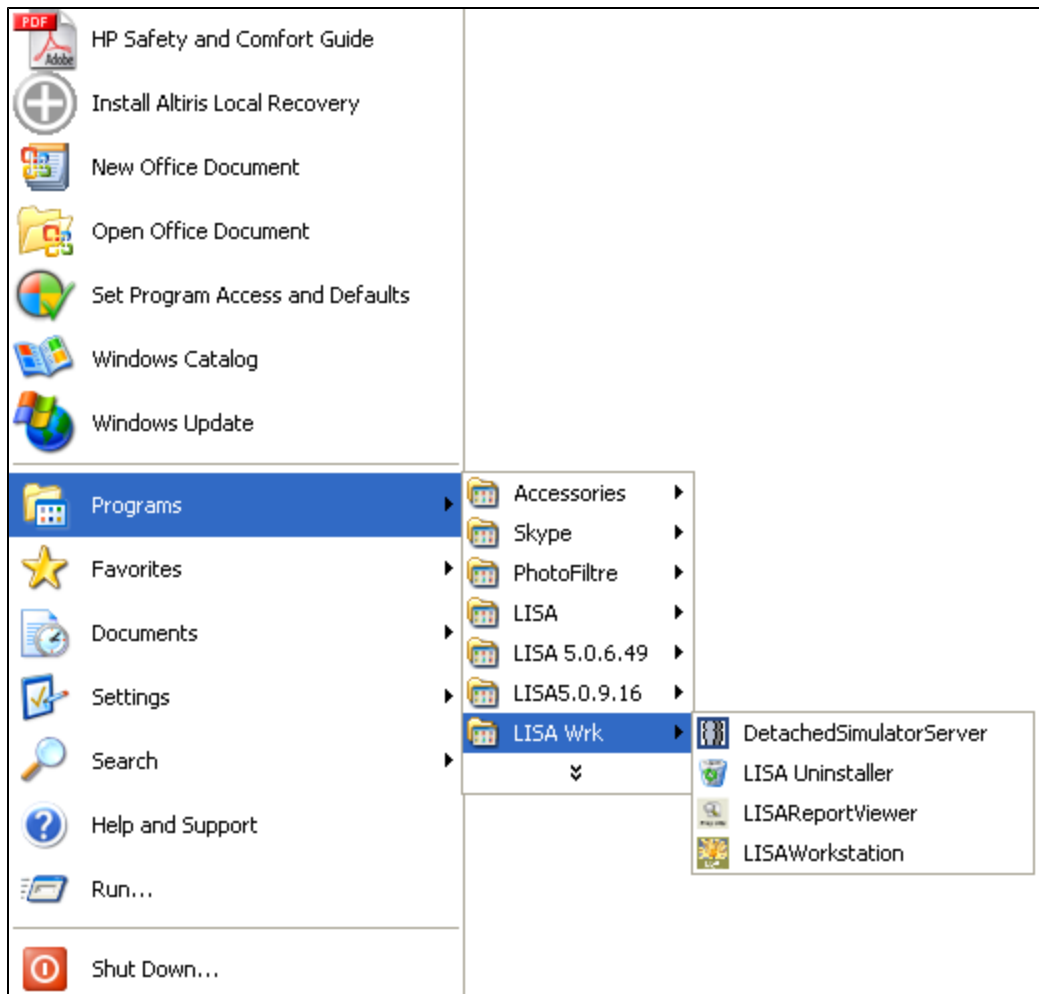
NOTE: LISA Workstation components are different than the LISA Server components.

There are several ways to start the LISA Workstation software on Windows:

1. **Desktop:** By Double-clicking the LISA icon on your desktop (if you have chosen to create a desktop icon during installation).



2. **Start Menu:** Click **Start > Programs > LISA > LISA Workstation** from start menu.



3. **Command Line:** By using a startup script from a command window. Open a command prompt, and navigate to the bin directory in the LISA install directory (the one you specified above) and running **LISAWorkstation.exe**.

```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

M:\>c:
C:\>cd lisa
C:\Lisa>cd bin
C:\Lisa\bin>LISAWorkstation.exe

```

When LISA opens, the LISA Registry dialog box will open. For more information on the LISA Registry, see [1.4 LISA Registry](#).

1.4 LISA Registry

1.4 LISA Registry

A LISA Registry is very important part of LISA and it is mandatory to start the LISA Workstation.

All the major services within LISA Workstation, like the Reporting database, VSE database, Pathfinder Agent and all other services, are provided through the LISA Registry.

By default the Registry dialog pops up every time you start LISA Workstation/Server.



You can choose from either any of the **LISA Registry** or the **Local Registry**.

LISA Registry

This is the name of the LISA registry you want to connect to.

If you are using LISA for the first time, you may need to create one. See [Creating a LISA Registry](#).

If you have already used LISA and created registries earlier, the LISA Registry drop down will list all of them.

Local Registry

At times you may not have access to a Registry, or may not have a license to run on own.

For this scenario, local mode of Registry exists. Local Mode is a Registry that's run inside of LISA Workstation. It does not require a Registry license, however, it is limited to the local machine.

The first time the user selects Local Mode, a Registry instance will be launched inside LISA Workstation. This in turn will launch the necessary Database, Web Portal, and LISA Agent processes.

NOTE: As long as LISA Workstation is running the Local Registry, associated services will also be running. Shutting down LISA Workstation will shut down the Local Registry and all services as well.

Once the Local Registry is started, you can switch between it and other LISA registries without having to restart the Local Registry. See [Switching Registries](#).

Attaching a LISA Registry

Every time you start the LISA Workstation, the Registry dialog pops up:

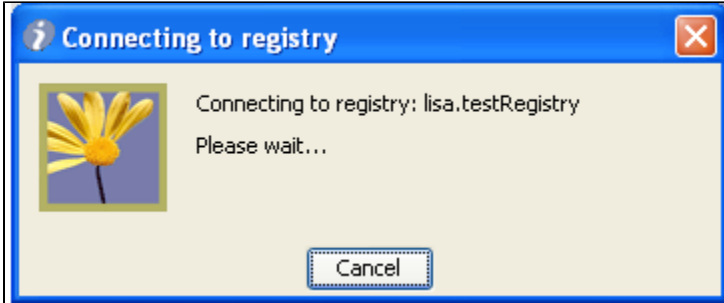


- Select the **LISA Registry** from the available drop down list. Else you can Create one if needed.
- Check or Un-check the "**Prompt on Startup**" check box. By default it is checked, so that the Set LISA Registry dialog always opens when you start LISA Workstation.

Note - When 'Prompt on Startup' is unchecked, the LISA Workstation will automatically connect to the last connected registry on start up. If the last connected registry was the Local Registry, then the Local Registry will be automatically launched.

- Click **OK** to connect to the selected Registry.
- Click on the '**Help**' button on the Registry dialog, to view the ITKO Help page, that documents the new Registry features and requirements.

The following screen appears to show that it is being connected:



Once the LISA registry connects, the LISA Workstation opens up.

1.4.1 Creating LISA Registry

1.4.1 Creating LISA Registry

A LISA Registry is very important part of LISA and is mandatory for LISA Workstation.

All the major services within LISA Workstation, like the Reporting database, VSE database, Pathfinder Agent and all other services, are provided through the LISA Registry.

Creating LISA Registry

A LISA Registry is needed to run many applications within the LISA Workstation as most of them, like the Reporting database, VSE database, Pathfinder Agent, reporting console, and other services are all provided through the Registry.

Hence the LISA Registry is very important for LISA Workstation.

To create a "named" Test Registry, open a command window and run:

```
%LISA_HOME%\bin\TestRegistry -n TestRegistryName
```

This creates a new Test Registry process associated with the name **TestRegistryName**.

For example, to create a Test Registry called **registry1** on a machine where LISA is installed to **C:\Lisa5.0Alpha**, enter the following command in the command window:

```
C:\Lisa5.0Alpha\bin\TestRegistry -n registry1
```

Or, go to the "bin" directory and enter the following:

```
TestRegistry -n registry1
```

To start the Test Registry

- Go to **Start Menu>Program Files>Lisa5.0Alpha>TestRegistry**
- Or Go to **C:\Lisa5.0Alpha\bin\TestRegistry.exe**

1.4.2 Switching Registries

Switching Registry

You can switch to a different Registry at any time.

- Open the **Set LISA Registry** dialog box.
- Click on the desired **registry** from the drop down list.
- Click **OK** to change the Registry.

At times, if the current Registry gets disconnected for some reason, LISA will automatically prompt to switch to a different Registry.

LISA will also give an option of launching a Local Registry.

1.5 Post Installation Directories & Files

1.5 Post Installation Directories & Files

Post installation, there is a directory created **%LISA_HOME%** (as specified in the installer).

There are several directories and files installed in the **%LISA_HOME%**. Certain folders and files which are important and which will be used frequently.

[Installed Directories](#)

[Installed Files](#)

Installed Directories

The screen below shows Directories installed during installation:

- **.install4j** – This is the Installer related directory and contains LISA installer.
























Name	Size	Type	Date Modified
.install4j		File Folder	4/1/2010 11:38 AM
addons		File Folder	4/1/2010 11:36 AM
agent		File Folder	4/1/2010 11:36 AM
bin		File Folder	4/1/2010 11:36 AM
database		File Folder	4/1/2010 11:36 AM
defaults		File Folder	4/1/2010 11:36 AM
doc		File Folder	4/1/2010 11:36 AM
embedded_servers		File Folder	4/1/2010 11:36 AM
examples		File Folder	4/1/2010 11:37 AM
examples_src		File Folder	4/1/2010 11:37 AM
hotDeploy		File Folder	4/1/2010 11:38 AM
incontainer		File Folder	4/1/2010 11:37 AM
jre		File Folder	4/1/2010 11:38 AM
legacy_reports		File Folder	4/1/2010 11:37 AM
lib		File Folder	4/1/2010 11:37 AM
licenses		File Folder	4/1/2010 11:37 AM
reports		File Folder	4/1/2010 11:37 AM
snmp		File Folder	4/1/2010 11:37 AM
sql		File Folder	4/1/2010 11:37 AM
tmp		File Folder	4/1/2010 11:39 AM
umetrics		File Folder	4/1/2010 11:37 AM
webserver		File Folder	4/1/2010 11:37 AM

- **install4j** - Installer related files are in this directory.
- **addons** – LISA Addons are installed in this directory. Currently only lisa-invoke & webapp are added.
- **agent** - LISA Agent related files are in this directory.
- **bin** – This is the directory for LISA executables. **TestManager.exe** and **TestRunner**, a headless version of LISA Workstation (LISA workstation) is also in this directory. For information on using TestRunner see [LISA User Guide - 36. TestRunner](#).
- ***database** - The database used for reporting portal.
- ***default** - Lists the Staging documents common to LISA across all projects. Project specific staging documents can be seen in the Project directory.
- **doc** – LISA documentation files (PDF's) are installed in this directory.

- **embedded Server** – Tomcat Server is installed in this directory.
- **examples** – This is a default LISA project containing LISA examples that use the Demo Server.
- **examples_src** – LISA examples source files and the Kiosk related files.
- **hotDeploy** – A directory monitored by LISA that contains Java classes and jar files. Java classes and jar files in this directory are on the LISA classpath. Any new files or directories added to this directory will be dynamically added to the LISA classpath.
- **incontainer** – LISA in-container testing instructions are installed in this directory.
- **jre** – jre required for LISA is installed in this directory.
- legacy reports -
- **lib** -- The jar files required for running LISA are installed in this directory.
- **licenses** – contains the license files required for running LISA.
- **reports** – XML-based test reports created by LISA are stored in this directory.
- **snmp** – SNMP related files are installed in this directory.
- **sql** – SQL required to run LISA reports is installed in this directory.
- **tmp** – Logging files created by LISA are stored in this directory. If you communicate with iTKO support on an issue you will probably be asked to send one or more of the files from this directory.
- **umetrics** - Metrics collection related files on Operating system basis.
- **webserver** - Webserver related files are in this directory.

Installed Files:

The screen below shows the files installed during installation.

	uninstall.exe	146 KB	Application	3/29/2010 12:16 AM
	README	1 KB	File	3/29/2010 12:09 AM
	rmi-keystore.jks	2 KB	JKS File	3/29/2010 12:05 AM
	rmi-truststore.jks	1 KB	JKS File	3/29/2010 12:05 AM
	webrekeys.ks	2 KB	KS File	3/29/2010 12:05 AM
	.lisa.properties.lock	0 KB	LOCK File	4/1/2010 11:38 AM
	.local.properties.lock	0 KB	LOCK File	4/1/2010 11:38 AM
	configure.bat	1 KB	MS-DOS Batch File	3/29/2010 12:05 AM
	setLISA_HOME.bat	1 KB	MS-DOS Batch File	3/29/2010 12:05 AM
	lisa.permissions	1 KB	PERMISSIONS File	3/29/2010 12:05 AM
	appletviewer.policy	1 KB	POLICY File	3/29/2010 12:04 AM
	_local.properties	17 KB	PROPERTIES File	3/29/2010 12:04 AM
	_site.properties	5 KB	PROPERTIES File	3/29/2010 12:04 AM
	learnmore.properties	3 KB	PROPERTIES File	3/29/2010 12:05 AM
	lisa.properties	40 KB	PROPERTIES File	3/29/2010 12:05 AM
	logging.properties	2 KB	PROPERTIES File	3/29/2010 12:05 AM
	typemap.properties	43 KB	PROPERTIES File	3/29/2010 12:05 AM
	contenttypes.xml	3 KB	XML Document	3/29/2010 12:05 AM
	desensitize.xml	8 KB	XML Document	3/29/2010 12:05 AM
	httpbrowsers.xml	2 KB	XML Document	3/29/2010 12:05 AM
	j2eeservers.xml	9 KB	XML Document	3/29/2010 12:05 AM
	mibdefs.xml	35 KB	XML Document	3/29/2010 12:05 AM
	wizards.xml	114 KB	XML Document	3/29/2010 12:05 AM

Files which you need to know and will use are:

- **lisa.properties** – This file contains properties used by LISA. This file **should not** be edited as it will be overwritten when LISA is updated to a later version. Use **local.properties** for any properties you wish to add.
- **local.properties** – This file contains properties you have added to LISA and will not be edited by LISA. This file can be created from scratch, or you can use the **_local.properties** template file by copying it and removing the leading underscore character. This file is read immediately after the **lisa.properties** file, and will override properties set in that file.
- **_local.properties** – This file is a template file for **local.properties** file.
- **logging.properties** – This file stores logging properties. You will sometimes find it useful to change the logging level in this file (log4j.rootCategory) to get more logging information from LISA Workstation.

All other files in the install directory are either of not much concern to the user or they are specialized files or directories that will be used very infrequently. When they are required, they will be referenced in the [LISA User Guide](#) or the [LISA Reference Guide](#).

1.6 LISA Installation Notes

1.6 LISA Installation Notes

Using LISA with Your Java Environment

You can replace the default JRE used by the LISA installation - with your own Java environment. To do so you must understand how LISA selects the JRE to use.

LISA uses the following **priority** to select what Java VM to use when launching LISA applications:

1. The LISA installed JRE (in a directory called jre in the LISA install directory)
2. LISA_JAVA_HOME environment variable
3. JAVA_HOME environment variable
4. JDK_HOME environment variable

To use your JAVA environment:

1. Rename the JRE directory in the LISA installation directory (to something like 'jre_default' for example).
2. Point the LISA_JAVA_HOME environment variable to your Java installation directory.

Environment Settings

LISA documentation mentions of a token called **%LISA_HOME%** or **\$LISA_HOME**. This is the location that LISA was installed.

On all supported platforms, an environment variable is set with this name automatically from the launch scripts or programs.

For example, if you installed into C:\LISA, that would be the value of %LISA_HOME%. LISA Workstation also has access to the value of this variable in a property called LISA_HOME.

- If you need to put additional jars, zips, or directories in LISA's CLASSPATH, you have two options:
 1. Define the environment variable LISA_POST_CLASSPATH and set the resources you want there.
 2. Put them in the %LISA_HOME%\hotDeploy directory,
- If you need to send additional parameters to the underlying Java engine when LISA starts, there is a LISA_MORE_VM_PROPS environment variable for that purpose.

For example, you can set **LISA_MORE_VM_PROPS=-DSERVER=www.itko.com** before you start LISA, and every test run will start with a LISA property called SERVER with that value.

For more information on Environment settings, refer to [Common LISA properties and Environment variables](#).

Generic UNIX installer

If you encounter any difficulties using the MAC or Unix installers, we provide a Generic Unix tar file on the download site: {+}
<http://www.itko.com/downloads/ga/+>

1. Download and expand the tar file.
2. Run **configure.sh** - you may have to change the permissions on this file.
3. Copy the **tools.jar** file that came with your Java install into either the LISA "lib" directory or the LISA "hotDeploy" directory. The "lib" and "hotDeploy" directories are in the LISA installation directory. The jar files in the "lib" directory are only loaded once, but the jar files in the "hotDeploy" directory are loaded whenever there is a change to the "hot deploy" directory contents.

LISA Known Issues

For issues related to LISA, please visit:

[LISA Known Issues 5.0](#)

2. Installing and Running LISA Demo Server

2. Installing and Running the LISA Demo Server

LISA Demo Server comes with a **variety of examples** that should help you get started with LISA.

There are two ways to access the Demo Server to view LISA samples:
Either by installing it locally on your machine or by accessing it remotely from the iTKO web site.

Local Demo Server

iTKO provides a bundle of the entire server hosted on **examples.itko.com** that you can download and run by yourself. We recommend that you **download the LISA Demo Server** with your LISA install so that you can launch the demos from the desktop shortcut or the Windows start menu.

If you download the "**Windows Workstation with samples**" installer as described in previous chapter ([1.2.1 Installation Login](#)), the JBOSS Demo Server is already included in that installation. It is also included in all LISA Server installers.

Remote Demo Server

The same local demo server, which demonstrates LISA's features and capabilities is also available on the iTKO website.

You can run the examples directly from iTKO website at: [www.http://examples.itko.com](http://examples.itko.com)

However, several of these examples utilize *non-web ports.

For example, many J2EE Servers use port 1099 to look up JNDI names. If you are having success with the web site examples but not the others, you are probably behind a **firewall** that is preventing you from accessing those services.

This chapter describes how to install and run the **LISA Demo Server Locally**.

The following topics are available.

- [2.1 Installing the JBOSS Demo Server](#)
- [2.2 Starting the Demo Server](#)
- [2.3 Example Test Case using Demo Server](#)

2.1 Installing the JBOSS Demo Server

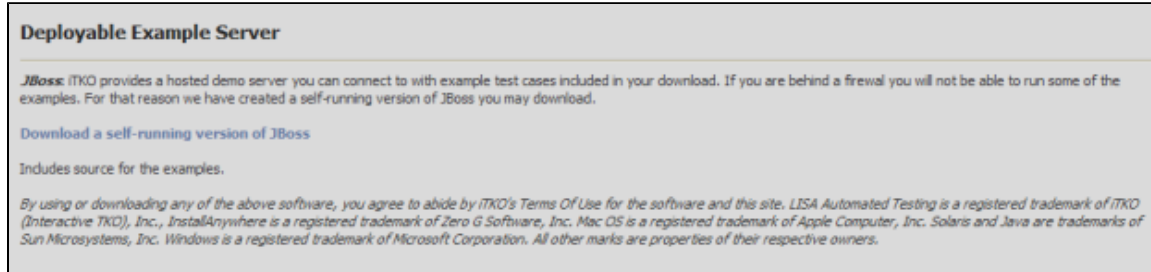
2.1 Installing the JBOSS Demo Server

You can download the;"**Demo Server**" from the same location where you downloaded LISA:

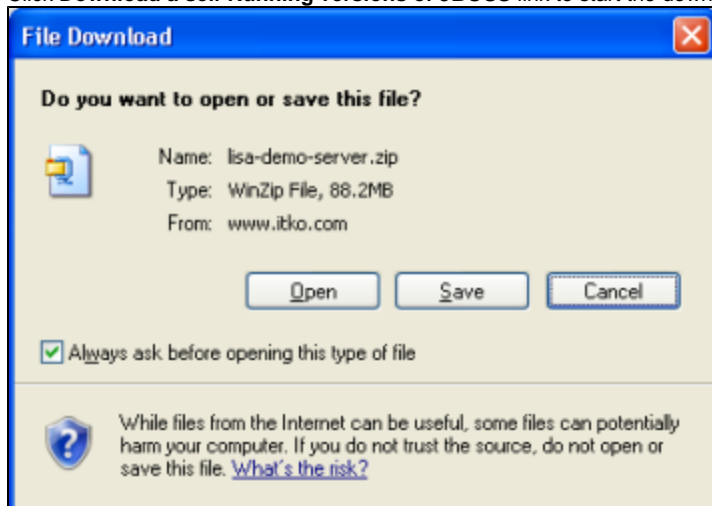
<http://www.itko.com/downloads/ga/http://www.itko.com/downloads/ga/>

This site is password protected. Contact your iTKO [sales representative](#) if you have any questions on your login credentials.

Enter the given **Username** and **Password** and click **OK**, to open the LISA GA download page. Scroll down to **Deployable Example Server** section as shown below:



Click **Download a self Running versions of JBOSS** link to start the download.



The JBOSS Demo Server download file "**lisa-demo-server.zip**" is around **88 mb** and includes the JBOSS Server with all the LISA examples.

To install the JBOSS deployable Server,

Once the Zip file is downloaded, extract the zip file contents into a local directory on your machine.

Notes:

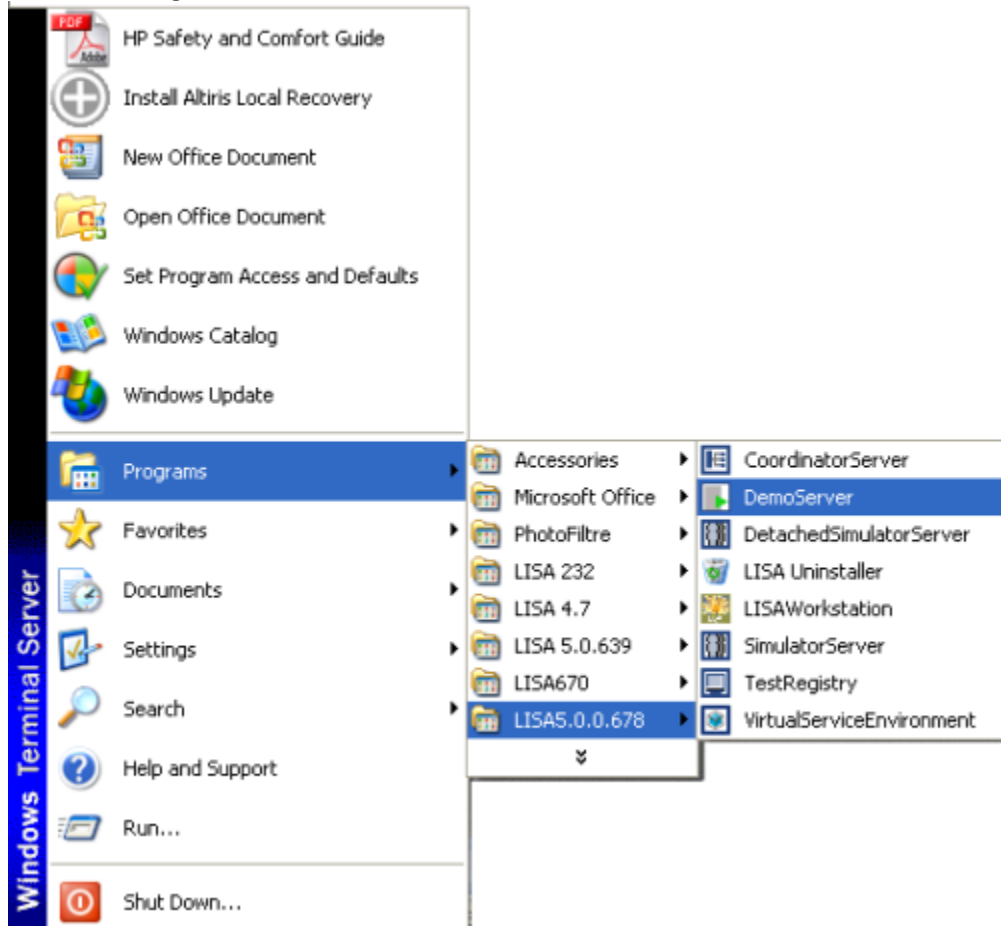
- 1> Make sure you set an environment variable "JAVA_HOME". This is required for JBOSS to compile and run JSP files etc.
- 2> Do not put the JBOSS Server directory on your desktop or any path with spaces in it. JBOSS will not be able to compile the JSPs if there is a space in the path to its directory.

2.2 Starting the Demo Server

2.2 Starting the Demo Server

To start the example Server,

Select **Start>Programs>LISA>Demo Server**



Or click **LISA Demo Server**, from the Desktop icons (if installed there).

There is a **start up script** for each different operating system.

For example, the file **start-windows.bat** is for Windows OS and **start-osx.command** will be installed for the MAC OS.

There is also an **info.txt** file which tells you how to run the demo app as shown below:

Name ▲	Size	Type	Date Modified
jboss		File Folder	8/10/2009 11:57 AM
hsqldb.jar	603 KB	Executable Jar File	7/15/2009 9:49 PM
info.txt	2 KB	Text Document	7/15/2009 9:49 PM
start-windows.bat	1 KB	MS-DOS Batch File	7/15/2009 9:49 PM

2.3 Example Test Case using Demo Server

2.3 Example Test Case using Demo Server

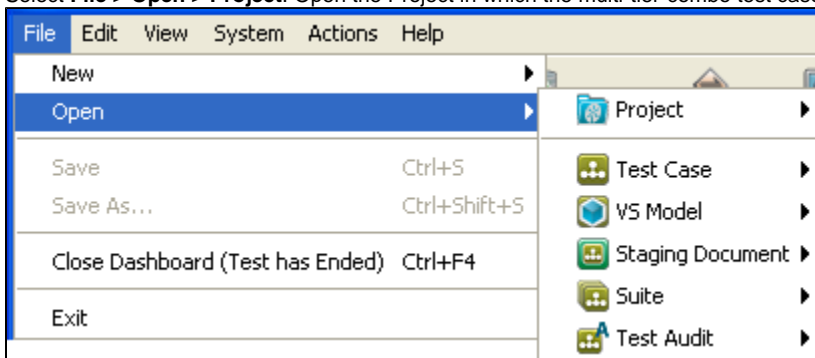
This section explains how to load and run the sample Test Case **multi-tier-combo.tst** using the demo server. The **multi-tier-combo.tst** test case is located in the %LISA_HOME%\examples directory.

Note: Remember to [start the local Demo Server](#) before opening the test case.

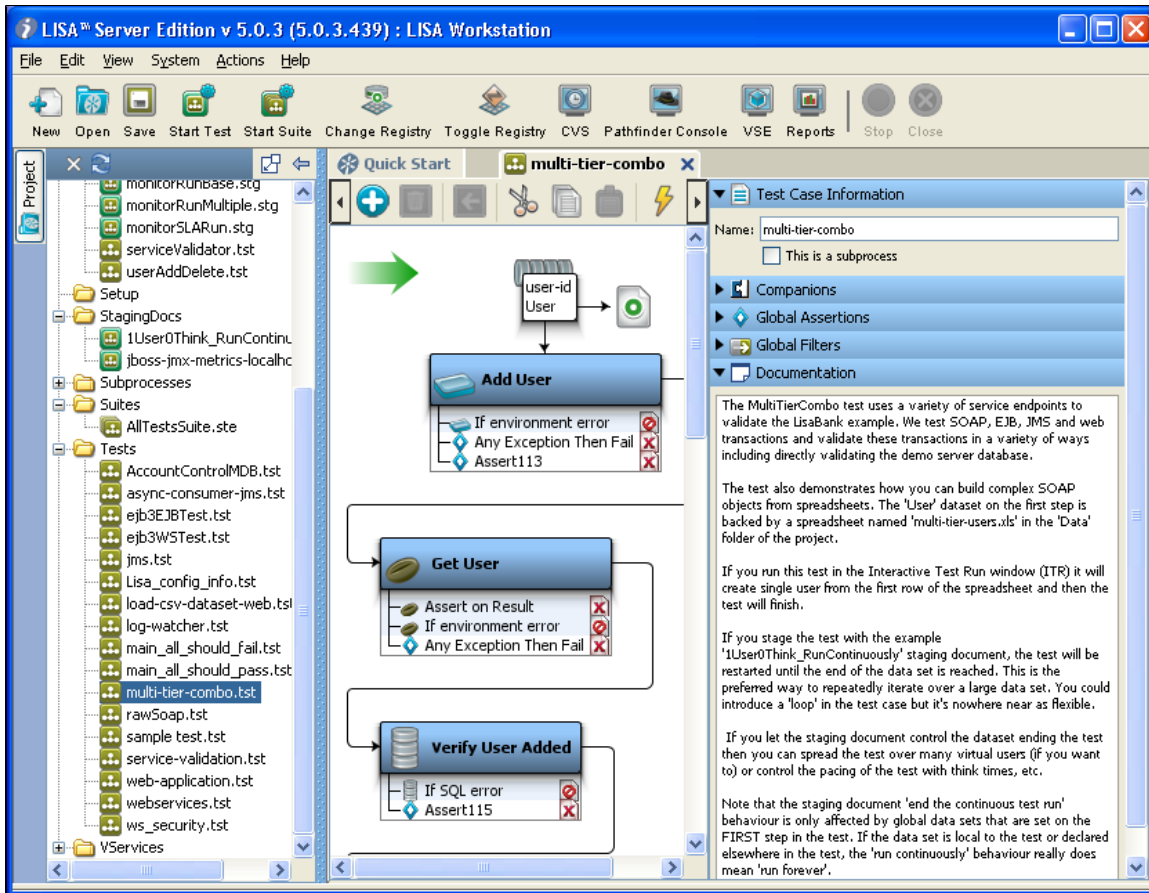
Viewing the Example Test Case

To view the example Test Case, complete the following Steps:

Select **File > Open > Project**. Open the Project in which the multi-tier-combo test case is present.



Within the Project, click the test case to open in the Model Editor. This will open the test case in the **LISA Model Editor** as shown below:

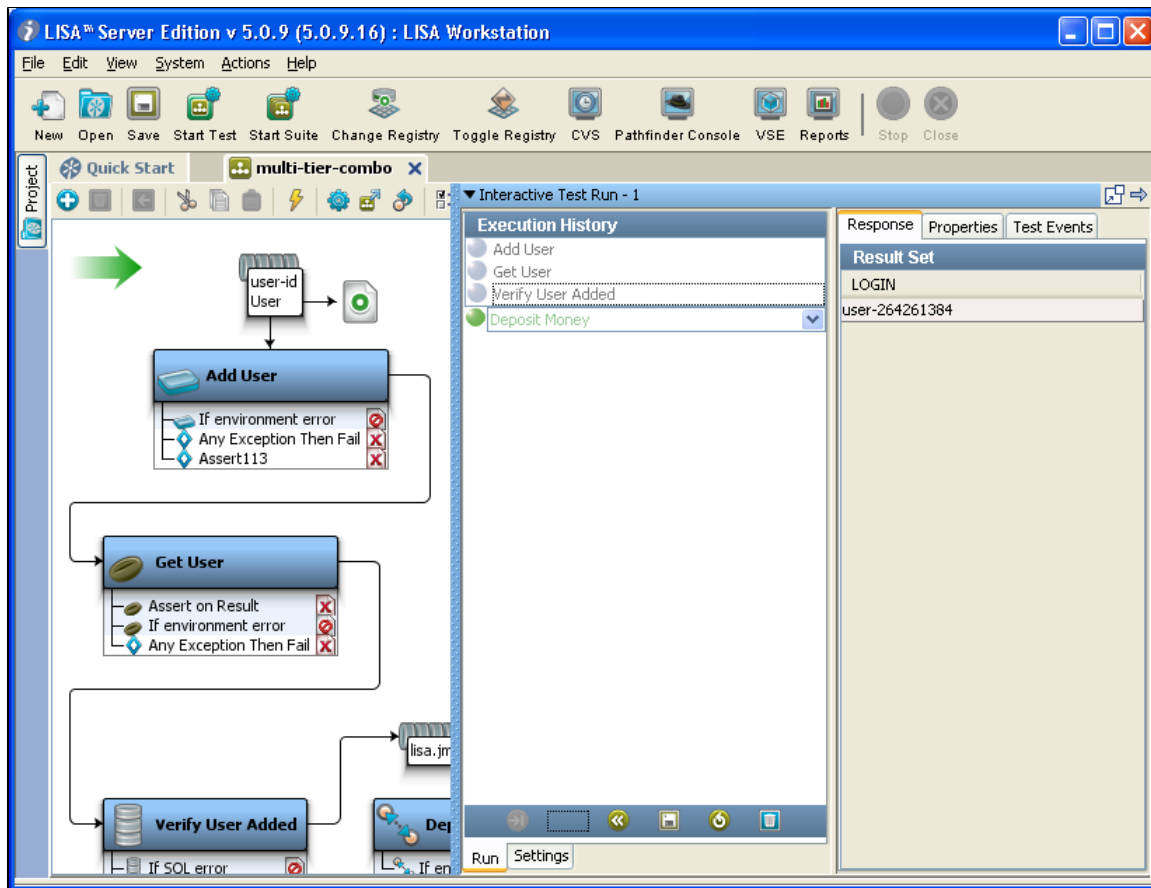


The demo can be run against the **iTKO example Server** hosted on the internet or against the deployable example **Server running locally**. To run the example using the JBOSS Demo Server running locally, select the **localhost** configuration. For more information on the available demo servers, see [2.1 Installing the JBOSS Demo Server](#).

Running the Example Test Case

You can run the created test case in the LISA ITR – Interactive Test Runner. To run the example Test Case in LISA's ITR, Select **Commands>Start a New Interactive Test Run**.

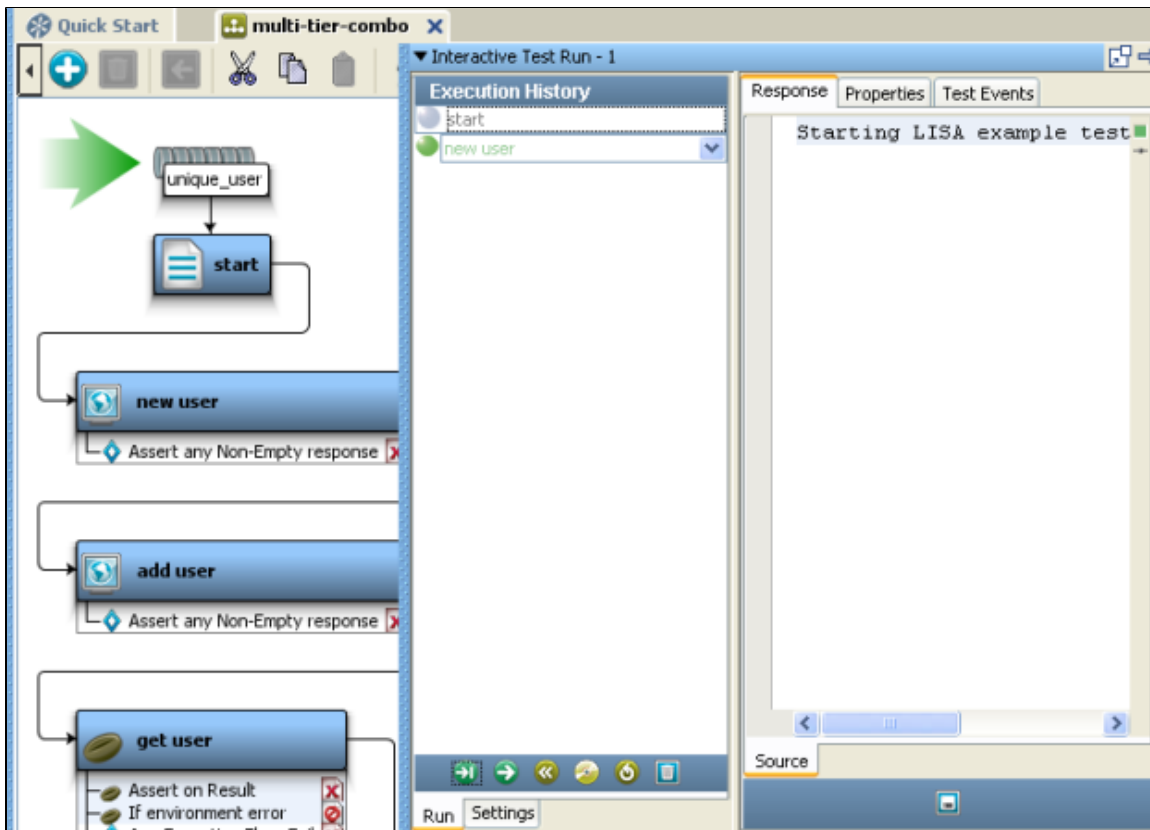
The Interactive Test Runner window slides from the right side as shown below:



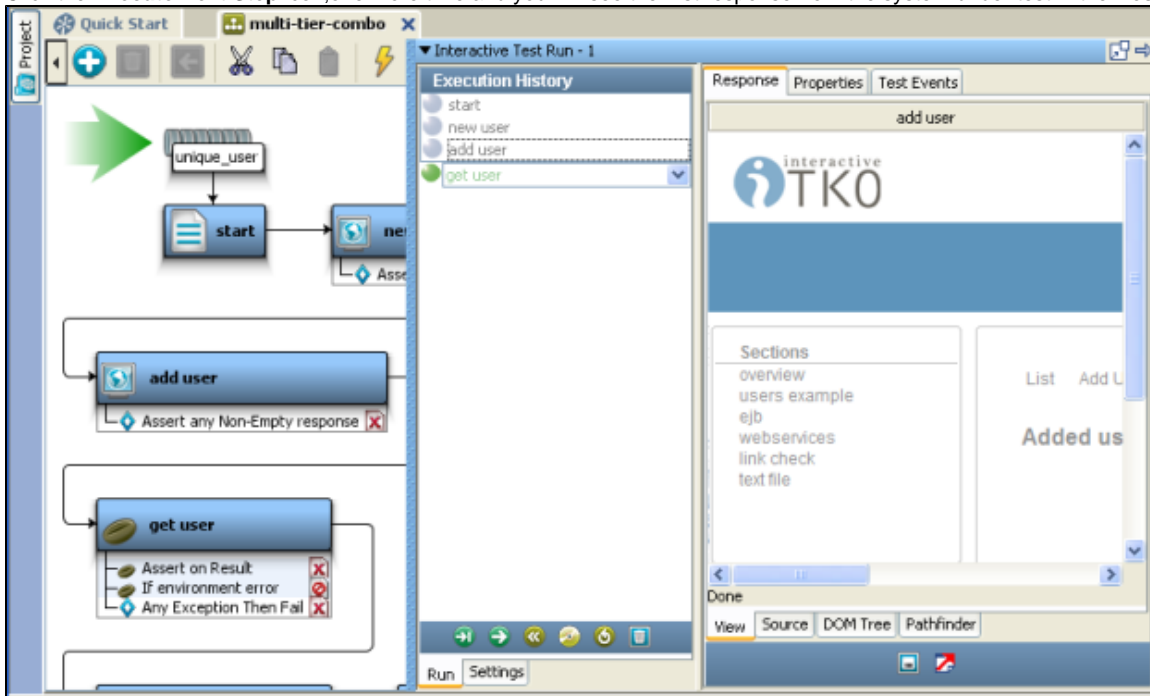
To start executing the test, one Step at a time, click **"Execute Next Step"** icon on the ITR toolbar.

To run the entire test case, in one shot, click **"Automatically Execute Test"** icon !2.3 Example Test Case on the ITR toolbar.

When one test Step is complete, you will see the following:

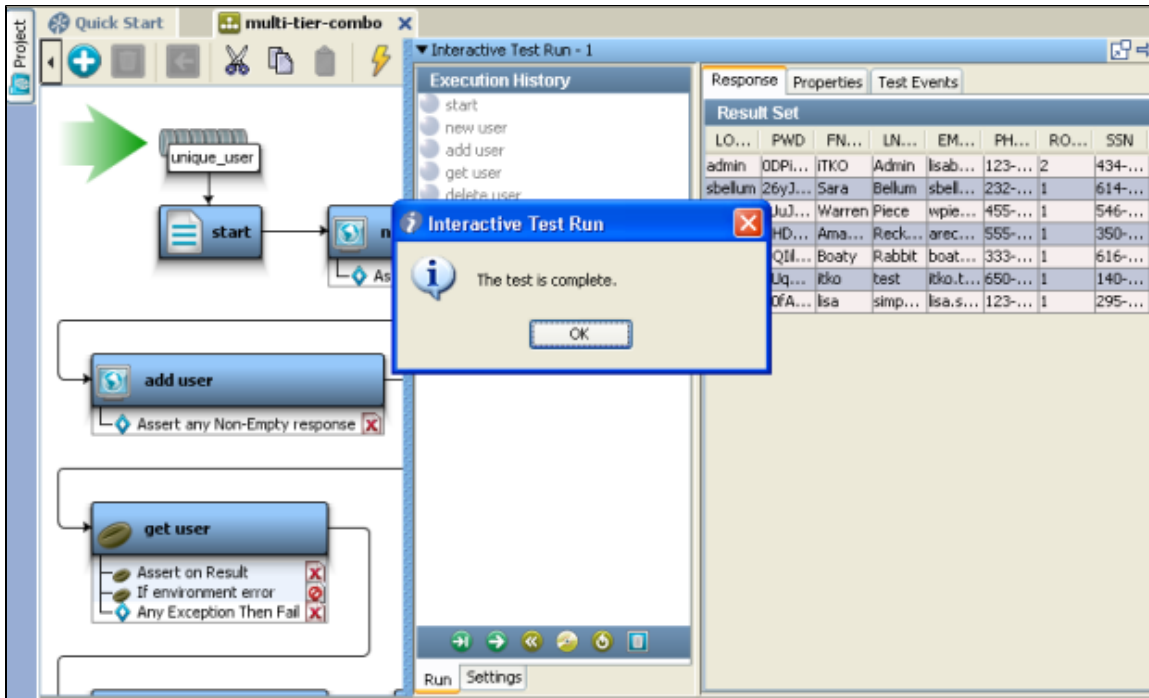


Click the **Execute Next Step** icon; one more time and you will see the first response from the system under test in the **Response** tab.



You may continue the entire test's execution or click on the various tabs located across the top of the screen (**Response**, **Properties** and **Test Events** tabs) to see other data that is provided by LISA as the test is executed.

Finally when the test ends, there is a **Test is Complete** dialog box as shown below:



For more information on what these screens provide, please see the [LISA User Guide](#)

3. Installing and Running LISA Server

3. Installing and Running LISA Server

LISA Server is the main center of LISA. It is connected to one or many LISA workstations. Some of the components of LISA server are different than LISA Workstation.

The following chapters are available.

- [3.1 Introduction to LISA Server](#)
- [3.2 LISA Server Hardware Requirements](#)
- [3.3 LISA Server Components](#)
- [3.4 LISA Server Architecture](#)
- [3.5 Installing and Configuring the LISA Server](#)
- [3.6 Miscellaneous Configuration Notes](#)

3.1 Introduction to LISA Server

3.1 Introduction to LISA Server

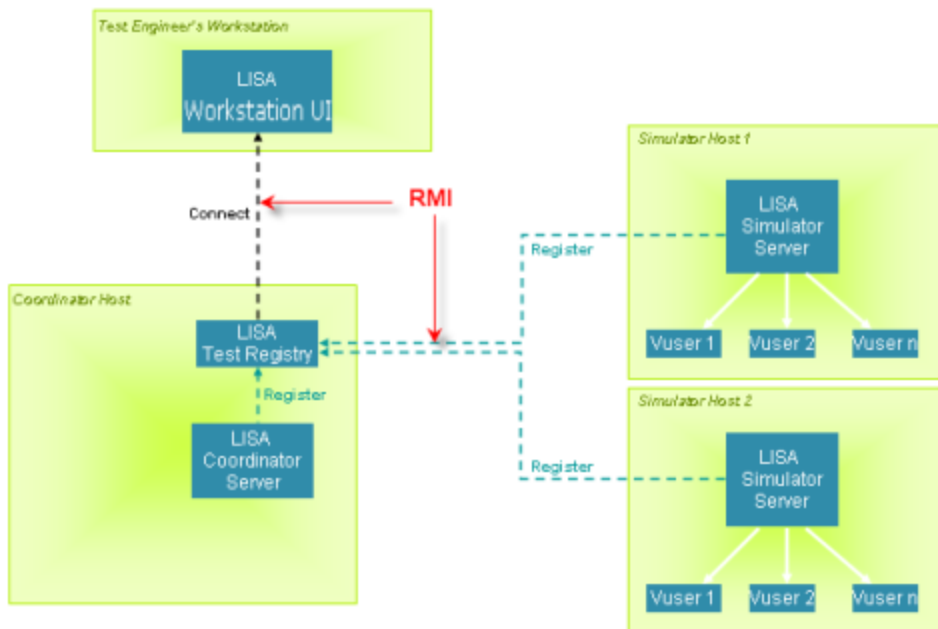
The architecture of a LISA Server is more complex than the LISA Workstation. The LISA Server uses the **testmanager.exe** to create and run the test as well as store and produce Test Reports.

The LISA Server is made up of four components:

1. LISA Workstation
2. Registry
3. Coordinator Server
4. Simulator Server

The LISA Workstation is used to create and monitor the tests, but the test cases are run in the LISA Server RMI environment. A Coordinator Server and a Simulator Server are embedded in the LISA Workstation. These are described in detail in the next section.

The LISA Server architecture is as shown below:



All the tests here are run by the virtual users (or Simulators) spawned by the Simulator Server under the supervision of a Coordinator in the Coordinator Server.

Each Simulator connects to and invokes actions on the system under test. A load test results when the virtual users are running in a parallel mode.

A LISA Server can be configured to run on a single CPU, including the one that the LISA Workstation is using, or on a large distributed environment incorporating many CPUs. Your testing requirements dictate the Server architecture to be used.

All LISA Server components communicate with each other using **Java RMI** (Remote method Invocation)

LISA Server components communicate with the client system and the LISA Workstation communicates with the LISA Server.

3.2 LISA Server Hardware Requirements

3.2 LISA Server Hardware Requirements

- CPU 2X2GHz or faster
- RAM 2-4GB or more
- Disk Space 5GB free space
- OS Windows 2000, 2003, XP, Linux, Solaris
- Database Not required but recommended to use MySQL, Oracle or SQL Server for production usage. DB can reside on a different system.

3.3 LISA Server Components

3.3 LISA Server Components

In **LISA Server** the tests are run in the Server environment. LISA Workstation connects to the Server to deploy and monitor tests that were developed in Workstation.

Workstation is one of four applications that make up the LISA Server environment.

In addition to Workstation, there is at least instance of each of the following:

- **LISA Registry:** A registry that keeps track of all the Coordinator and Simulator Servers. It is not common to have more than one LISA Registry in a LISA Server installation. Workstation attaches to this registry
- **Coordinator Server:** Receives the test run information, in the form of documents, and coordinates the tests that are run on one or more Simulator Servers. The Coordinator Server manages metric collection and reporting. A LISA Server environment can have, and commonly does have more than one Coordinator Server.
- **Simulator Server:** Runs the tests under the supervision of the Coordinator Server. The Simulator Servers instantiate virtual users to run test instances against the system under test. For large tests with many virtual users, virtual users can be distributed amongst several Simulator Servers. For information on "Calculating Simulator Instances" and "Starting two Simulator Instances at the same time" refer the [LISA Installation and Configuration Guide]confluence/display/DOC/LISA+Installation+and+Configuration+Guide].

Any number of Workstations can attach to the LISA Registry and share the Server environment.

The design of LISA Server architecture, and the installation and configuration of LISA Server are beyond the scope of this section.

For more information on LISA Server architectures and detailed instructions on how to install and configure LISA Server see the [LISA Installation and Configuration Guide][confluence/display/DOC/LISA+Installation+and+Configuration+Guide](#)].

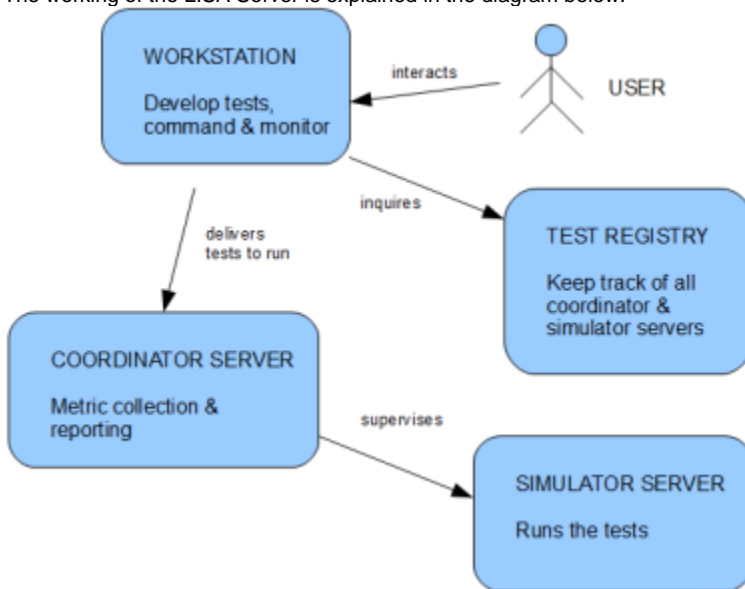
LISA SERVER Components

LISA Server consists of four major components:

- LISA Workstation
- Test Registry
- Coordinator Server
- Simulator Server

The last three are headless Java applications that run in separate Virtual Machines. All these components are explained later in this section.

The working of the LISA Server is explained in the diagram below:



There must be at least one of each of these components in a minimal LISA Server configuration. There can be as many instances of each type as is needed for a specific testing environment. Typically there will be only **one Test Registry**, **one Coordinator Server**, but **more than one Simulator Server** in a LISA Server configuration.

NOTE - For LISA VSE, it is a server level service. It can co-exist with a TestRegistry that has a Coordinator and Simulator attached to it, but the Simulator and Coordinator are not mandatory to run VSE.

LISA Workstation

All tests are created, monitored and run in the LISA Workstation.

For more information, see [1. Installing and Running LISA Workstation](#).

Test Registry

The Test Registry is very much like an RMI registry. Its primary function is to keep track (or be a registry) of the Coordinator and Simulator Servers that are part of the LISA Server installation. LISA Workstation connects to the Test registry to keep track of the Coordinator and Simulator Server.

For more information, see [1.4 LISA Registry](#)

Coordinator Server

The Coordinator Server is a Java application whose responsibilities include:

- Receiving the information required to run a test, from the LISA Workstation.
- Coordinating the test runs on the Simulator Server(s).

- Collecting the result data for use in LISA reports.
- Communicating the test data to LISA Workstation for monitoring purposes.

For more information, see [3.4.5 Creating and Running Coordinator Servers](#).

Simulator Server

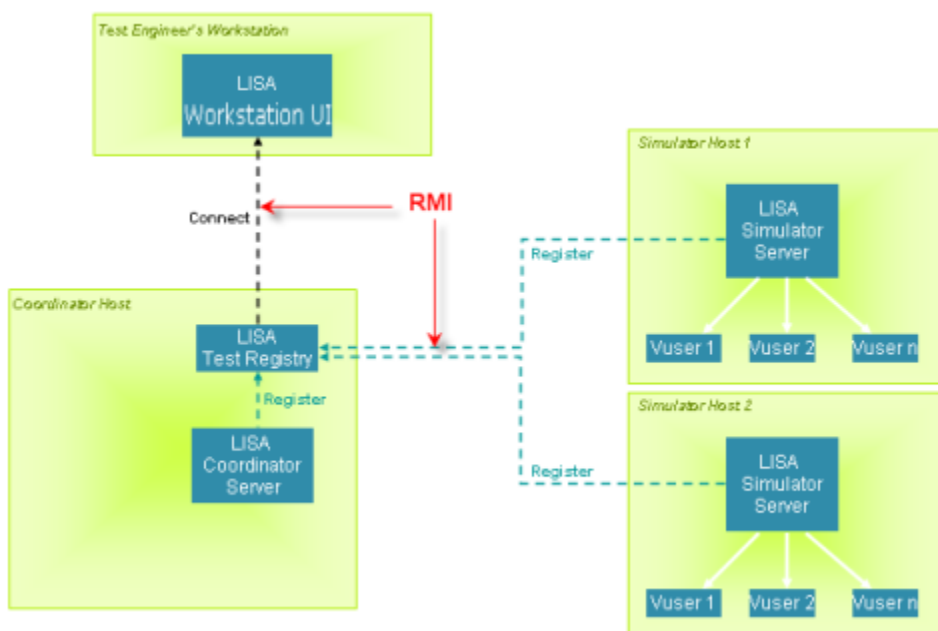
The Simulator Server(s) is responsible for running the tests, and reporting the results back to the Coordinator Server. Virtual users or test instances will be created and run on the Simulator Servers(s). The number of virtual users, and hence the number of Simulator Servers deployed will depend on the nature of the tests being performed. Each virtual user will be in communication with the client system.

For more information, see [3.4.6 Creating and Running Simulator Servers](#).

3.4 LISA Server Architecture

3.4 LISA Server Architecture

The LISA Server architecture is as shown below:



All the tests here are run by the virtual users (or Simulators) spawned by the Simulator Server under the supervision of a Coordinator in the Coordinator Server.

Each Simulator connects to and invokes actions on the system under test. A load test results when the virtual users are running in a parallel mode.

A LISA Server can be configured to run on a single CPU, including the one that the LISA Workstation is using, or on a large distributed environment incorporating many CPUs. Your testing requirements dictate the Server architecture to be used.

All LISA Server components communicate with each other using **Java RMI** (Remote method Invocation)

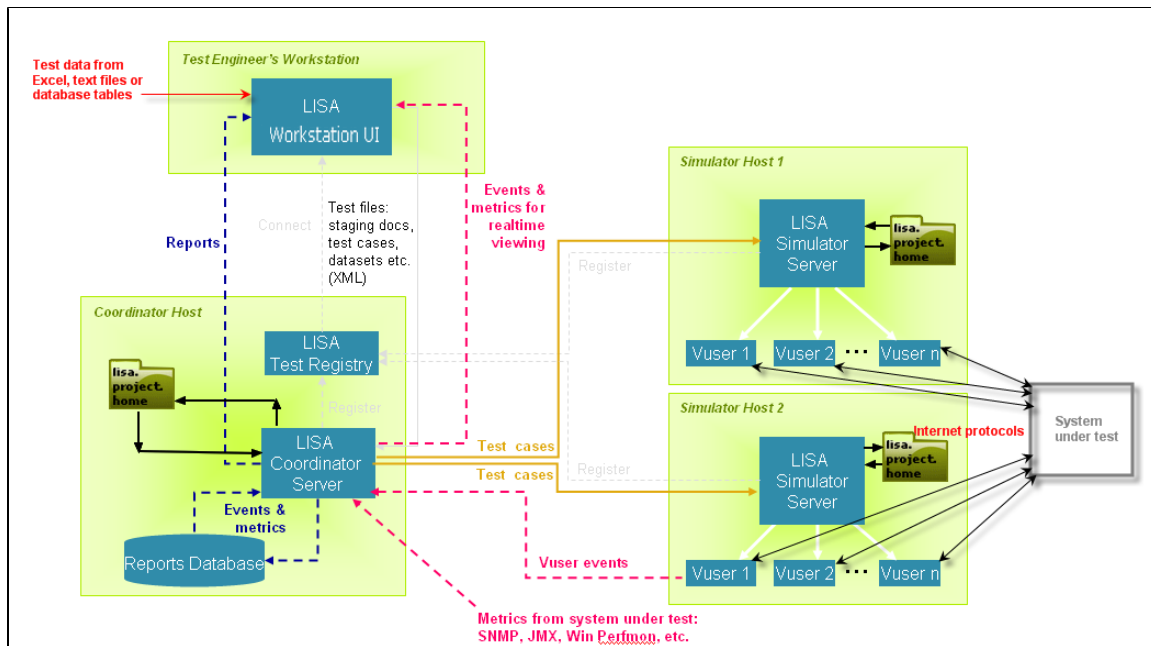
LISA Server components communicate with the client system and the LISA Workstation communicates with the LISA Server.

3.4.1 Data Flow within LISA Server

3.4.1 Data Flow within LISA Server

In LISA Server, the data flow is more complex because of the **inter-component** communication happening within the Server.

The data flow is graphically shown as below:



The data flow between components is explained below:

a. For tests/suites within a LISA project,

- 1) Copy the project anywhere under the folder defined by the property **lisa.projects.home** in local.properties of all coordinator & simulator servers.
- 2) Once set, all the test assets, including datafiles, staging docs, suites etc, can be found server side at runtime.

b. For tests/suites not within a LISA project,

- 1) Text Data comes from Excel sheets or text files and is read from external datasets by LISA Workstation.
- 2) Test case documents, staging documents, and test suite documents (all XML) are transmitted to the Coordinator Server.
- 3) Data sets are serialized and transmitted to the Coordinator Server.
- 4) Test cases are transmitted from the Coordinator to the Simulators Servers.
- 5) Each virtual user (test instances) communicates with the system under test via various internet protocols.
- 6) Virtual users send Event data back to the Coordinator and is stored for later reporting.
- 7) Metrics are sent from the system under test to the Coordinator and is stored for later reporting.
- 8) Event and Metric data is sent to LISA Workstation during the test for real time monitoring.
- 9) Reports are sent to LISA Workstation on request after the test runs are complete.

To know more about running the tests in Server Environment, see [Running Tests in Server Environment](#).

3.5 Installing and Configuring the LISA Server

3.5 Installing and Configuring the LISA Server

The procedure to download the LISA Server is identical to that for LISA Workstation, and is described in detail in [1. Installing and Running LISA Workstation](#). Once you have downloaded the installer for LISA Server, use it to install LISA on each machine that you plan to use in your Server environment as explained in [1. Installing and Running LISA Workstation](#).

Configuring LISA Server Licenses

The license information for LISA Server must be added to each of your **local.properties** files.

The easiest way to do this is to make a copy of the **_local.properties** file and remove the leading underscore.

When you open this file in a **text editor** you will see the properties already there (commented out and with dummy values):

```
1. =====
   #license properties
2. =====
   #laf.server.url=https://license.itko.com
   #laf.domain=iTKO/LISA/YOURCO
   #laf.username=YOURUSERNAME
   #laf.password=YOURPASSWORD
```

Remove the # signs (comment symbol) and copy and paste the values of LISA Server, domain, username and password from your license email into these properties.

If you are using an **HTTP proxy Server** you will have to provide additional information.

The properties that you must add are also in the **local.properties** template immediately after the properties shown above:

```
1. =====
   #license properties if using an http proxy server
2. =====
   #laf.usehttpproxy.server=true
   #laf.httpproxy.server=my_proxyserver.com
   #laf.httpproxy.port=3128
3. === if your proxy server requires credentials - leave blank to use native NTLM authentication
   #laf.httpproxy.domain=if needed for NTLM
   #laf.httpproxy.username=if needed
   #laf.httpproxy.password=if needed
```

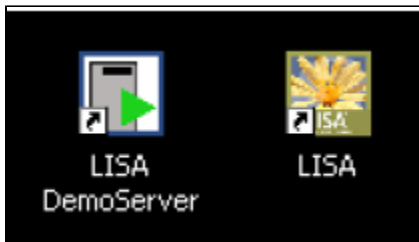
3.4.1 Starting LISA Server

3.4.1 Starting LISA Server

Once you install the LISA Workstation, you are ready to get started. When you install LISA Server, there will be additional items added in the **LISA start menu** – Coordinator Server, Simulator Server, Test Registry.

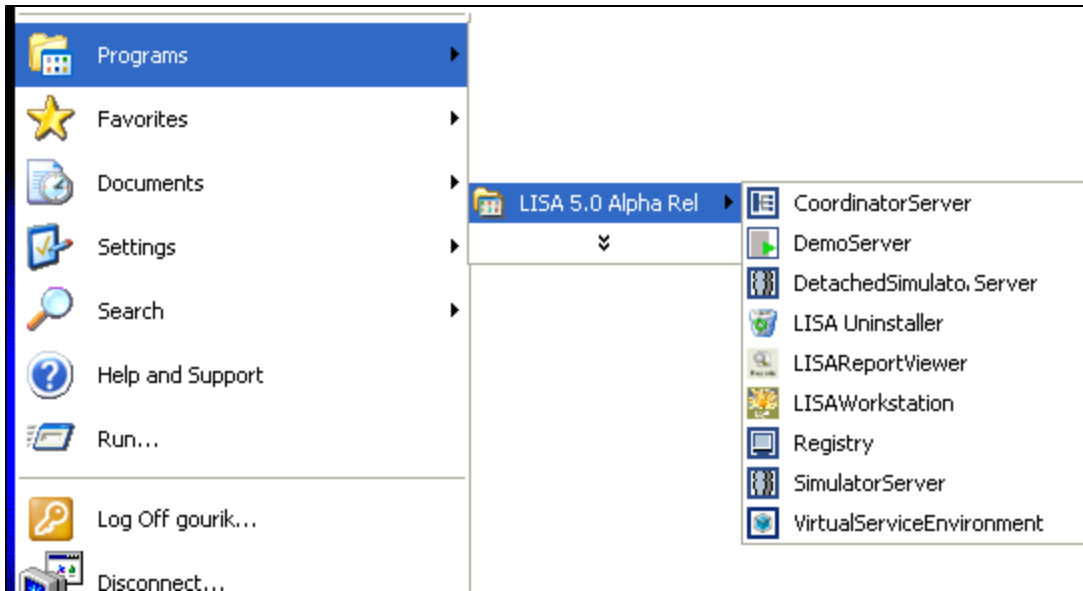
There are several ways to start the LISA Server on Windows:

1. Desktop: By Double-clicking the LISA icon on your desktop (if you have chosen to create a desktop icon during installation).



2. Command Line: By using a startup script from a command window. Open a command prompt, and navigate to the bin directory in the LISA install directory (the one you specified above) and running LISAWorkstation.exe.

3. Program Menu: Select *Start > Programs > LISA > *as shown in the figure below:



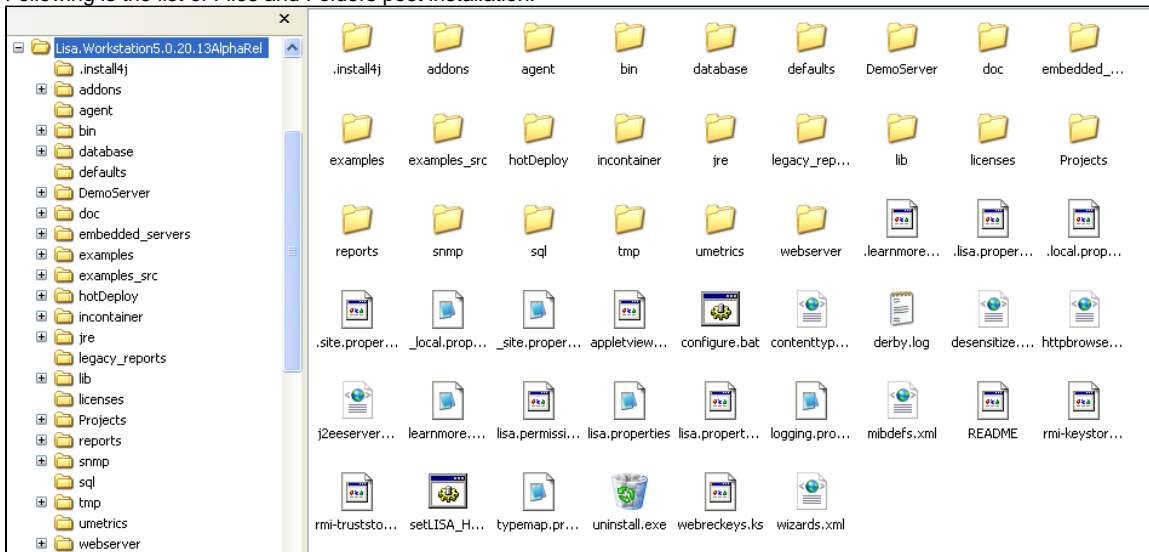
- Click on **LISAWorkstation** icon to start the LISA Workstation.

You can also start a default instance of each Server component from this menu simply by starting the three components in order:

- Registry
- CoordinatorServer
- SimulatorServer

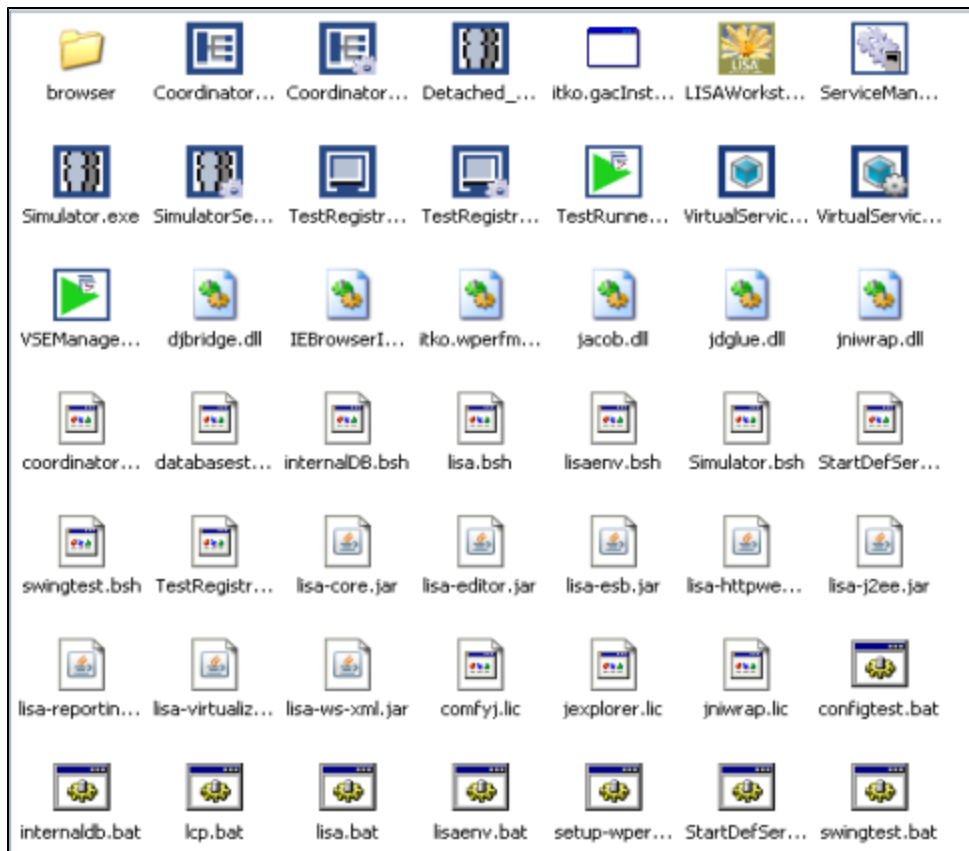
Post Installation Directories

Following is the list of Files and Folders post installation:



Note - The "Projects" directory is only created once you create a Project within LISA Workstation. All the created projects are listed in this directory.

Clicking on %LISA_HOME%/bin directory, will list all LISA Workstation related executables:



There is also a simple ".bat" file - **StartDefServers.bat**, which will start the default Servers. On non-Windows platforms you will find the equivalent executables in the "bin" folder.

Starting as Services

All the above mentioned three components (Registry, Coordinator Server, Simulator Server) can be started as "**Services**" on Windows.

Within LISA, there are three executables to do this in the %LISA_HOME%\bin directory. All of these have the word "Service" appended to the executable name as shown below:

browser	
xulrunner-windows	
CoordinatorServer.exe	189 KB
CoordinatorService.exe	208 KB
CVSManager.exe	205 KB
Detached_Simulator.exe	149 KB
itko.gacInstall.exe	8 KB
LISAReportViewer.exe	457 KB
LISAWorkstation.exe	513 KB
Registry.exe	189 KB
ServiceManager.exe	205 KB
Simulator.exe	149 KB
SimulatorService.exe	192 KB
TestRegistry.exe	189 KB
TestRegistryService.exe	192 KB
TestRunner.exe	205 KB
VirtualServiceEnvironment.exe	142 KB
VirtualServiceEnvironmentService.exe	146 KB
VSEManager.exe	205 KB

Note: Every Server component MUST have a name, and Coordinator Servers and Simulator Servers must name a Test Registry to attach to

itself.

Also see [Starting and Stopping Services](#)

3.4.2 Getting Started

3.4.2 Getting Started

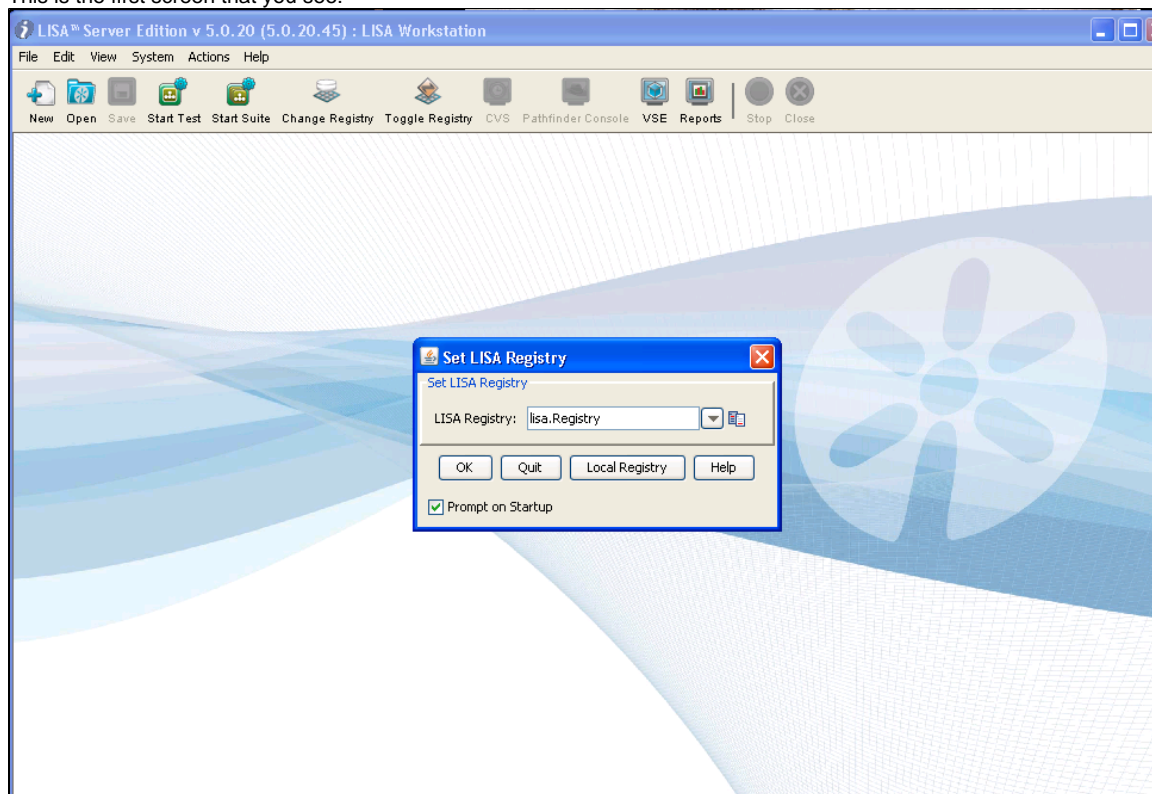
Once you install the LISA Server, you are ready to get started working in LISA.

Note that after Server installation, there will be additional items added in the **LISA start menu** –

-Coordinator Server
-Simulator Server
-Registry

- Click on the LISA Workstation icon to get started.
- You will need to enter your [license credentials](#). Upon entering them correctly, you can proceed to work in LISA Workstation environment.

This is the first screen that you see:



You will be asked to [Set the Registry](#) at start.

After selecting the Registry, click OK to get started within LISA Workstation Environment.

3.4.3 LISA Server Custom Configurations

3.4.3 LISA Server Custom Configurations

Some testing scenarios call for the test load to be distributed over several machines.

In this case, you can create as many named instances of the Server components as your scenario specifies.

- [Creating and Attaching Test Registries](#)
- [Creating and Running Coordinator Servers](#)
- [Creating and Running Simulator Servers](#)

3.4.4 Creating a Test Registry

3.4.4 Creating a Test Registry

We need to create a LISA Test Registry as the Reporting database, VSE database, pathfinder Agent, reporting console, and other services are all provided through the Registry.

Hence the Registry is very important for LISA Workstation.

To create a "named" Test Registry, open a command window and run:

```
%LISA_HOME%\bin\TestRegistry -n TestRegistryName
```

This creates a new Test Registry process associated with the name **TestRegistryName**.

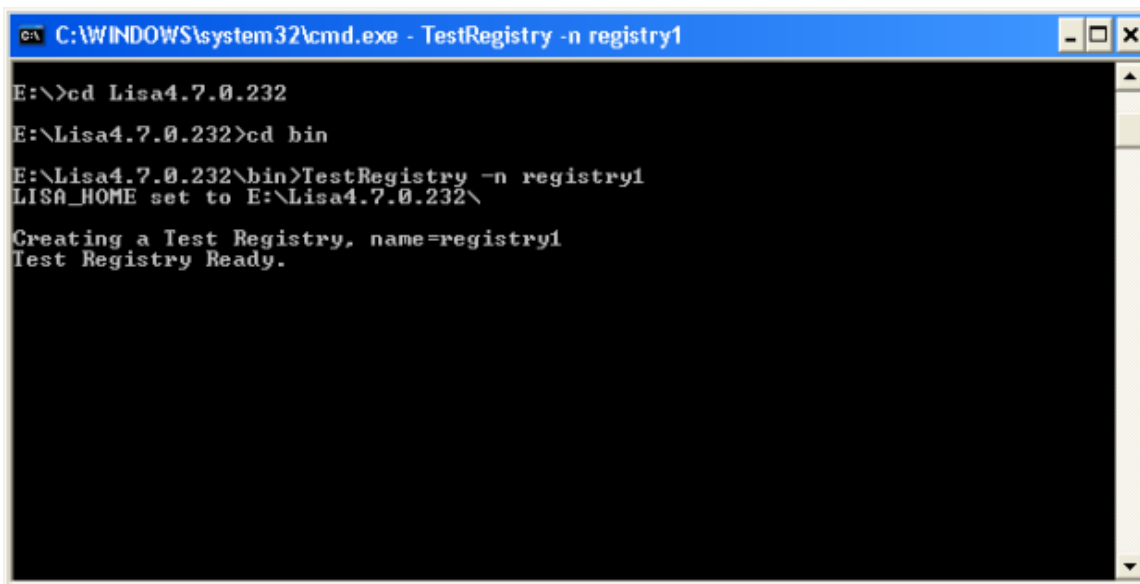
For example, to create a Test Registry called **registry1** on a machine where LISA is installed to **C:\Lisa404Alpha4623**, enter the following command in the command window:

```
C:\Lisa404Alpha4623\bin\TestRegistry -n registry1
```

Or, go to the "bin" directory and enter the following:

```
TestRegistry -n registry1
```

The TestRegistry starts, as seen below:



```
C:\WINDOWS\system32\cmd.exe - TestRegistry -n registry1

E:\>cd Lisa4.7.0.232
E:\Lisa4.7.0.232>cd bin
E:\Lisa4.7.0.232\bin>TestRegistry -n registry1
LISA_HOME set to E:\Lisa4.7.0.232\
Creating a Test Registry, name=registry1
Test Registry Ready.
```

For more information on Test Registries, see

[Setting LISA Registry](#) and Local mode of Registry

3.4.4.1 Changing Test Registry

3.4.4.1 Changing Current Test Registry

To attach a new Test Registry or change current Registry:

- Start LISA Workstation, upon startup it will show the "Set LISA Registry" dialog.
- Or Within LISA Workstation, Click **System > Registries > Set LISA Registry**, from the main menu.

The **Set Test Registry** window will appear.



Enter the **Name** of the Test Registry or select from the drop down and click **OK**.

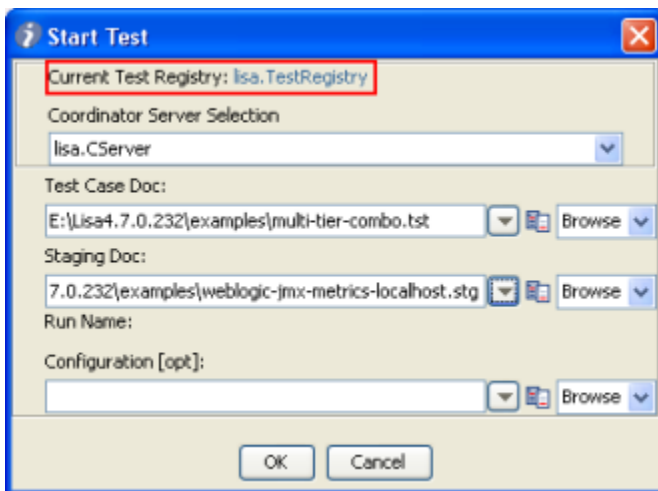
If you just type in the Test Registry name, LISA will default to "**rmi://localhost/**".

If the Test Registry is on a different machine than the LISA Workstation, specify the full Test Registry address/path in the **Test Registry** field in the above dialog.

For example, to connect to a Test Registry called **remoteTR** on a Server called **registryHost**, enter the value:

rmi://registryHost/remoteTR.

When you open the **Start Test** dialog as shown below, you can see the current Test Registry attached - as highlighted in Red.



3.4.5 Creating and Running Coordinator Servers

3.4.5 Creating Coordinator Servers

To create a Coordinator Server, open a command window and run:

```
%LISA_HOME%\bin\CoordinatorServer --n CoordinatorServerName -m TestRegistryName
```

This creates a new Coordinator Server process associated with the name **CoordinatorServerName** and attaches it to the Test Registry known as **TestRegistryName**.

For example, to create a Coordinator Server called **coordinator1** attached to a non-default Test Registry called **registry1** on a machine where LISA is installed to **C:\Lisa4.7.0.232**, enter the following command in the command window:

```
C:* *Lisa4.7.0.232\bin\CoordinatorServer -n coordinator1 -m registry1
```

or, go to the bin directory and type:

```
CoordinatorServer -n coordinator1 -m registry1
```

The Coordinator Server starts, as seen below:

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

M:\>e:

E:\>cd Lisa4.7.0.232

E:\Lisa4.7.0.232>cd bin

E:\Lisa4.7.0.232\bin>CoordinatorServer -n coordinator1 -m registry1
LISA_HOME set to E:\Lisa4.7.0.232\
Creating a CoordinatorServer, name=coordinator1, testRegistry=registry1
Aug 20, 2009 10:43:29 AM pt.ipb.agentapi.engine.snmp.JoeSnmpEngine open
INFO: Starting JoeSnmpEngine on port 1161
Then we will add this server to the Test Registry.

E:\Lisa4.7.0.232\bin>_
```

If the associated Test Registry is specified in the **LISA Workstation**, the Coordinator Server now appears in the **Coord Servers** tab of the **Test Registry Monitor**, as seen below:

Test Registry Monitor	
Tests	Simulators
Coord Servers	Virtual Environs
Server	Coordinators
lisa.CServer	1



To open the Test Registry monitor window, click **Toggle Registry** icon.

3.4.6 Creating and Running Simulator Servers

3.4.6 Creating Simulator Servers

To create a Simulator, open a command window and run:

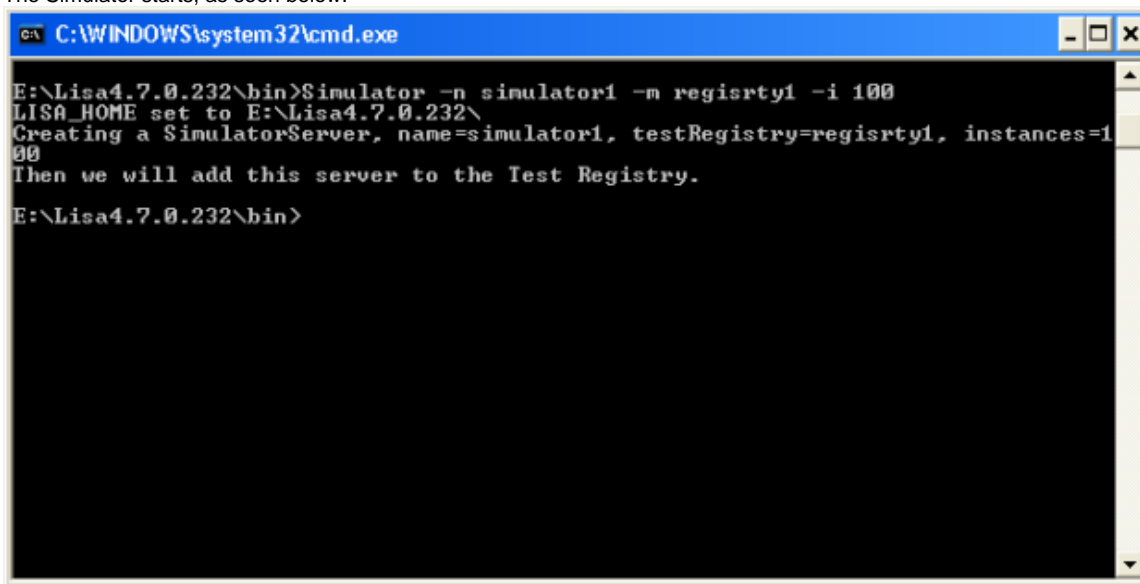
```
%LISA_HOME%\bin\Simulator -n SimulatorName -m TestRegistryName
```

This creates a new Simulator process associated with the name **SimulatorName** and attaches it to the Test Registry known as **TestRegistryName**.

For example, to create a Simulator called simulator1 attached to a Test Registry called **registry1** on a machine where LISA is installed to **E:*Lisa4.7.0.232**, enter the following command in the command window:

```
E:* *Lisa4.7.0.232\bin\Simulator -n simulator1 -m registry1
```

The Simulator starts, as seen below:

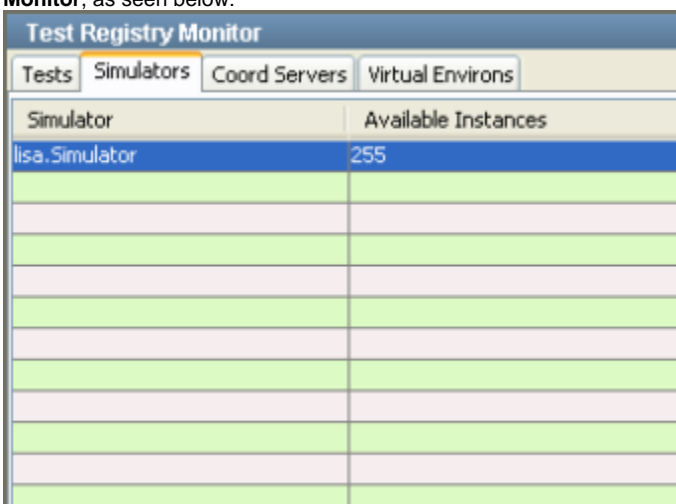


```
C:\WINDOWS\system32\cmd.exe

E:\Lisa4.7.0.232\bin>Simulator -n simulator1 -m regisrty1 -i 100
LISA_HOME set to E:\Lisa4.7.0.232\
Creating a SimulatorServer, name=simulator1, testRegistry=regisrty1, instances=100
Then we will add this server to the Test Registry.
E:\Lisa4.7.0.232\bin>
```

Note that you can specify the number of virtual users for this Simulator using the `--i` parameter (set to 100 in the figure above.)

If the associated Test Registry is specified in the LISA Workstation, the Simulator now appears in the **Simulators** tab of the **Test Registry Monitor**, as seen below:



Simulator	Available Instances
lisa.Simulator	255

Running Multiple Simulators

For Running multiple Simulators,

PI use command prompt to create simulators as shown above.

To avoid port conflict,

set `lisa.rmi.port.strategy=anon` in local.properties file.

For more info about LISA port usage,

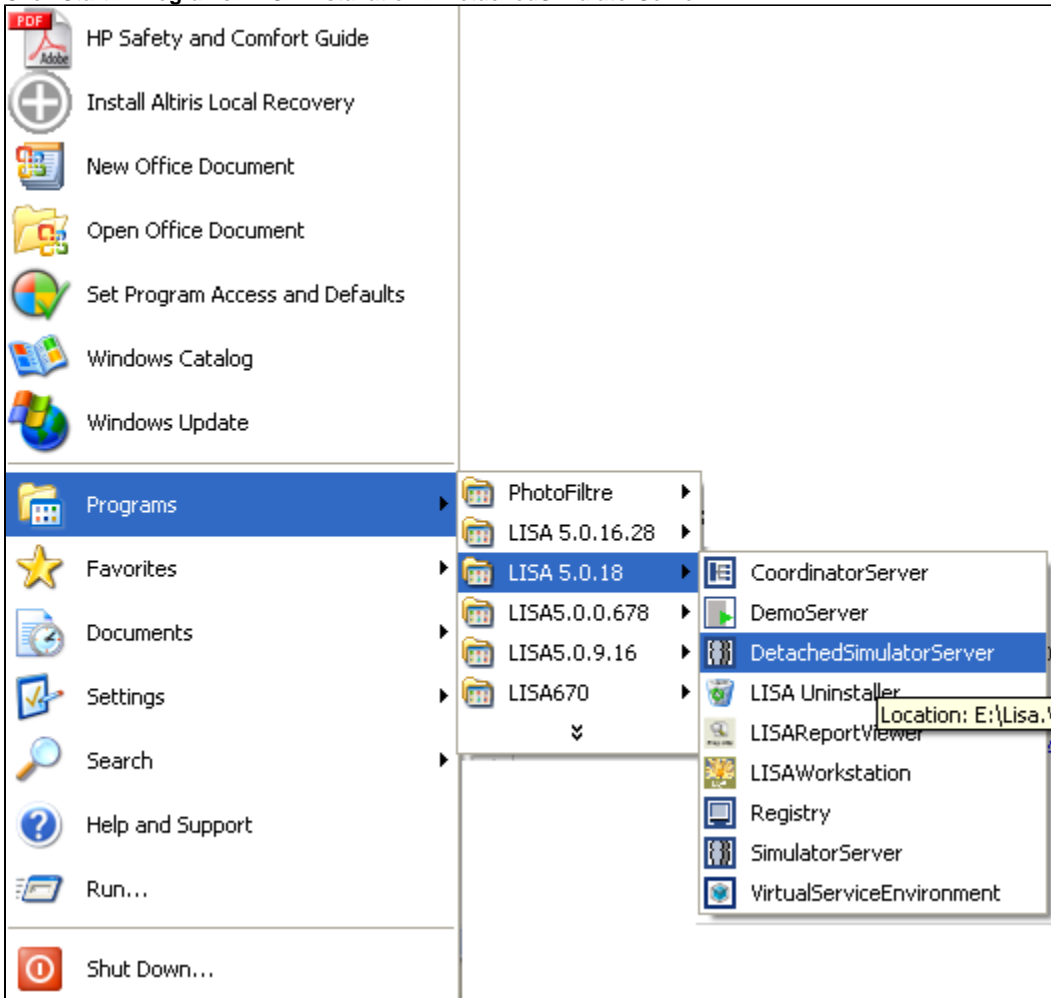
see [<https://support.itko.com/confluence/display/technotes/LISA+Port+Usage+-+4.x>][|](https://support.itko.com/confluence/display/technotes/LISA+Port+Usage+-+4.x)[confluence/display/technotes/LISA+Port+Usage+-+4.x](https://support.itko.com/confluence/display/technotes/LISA+Port+Usage+-+4.x)]

3.4.7 Running Detached Simulator Servers

3.4.7 Running Simulator in Detached Mode

You can detach the Simulator from the Test Registry either from the main menu or from the command line option as shown below:

- Click **Start > Programs> LISA Installation > DetachedSimulatorServer**



You can also detach the Simulator from the command line -

- **To run the Simulator Server in a detached mode**, add the **"-detached"** option to the Simulator Server's command line.

When a detached Simulator Server boots up, it will not attach itself to the Registry.

To run tests on a detached Simulator Server, set the **"Execute On"** property of a Step to the RMI address of the Simulator Server, such as "<rmi://localhost/lisa.Simulator>".

3.4.8 Starting and Stopping Service

3.4.8 Start and Stop Service

If you intend on leaving the Server components running most of the time, you can use the "Service" executables in **%LISA_HOME%\bin**:

To start the services,

Enter the following in the command prompt:

```
TestRegistryService start
CoordinatorService start
SimulatorService start
```

To Stop the services,

Enter the following in the command prompt:

```
SimulatorService stop
CoordinatorService stop
TestRegistryService stop
```

We recommend that you start and stop the components in the order as shown above.

Note: If the LISA Workstation started the report database (Derby), it gets shut down when LISA Workstation shuts down, but, if Coordinator started the database (Derby), it is **not** shut down when Coordinator shuts down.

When you start the Servers as described above, LISA will look in **lisa.properties** for default names.

The relevant section of **lisa.properties** file is shown below:

- **Test Registry** - This is the default name of the Test Registry to attach to. This is also the default name of the Test Registry when you start it without a name:

```
lisa.testRegistry=lisa.TestRegistry
```

- **CoordinatorServer** - This is the default name of the Coordinator Server when started without an explicit name. Also used when the TestRunner needs a Coordinator Server and you don't specify which Coordinator server (CS) to use.

```
lisa.csName=lisa.CServer
lisa.csTestRegistry=lisa.TestRegistry
```

- **Simulator Server** - This is the default name of a Simulator server in case you do not specify.

```
lisa.simulatorName=lisa.Simulator
lisa.simulatorTestRegistry=lisa.TestRegistry
lisa.simulatorInstances=256
```

- **Timeout value** - This is the timeout value in seconds, which the coordinators and simulators will use when connecting to a test registry. A value of 0 indicates an infinite timeout.

```
lisa.testRegistryConnectionTimeoutSeconds=90.
```

If you wish to override any of these properties, please re-specify the property value in your **local.properties** file which is in the %LISA_HOME% folder.

3.6 Miscellaneous Configuration Notes

3.6 Miscellaneous Configuration Notes

The following topics are available in this section.

```
3.6.1 Project Directory Structure
3.6.2 Load and Performance Server Sizing
3.6.3 Calculating Simulator Instances
3.6.4 Configuring LISA to a Different Database
3.6.5 Running Tests in LISA Server
```

3.6.1 Project Directory Structure

3.6.1 Project Directory Structure

As of LISA 5.0, it is a best practice to make test assets (i.e. Projects) available to the server components using them.

From a server perspective, this is accomplished by setting the **lisa.projects.home** property in local.properties.

The value of that property can be a local path on the machine, or it can be a UNC path (Just remember to use forward slashes or escape backslashes if you must use them).

There are two basic approaches to manage access to these assets and both share the following prerequisites:

- Naming standards are a **MUST**. Different teams may be using the same server environment, so having naming standards to differentiate ownership and purpose are important for maintaining order.
- Project names must be unique. On the server environment, if two deployed projects have the same name, unexpected things **WILL** happen.
- All projects must be under `lisa.projects.home`.

By default this property is set to `LISA_HOME`. It can be set to anything you'd like, but the server will recursively search that directory looking for Projects, so the narrower you can make it, the better.

For example, `"lisa.projects.home=C:/"` is probably not a good idea, but `"lisa.projects.home=C:/lisa_projects"` would be fine.

Just make sure that anything you want the server to be able to run, lives under that directory.

3.6.2 Load and Performance Server Sizing

3.6.2 Load and Performance Server Sizing

There is no easy way to calculate how many simulation Servers are needed for a particular load test.

It will depend on many factors, including:

- Server host configuration (number of CPUs, amount of RAM etc.)
- Test Case footprint (number of test steps, type of test steps etc.)
- Other test requirements (number of reports, size of datasets etc.)

A recommended practice is to make several test runs of your performance test, just to collect data that can be helpful in determining the configuration of your LISA Server environment. Collecting Metrics, and monitoring memory usage and CPU usage will be invaluable for estimating the number of virtual users you can safely use on a given Simulator Server.

The Test Registry is lightweight and requires very few computing resources. It can be run from virtually any machine in your network.

The Coordinator Server requires resources, so although it does not require its own machine, it is a common practice to install it on a separate machine. This is especially true if there are lots of metrics being collected and many reports requested.

LISA uses Simulator Servers to **simulate thousands of virtual users**. It is a recommended practice to run one Simulator Server per physical Server. Technically a single Simulator Server can be started with as many instances as you want. However, the number of instances per Simulator is generally limited by Server memory size and speed. A good **upper** limit is around 250 virtual users.

Vertical or horizontal scaling can be used for **Server sizing**. In vertical scaling, you increase CPU speed and available memory, which are generally limited. In horizontal scaling, you add more Servers. To increase the number of virtual users, horizontal scaling is recommended.

Since the number of instances per Simulator is dependent on a lot of factors, it is not possible to use a simple rule to calculate the maximum number of instances.

3.6.3 Calculating Simulator Instances

3.6.3 Calculating Simulator Instances

Use the following analysis to calculate the number of instances for a given Simulator:

1. Start LISA Workstation and note the memory usage from **Help LISA Runtime Info**.
2. Run the test suite locally and note the memory usage from **Help> LISA Runtime Info**.
3. Take the difference between the memory usage in step 2 and step 1
4. Multiply your available RAM by 60%
5. Divide the available RAM in step 4 by memory usage in step 3
6. The resulting number in step 5 is a good starting estimate of the number of virtual users (instances) that you should configure in your Simulator Server
7. If the Coordinator Server and Test Registry also run on the same Server as the Simulator Server then multiply available RAM by 40% in step 4 instead of 60%. (The Coordinator Server collects all reports and metrics and therefore consumes RAM).

This technique will give you a starting point. Please use several iterations and other intuitive methods to get to the correct number of instances per Simulator.

3.6.4 Configuring LISA to a Different Database

3.6.4 Configuring LISA to a Different Database

If you need to configure LISA to use a different database, here is how to do it.

LISA Database can be changed using the **site.properties** file located in %LISA_HOME%

Under LISA_HOME/database can be found a template `site.properties` file for each of the five supported database types:

- `db2-site.properties` : IBM DB2
- `derby-site.properties` : Java Derby DB
- `mysql-site.properties` : MySQL
- `oracle-site.properties` : Oracle
- `sqlserver-site.properties` : Microsoft SQL Server

To Change the Database

1. Copy the appropriate template `database/<type>-site.properties` file to LISA_HOME and rename it to `site.properties`
2. Modify the new `site.properties` file, changing a few of the properties in the **Properties to configure** section to match the target database.
3. Launch the Registry or Workstation

Note - If you're running LISA Server, you do **not** have to reconfigure each and every LISA Workstation installation.

The configuration in `site.properties` will be propagated to each Workstation, VSE, Coordinator, Simulator Server, and any other LISA component that connects to the Registry.

Example - MySQL and LISA

Here is an example, from start to finish, of setting up MySQL and LISA.

Set Up MySQL

1. Download MySQL 5.X from <http://www.mysql.com> and install it. In most cases the standalone version will be fine.
2. Launch MySQL. Depending on the operating system and how you installed it, MySQL will either be automatically started or you will have to manually start it. For Linux and OS X the commands to start MySQL will look similar to this:
`cd /usr/local/mysql`
`sudo ./bin/mysqld_safe`
3. Create a MySQL database for LISA's use. For this example, the database name is ***lisadb***:
`mysqladmin -p -u root create lisadb`
You may also want to take this opportunity to change the root password on your MySQL server:
`mysqladmin -p -u root password <new password>`
4. Connect using the MySQL client:
`mysql -p -u root`
5. Open the ***lisadb*** database and create a user, named ***lisauser***, with password ***lisapass***:
`use lisadb;`
`grant all on lisadb.* to 'lisauser'@'localhost' identified by 'lisapass';`
`exit`

At this point we have created a MySQL server running in our local host, a database on that server, and a login to that database.

The key configuration options are in ***bold italic*** above, here they are again:

- Database name: ***lisadb***
- User name: ***lisauser***
- Password: ***lisapass***

Set up site.properties

1. Copy `LISA_HOME/mysql-site.properties` to LISA_HOME and rename it to `site.properties`
2. Edit `site.properties` and find the following four lines:
`lisadb.mysql.server=server`
`lisadb.mysql.databasesname=database`
`lisadb.username=username`
`lisadb.password=password`
3. Change each of those lines to these:
`lisadb.mysql.server=lisa.server.hostname`
`lisadb.mysql.databasesname=lisadb`
`lisadb.username=lisauser`
`lisadb.password=lisapass`

Note that we have set the host name to `{{lisa.server.hostname}}`. This is a special feature we can use when MySQL, or any other database, is running on the same physical server as the LISA server. This will automatically be populated with the external host name of the server. When `site.properties` is propagated to other LISA components running remotely, this host name will allow them to find the MySQL server by its external host name.

The rest of the properties in `site.properties` should be fine to leave at their defaults.

Set up remote LISA clients

Again, there is **nothing** that needs to be done with remote installations of LISA Workstation, Coordinator, Simulator Server, VSE, or any other remote LISA component. They will all receive the `site.properties` from the Registry when they connect and will configure their database access accordingly.

Start LISA

At this point you should be able to launch the Registry or LISA Workstation and start using LISA. All reporting data, VSE service images, CVS schedules, and Pathfinder data will go to your MySQL database.

3.6.5 Running Tests in Server Environment

3.6.5 Running Tests in Server Environment

Tests staged in a LISA Server environment are staged, and monitored with LISA Workstation, but they are managed and run by the LISA Server. A LISA Server can be configured to run on a single CPU, including the one that Workstation is using, or on a large distributed environment incorporating many CPUs. This way you can gain a dramatic increase in computer power, and your testing configuration options increase dramatically. Your testing requirements should dictate the Server architecture to be used.

In **LISA Server** the tests are run in the Server environment. LISA Workstation connects to the Server to deploy and monitor tests that were developed in Workstation.

Workstation is one of four applications that make up the LISA Server environment.

In addition to Workstation, there is at least instance of each of the following:

- **LISA Registry:** A registry that keeps track of all the Coordinator and Simulator Servers. It is not common to have more than one LISA Registry in a LISA Server installation. Workstation attaches to this registry
- **Coordinator Server:** Receives the test run information, in the form of documents, and coordinates the tests that are run on one or more Simulator Servers. The Coordinator Server manages metric collection and reporting. A LISA Server environment can have, and commonly does have more than one Coordinator Server.
- **Simulator Server:** Runs the tests under the supervision of the Coordinator Server. The Simulator Servers instantiate virtual users to run test instances against the system under test. For large tests with many virtual users, virtual users can be distributed amongst several Simulator Servers.
For more information see also "[Calculating Simulator Instances](#)".

Any number of Workstations can attach to the LISA Registry and share the Server environment.

The design of LISA Server architecture, and the installation and configuration of LISA Server are beyond the scope of this section.

Prerequisites

To use your LISA Server for testing you must obtain some configuration information for the LISA Server environment you are going to use. This should be available to you if you see your Server administrator.

The information you would need is:

1. The name of the LISA Registry
2. The names of all Coordinator Servers
3. The names of all Simulator Servers
4. The number of virtual users your LISA license allows
5. For testing independent tests/suites, that are not a part of any LISA project, we need a structure of any document repositories set up as part of the Server environment.

Usually a company will set up a standard repository for the storage of all artifacts related to testing. LISA does not have any built in repository, nor does it have any dependencies on any particular repository. All LISA documents are XML documents.

For testing all project related tests/suites, we need to have the LISA project(s) that would be used in testing under a specific directory setup as `projects-home`(See Note #1) directory for all coordinator and simulator servers.

Staging and running tests in a LISA Server environment builds on the capabilities that are available in LISA Workstation. It is assumed that you understand the concepts described in the earlier sections of this chapter. In this section we concentrate on the extra capabilities available in the Test Monitor when you are using LISA Server.

Staging and running a test on LISA Server starts with Workstation. You still configure and stage the test, start the test, monitor the test while it is running, and view the reports at the conclusion of the test. The difference is the test is running on the Server, and reports and metrics are collected on the Coordinator Servers.

Note - For each LISA server there is a `local.properties` file that specify the system properties.

To run any test/suite defined within a LISA project, we should have a copy of the whole project somewhere under the project-home directory, defined as `"lisa.projects.home"` in `local.properties`, for each Coordinator and Simulator used in the server environment.

3.7 LISA Server Environment

Running Tests in LISA Server Environment

To use your LISA Server for testing you must obtain some configuration information for the LISA Server environment you are going to use.

Note - This should be available to you from your Server administrator.

The information you would need is:

1. The name of the LISA Registry
2. The names of all Coordinator Servers
3. The names of all Simulator Servers
4. The number of virtual users your LISA license allows
5. For testing independent tests/suites, that are not a part of any LISA project, we need a structure of any document repositories set up as part of the Server environment.

Usually a company will set up a standard repository for the storage of all artifacts related to testing. LISA does not have any built in repository, nor does it have any dependencies on any particular repository. All LISA documents are XML documents.

For testing all project related tests/suites, we need to have the LISA project(s) that would be used in testing under a specific directory setup as projects-home directory for all coordinator and simulator servers.

For more information, please refer to creating Staging documents and [Test Suite documents](#) .

Staging and running tests in a LISA Server environment builds on the capabilities that are available in LISA Workstation. It is assumed that you understand the concepts described in the earlier sections of this chapter. In this section we concentrate on the extra capabilities available in the Test Monitor when you are using LISA Server.

Staging and running a test on LISA Server starts with Workstation. You still configure and stage the test, start the test, monitor the test while it is running, and view the reports at the conclusion of the test. The difference is the test is running on the Server, and reports and metrics are collected on the Coordinator Servers.

Note - For each LISA server there is a local.properties file that specify the system properties.

To run any test/suite defined within a [LISA project](#), we should have a copy of the whole project, under the project-home directory. This is defined as "lisa.projects.home" in local.properties, for each Coordinator and Simulator used in the server environment.


Running LISA Server

When tests are running on LISA Server, you connect to the Server by attaching to the LISA Registry.

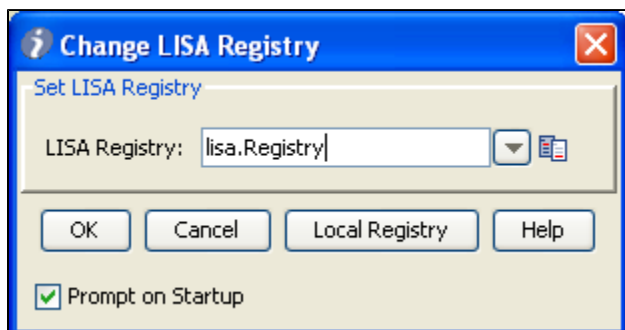
Note: If you have not set the LISA Registry, Please do so now in the "**local.properties**" file in LISA root directory.

Note - You can also set the LISA Registry, from the command line by invoking **TestRegistry.exe**.

To set/change a LISA Registry:

- Click the **Change Registry**  icon on the toolbar.
Or from the main menu, select **System-> Change LISA Registry**

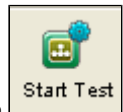
The **Change LISA Registry** pop-up screen is displayed showing a list of available Test Registries:



Select the appropriate LISA Registry from the drop down list and click **OK**.

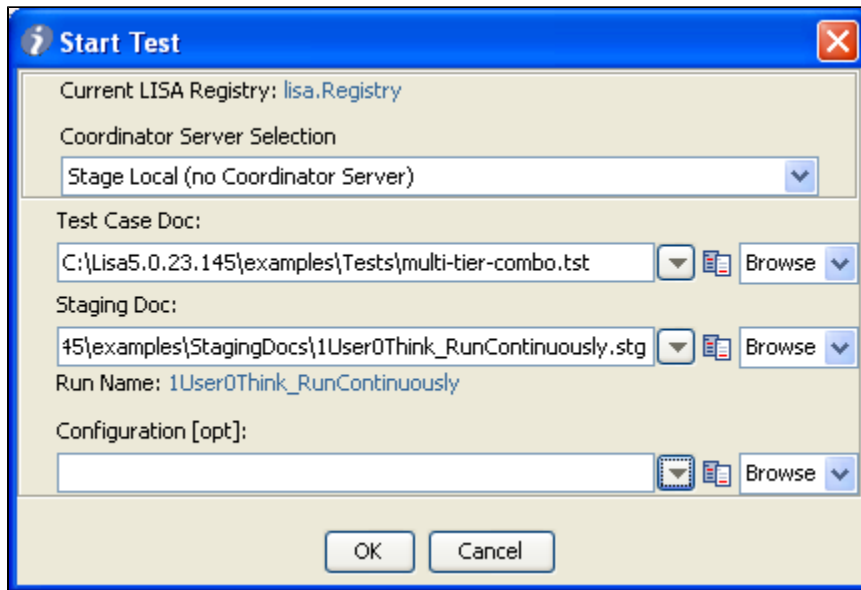
Or Click **Local Registry** to attach Local Registry as LISA Registry.

To stage a test with LISA Server,



- Click the Test Case Execution  icon on the toolbar.
Or from the main menu select **Start -> Test Case Execution**

The **Start Test** pop-up screen is displayed:



All the fields here are similar to that for LISA Workstation.

There are two differences:

- The **Current LISA Registry** value now has the name of the LISA Registry (lisa.Registry).
- **Coordinator Server Selection**: You choose which Coordinator Server you want to use for your test.

3.8 LISA Registry Monitor

3.8 LISA Registry Monitor

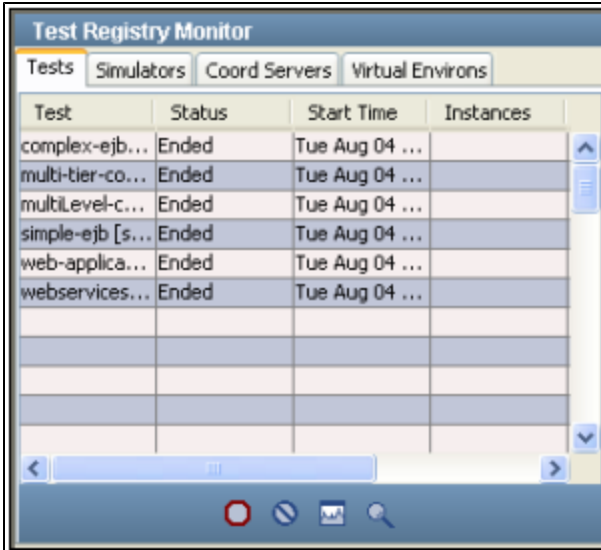
The Test Registry Monitor for LISA Server looks very similar to the test monitor for LISA Workstation, except there are some extra features like the **LISA Registry Monitor** Panel and the **Optimizers** Panel.

You can open the Registry Monitor for a Test Suite, where in you can monitor all the tests, Simulators, Coordinators and Virtual Environments.

To open the LISA Registry monitor for the suite document,

- Click the **Toggle Registry** icon on the toolbar.
- Or from the main menu, select **View -> Toggle LISA Registry Monitor**
- Or use the shortcut **CTRL + SHIFT + Y**

The below screen shows the LISA Registry Monitor and the Optimizers in the left panel.



The top panel, the "LISA Registry Monitor" has four tabs for

- LISA Tests
- Simulators
- Coord Servers
- Virtual Environment

For more information about all this, refer to [LISA Registry Monitor for LISA Server](#) in the User Guide.

3.8.1 Using the Load Test Optimizer

3.8.1 Load Test Optimizer

The Load Test Optimizer allows you to run a simple load test on the system under test. A load test determines how many users the system under test supports.

In the Load Test Optimizer, the system under test is run continuously while more and more simulated users access it, until a specific target is reached, usually a predefined average response time.

The Load Test Optimizer in the LISA Registry Monitor helps determine how many users the system under test supports. An Optimizer can be set on any LISA metric like average response time or a metric pulled from an external source (e.g. SNMP, JMX, Windows Perfmon). It increments the number of users at a preset frequency and informs you when a preset metric threshold is achieved. For example, you can configure the optimizer to increment the number of test instances spawned by the Simulator by five every ten seconds until the average response time hits two seconds.

To configure and start an Optimizer, **select a test from the running tests list**, and click Optimize Test,  icon to open the optimizer for that test.

The Optimizer panel in the LISA Registry Monitor opens with the name of the staging document listed at the top:

The following parameters are listed:

- **Metric:** The metric to use for the optimization. Select from the pull-down list
- **Simulator:** The Simulator used to spawn test instances. Select from the pull-down list
- **Threshold Low:** The metric value at which the optimizer reports that the system will require more virtual users. Enter a numerical value.
- **Threshold High:** The metric value at which the optimizer reports that the system cannot support any more virtual users. Enter a numerical value.
- **Increment (#instances):** The number of additional virtual users to add to the system at the update frequency. Enter a numerical value.
- **Update Frequency (millis):** the number of milliseconds between increments


Tip: To optimize the test, you need it in the running stage.

Click Start Optimizer icon to start the optimizer.

As the optimizer increments the number of virtual users, it displays the number of virtual users on both the Optimizers section and in the Instances column of the Tests tab in the LISA Registry Monitor.

Unable to render embedded object: File (worddavic4a2b870062c2bb98c500bc1526c0498.png) not found.

As the optimizer executes, you can change the parameters specified above.

To update the optimizer with the changed information, click the Update,  icon.

To close the optimizer click the Close, icon.

4. Configuring Java Tools

4. Configuring Java Tools

While working in LISA, you might need to configure certain java tools.

The following topics provide information on how to do this.

- [4.1 Installing Perfmon](#)
- [4.2 Installing and Configuring SNMP](#)
- [4.3 Running TCPMon](#)
- [4.4 Installing Mercury Test Director Plugin](#)

4.1 Installing Perfmon

4.1 Installing Performance Monitor (Perfmon)

Performance Monitor (**Perfmon**) is a utility that demonstrates monitoring the performance of the local or remote system. It demonstrates how to monitor **System Performance** Using Performance Counters.

To use **Perfmon** to monitor a windows machine performance:

- You must have version 2.0 of the Microsoft .Net framework installed
- From a command prompt* you need to run the **setup-wperfmon.bat** file located in the LISA_HOME/bin directory.
- On a Vista / 2008 machine or later command prompt must be 'Run as Administrator';

In addition, make sure the following are **True**:

1. The UserID is the same on both machines
2. The UserID has administrator privileges on both machines.
3. File and Printer sharing is turned ON
4. Simple File sharing is OFF
5. The default C\$ and/or ADMIN\$ shares are enabled

NOTE: Sometimes the firewall on the machine to be monitored has to be stopped.

LISA and Windows use the same technology to do remote monitoring.

You can verify that remote monitoring is working by:

Clicking **Start->Control Panel->Administrative Tools->Performance** and adding a monitor to the machine that you wish to observe.

Since LISA and Windows use this same technology, if Windows monitoring works then LISA monitoring should work.

To use Perfmon to gather Metrics within LISA, see the [LISA User Guide - 32.1.6 Windows Perfmon Metrics](#).

4.2 Installing and Configuring SNMP

4.2 Installing and Configuring SNMP

The Microsoft Windows implementation of the Simple Network Management Protocol (**SNMP**) is used to configure remote devices, monitor network performance, audit network usage, and detect network faults or inappropriate access.

SNMP support on Windows --

Windows 2000, XP and Vista provide an agent that is able to answer SNMP requests and send Traps.

SNMP support on Unix --

This is available from your operating system vendor, or you can try the Net-SNMP open-source SNMP package. See its accompanying documentation for installation and configuration directions.

For more information, see the following topics.

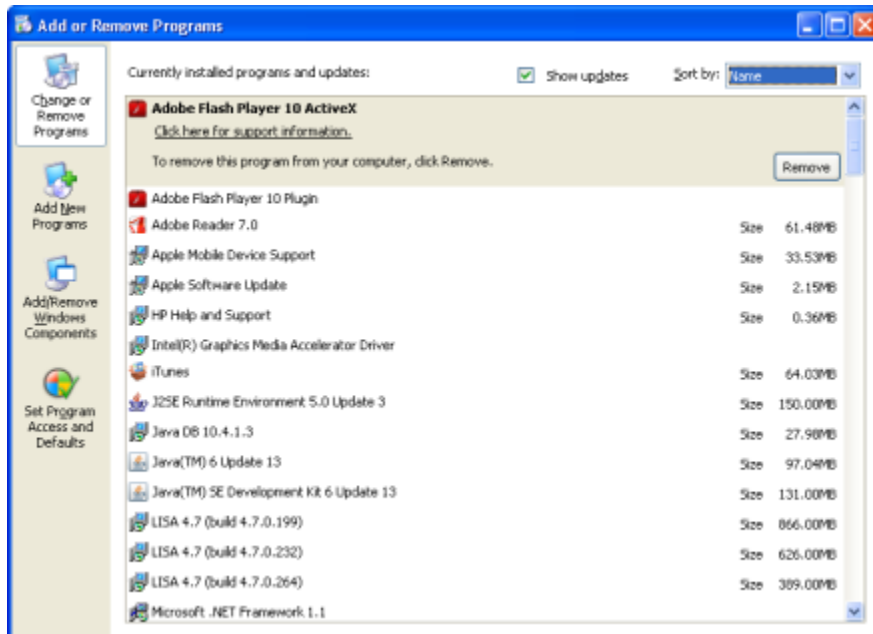
4.2.1 Installing SNMP Agent

4.2.2 Configuring SNMP Agent

4.2.1 Installing SNMP Agent

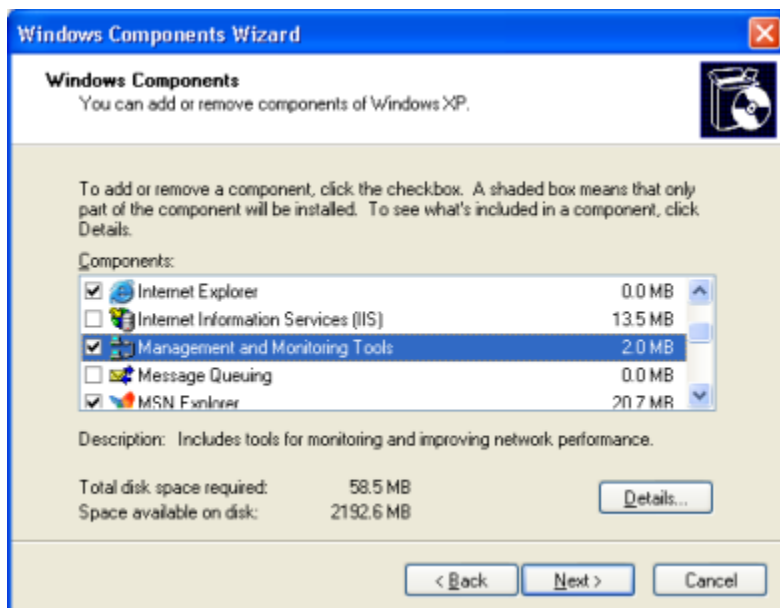
Installing the Microsoft SNMP Agent

1. Open the Windows **Control Panel** display, double-click the **Add or Remove Programs** icon. The Add or Remove Programs window displays.

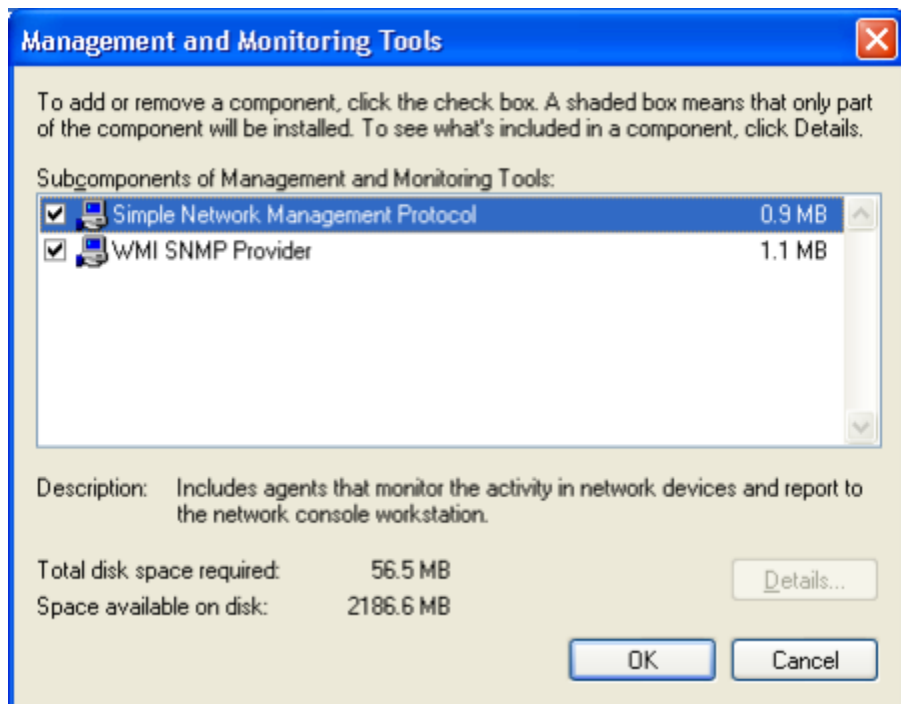


2. In the Add or Remove Programs window, click the **Add/Remove Windows Components** icon which is on the left-hand side of the window. The Windows Component Wizard opens.

3. In the Windows Components Wizard, select **Management and Monitoring Tools** in the **Components** list and click **Details**. The Management and Monitoring Tools window opens.



4. In the Management and Monitoring Tools window, select **Simple Network Management Protocol** from the **Subcomponents of Management and Monitoring Tools** list and click **OK**.

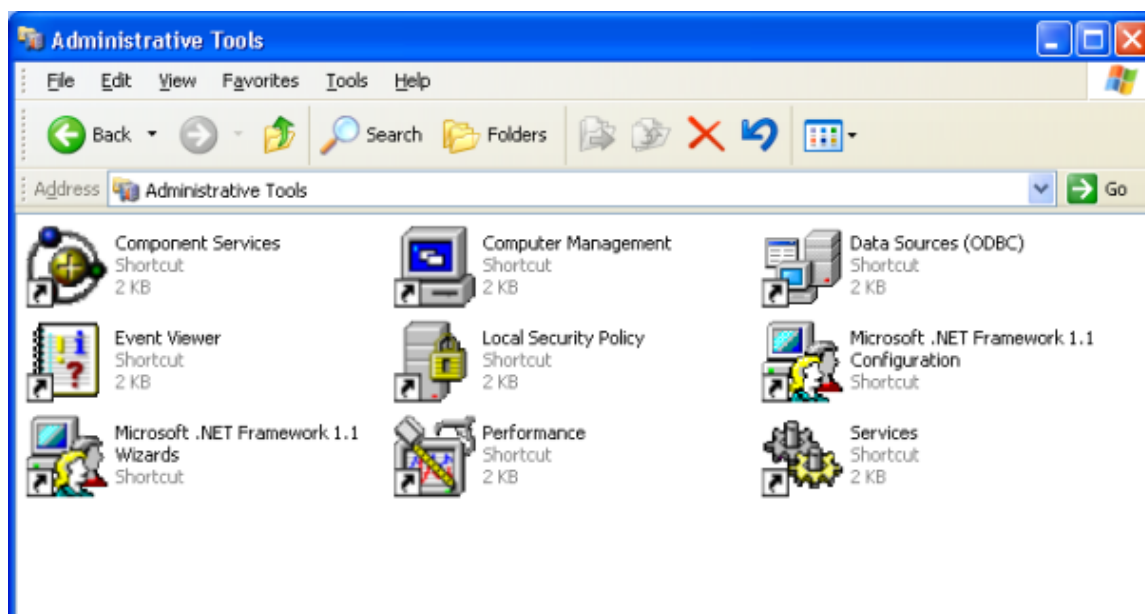


5. Click **Next**. The Windows Components Wizard installs the Microsoft SNMP agent. When complete, click **Finish**.

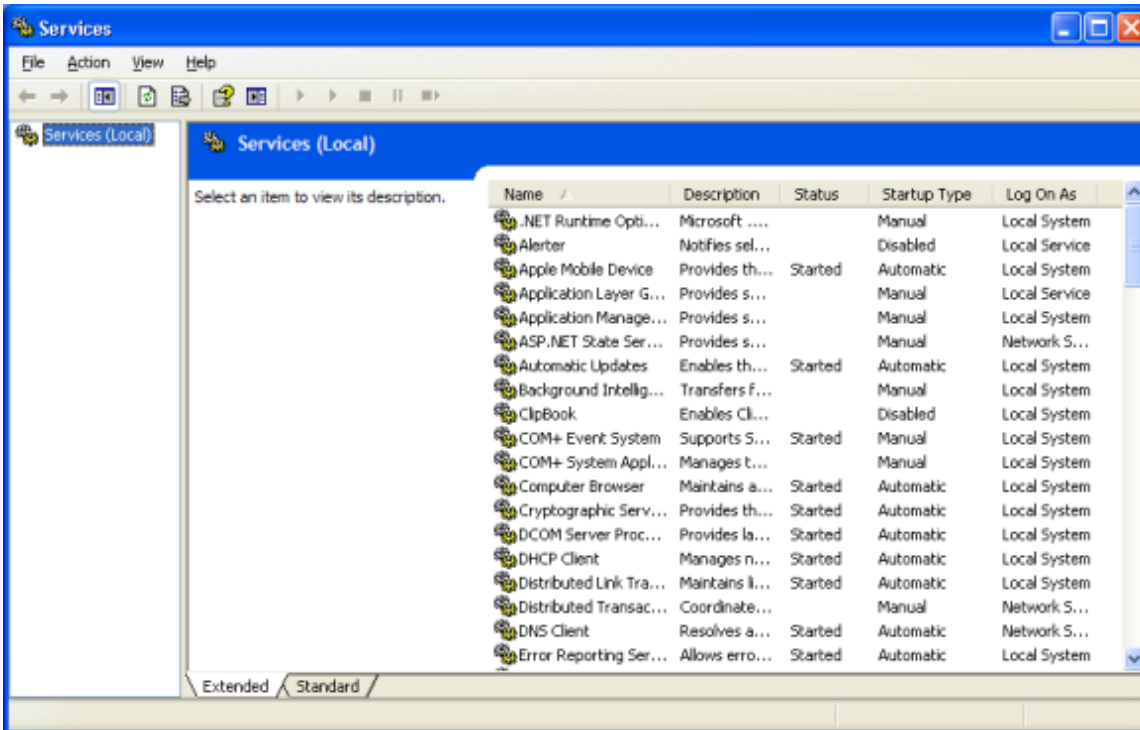
4.2.2 Configuring SNMP Agent

Configuring the Microsoft SNMP Agent

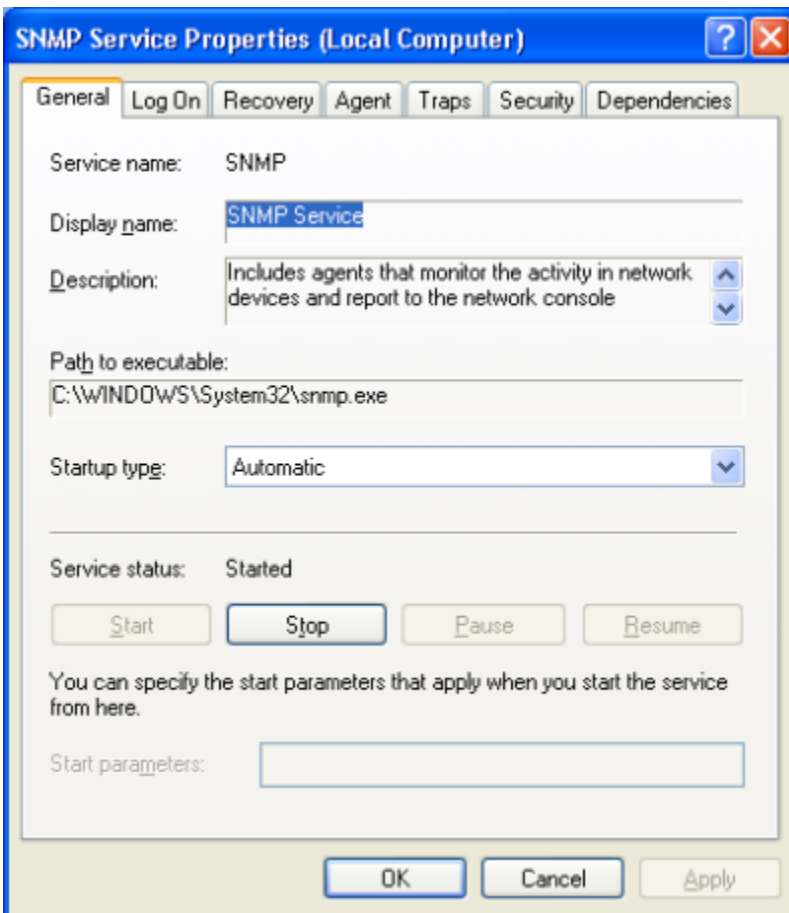
1. Open the Windows **Control Panel display**, double-click the **Administrative Tools** icon. The Administrative Tools window displays.



2. From the **Administrative Tools** window, double-click the **Services** icon. The Services window displays.



3. In the **Services** window, double-click the **SNMP service**. The SNMP Service Properties window displays.



4. In the **General tab** of the SNMP Service Properties window, change **Startup Type** to **Automatic**, as shown above.

This configures the SNMP service to start the Microsoft SNMP agent on system startup. Click **OK**.

To use Windows SNMP to gather metrics, see the [LISA User Guide](#).

4.3 Running TCPMon

4.3 Running TCPMon

TCPMon is a utility that allows the user to monitor the messages passed in a TCP based conversation.

TCPMon consists of:

For windows -

a .jar file, a .bat file

For Unix -

a shell script

To run TCPMon on Windows

- Double click the .bat file on Windows or execute the shell script on Unix.

There is a **tcpmon.bat** file in the LISA_HOME/bin directory. You can obtain the latest version of TCPMon from: <http://ws.apache.org/commons/tcpmon/download.cgi>.

Note: This chapter documents the TCPMon version from Apache. It contains the 'Sender' tab that is not available in the TCPMon version currently distributed with LISA.

See the following topics also.

[4.3.1 Using TCPMon as Explicit Intermediate](#)

[4.3.2 Using TCPMon as a Request Sender for Web Services](#)

4.3.1 Using TCPMon as Explicit Intermediate

4.3.1 Using TCPMon as an Explicit Intermediate

The most common usage pattern for the TCPMon is as an **intermediary**. It is called **explicit** since the client has to point to the intermediary, rather than the original endpoint in order to monitor the messages.

The following figure explains this concept:



In order to **start the TCPMon** in this configuration,

You have to provide the **listen port**, the **host name** and the **port for the listener** in the **Admin** tab as shown in the figure below:

TCPMonitor

Admin

Create a new TCP/IP Monitor...

Listen Port #

Act as a...

☒ Listener

Target Hostname

Target Port #

☐ Proxy

Options

☐ HTTP Proxy Support

Hostname

Port #

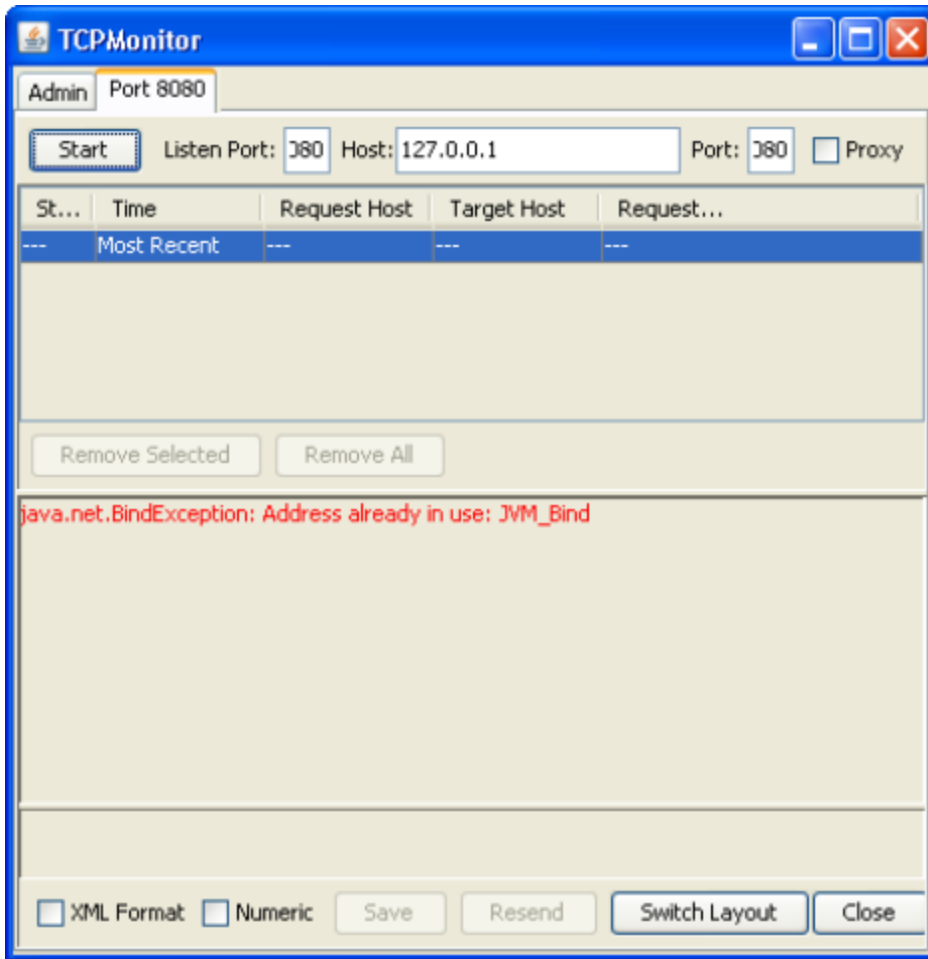
☐ Simulate Slow Connection

Bytes per Pause

Delay in Milliseconds

Add

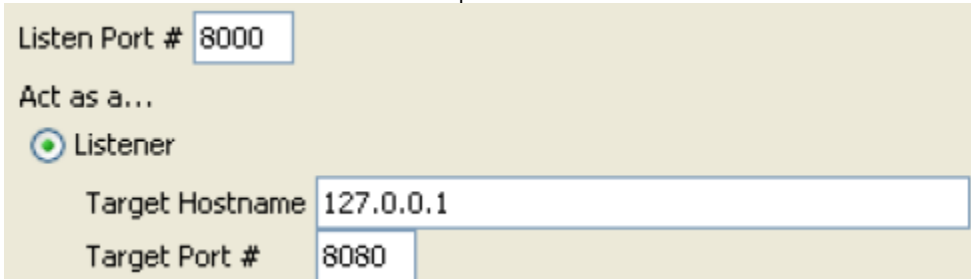
Click the **Add** button to open up a new tab (Port 8000) that allows the messages to be seen. This is shown in the figure below:



Requests should now point to the listener port of the TCPMon instead of the original endpoint.

As shown above, all messages passed to and from localhost:8080 will to be monitored:

- We set the listener to port 8000 which may be any unused port in the local machine
- We added a listener with host as **localhost** and port as **8080**



- Point the browser to **localhost:8000** instead of localhost:8080.

4.3.2 Using TCPMon as a Request Sender for Web Services

Using TCPMon as a Request Sender for Web Services

TCPMon can also be used as a request sender for web services.

- The request SOAP message can be pasted into the Sender screen and then sent directly to the Server.
- The web service endpoint is entered in the connection Endpoint textbox.

For more information on TCPMon, see [TCPMon documentation](#).

4.4 Installing Mercury Test Director Plugin

4.4 Installing HP - Test Director Plugin

HP TestDirector™ -

HP (formerly Mercury) TestDirector™ is a single, Web-based application for all essential aspects of test management — Requirements Management, Test Plan, Test Lab, and Defects Management.

For more information on HP TestDirector, see [HP Quality Test Professional](#)

iTKO integrates with TestDirector™ -

iTKO provides a way to import into and **run LISA tests from TestDirector** so you can take advantage of all TestDirector features while harnessing the power of LISA testing.

iTKO has developed a plugin - **LISA Mercury Quality Center 9.0** plugin for this. You can download this plugin from the iTKO download site.

This plugin allows you to load and run LISA Test Cases as Test Director Tests from within the TestDirector suite.

By simply loading a LISA test case into Test Director, users get real-time execution of LISA tests, with full capture of the test results and LISA callbacks returning from any system under test.

Now iTKO LISA tests are executable within the workflow of Test Director, and they report back results to maintain the context and status of the testing process.

This section covers the following topics:

- System requirements
- Running LISA Tests from TestDirector (Debug and Run mode)
- Command line Interface
- Troubleshooting

System Requirements

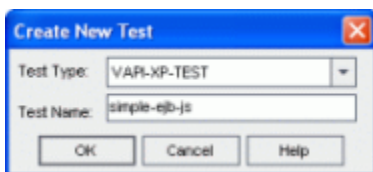
Following needs to be installed -

- An install of LISA 3.5 or above
- An install of Mercury's TestDirector for Quality Center 9.0 plugin or above
- An install of the .NET 2.0 runtime

Running LISA tests from TestDirector

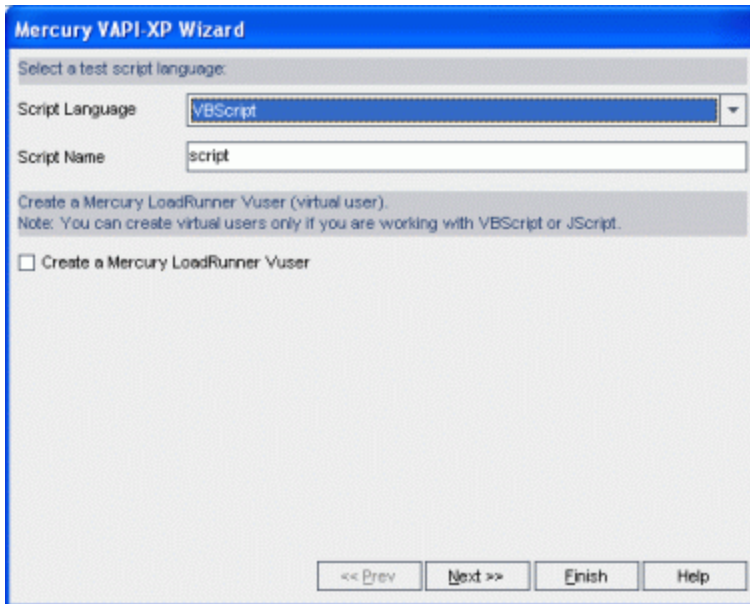
The LISA plugin uses **VAPI-XP** to integrate with TestDirector.

To use LISA together with TestDirector, simply create VAPI-XP tests like you normally would in TestDirector.



Click **OK** to open the VAPI_XP wizard.

Select the scripting language in the VAPI-XP wizard (only VBScript and JavaScript have been tested).

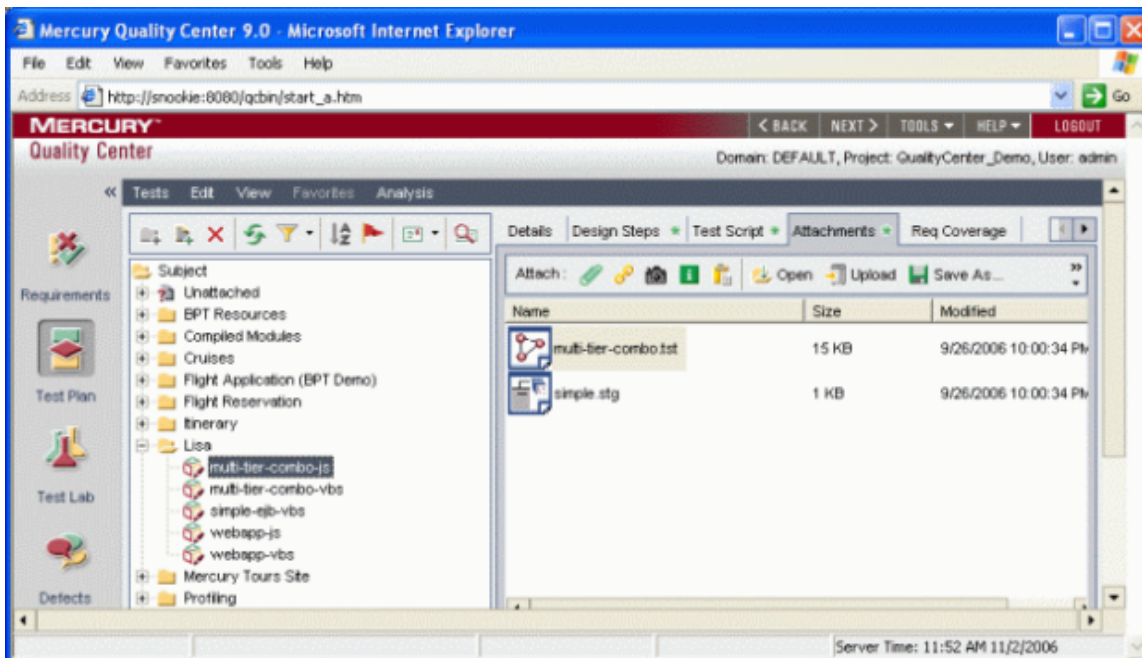


Click **Finish**.

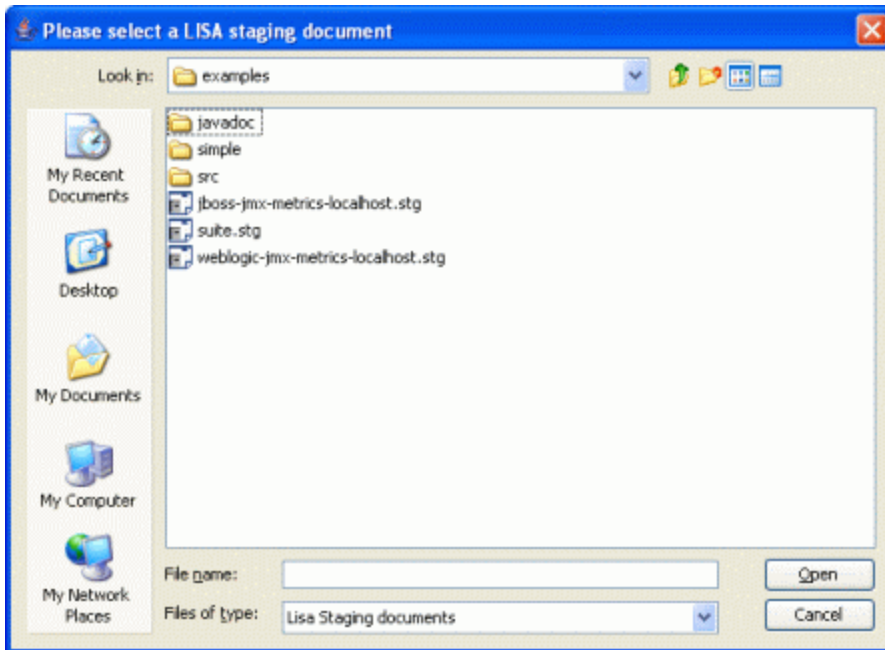
Then simply attach LISA test and/or suite files and/or config files to them.

Note: you can attach the files themselves or a URL to the files. If a test or suite file is not found during a run, the user will be prompted to supply one. If a config file is not found during a run, the LISA_CONFIG environment variable will be used. If LISA_CONFIG has not been set the user will be prompted to supply a configuration during the run. When you double-click the attached files, LISA's TestManager will open them so you can easily edit them.

Also note that if you attached the files themselves, you will need to reattach them after editing. If you attach only the URLs, this is not necessary.



Optionally, attach a LISA staging document. If it is not provided, it will be asked for during the test run.



The LISA plugin main interface is a COM object whose ProgID is **MercuryLisaBridge.MercuryTestRunner**.

It has the following API:

```
// Call this before using other APIs, supplying the intrinsic IDConnection and IDOutput objects from the script.
void Init(IDConnection connection, IDOutput output);

// This is used to connect to TestDirector when running standalone. It is not needed in VAPI-XP scripts.
void Connect(string server, string user, string pwd, string domain, string project);

// This is used to disconnect from TestDirector when running standalone. It is not needed in VAPI-XP scripts.
void Disconnect();

// Call this to (re)load the test data from LISA files into TestDirector's database.
void Reload(ITest test);

// Call this to run the attached LISA test in debug mode (i.e. without persisting the run and test results in ID).
void Debug(ITest test);

// Call this to run the attached LISA test.
void Run(IITest2 test, IRun2 run);
```

To debug or run the test, you can use the following VAPI-XP script templates. **For VBScript:**

```
Sub Test_Main(Debug, CurrentTestSet, CurrentTest, CurrentRun)

    On Error Resume Next
    IDOutput.Clear

    Set lisa = CreateObject("MercuryLisaBridge.MercuryTestRunner")
    lisa.Init IDConnection, IDOutput
    ' Optionally reload the test in ID's database
    lisa.Reload ThisTest

    If Debug Then
        lisa.Debug ThisTest
    End If

    If Not Debug Then
        lisa.Run CurrentTest, CurrentRun
    End If

    If Err.Number <> 0 Then
        IDOutput.Print "Run-time error [" & Err.Number & "] : " & Err.Description
    End If

End Sub
```


For JavaScript:

```
function Test_Main(Debug, CurrentTestSet, CurrentTest, CurrentRun)
{
    try
    {
        TDOutput.Clear();

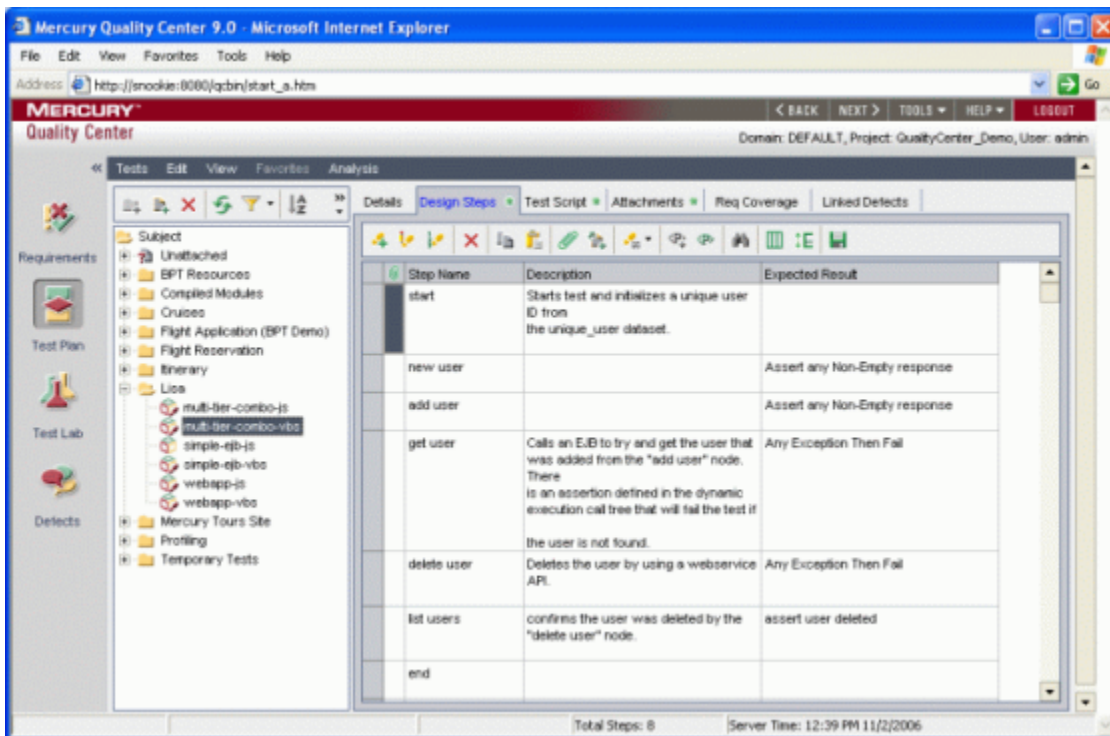
        lisa = new ActiveXObject("MercuryLisaBridge.MercuryTestRunner");
        lisa.Init(TDConnection, TDOutput);

        // Optionally reload the test in TD's database
        lisa.Reload(ThisTest);

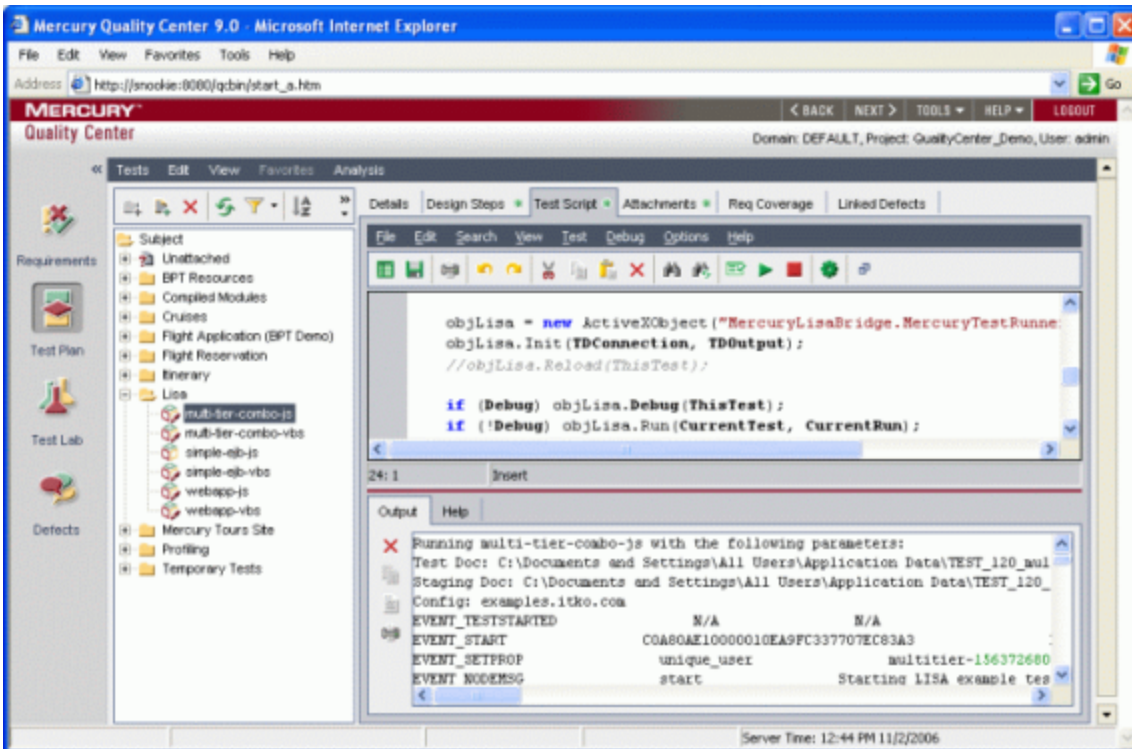
        if (Debug) lisa.Debug(ThisTest);
        if (!Debug) lisa.Run(CurrentTest, CurrentRun);
    }
    catch(e)
    {
        TDOutput.Print("Run-time error [" + (e.number & 0xFFFF) + "] : " + e.description);
    }
}
```

The result of running the **Reload API** can be seen in the following screenshot.

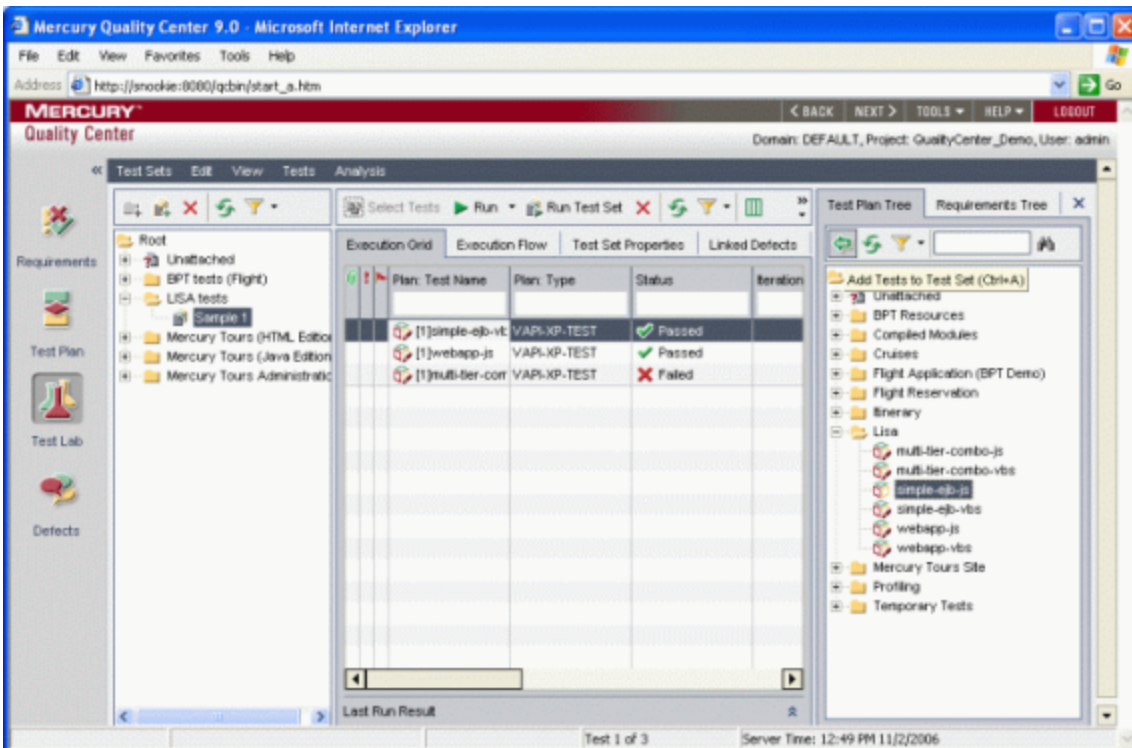
The design steps have been populated. Similarly the Details screen contains the LISA supplied description of the test.



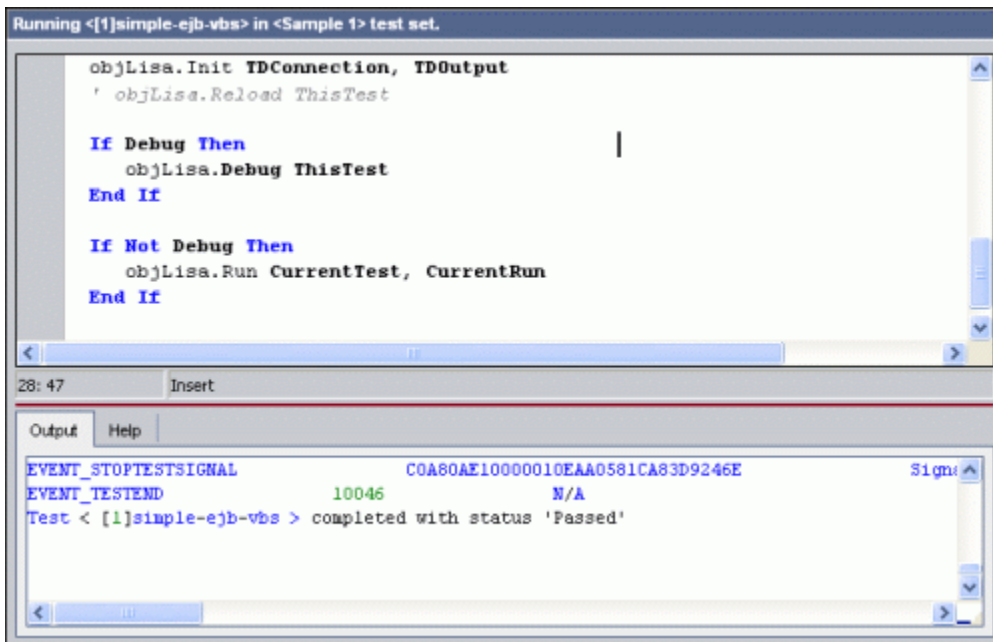
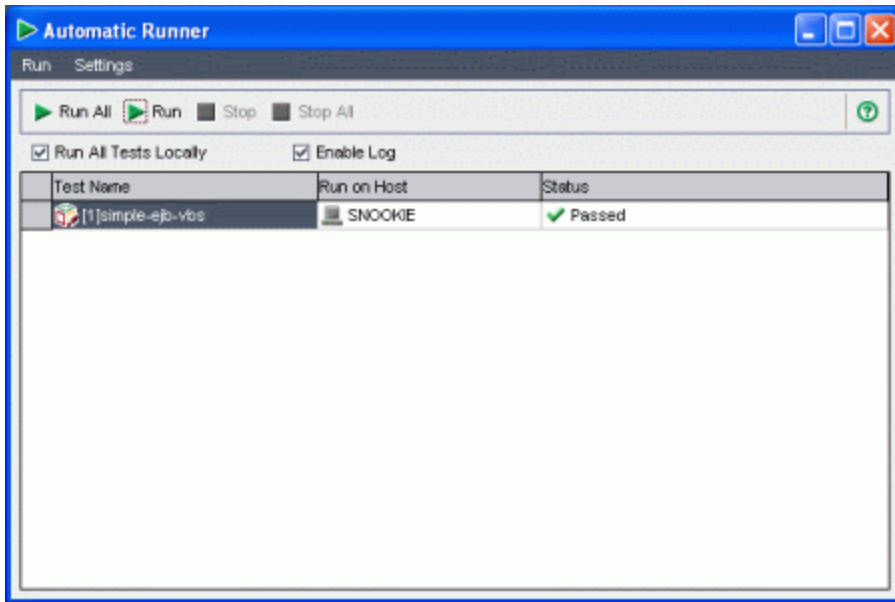
Now we can run the test in **Debug mode**:



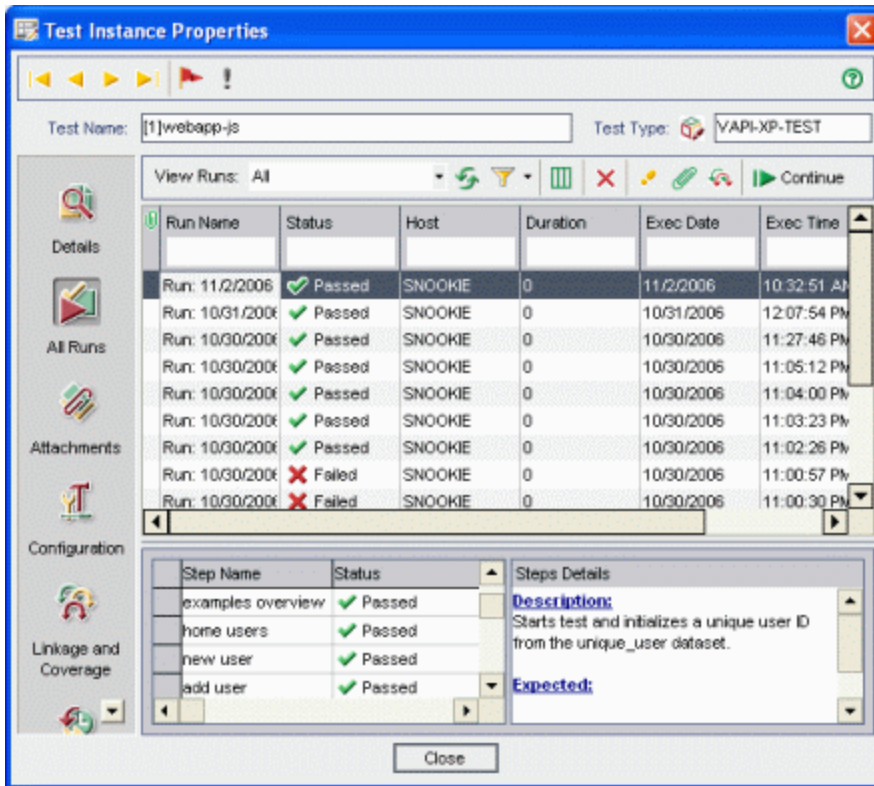
Finally, you can create **Execution Tests** or Suites in the Test Lab that reference our LISA tests.



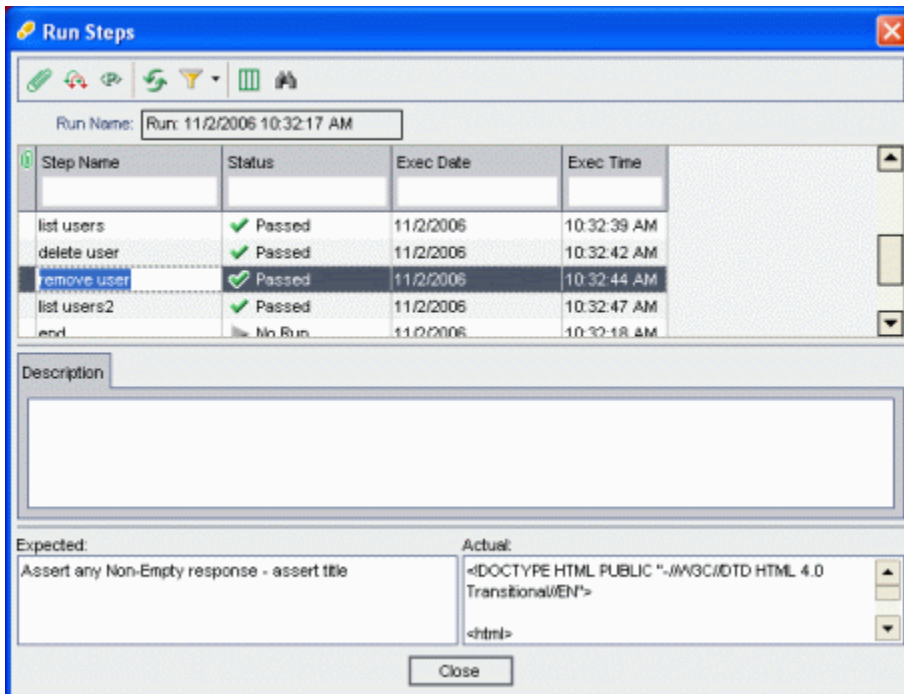
And run them from there:



Looking at the test instance properties you can examine your previous runs (only those not run in debug mode).



And you can specifically inspect the steps that were successfully run or failed.



Command-line interface

In addition to loading and running tests from TestDirector, the LISA TestDirector plugin gives you the ability to debug or run TestDirector LISA tests from the command line and to persist results in the TestDirector database for later viewing.

The <LISA_HOME>\bin directory contains the **MercuryLisaRunner.exe** executable which allows you to do this. The general command line should be:

```

MercuryLisaRunner.exe [options] <command> <arg>

where options are:
-h <host>                (defaults to localhost)
-P <port>                (defaults to 8080)
-u <user>                (defaults to admin)
-p <password>            (defaults to admin)
-D <domain>              (defaults to DEFAULT)
-l <project>             (defaults to QualityCenter_Demo)

commands are:
run                      runs the test name specified as arg
debug                   runs the test name specified as arg in debug mode
reload                  reloads the test name specified as arg or all LISA tests if arg is all

and arg is one of:
test name
all

```

```

C:\WINDOWS\system32\cmd.exe
C:\itko\lisa-mercury\dist\bin>MercuryLisaRunner.exe debug simple-ejb-vbs
Connecting...
Connected
Running simple-ejb-vbs with the following parameters:
Test Doc: C:\Documents and Settings\All Users\Application Data\ITSI_117_simple-ejb.tst
Staging Doc:
Config: examples.itko.com
Activating 587d87cc-378d-42cc-a8f4-7850ea88f771
EVENT_TESTSTARTED N/A N/A
EVENT_START CBAB81648000010EAC698278CD11DF8E N/A
EVENT_LOGMSG Start Test N/A
EVENT_SETPROP unique_user EJBUser-391013595
EVENT_MODEMSG rootNode Start Test
EVENT_LOGMSG Will execute the default next step N/A
EVENT_TRANSACTION addUser
EVENT_MODEMSG addUser con.itko.examples.ejb.UserControlBean
EVENT_SETPROP lisa.hidden.ejbobj.js.jsp://examples.itko.com:1099 javax.naming.InitialContext@26dfcc
EVENT_SETPROP lisa.hidden.ejbobj.js.jsp://examples.itko.com:1099con.itko.examples.ejb.UserControlBeanEJBNone co
n.itko.examples.ejb.UserControlBeanNone
EVENT_SETPROP lisa.hidden.call.CBAB81648000010EAC68FF1DCCED7C90 <removed>
EVENT_SETPROP lisa.hidden.ejb.CBAB81648000010EAC68FF1DCCED7C90 <removed>
EVENT_SETPROP lisa.hidden.ejblocalboneproxy.callCBAB81648000010EAC68FF1DCCED7C90 <removed>
EVENT_SETPROP lisa.hidden.ejblocalobjectproxyCBAB81648000010EAC68FF1DCCED7C90 <removed>
EVENT_CALL addUser con.itko.examples.ejb.UserControl create( )
EVENT_CALLRESULT addUser <dynamic-proxy>
<interface>con.itko.examples.ejb.UserControl</
EVENT_SETPROP usercontrolbean con.itko.examples.ejb.UserControlBean:Stateless
EVENT_SETPROP lisa.hidden.ejbobj.js.jsp://examples.itko.com:1099con.itko.examples.ejb.UserControlBeanEJBRef co
n.itko.examples.ejb.UserControlBean:Stateless

```

Troubleshooting

If you have trouble determining the cause of a test failure, the full log of a given run is available at **<Documents and Settings>All Users\Application Data<Test name>** along with some Perfmon counters. To improve performance, the LISA bridge always keeps a reference to the LISA COM Server so that it doesn't need to be instantiated for each API call. When the process hosting the bridge terminates, this reference is released unless the host is a native app like a web browser so the **LisaComServer.exe** process will stay alive. This is normally not a problem unless something gets in a bad state (because of an abrupt termination for example). In that case, think about manually terminating the lingering **LisaComServer.exe** process before proceeding.

LISA Pathfinder Pro

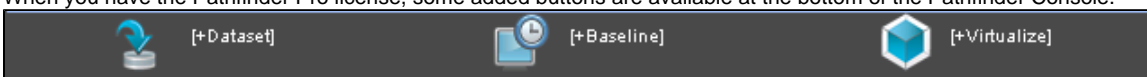
LISA Pathfinder Pro (UNDER CONSTRUCTION)

From version 5.0, ITKO introduced Pathfinder Pro. This **requires an additional license** and includes many added features. For more information, please contact ITKO support.

Some of the additional features of Pathfinder Pro include the following.

1. Creating a Data Set
2. Creating a Baseline Test
3. Creating a Virtual Model & Image
4. Creating a Defect Case
5. Editing a Defect Case
6. Filtering Defect Cases

When you have the Pathfinder Pro license, some added buttons are available at the bottom of the Pathfinder Console.



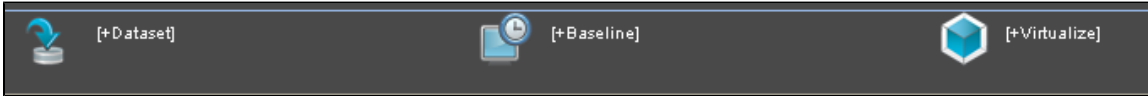
1. Creating a Data Set

1. Creating a Data Set

The Data Set feature is only available in Pathfinder Pro version.

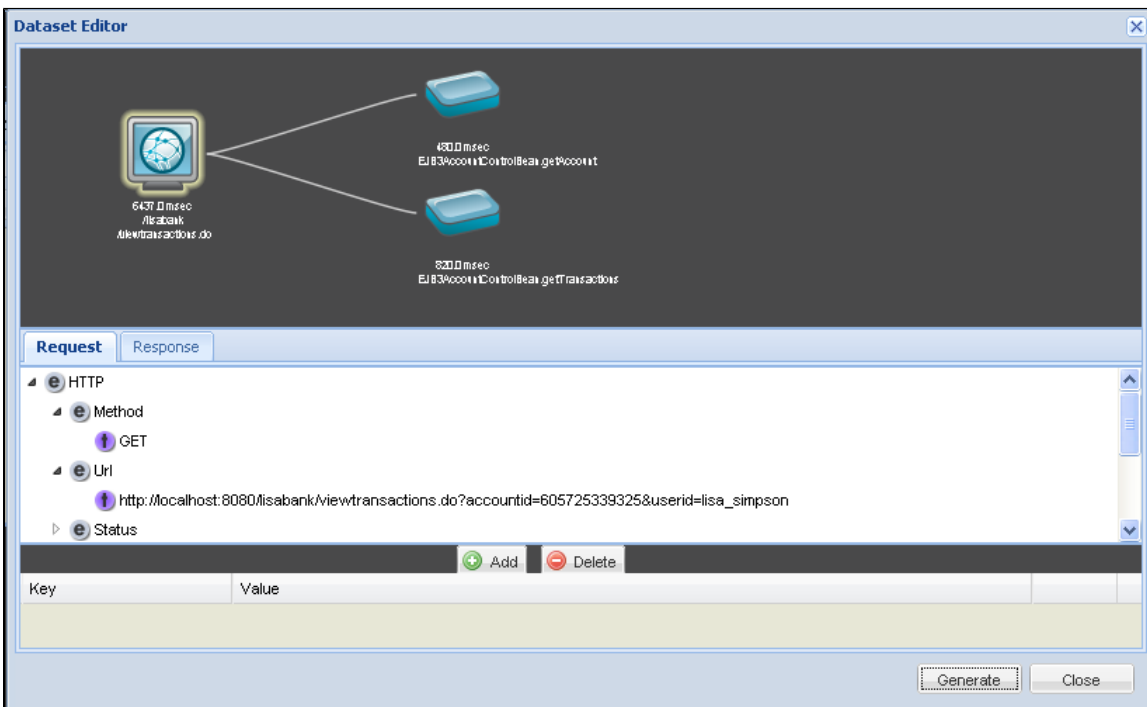
To create a Data set in pathfinder,

- Open the Pathfinder Console.
- Click the **Data Set** icon at the bottom of the tab as shown below:



Note - Similar to a Data Set, you can also create a Baseline or Create a Virtualized model within Pathfinder, but currently these two will only work for Web Service components.

- The **Data Set Editor** window opens as shown below:



You can select any criteria from the Request or Response tab. On the selected criteria, the Data Set will be generated.

- Click **Add** to add the criteria to the list. You can add more than one criteria for the Data set.
- Click **Generate** to generate the Data set.

2. Creating a Baseline Test

2. Creating a Baseline Test

The Baseline Test feature is only available in Pathfinder Pro version.

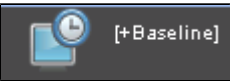
You can create a Baseline Test within the Pathfinder Console, by using its data.

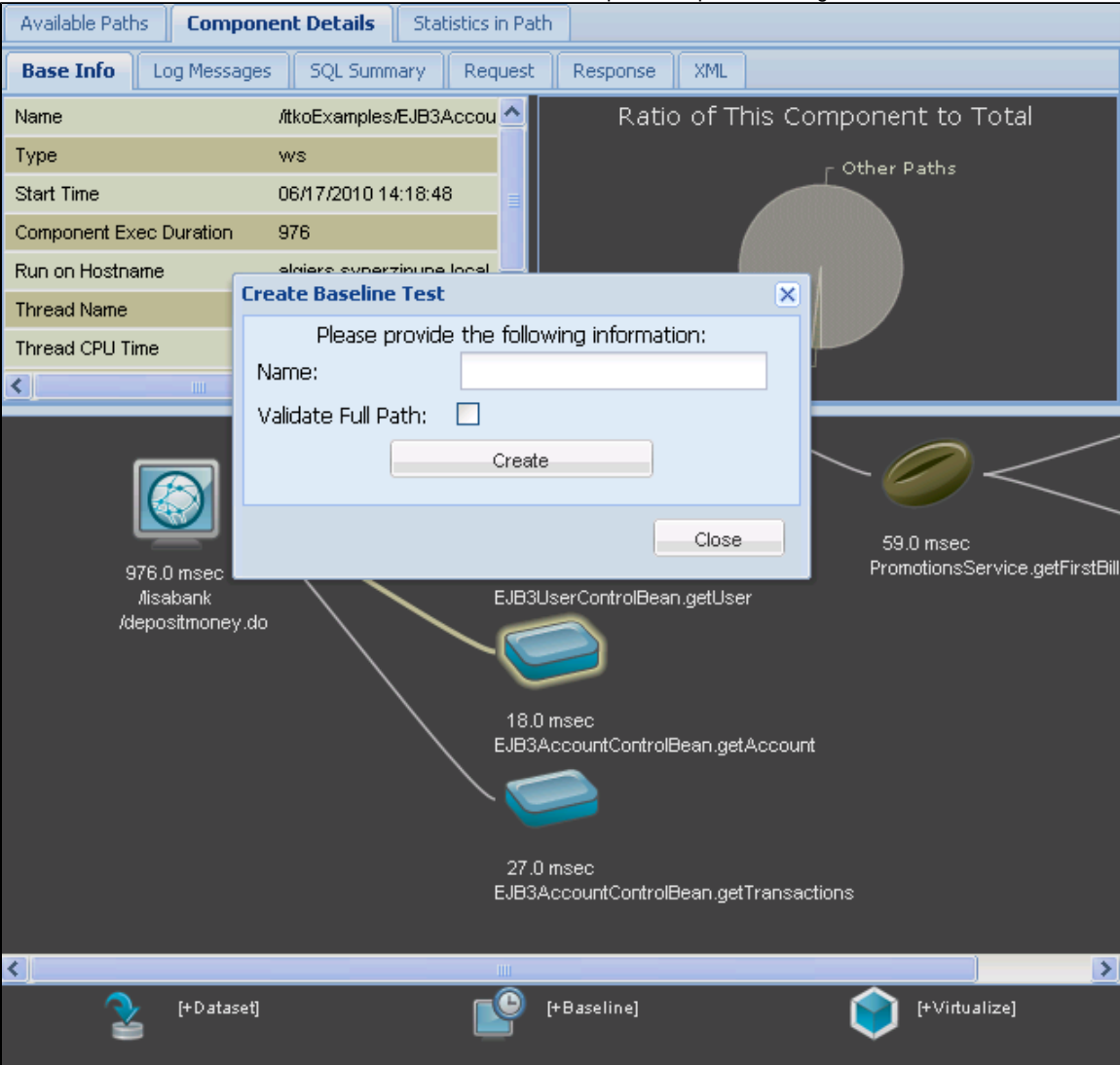
- Open a pathfinder console with transactions as shown below



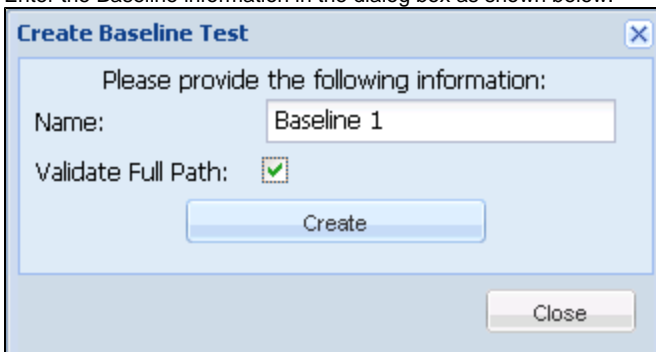
- Select the **Web Services** Transaction Component in the Path Graph.

Note - Currently this feature is available only for Web Service Transactions.

- Click on the **Baseline**  icon in the bottom panel to open the dialog as shown below:



- Enter the Baseline information in the dialog box as shown below:



Create Baseline Test

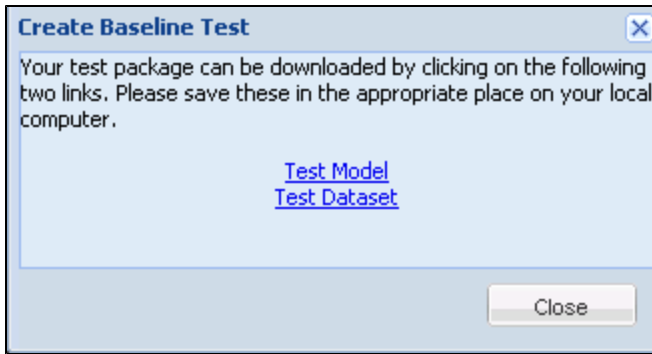
Please provide the following information:

Name:

Validate Full Path: ☒

- Enter the **Name** of the Baseline Test.
- Check the **Validate Full Path** button to validate the path.
- Click **Create** to create the Baseline or **Close** to close the dialog box.

This will start the process of baseline creation. A message box appears when this is done as shown below:



- Click on **Test Model** or Test **Dataset** to download the same.
- Save the file in proper %LISA_HOME% folder.

You can now open the saved Test Model or Test Dataset within LISA Workstation.

Some guidelines to follow while creating a Baseline test -

The following requirements must be met when naming the dataset file or else the test case will not be able to locate the dataset information.

- The name of the dataset file must be the same as the name initially given for the baseline test
- The dataset file must have the lds extension (large data set).
- The dataset file must be located in the same directory as the test model file

For example, if the name given for the baseline test is **test1**, then the dataset file must be named **test1.lds**.

3. Creating a Virtual Model & Image

3. Creating a Virtual Model & Image

The Virtualize feature is only available in Pathfinder Pro version.

You can create a Virtual Service Image or a Virtual Model (.vm) within Pathfinder Console using its data.

Note - Currently this feature is available only for Web Service Transactions.

To create a Virtual Service,

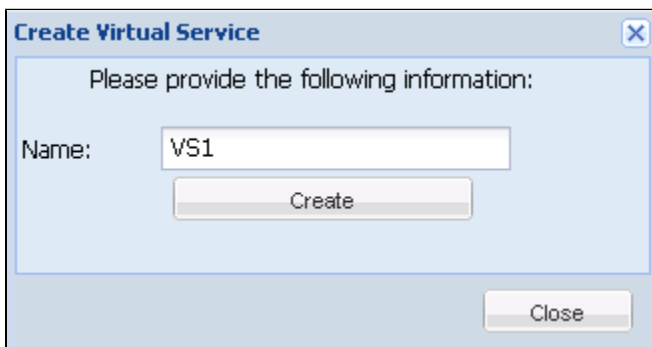
- Open a Pathfinder Console with transactions as shown below:
- Select a **Web Service transaction** Component in the Path Graph. Within the Path Graph, a web services component is denoted by



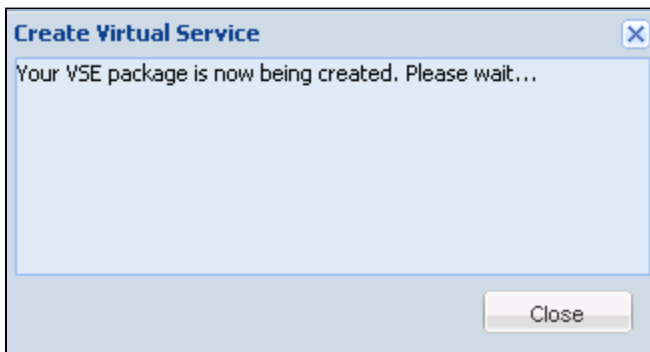
icon.



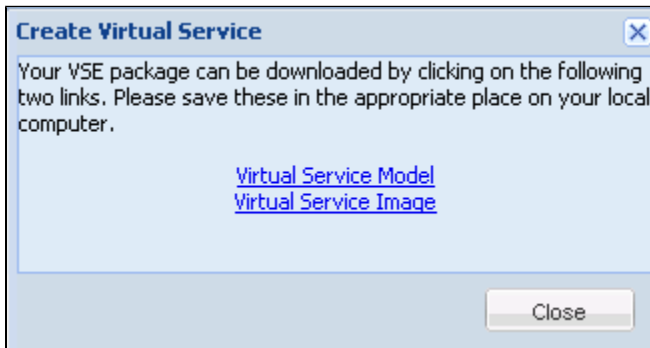
- Click on the **Virtualize** icon in the bottom panel to open the "Create Virtual Service" dialog as shown below:



- Enter the Name of the Virtual Service to be created. Ex: VS1.
- Click Create to create the Virtual service. It will pop up a message box saying it is creating a VSE package as shown below:

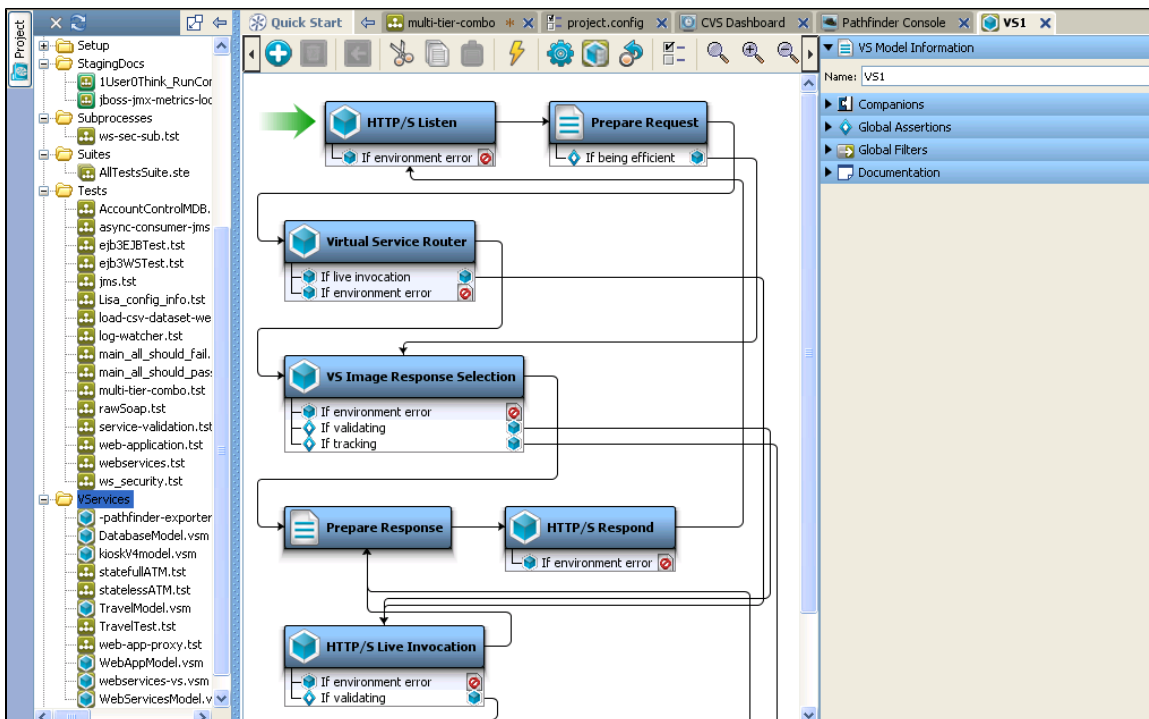


Once the package is created, you need to download it and save it.



- Click on the above link to download the Virtual Service Model or Virtual Service Image.
- Save the file under appropriate project directory.

For the purpose of illustration, we have opened the VS1 document within LISA Workstation as shown below:



4. Creating a Defect Case

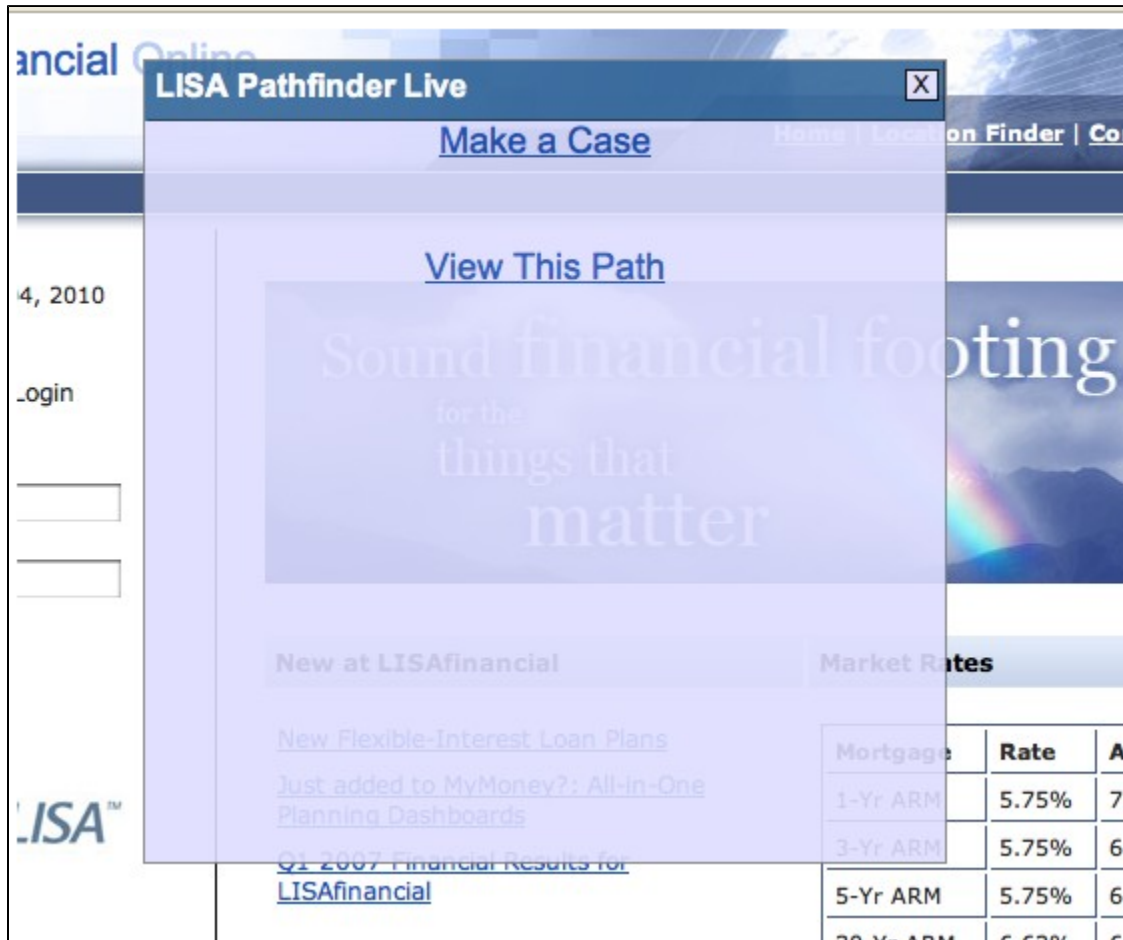
4. Creating a Defect Case

With the help of Pathfinder Pro, if a Path contains functional bug, it is easy to capture all the context transactions needed to properly document the case.

You can create a **Defect case** within the Application and view it through the Pathfinder Console.

For the purpose of illustration, we are taking example of LISA Bank application.

- Open LISA Bank application.
- Press **Alt + Click** on screen to Make a Case.
A pop-up will be displayed that allows you to enter the details of a case.



- Click **Make a Case** to create a new defect.
- Click **View This Path** to view the current Path in Pathfinder Console.
- Clicking on Make a Case will open a Case entering dialog as shown below:

LISA Pathfinder Live

Home | Location P

X

Title:

Welcome lisa simpson

Email:

Activity

External Defect Tracking Num:

Severity:

Value

Low

857422319096

Description:

derby

\$120000.00

	Description	Debits	Credits	Balance
1.0	deposit		\$100000.00	
1.0	Initial Deposit		\$20000.00	

Submit

☒ Take screenshot

Submit

- Fill in all the details regarding the case like Title, email, Defect number, Description and Severity in the given fields as shown in example below:
- Check the **Take Screenshot** button to take a screen shot of the defect. This screenshot is attached to the defect case.

LISA Pathfinder Live

Home | Location P

X

Title:

LISA Bank application

Email:

abc@itko.com

Activity

External Defect Tracking Num:

#12345

Severity:

Value

Critical

857422319096

Description:

derby

Not able to do transactions in LISA Bank

\$120000.00

	Description	Debits	Credits	Balance
1.0	deposit		\$100000.00	
1.0	Initial Deposit		\$20000.00	

Submit

☒ Take screenshot

Submit

- Click **Submit** to submit the defect.

This will open a message box informing us that the Case has been submitted.

Date	Description	Debits	Credit
06/22/10	deposit		\$1
06/22/10	Initial Deposit		\$

- Click on **View Case** to view case in Pathfinder Console as shown below:

Cases

Start Date: 06/22/2010 00:00:00
End Date: 06/22/2010 23:59:59
Status: ☒ New ☐ Identified ☐ Closed
Reporter: Any
Title:
Defect #:
Pathfinder #: 464e3823-2b69-4950-b507-f3b6aa

Case Details

Title: LISA Bank application Date: 06/22/2010 10:23:15
Reporter: abc@itko.com Status: ☒ New ☐ Identified ☐ Closed
Defect: #12345 Severity: Critical
Description: Not able to do transactions in LISA Bank

Cancel Save

LISAfinancial Online

Account Activity

Date	Description	Debits	Credits	Balance
06/22/10	Initial Deposit		\$120,000.00	\$120,000.00

Note - For new cases, the Date range is disabled as shown above.

The screen shot of the defect is seen in the lower panel of the window as shown above.

- Click on **Save** to save the case. You can also save the case as a PDF.

Viewing a Defect Case

You can also view the earlier created defect cases in Pathfinder Console.

- Click on the **Cases** tab in the left panel. This opens the case panel as shown below:

Paths

Cases

Start Date:

06/14/2010

00:00:00

End Date:

06/14/2010

23:59:59

Status:

☒ New

☐ Identified

☐ Closed

Reporter:

Any

Title:

Defect #:

Pathfinder #:

Defect	Reporter	Title
1112	gauri.kulkarni@synerzip.com	My test case

- Click on the **Identified** or **Closed** Status to view those defects. The list of filtered defect cases will appear in the panel below.
- Click on the defect in the list to open the same in the right panel.

Paths

Cases

Start Date:

06/14/2010

00:00:00

End Date:

06/14/2010

23:59:59

Status:

☒ New

☐ Identified

☐ Closed

Reporter:

gauri.kulkarni@synerzip.com

Title:

My Test Case

Defect #:

Pathfinder #:

Defect	Reporter	Title
1112	gauri.kulkarni@synerzip.com	My test case

Page 1 of 1

Displaying 1 - 1 of 1

Case Details

Available Paths

Component Details

Statistics in Session

Title:

My test case

Reporter:

gauri.kulkarni@synerzip.com

Defect:

1112

Description:

This is my test case example.

Date:

06/08/2010 12:08:04

Status:

☒ New

☐ Identified

☐ Closed

Severity:

Low

Cancel

Save

Case Details

Available Paths

Component Details

Statistics in Session

Case Details

Available Paths

Component Details

Statistics in Session

Note - Pathfinder filters the Cases/defects by Start Date, End Date, Identified and Closed cases.

Editing a Defect Case

All the cases in Pathfinder are editable.

- Open the Case by clicking it from the list of cases.
- The **Case Details** tab will open in the right window.
- Edit the Title, Reporter name, Defect no and/or the Description of the Defect.
- Click **Save** to save the changes.

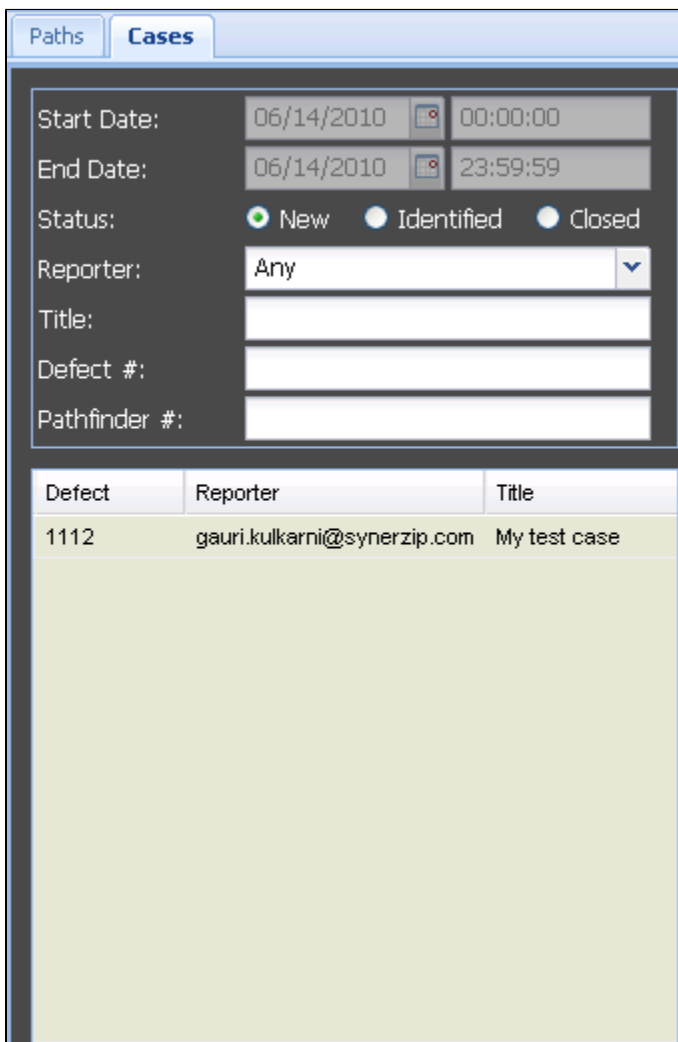
The changes are saved to the defect case and will be seen next time you reopen the case.

5. Editing a Defect Case

5. Editing/Viewing a Defect Case

All the cases in Pathfinder are editable.

- Click on the **Cases** tab in the left panel. This opens the case panel as shown below:



The screenshot shows the 'Cases' tab in the Pathfinder application. The interface includes a form for editing a defect case and a table listing defect cases.

Form Fields:

- Start Date: 06/14/2010 00:00:00
- End Date: 06/14/2010 23:59:59
- Status: ☒ New ☐ Identified ☐ Closed
- Reporter: Any (dropdown menu)
- Title: (text input)
- Defect #: (text input)
- Pathfinder #: (text input)

Defect Cases Table:

Defect	Reporter	Title
1112	gauri.kulkarni@synerzip.com	My test case

Note - For old cases, you can select the Date range.

- Click on the **Identified** or **Closed** Status to view those defects. The list of filtered defect cases will appear in the panel below.
- Open the Case by selecting it from the list of cases.
- The **Case Details** tab will open in the right window with all the saved description.
- This is in editable form and you can Edit the Title, Reporter name, Defect no and/or the Description of the Defect.
- Click **Save** to save the changes.

The changes are saved to the defect case and will be seen next time you reopen the case.

Viewing a Defect Case

You can view the earlier created defect cases in Pathfinder Console.

- Click on the **Cases** tab in the left panel. This opens the case panel as shown below:

Paths

Cases

Start Date:06/14/201000:00:00

End Date:06/14/201023:59:59

Status:☒ New ☐ Identified ☐ Closed

Reporter:Any

Title:

Defect #:

Pathfinder #:

Defect	Reporter	Title
1112	gauri.kulkarni@synerzip.com	My test case

- Click on the **Identified** or **Closed** Status to view those defects. The list of filtered defect cases will appear in the panel below.
- Click on the defect in the list to open the same in the right panel.

Paths		Cases	Case Details	Available Paths	Component Details	Statistics in Session		
Start Date:	06/14/2010	00:00:00	Title:	My test case			Date:	06/08/2010 12:08:04
End Date:	06/14/2010	23:59:59	Reporter:	gauri.kulkarni@synerzip.com			Status:	<input checked="" type="radio"/> New <input type="radio"/> Identified <input type="radio"/> Closed
Status:	<input checked="" type="radio"/> New <input type="radio"/> Identified <input type="radio"/> Closed		Defect:	1112			Severity:	Low
Reporter:	gauri.kulkarni@synerzip.com		Description:	This is my test case example.				
Title:	My Test Case							
Defect #:								
Pathfinder #:								

Defect	Reporter	Title
1112	gauri.kulkarni@synerzip.com	My test case

Cancel

Save

Note - Pathfinder filters the Cases/defects by Start Date, End Date, Identified and Closed cases.

6. Filtering Defect Cases

6. Filtering Defect Cases in Pathfinder

You can filter cases in pathfinder by clicking on the Cases tab in the left panel.

This opens a filter as shown below:

Paths

Cases

Start Date:

06/14/2010

00:00:00

End Date:

06/14/2010

23:59:59

Status:

☒ New
 ☐ Identified
 ☐ Closed

Reporter:

gauri.kulkarni@synerzip.com

Title:

My Test Case

Defect #:

Pathfinder #:

Defect	Reporter	Title
1112	gauri.kulkarni@synerzip.com	My test case

Case Details

Available Paths

Component Details

Statistics in Session

Title:

My test case

Date:

06/08/2010 12:08:04

Reporter:

gauri.kulkarni@synerzip.com

Status:

☒ New
 ☐ Identified
 ☐ Closed

Defect:

1112

Severity:

Low

Description:

This is my test case example.

Cancel

Save

Page 1 of 1

Displaying 1 - 1 of 1

LISA Reference Guide

The Reference Guide online documentation is divided into eight parts.

PART 1 - Introduction
PART 2 - Test Steps
PART 3 - Filters
PART 4 - Assertions
PART 5 - Data Sets
PART 6 - Companions
PART 7 - Events
PART 8 - Appendix

PART 1 - Introduction

PART 1 - Introduction

Welcome to the LISA Reference Guide. This manual contains detailed instructions on how to create and configure LISA's built in components. As such it assumes that you are familiar with the concepts and usage of these components, and is complementary to the [LISA User Guide](#) and the [LISA Installation and Configuration Guide](#).

Each chapter in this document discusses one of the major building blocks of a LISA test case. You will find out what options are available for each component, and what parameters have to be provided when configuring them. For an introduction to these components, and a general discussion of their use we refer you to the [LISA User Guide](#).

Several components, especially test steps, require parameters that must be obtained from people knowledgeable about the system under test. This information is collected together under 'Prerequisites'. You will rarely be able to proceed with building and running the component without this information.

Several components use common LISA functionality such as the Complex Object Editor. Rather than explain their use repeatedly in this guide, we have explained how to use them in the User Guide. We will often refer you to that document in our discussions. Since these are commonly used in LISA, we recommend you become familiar with them early on in your use of LISA. The same is true with LISA Properties. They are ubiquitous in LISA test cases, and must be understood to take full advantage of LISA functionality. Properties can be used virtually anywhere in LISA where data is used, including embedding them in text. Again we make the assumption that the reader is comfortable with properties and their use. Detailed information about using properties can be found in the [LISA User Guide - 5. Using Properties](#).

The figures used in this guide rarely show a full screen view. Full screen views are displayed in the User Guide. Here we concentrate on the section of the screen that is relevant to the discussion at hand.

Several components require parameters that must be obtained from people knowledgeable about the system under test. This information is identified throughout the guide. You will rarely be able to proceed with building and running the component without this information.

PART 2 - Test Steps

PART 2 - Test Steps

Test Steps

This section describes each of the Test Steps that are available in LISA. For a more detailed understanding of the test steps, see the [LISA User Guide - PART 2](#).

Some test steps have prerequisites and parameter requirements listed. These are there to help you prepare for the creation of a test case. They are action items that need to be done before you start to create that particular test step.

For convenience the test steps in this chapter have been organized to correspond with the New Step menu in the workstation. Some test steps, however, appear in more than one group. For these test steps you may have to look through the table of contents to find them.



You can add a test step by clicking on the **Add button** on the Test Step toolbar or

Following Type of Test Steps are present in LISA:

1. Web_Web Services Steps

- Http/HTML Request
- REST Step
- Next Generation Web Service Execution (alpha)
- Web Service Execution
- WSDL Validation
- Raw Soap Request
- Base64 Encoder Step
- Multipart MIME Step
- SAML Assertion Query
- Start or Stop Web Server
- Create a Virtual Web Service

2. Java_J2EE Steps

- Dynamic Java Execution - TO DO
- RMI Server Execution
- Enterprise JavaBean Execution

3. Other Transaction Steps

- SQL Database Execution (JDBC)
- Corba Execution

4. Utilities Steps

- Save Property as Last Response
- Output Log Message
- Write Properties To File
- Read Properties from a File
- Do-Nothing Step
- Parse Text as Response
- Base64 Encoder Step
- Checksum Step
- Convert XML to Element object
- Compare Strings for Response Lookup
- Compare Strings For Next Step Lookup

5. External_Sub Process Steps

- Execute External Command
- File System Snapshot
- Execute Sub Test Case (deprecated)
- Execute Sub Process
- Execute JUnit TestCase/Suite
- Read a File(Disk, URL or Classpath)
- FTP Step

6. JMS Messaging Steps

- JMS Messaging (JNDI)
- Message Consumer

7. Bea Steps

- Weblogic JMS (JNDI)

8. Sun JCAPS Steps

- JCaps Messaging (Native)
- JCaps Messaging (JNDI)

9. Oracle Steps

- Oracle OC4J (JNDI)
- Oracle AQ (JMS)
- Oracle AQ (JPUB)

10. TIBCO Steps

- TIBCO Rendezvous Messaging
- TIBCO EMS Messaging
- TIBCO Direct JMS

11. Sonic Steps

- SonicMQ Messaging (Native)
- SonicMQ Messaging (JNDI)

12. WebMethods Steps

- WebMethods Broker
- webMethods Integration Server Services

13. IBM Steps

- IBM Websphere MQ

14. Virtual Service Environment Steps

- Virtual Service Router
- Virtual Service Tracker
- Virtual Conversational/Stateless Response Selector
- Virtual HTTP/S Listener
- Virtual HTTP/S Live Invocation
- Virtual HTTP/S Responder
- Virtual JDBC Listener
- Virtual JDBC Responder
- Messaging Virtualization Marker
- Compare Strings for Response Lookup
- Compare Strings for Next Step Lookup

15. Custom Extension Steps

- Custom Test Step Execution
- Java Script
- Swing Test
- Java Protocol Request Listener
- JavaProtocol.Responder
- Java Pass Through
- JMS Pass through
- MQ Pass through

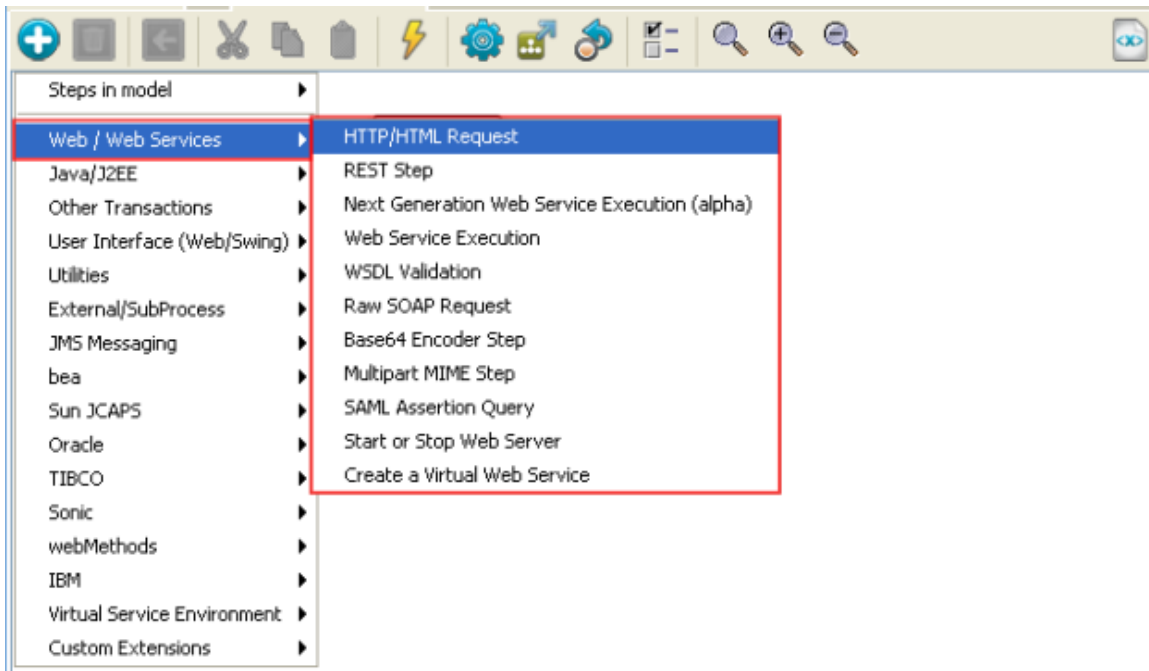
16. Standard Test Steps

- Abort the Test
- End the Test
- Fail the Test

1. Web_Web Services Steps

1. Web_Web Services Steps

Web Services Test Steps have the following Steps in the sub menu:



This chapter further explains these steps:

- 1.1 HTTP_HTML Request
- 1.2 REST Step
- 1.3 Web Service Execution (XML)
- 1.9 Web Service Execution (Legacy)
- 1.4 WSDL Validation
- 1.5 Raw SOAP Request
- 1.6 Base64 Encoder
- 1.7 Multipart MIME (Multipurpose Internet Mail Extensions) Step
- 1.8 SAML Assertion Query
- 1.10 Start or Stop Web Server
- 23.2.4 Email Notification Settings

1.1 HTTP_HTML Request

1.1 HTTP_HTML Request

This step is used while testing a traditional Web application to send and receive HTTP(S) requests, including GET and POST parameters and optionally, embedded images as a response. You can also record HTTP steps using the Web Site Proxy Recorder.

When you add this step to a test case, the step editor includes three tabs as shown below:

☒ Specify URL in parts ☐ Use property

Protocol: User:

Host Name: Password:

Port (opt):

Path:

URL Parameters	
Key	Value
Key	Value

Find:

POST Parameters		
Key	Value	Encrypt

Find:

Form Encoding:

☐ Download images referenced (test bandwidth)

URL Transaction Info HTTP Headers Response

All Known State	
Key	Value
LISA_LAST_...	
instance	0
robot	0
testCaseId	3434303...
lisa.Step2.rsp	<html>...
LISA_TC_P...	C:\Test...
LISA_HOST	Diana-PC
testCase	Test Case
lisa.Step1.rsp	VXNickE...
lisa.designti...	com.itko...
LISA_USER	dbeebe
LISA_TC_URL	file:/C:/...

URL Transaction Info tab

Use the **URL Transaction Info** tab to specify the information used to construct the URL. You can set up the URL transaction information with either of these options:

- [Specify URL in parts](#)
- [Use property](#)

Specify URL in parts

Select the **Specify URL in parts** option (default) to specify the URL in its essential pieces. All the fields in this tab are explained below:

Field	Description
Protocol	The protocol that is used to communicate with the web server. The default is http.
Host Name	The host name of the web server. Use the LISA property SERVER or enter hostname or IP address of your application server. This can be a domain name, such as www.mycompany.com or an IP address, such as 123.4.5.6. For a local web server, use the host name localhost or the IP address 127.0.0.1.
Port	Optional. Use the LISA property PORT or the port on the web server used to access the web server, if necessary. For example, the port required to access the Apache Tomcat web server by default is 8080.
Path	The path to the file to access. For example, if the URL to access is http://localhost:8080/mysite/index.jsp , enter mysite/index.jsp in the Path field.
User	Enter if a User ID is required for the application server.
Password	Enter if a password is required for the application server.
URL Parameters	GET (or URL) Request Parameters: these request parameters are passed as part of the URL, and so they are exposed to the user in address bar of the web browser.
POST Parameters	POST Request Parameters: the request parameters are passed as part of the body of the page request, and so they are not exposed to the user in the address bar of the web browser.

Form Encoding	During a step execution, parameters are URL Encoded by LISA as they are sent. The MIME type used is application/form-urlencoded.
All Known State	All known properties, such as test case properties, data sets, and filters, are listed.
Download images referenced (test bandwidth)	If this element is selected, the step will download web page images into the test environment. If you do not select this box, there will be no images downloaded.

Other functions which are part of the Toolbar available on the URL Parameters and POST Parameters sections are:

Field	Icon	Description
Add		Add a request parameter
Up		Move an existing parameter up in the list of parameters
Down		Move an existing parameter down in the list of parameters
Delete		Delete an existing parameter
Find		Find text.
Auto-Generate a Filter		From the referring Step to make this parameter dynamic. Create a new filter to auto-populate this property at run time. For more information on filters, see the Filters chapter.
Apply selected All Known State property		Apply state to the parameter. For more information on applying state, see the All Known State section which follows
Auto Apply all Known State properties		Apply all state to all properties possible by patterns. For more information on applying state, see the All Known State section which follows.

All Known State

All known properties, such as test case properties, data sets and filters, are displayed in the All Known State pane, as seen below in this URL Transaction element of an HTML step:

URL Parameters		All Known State	
Key	Value	Key	Value
u_login	vvv	LISA_LAST_STEP	
		lisa.designb...	com.itko.lisa.editor.T...
		lisa.null.rsp	<html><body></bo...
		lisa.Step1.rsp	<html><body></bo...
		LISA_TC_PATH	D:\new tutorials
		robot	0
		LISA_HOST	istanbul
		instance	0
		LISA_USER	Vanishri

You can assign the values of properties to URL request parameters. For example, to assign the value of the **LISA_USER** data set key to the **u_login** request parameter in the example above, select the u_login key in the URL Parameters pane, and then the **LISA_USER** key in the All Known State pane.

Then click **Apply selected All Known State property to current parameter**:

URL Parameters		All Known State	
Key	Value	Key	Value
u_login	vvv	LISA_LAST_...	
		lisa.designb...	com.itko.lis...
		lisa.null.rsp	<html><b...
		lisa.Step1.rsp	<html><b...
		LISA_TC_P...	D:\new tuto...
		robot	0
		LISA_USER	Vanishri


LISA warns you of the impending change:



Click **OK**.

The new property is displayed in the URL Parameters pane:

URL Parameters		All Known State	
Key	Value	Key	Value
url_high	{LISA_URL_P...}	iso HTML request type	<html> <body> </body> </html>
		testCase	test Case
		Instance	-
		iso http url	http://lisa.test:8080/comm/Trans@?e...
		testCaseId	00400000000000000000000000000000
		LISA_POST_STEP	-
		robot	-
		LISA_TC_PATH	C:\Documents and Settings\Cyntia...
		LISA_USER	Cyntia Luby

If all the names of the URL Parameter keys are the same as the names of the All Known State keys, you can quickly assign all the properties to the associated parameters by clicking Apply to All .

Use property option

If the **Use property** radio button is selected the following parameters can be specified:

- **Property Key:** Specify a property that contains the connection information.
- **Download images referenced (test bandwidth):** If this element is selected, the step will download web page images into the test environment. If you do not select this box, there will be no images downloaded.


☐ Specify URL in parts ☒ Use property

Property Key:  

☐ Download images referenced (test bandwidth)

URL Transaction Info HTTP Headers Response

HTTP Headers Tab

On the HTTP Headers tab, create any custom HTTP headers. The top section Custom HTTP Headers (Current Only) is for headers that are only sent to the server for this request. The bottom section Custom HTTP Headers (Persist) is for headers that are sent on this transaction and every other transaction in the test. To create a request parameter in either section, click Add  and change the key and value to the desired text.

Custom HTTP Headers (Current Only)	
Key	Value
<div> + ↑ ↓ ✕ </div>	

Custom HTTP Headers (Persist)	
Key	Value
<div> + ↑ ↓ ✕ </div>	

URL Transaction Info
HTTP Headers
Response

Response Tab

On the Response tab, view the HTTP response returned by the server when this test was recorded.

about:blank

Done

View
Source
DOM Tree

Select a Command

URL Transaction Info
HTTP Headers
Response

You can view the Source of the response.

HTTP/HTML Request - Step1

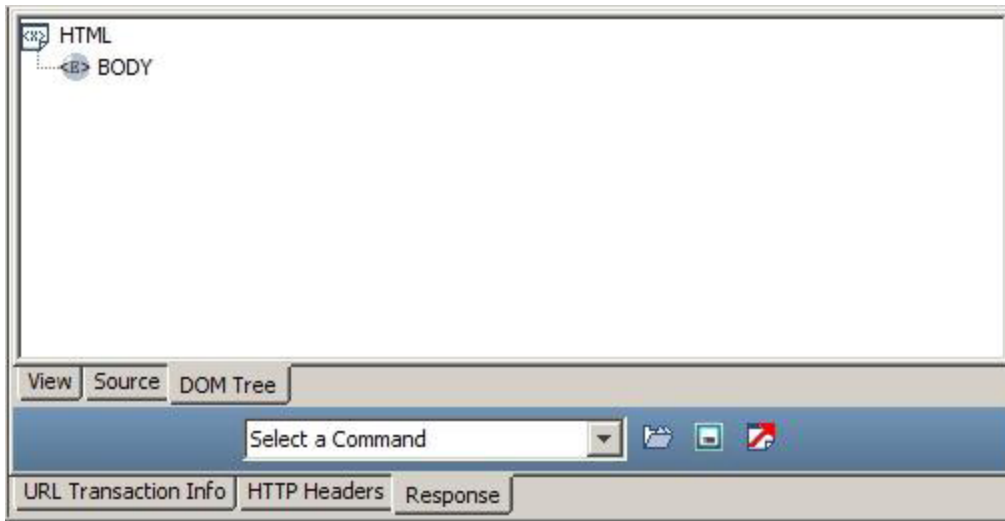
<html><body></body></html>

View
Source
DOM Tree

Select a Command

URL Transaction Info
HTTP Headers
Response

You can view the DOM Tree of the response.

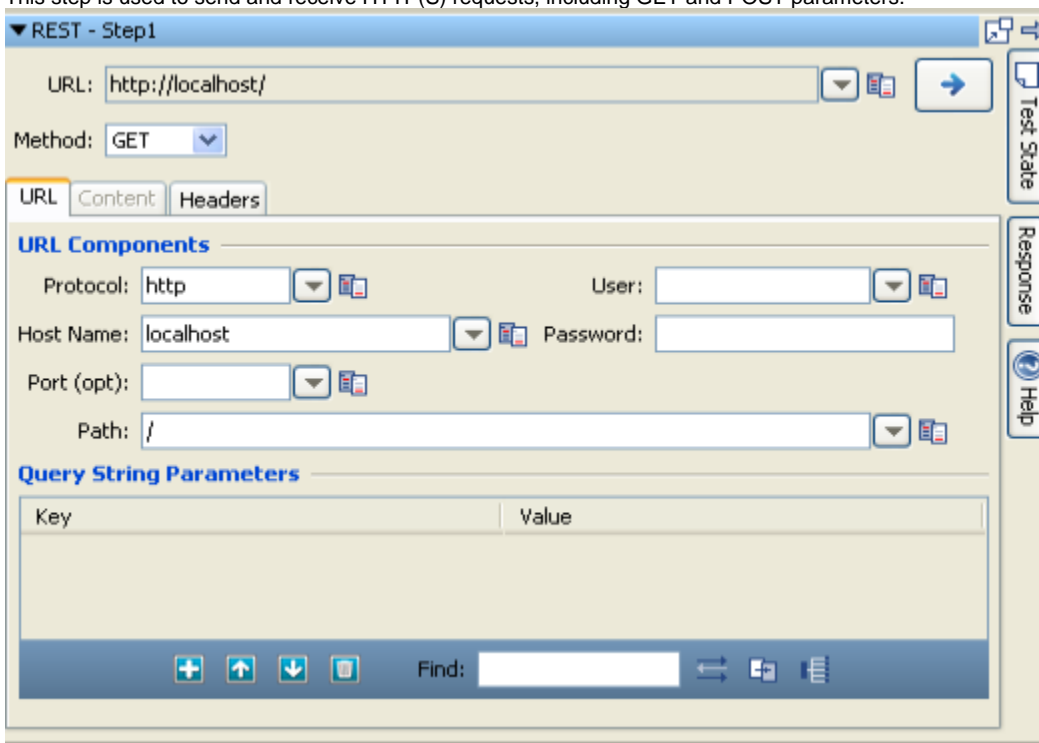


1.2 REST Step

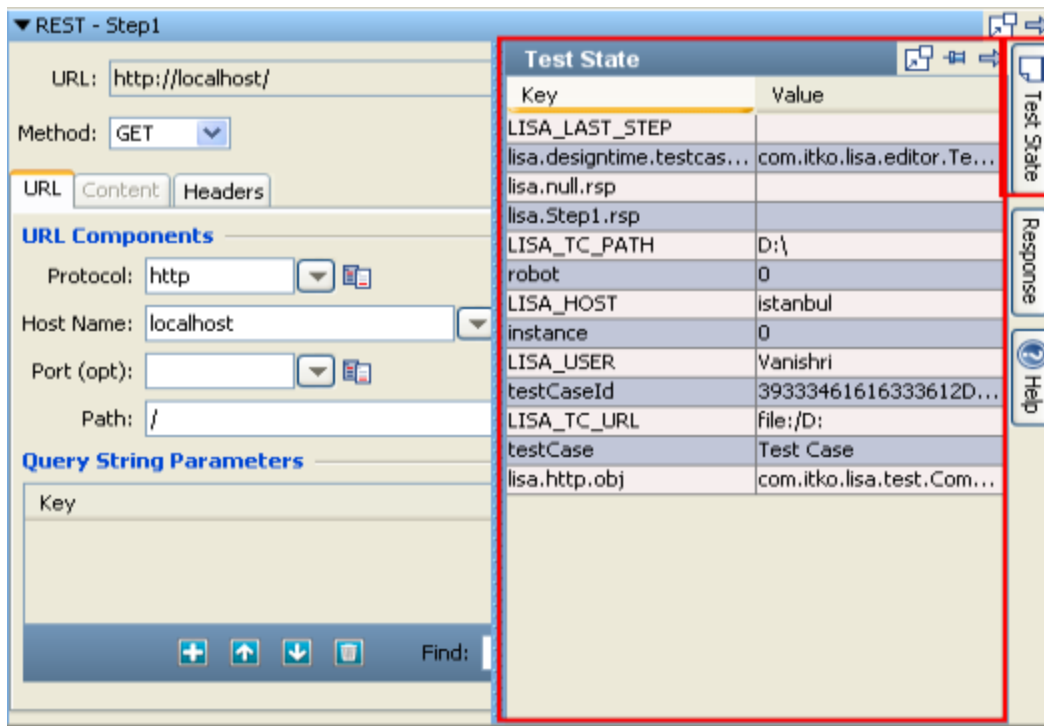
1.2 REST Step

The REST step should be used when testing REST applications.

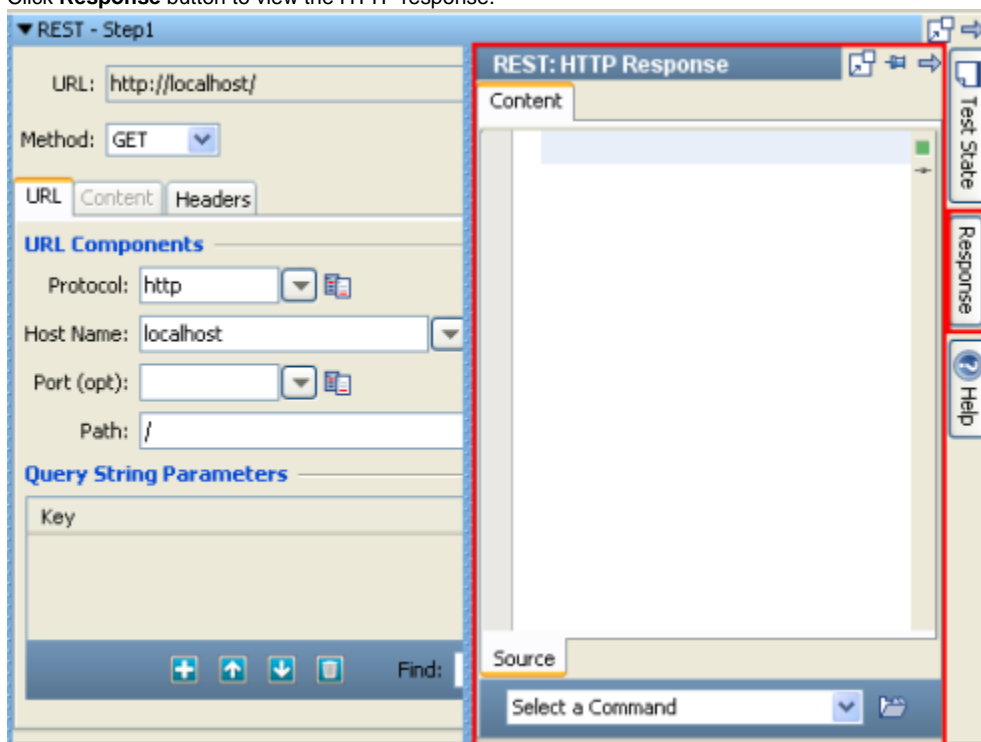
This step is used to send and receive HTTP(S) requests, including GET and POST parameters.



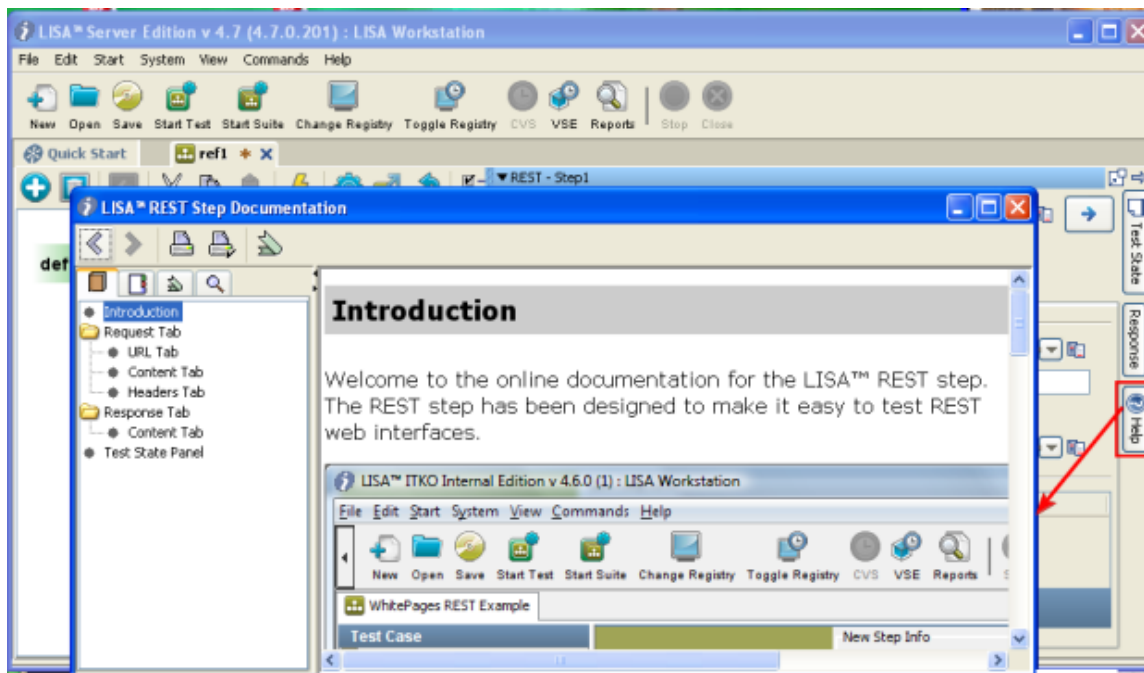
Click on the **Test State** button, on the right vertical bar to view parameters. The Test State area slides open as shown below: You can dock, pin, and hide the list.



Click **Response** button to view the HTTP response.



Click **Help** to view the REST step documentation.



1.3 Web Service Execution (XML)

1.3 Web Service Execution(XML)

The **Web Service Execution (XML)** step is designed to execute an operation on a SOAP based Web Service using an HTTP POST or JMS message. This new step has been redesigned from the ground up, but it uses many of the same configuration options as the legacy Web Service Execution step, so many of the options will be familiar to users of the old step.

One of the major differences is that access to a WSDL is no longer required, but rather it is a recommended but optional piece of configuration information. If a WSDL is configured, it helps in the process of building a SOAP message to be sent to the service. The second major difference is that the step allows you to manipulate the raw SOAP message (xml) directly. You will find this gives you a great deal more flexibility and power, but does expose you to a bit more of the nitty-gritty of how web services work.

In general, the top portion of the editor is dedicated to how and where to send the SOAP message, where the bottom portion is dedicated to what the message will say.

Once the test step opens, it has two tabs, with each tab having multiple sub tabs within as shown below:

Connection Tab

The Connection tab has fields for connection. It has sub tabs on the top bar as well as bottom bar.

Web Service Execution (XML) - Step1

Connection

WSDL URL: `http://{{WSSERVER}}:{{WSPORT}}/itko-examples/services/UserControlService?wsdl`

Service: `UserControlServiceService` Port: `UserControlService`

Operation: `addUser` On Error: `Abort the Test`

Endpoint: `{{ENDPOINT}}`

Visual XML | Raw XML | Headers | Attachments

Node	Occurs	Nil	Nilable	Value
soapenv:Envelope	1..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
soapenv:Body	1..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
ns1:addUser		<input type="checkbox"/>	<input type="checkbox"/>	
encodingStyle		<input type="checkbox"/>	<input type="checkbox"/>	<code>http://schemas.xmlsoap.org/soap/encoding/</code>
login		<input type="checkbox"/>	<input type="checkbox"/>	<code>iyapf9we {[:login:A*(5-15)]}</code>
xsi:type		<input type="checkbox"/>	<input type="checkbox"/>	<code>soapenc:string</code>
cleartextPassword		<input type="checkbox"/>	<input type="checkbox"/>	<code>1y7P3 {[:cleartextPassword:A*(5-15)]}</code>
xsi:type		<input type="checkbox"/>	<input type="checkbox"/>	<code>soapenc:string</code>

Validation Results | View XML Schema Source | Error Log

Request Editor | Request | Response

Top bar - for viewing the Visual XML, Raw XML, Headers, Attachments.

Bottom bar – for Request and Response

Basic Configuration

Connection


Connection

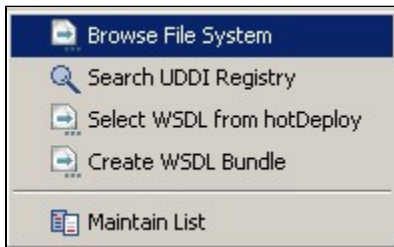
WSDL URL: `http://{{WSSERVER}}:{{WSPORT}}/itko-examples/services/UserControlService?wsdl`

Service: `UserControlServiceService` Port: `UserControlService`

Operation: `addUser` On Error: `Abort the Test`


Endpoint: `{{ENDPOINT}}`

WSDL URL --- The WSDL URL is an optional but recommended field (denoted by it's slightly grey label). It must be a URL (either [file:/](#), [http:/](#), or [https:/](#)). From the More Options menu  you can choose to



- Browse the File System for a local WSDL or WSDL Bundle file,
- Search a UDDI Registry (which will populate the advanced UDDI access point lookup).
- Select WSDL from hotDeploy to migrate from the legacy WS step, or
- Create and use a WSDL Bundle. Creating a WSDL Bundle can also be accessed from the Actions menu, but from here it will automatically populate the WSDL URL with the resulting WSDL Bundle's file URL. Or if you already have a WSDL URL populated, it will pre-populate the WSDL URL in the WSDL Bundle dialog.

When a **WSDL URL** is entered, one that is not already a WSDL Bundle, LISA will create a WSDL Bundle and stores it locally in the Project's Data/wsdl's directory, caching the WSDL locally for quicker access. The WSDL is parsed and it's schema is used to build sample soap messages as well as being used by the Visual XML editor to help assist you when manually editing the SOAP Message. It will first try and load a cached WSDL Bundle whenever processing the WSDL. If the external WSDL has changed and you want to force the local WSDL Cache to update, use

the Refresh WSDL Cache button . At any time you can manually drop a WSDL Bundle into the Data/wsdl's and anytime the step tries to process the 'live' wsdl url it will use the cached bundle instead.

Service, Port, Operation — If the WSDL URL is populated, LISA will process the WSDL and populate the Service, Port, and Operation selection. These optional but recommended fields can be used to build a sample SOAP request message. Selecting a Port will also update the Endpoint URL to match the definition in the WSDL. Changing the WSDL URL will cause these items to be refreshed, and if the Endpoint URL and SOAP Message are unchanged, they will update as well to correspond to the new WSDL, Service, Port, and Operation that are selected.

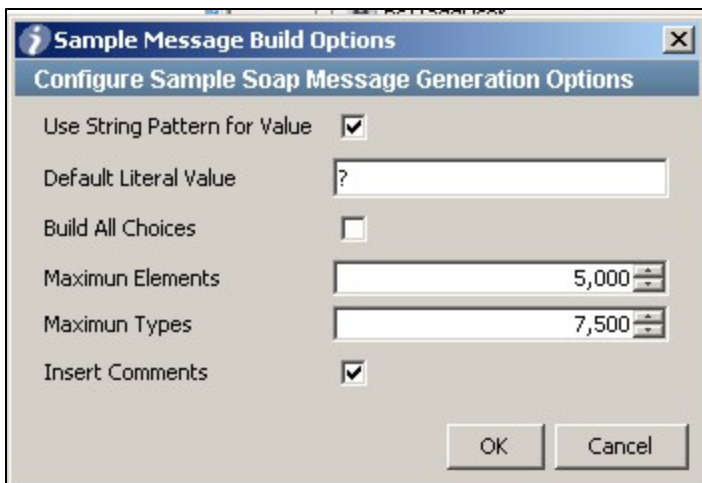
If the Endpoint was changed and no longer matches what is defined in the WSDL a Warning button will appear next to the field. A tooltip on the button will indicate what the difference are between the entered value and the WSDL definition. Clicking on the button will update the field to match the WSDL definition.

If the SOAP Request Message no longer matches the default, it will not be updated automatically. You can force the SOAP Request Message to be updated by using the Build Message button next to the Operation field.

- **Operation:** Any of the following options can be used here:

Build Options — When LISA builds sample SOAP messages it uses various build options to determine what to do in various situations.

- Build empty SOAP request message.
- Build full SOAP request message.



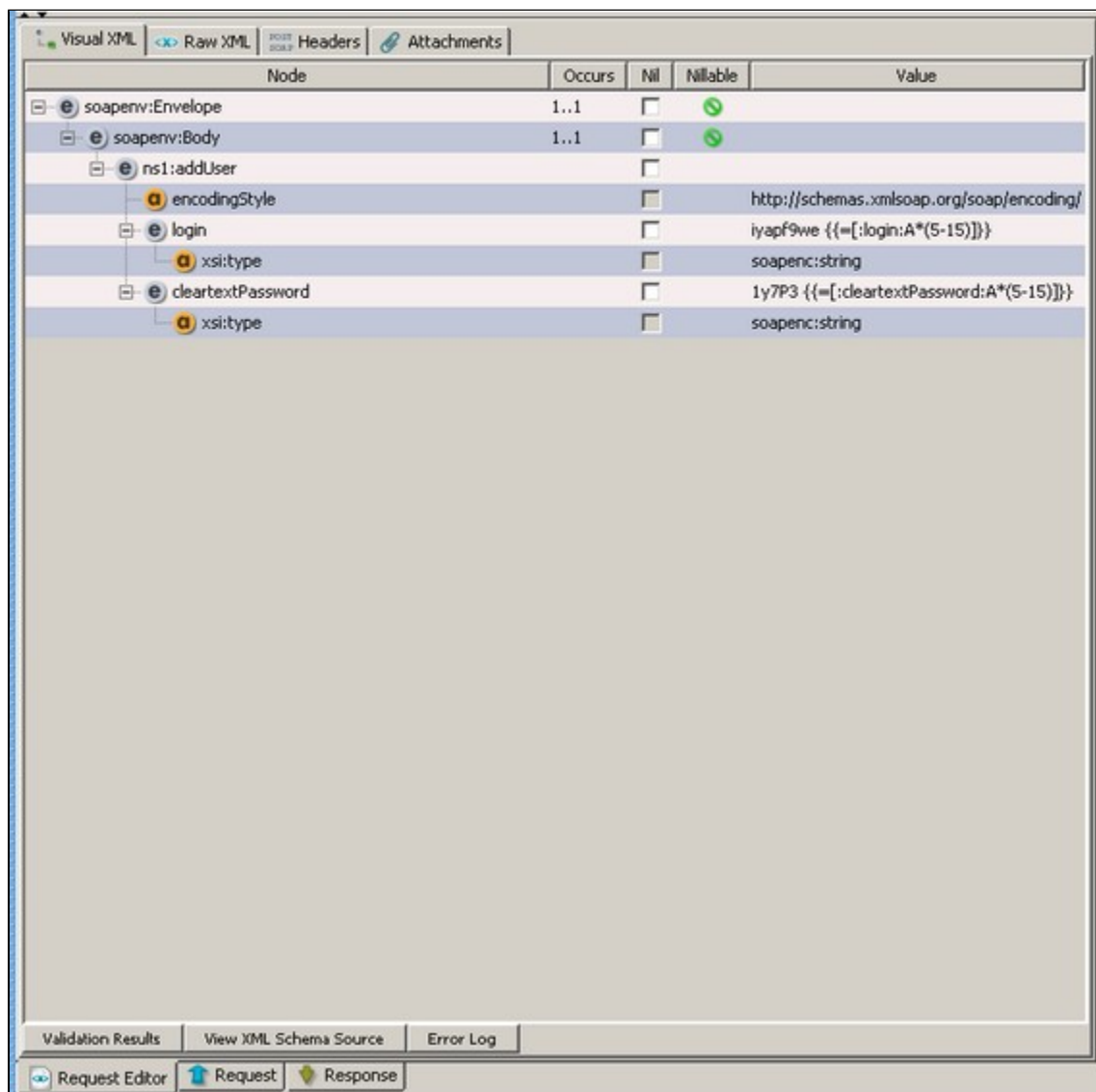
- **Use String Pattern for Value** — when checked it will populate element values using LISA String Patterns vs using a hard coded literal value
- **Default Literal Value** — when not using string patterns, use this literal value for all string values
- **Build All Choices** — by default LISA will only generate the first element in a xml schema choice. Check this option to build all possible

choice elements. Note: it will not be a valid SOAP request if you include more than one choice element, but it will show you a sample for each possible choice, making it easier to build a message when not using the first choice.

- **Maximum Elements** — this tells LISA when to give up building the sample message if it would end up creating a enormously large message - how many elements should it stop after
- **Maximum Type** — this tells LISA when to give up building the sample message if it would end up creating a enormously large message - how many complex schema types should it stop after
- **Insert Comments** — by default LISA will generate comments related to the schema (e.g. when an element is optional, alternative choices, nillable elements, etc). You don't see these comments in the Visual XML Editor, but they are visible in the Raw Editor.
- **Port**: This field indicates the server port on which the service is available.
- **On Error**: This field indicates what action to be taken when some error occurs during execution.
- **Endpoint**: The URL to the SAML Query API of the Identity Provider.

Visual XML Editor

The Visual XML Editor (or VXE) is a graphical editor for any XML document. Since a SOAP message (envelope) is just a XML document that conforms to the SOAP specification (SOAP schema), we can use it to build/edit the SOAP message.



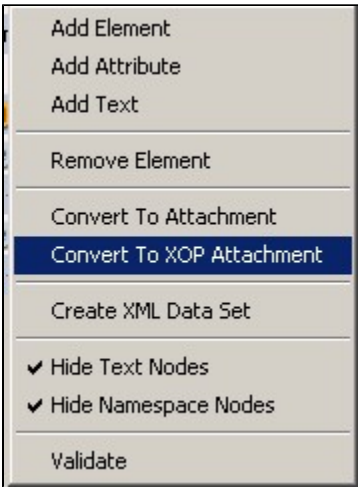
The table shows the XML document including the SOAP Envelope and Body elements. It has a **Occurs** column that indicates how many elements are expected. The first number indicates the minimum number of times the element can occur. Zero would mean that it's optional and can be removed. The second number indicated the maximum number of times the element can occur. Infinity would mean there can be an unlimited number of elements of that name.

The **Nil** column allows you to nil out or un-nil the element value. Checking nil will remove any element children or values but leave all attributes alone. Un-checking nil will populate all expected children and attributes as defined in the WSDL schema.

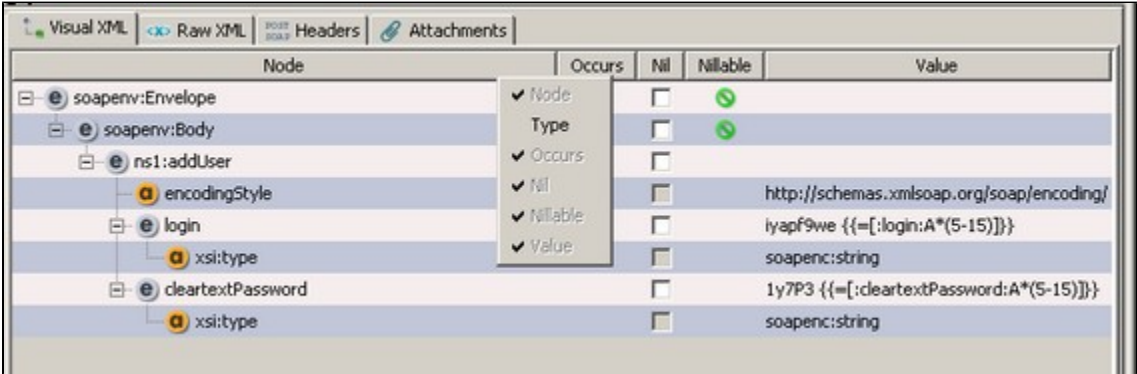
The **Nillable** column indicates whether the element can be nil. It will show a red icon!Picture 855.png! if the element is nil, but is non-nillable. And green if it is non-nillable and is not nil.

Element values can be typed directly into the Value column. If the element type is one of a set of known type, a specialize edit buttons will show up.

Context Menu — When you right click on the VXE you get a context sensitive menu which gives you options for manipulating the document.



- **Convert to Attachment** — This is a shortcut method for creating a standard referenced attachment (see Attachment section for more details)
- **Convert to XOP Attachment** — This is a shortcut method for creating a standard referenced XOP Include attachment (see Attachment section for more details)
- **Create XML Data Set** — This is a shortcut method for generating a XML Data Set. A typical use case would be to build a full SOAP message then select the section of the XML document that you want as the first record of the data set. Creating a XML Data Set will automatically populate the first record with the selected xml element tree and set the new data set LISA property as the value in the editor.
- **Hide Text Nodes** — By default we hide Text Nodes which are typically redundant (e.g. white space), but on some occasions it's useful to view them, including when the XML element is of mixed type (element that supports intermixed elements and text).
- **Hide Namespace Nodes** — By default we also hide namespace declarations and namespace prefix declarations. You may want to show them to confirm a prefix value or if you want to change prefixes or namespace scoping.
- **Show Type Column** – If you right-click on the table header you have the option to hide/show table columns. The Type column is hidden by default, but may be useful to some users. The column will show the XML schema type (local name) and has a tooltip to show the fully qualified name (qName) with namespace.



Visual XML					
Raw XML					
POST SOAP Headers Attachments					
Node	Occurs	Nil	Nullable	Value	Type
soapenv:Envelope	1..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>		Envelope
soapenv:Body	1..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>		Body
ns1:addUser		<input type="checkbox"/>	<input type="checkbox"/>		
encodingStyle		<input type="checkbox"/>	<input type="checkbox"/>	http://schemas.xmlsoap.org/soap/envelope/8body	
login		<input type="checkbox"/>	<input type="checkbox"/>	iyapf9we {[:login:A*(5-...	
xsi:type		<input type="checkbox"/>	<input type="checkbox"/>	soapenc:string	
cleartextPassword		<input type="checkbox"/>	<input type="checkbox"/>	1y7P3 {[:cleartextPass...	
xsi:type		<input type="checkbox"/>	<input type="checkbox"/>	soapenc:string	

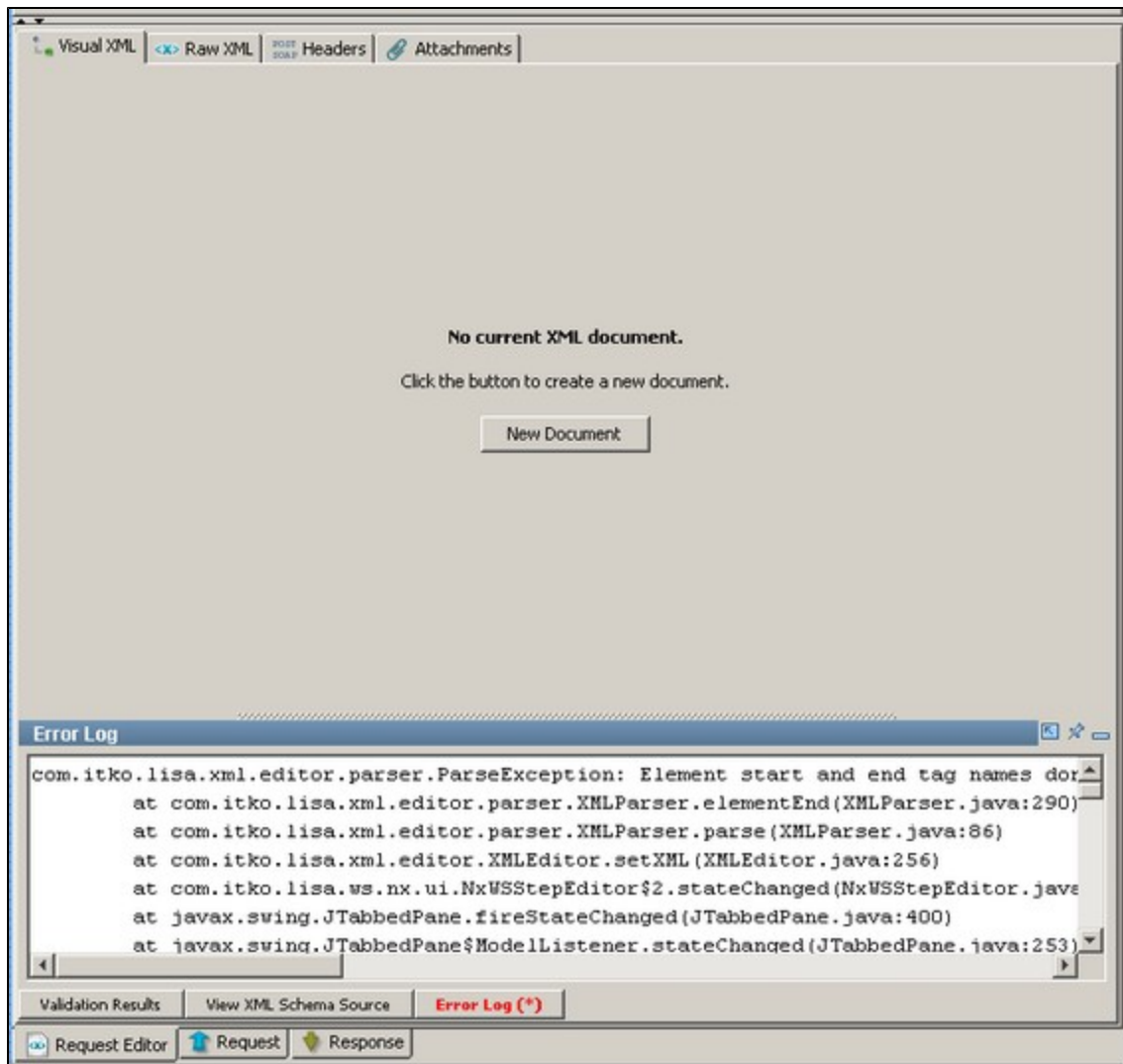
```

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns1="http://webservices.examples.itko.com">
  <soapenv:Body>
    <ns1:addUser xmlns:ns1="http://webservices.examples.itko.com" encodingStyle="http://schemas.xmlsoap.org/soap/envelope/8body">
      <login xsi:type="soapenc:string">{[:login:A*(5-15)]}</login>
      <cleartextPassword xsi:type="soapenc:string">{[:cleartextPassword:A*(5-15)]}</cleartextPassword>
    </ns1:addUser>
  </soapenv:Body>
</soapenv:Envelope>

```

Raw XML Editor

The Raw XML Editor is a text based editor that is XML aware and that allows you to manually edit the raw xml SOAP message. Any changes that are made will be seen, when you switch back to the VXE (and vice-versa). If you make an edit to the XML which causes the document to no longer be a valid XML document, the VXE may show an error message when you switch back to it.



Simply fix your changes in the Raw XML Editor and the VXE will begin working again.

Visual XML Raw XML POST SOAP Headers Attachments

Transport Headers

Key	Value
-----	-------

+ ↑ ↓ ✕

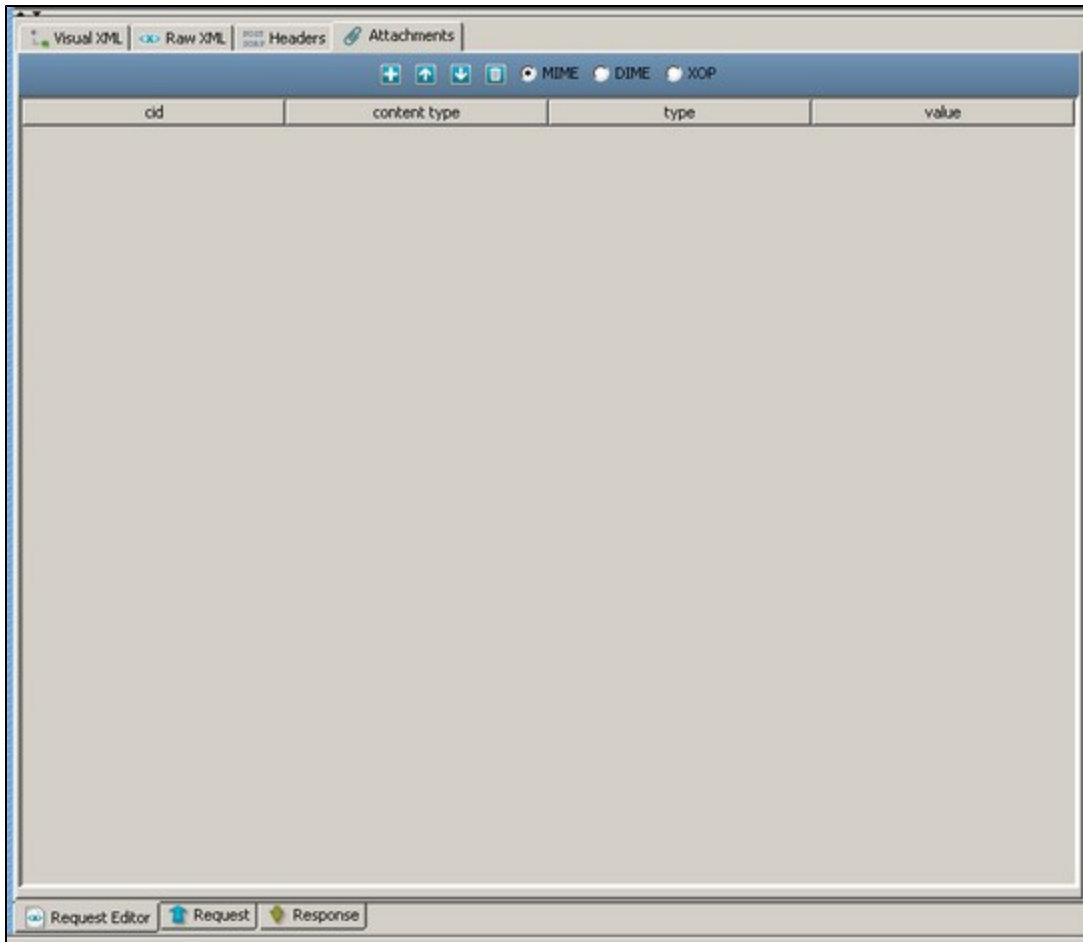
Request Editor Request Response

Headers

This is a place to insert headers that will be transmitted with the SOAP message (e.g. HTTP Headers or JMS properties).

Attachments

Explicitly showing attachments is one of the major usability differences from the legacy Web Service Execution step. In the legacy step, attachments were handled *automatically* by the generated Java classes, but they could be very difficult to configure the necessary Java objects to use them properly. We now have an explicit tab dedicated to the editing of Attachment data (as well as a new tab to show Sent/Received attachments in the Request/Response tabs described below).



If you are planning on using referenced attachments (attachments referenced in the SOAP message), then in the VXE you can right-click on the element that you want to be the references attachment and choose to Convert to Attachment (for MIME and DIME) or Convert to XOP Attachment (for MTOM/XOP Include style). This will automatically create the necessary elements and attributes and configure the content id used to match up the element with the attachment. It will then switch over to the attachments tab and pre-populate a new attachment with the content id, select a default content type and attachment type, as well as populate the value with any existing element data from the VXE.

If you are planning on using unreferenced (i.e. anonymous) attachments, switch over the Attachments tab and add an attachment manually.

Add, Up, Down, Delete — To add, remove, or rearrange the attachments in the table.

MIME/DIME/XOP — This controls how the attachments are sent, either using MIME, DIME, or XOP standards. *Note: XOP sends different content headers based on the SOAP version.*

cid — The Content ID which can be used in a href attribute in the SOAP message to link the element to the attachment data

content type — The mime encoding type used to assist the server in how to process the attachment data

type/value — The LISA attachment type determines how to edit and interpret the value data. Each type has it's own editor described below:

Type Editors

- XML — an XML aware text editor to edit the attachment value.
- Text — a Text based editor to edit the attachment value.
- Base64 Encoded — a text based editor to edit the attachment value and a bytes viewer to view the decoded binary data.
- Hex Encoded — a text based editor to edit the attachment value and a bytes viewer to view the decoded binary data.
- URL/Text — a url field to edit the attachment value and a text data viewer to view the results of loading the data from the URL.
- URL/XML — a url field to edit the attachment value and a XML aware text data viewer to view the results of loading the data from the URL.
- URL/Binary — a url field to edit the attachment value and a binary data viewer to view the results of loading the data from the URL.
- Property — a property field to edit the attachment value. If the resulting property is a String, it will send the attachment as text, otherwise it will send it as binary data.
- Property/URL — a property field to edit the attachment value. The property value is assumed to be a URL. The URL contents is loaded and sent as the attachment data.

Design Time Execution

Now that you have completed configuring the connection information and building the SOAP Request Message you can now test the step by executing it at design time.

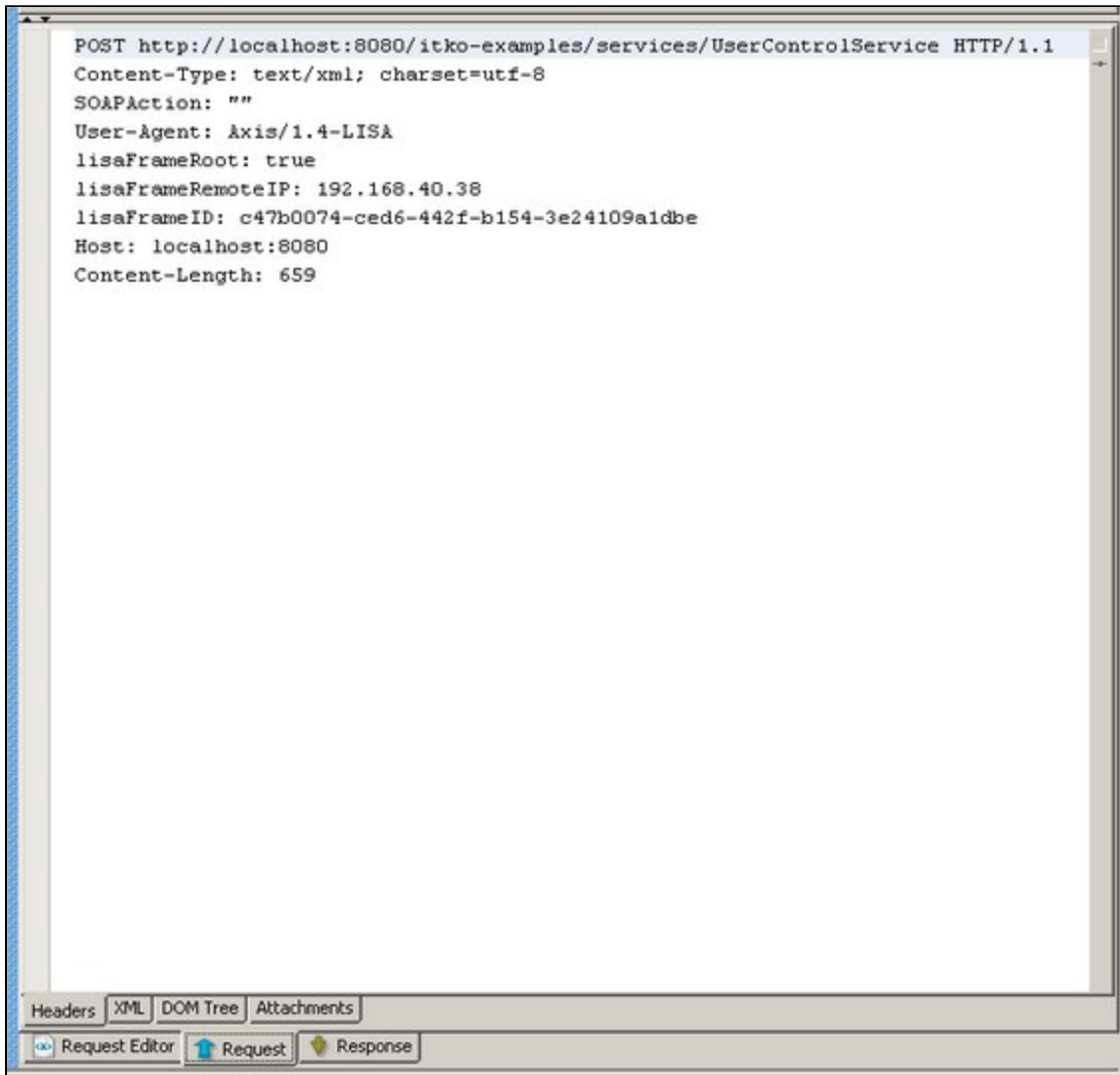
Execute the Web Service operation by clicking the **Execute** button  in the upper right corner. This will open the **Advanced** Tab.

Once executed, the Request and Response tabs will be populated and it will switch to the Response tab automatically.

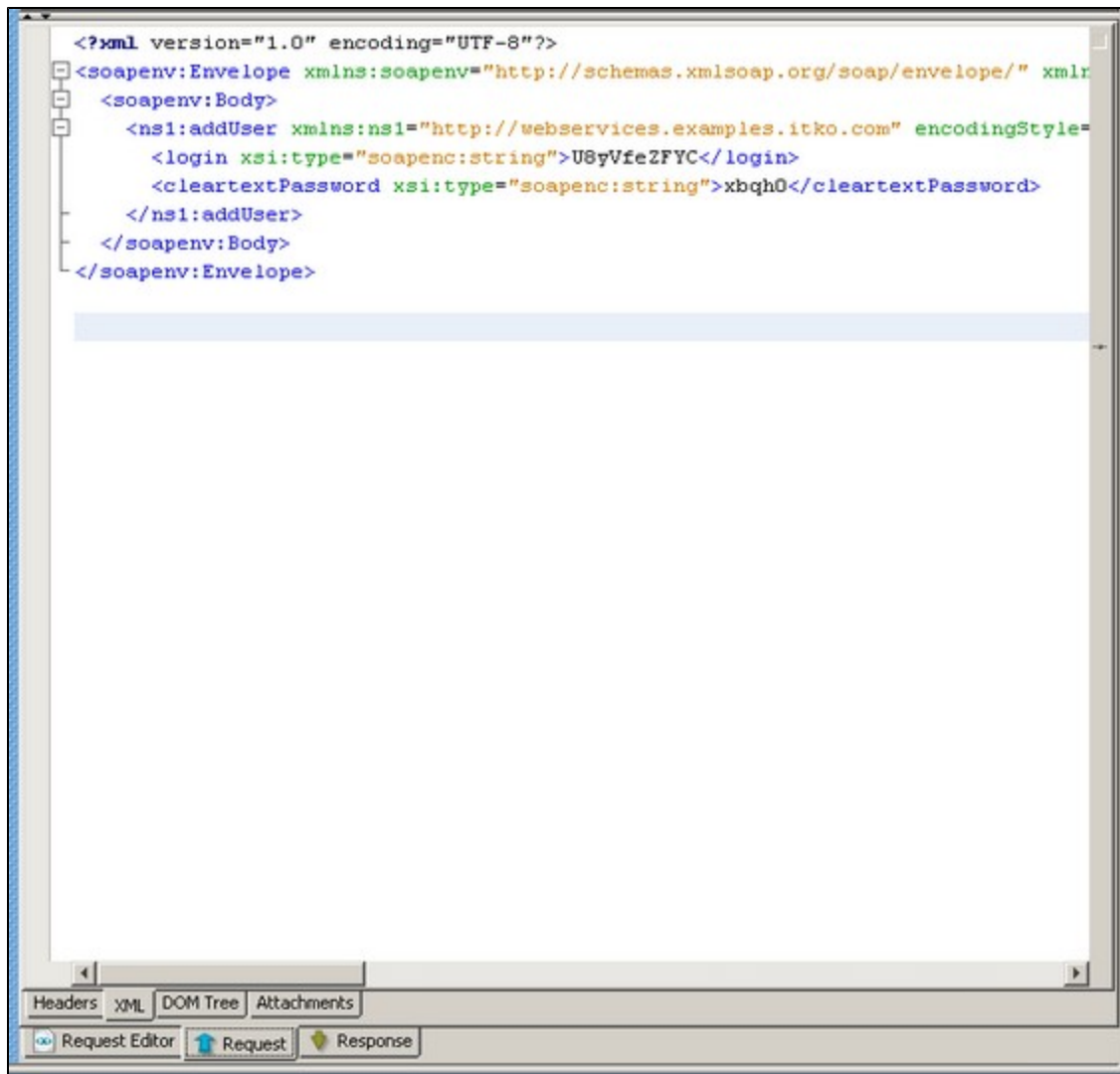
Request

The Request tab will show the resulting Request data that was sent after any post processing (e.g. substituting lisa properties). If the message contained any attachment, it will not show the raw MIME or DIME encoded message, but rather the processed message and attachments. If you want to see the raw message a tool like tcpmon should be used.

Header — Shows the Transport Headers that were sent for the Request.



XML — Shows the raw SOAP message that was sent after any advanced processing.



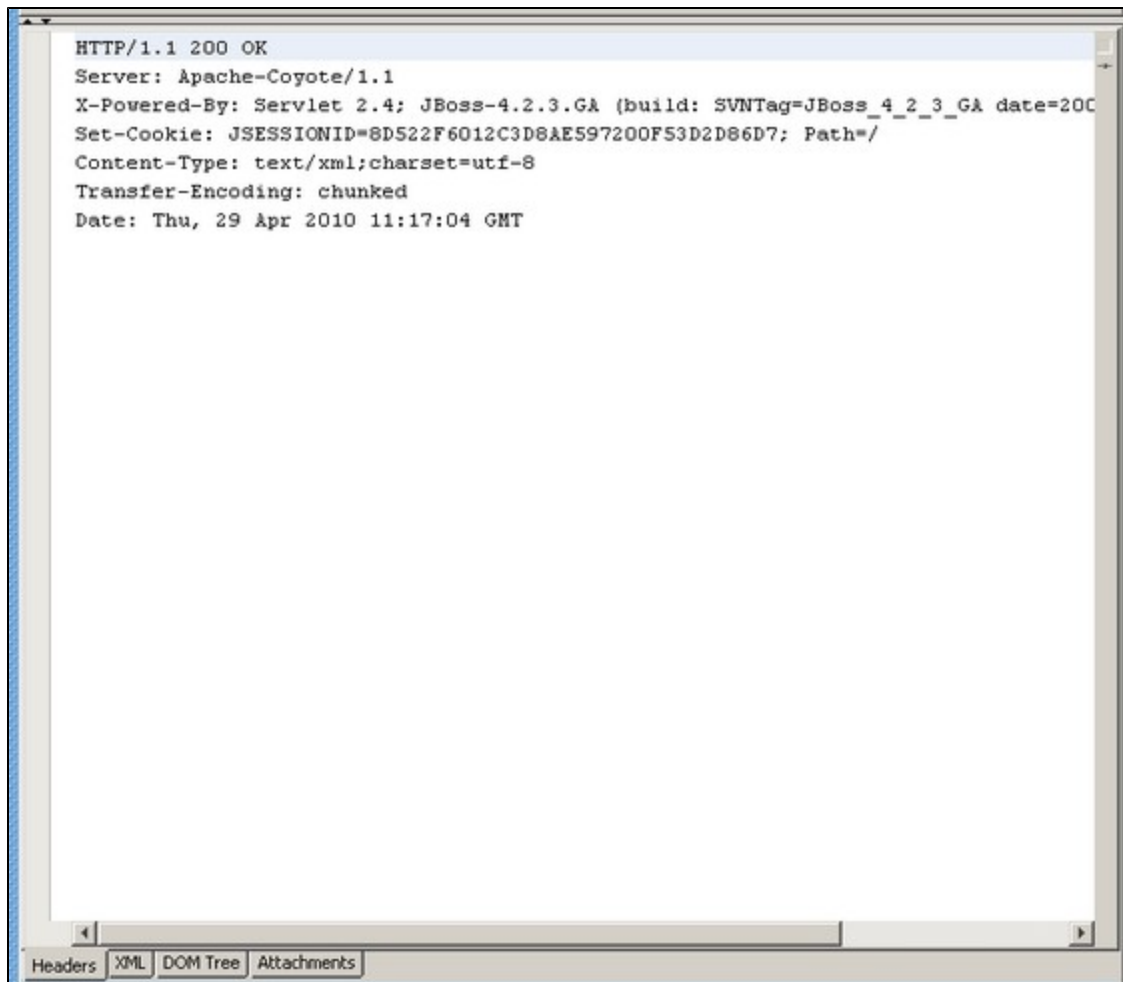
DOM Tree — Shows a DOM tree for the SOAP message. This will be replaced with a read-only VXE shortly. **Attachments** — Shows any Attachments that were sent.

Attachments — Shows any Attachments that were sent

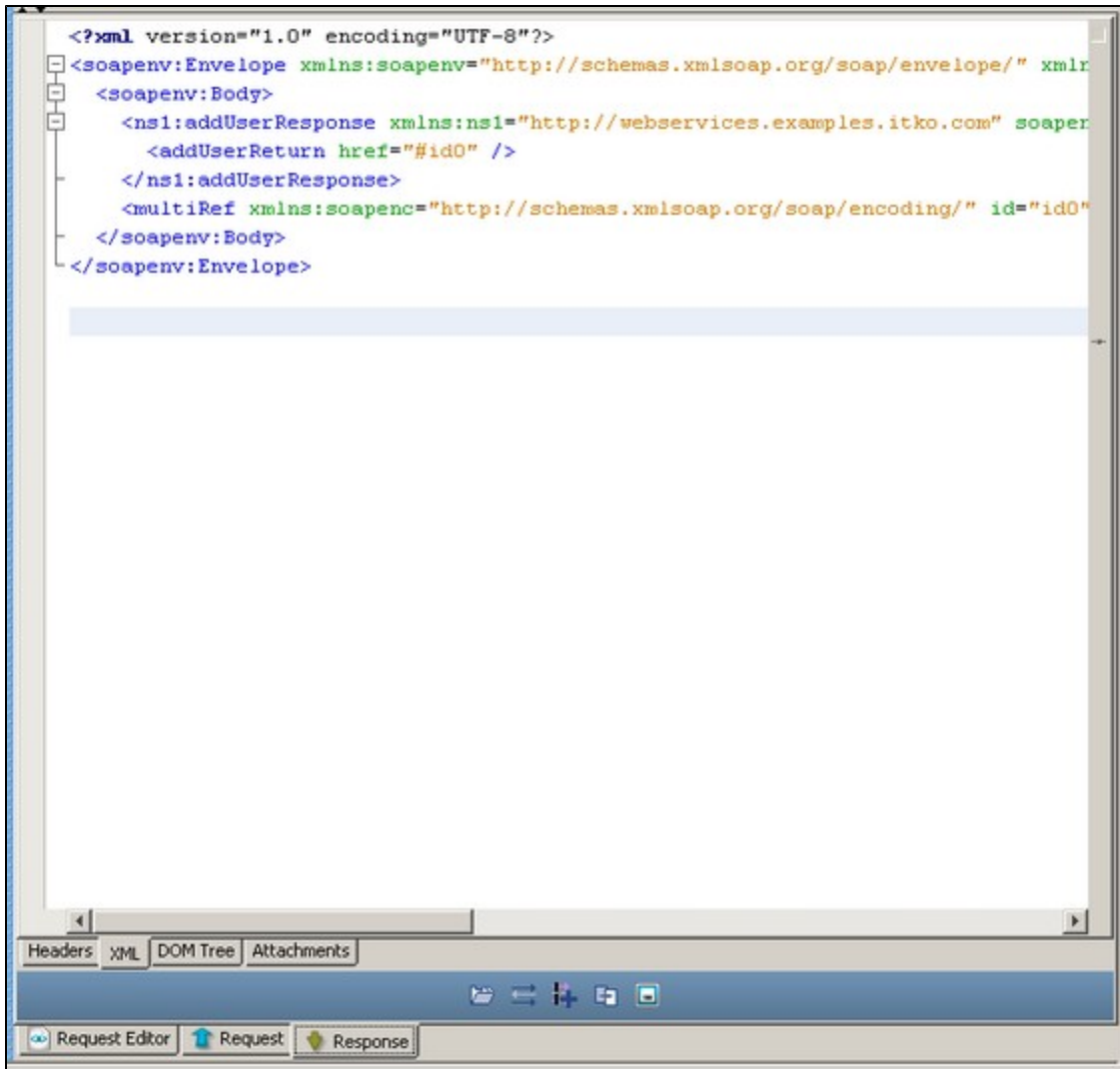
Response

The Response tab will show the resulting Response data that was received. If the message contained any attachment, it will not show the raw MIME or DIME encoded message, but rather the processed message and attachments. If you want to see the raw message a tool like tcpmon should be used. If there are any advanced post-processing options set (see below), the SOAP response message will be show post-processed.

Header — Shows the Transport Headers that were received form the Response.



XML — Shows the raw SOAP message that was received after any advanced processing.



DOM Tree — Shows a DOM tree for the SOAP message. This will be replaced with a read-only VXE shortly. From this tab you can quickly add filters and assertions on the resulting SOAP message.

Attachments — Shows any Attachments that were received. Received attachments can be accessed via automatically generated LISA properties (for use in later steps, filters, or assertions). For each attachment the following LISA properties are set:

Property	Value
<code>lisa.<step name>.rsp.attachment.<cid></code>	attachment value, byte[] or String
<code>lisa.<step name>.rsp.attachment.contenttype.<cid></code>	content type (mime type, e.g. text/plain)
<code>lisa.<step name>.rsp.attachment.<index></code>	attachment value, byte[] or String
<code>lisa.<step name>.rsp.attachment.contenttype.<index></code>	content type (mime type, e.g. text/plain)
<code>lisa.<step name>.rsp.attachment.contentid.<index></code>	Content Id (cid)

- `<step name>` is the LISA step name that is being executed.
- `<cid>` is the Content ID that is usually referenced in the SOAP message.
- `<index>` starts at 0 and is incremented for each attachment in the response attachments list.

Advanced Configuration

Transport



HTTP Version — This is 1.1 by default and control which HTTP protocol is used when sending the operation request.

SOAP Version — This is auto-populated based on the WSDL definition. SOAP Version will control the generation of a number of transport header (e.g. SOAPAction and contentType).

Call Timeout (ms) — This defines how long LISA will wait while trying to execute the operation. Once the timeout is hit, an exception will be thrown and the On Error handling will occur.

SSL



- **SSL Keystore File:** The name of the keystore file where the client identity certificate is stored. It can be in jks or pkcs format
- **SSL Keystore Password:** The password for the keystore

You can specify **global certificates properties** for SSL in your **local.properties** file.

For global certificates (web server, raw SOAP, and web service steps)

- `ssl.client.cert.path` = a full path to the keystore
- `ssl.client.cert.pass` = password for the keystore (this password will be automatically encrypted when Lisa runs)
- `ssl.client.key.pass` = an optional password for the key entry if you are using the JKS keystore and the key has different password from the keystore (this password will be automatically encrypted when Lisa runs)

Note: This is currently not an available option to be set in the WS Test step and if required must be set in the local.properties file.

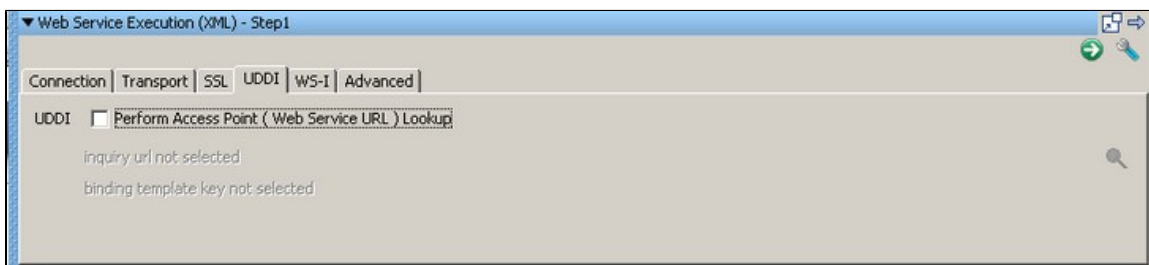
For web service steps only certificates (not raw soap steps))

- `ws.ssl.client.cert.path` = a full path to the keystore
- `ws.ssl.client.cert.pass` = password for the keystore (this password will be automatically encrypted when Lisa runs)
- `ws.ssl.client.key.pass` = an optional password for the key entry if you are using the JKS keystore and the key has different password from the keystore (this password will be automatically encrypted when Lisa runs)

Note: This is currently not an available option to be set in the WS Test step and if required must be set in the local.properties file.

Note: If you have values in local.properties and in the general tab, the values in the general tab will be used.

UDDI

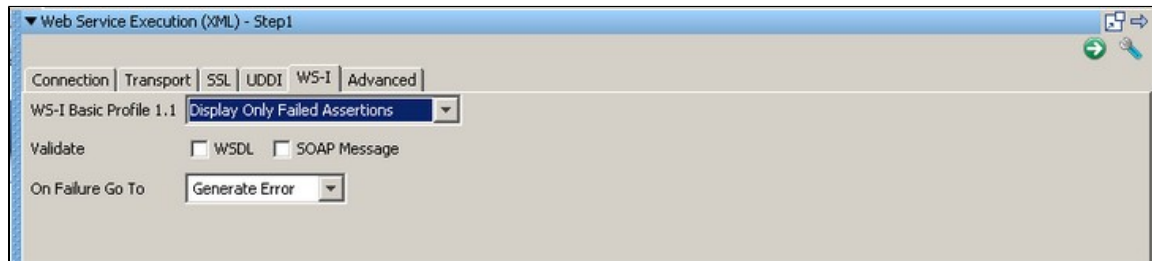


- **Perform Access Point (Web Service URL) Lookup**
- Select the **Inquiry URL** and **Binding Template**: use the Search UDDI Server find button to navigate to the correct Binding Template to

perform the lookup.

Note: when creating the step if you used the UDDI Search function when specifying the Web Service WSDL URL, these values will be automatically filled in. If the **Inquiry URL** is specified but the **Binding Template** is not, you may have preformed a Model search. In order to locate the Binding Template you need to perform a search at a higher level of the hierarchy and drill down to the TModel through a particular Binding Template.

WS-I



- **WS-I Basic Profile 1.1:**

You can choose four different validation levels in the pull down menu.

1. Display All Assertions
2. Display All But Info Assertions
3. Display Only Failed Assertions
4. Display Only Not Passed Assertions

- **Validate:**

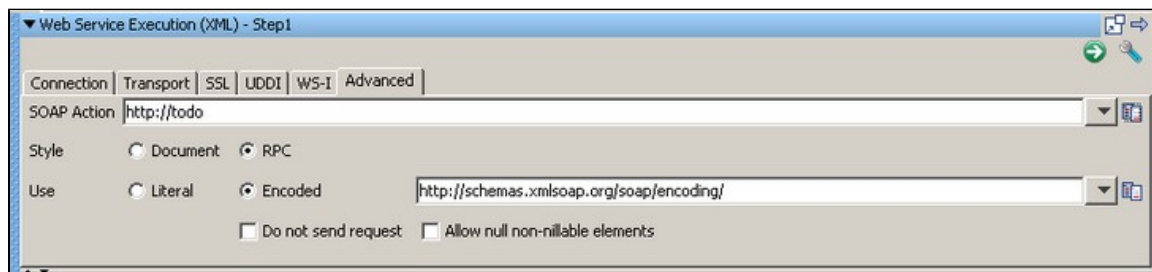
WSDL: Check to validate the WSDL

SOAP Message: Check to validate the SOAP message

- **On Failure GO To:** Select the step to redirect to on error

Note: Validation failures are common, but usually should not affect the outcome of the test. It is good practice to set the next step to continue so that you can complete the test.

Advanced



SOAP Action — This field is auto-populated based on the WSDL operation definition. It is used as the value of the SOAPAction transport header for SOAP 1.1 request message. Changing this field manually should only be done in rare cases.

Style — This field is auto-populated based on the WSDL operation definition. It is used to determine how a sample soap message is generated. Changing this field manually should only be done in rare cases.

Use — This field is auto-populated based on the WSDL operation definition. It is used to determine how a sample soap message is generated. If Encoded is selected, you can also edit the Encoded URI in the field next to the choice. Changing these fields manually should only be done in rare cases.

Do Not Send Request — When selected the step execution will preform all the normal SOAP message processing but will not send the generated SOAP message. Instead it will set the response to be the request message that would have been sent.

Allow null non-nilables — Left over from the legacy WS step. This is no longer used and will be removed.

Request Editor

Addressing

You can send a WS-Addressing header with your request. The WSDL does not specify if WSAddressing information is required so you must configure it.

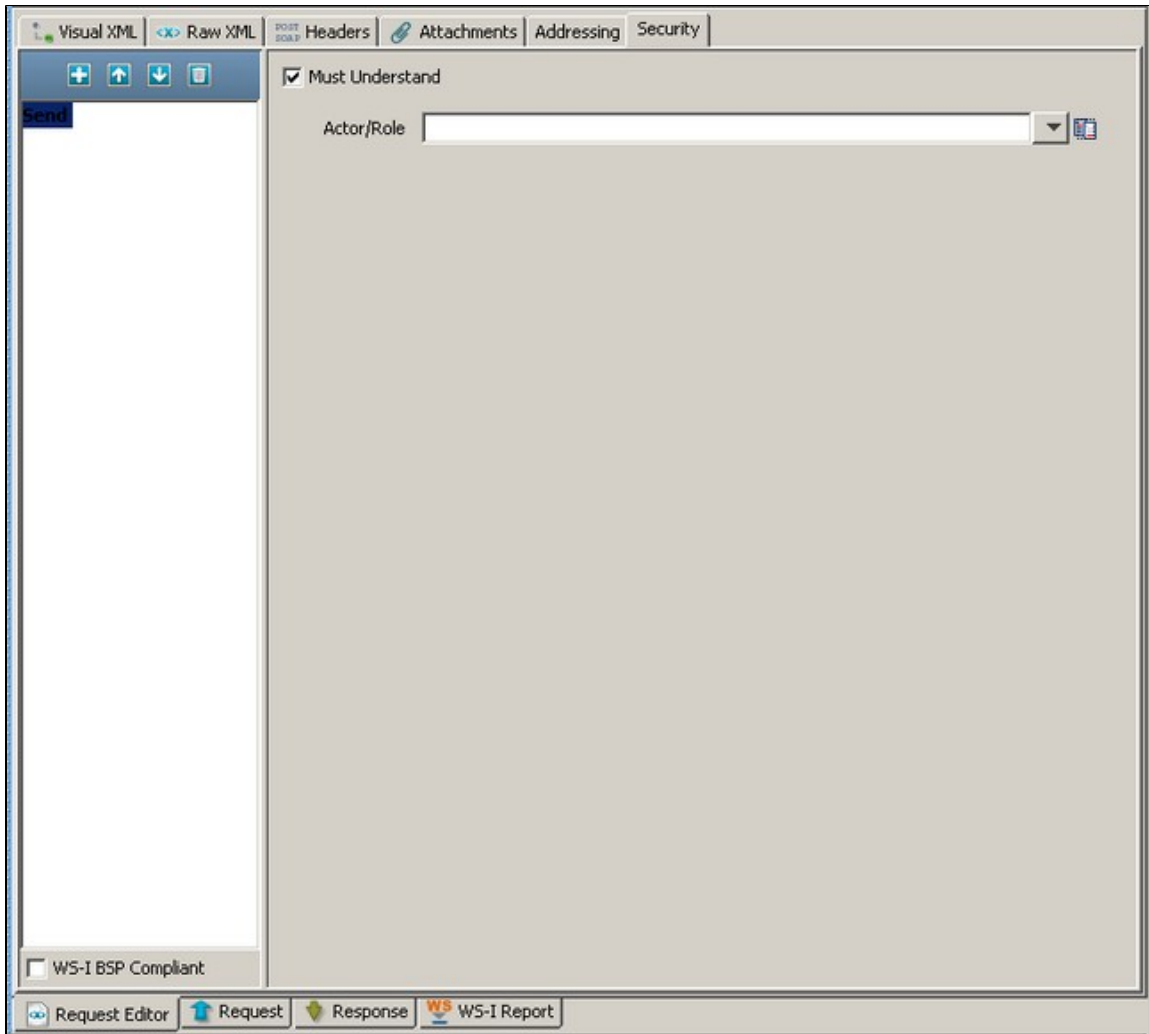
Click the **Addressing** tab, and then specify:

The screenshot shows a software interface with several tabs: Visual XML, Raw XML, POST SOAP, Headers, Attachments, Addressing (selected), and Security. The Addressing tab contains the following options:

- ☒ Use WS-Addressing
- Version: ☒ Final (2005/08) ☐ Submission (2004/08) ☐ Draft (2004/03) ☐ Draft (2003/03)
- Default and Override columns for the following elements:
 - To: ☒ Default, ☐ Override
 - From: ☒ Default, ☐ Override
 - Action: ☒ Default, ☐ Override
 - MessageId: ☒ Default, ☐ Override
 - ReplyTo: ☐ Default, ☐ Override
 - FaultTo: ☐ Default, ☐ Override
- ☐ Must Understand

- **Use WS-Addressing:** Click to use WS-Addressing
- **Version:** Select appropriate version. Several versions of the WS-Addressing specification are listed as options because some web services platforms (e.g. .Net) still use the older Draft specifications. You will need to determine which your web service platform is using.
- LISA will populate as many values as it can. Then you can choose to use the **Default** value or **Override** it. You can choose not to send some of the **Default** elements by un-checking the **Default** check box for that element.
- Click **Must Understand** check box if you want to assure that the web service can understand the WSAddressing header

Security



Click the **Security** tab. Click **Send**.

- Check '**Must Understand**' check box (if needed or if you want to ensure that the WS-Security header is processed by the server)
- Enter the **Actor/Role** name (if needed – most web services do not use multiple Actors/Roles)
- Click the **Add**, icon and select the security action type to add. You will be presented with the configuration panel for that security action type. These are discussed below.

Adding security verification to the response is very similar:

Click the **Security** tab. Click the **Add**, icon and select **Receive**.

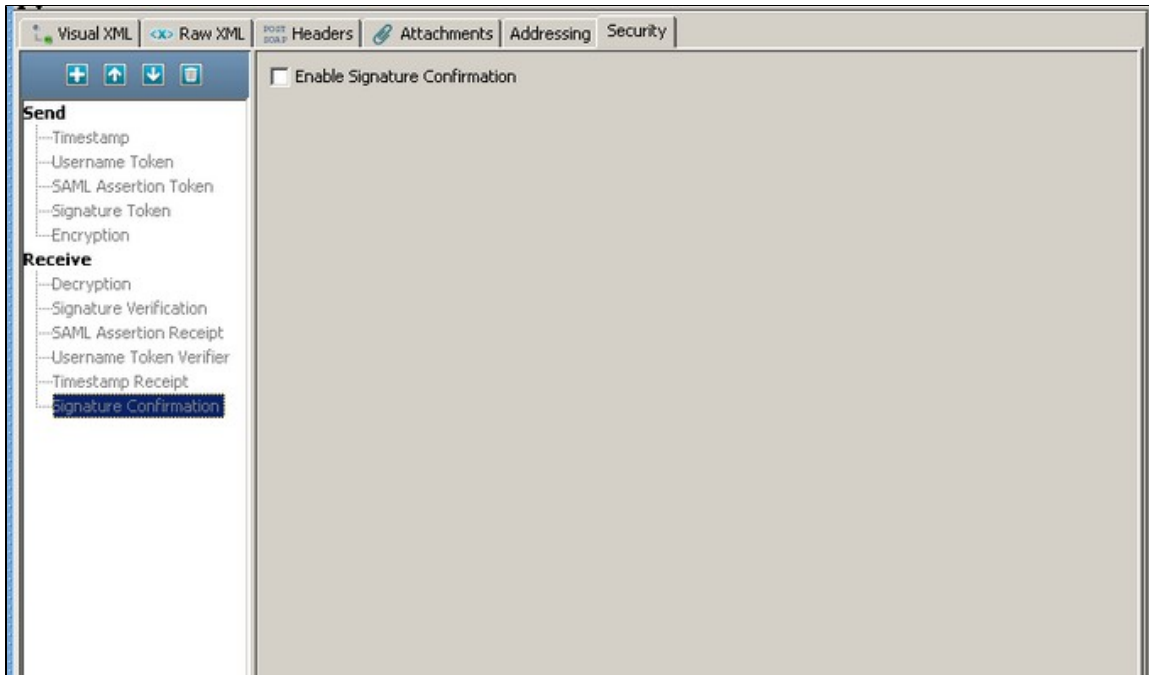
- Enter the **Actor/Role** name (if needed)
- Click the **Add**, icon and select the security action type. You will be presented with the configuration panel for that security action type. These are discussed below.

Note: You can add as many security types as are needed to execute your Web Service.

When a keystore is being used in a security action type configuration (described below for each type), you can verify your keystore settings to make sure you are using the correct format, password, alias and alias password. There is a **Verify** button on the editors for Signature, Encryption/Decryption and SAML Assertion Token. Clicking the **Verify** button will invoke the Keystore Verifier, which will produce a verification report.

If you do not know the expected alias name for a WS-Security setting, you can use the Keystore Verifier to list all of the aliases in the keystore. Leave the Keystore Alias and Alias Password boxes empty and click the **Verify** button. The Keystore Verifier is described at the end of this section on WS-Security.

An example screen with several security types chosen is shown below (disabled actions are shown in gray):

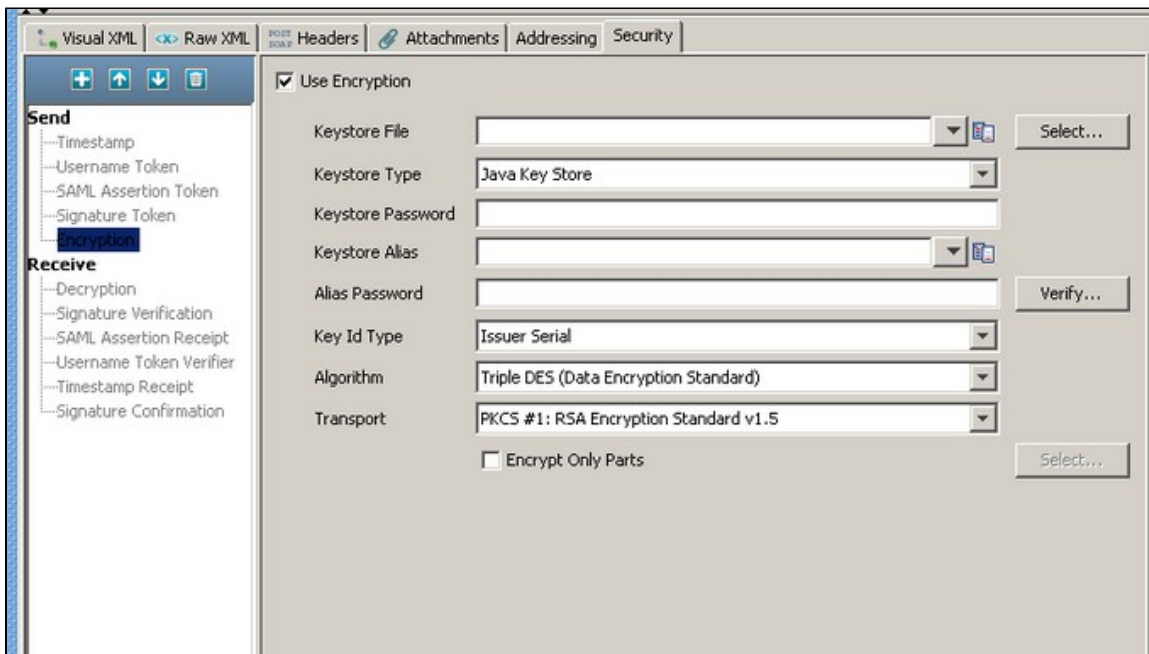


In the following sections we will describe the parameters needed to run the ws-security example

1) XML Encryption/Decryption

The parameters required configuring XML Encryption and Decryption are shown below:

a) Encryption

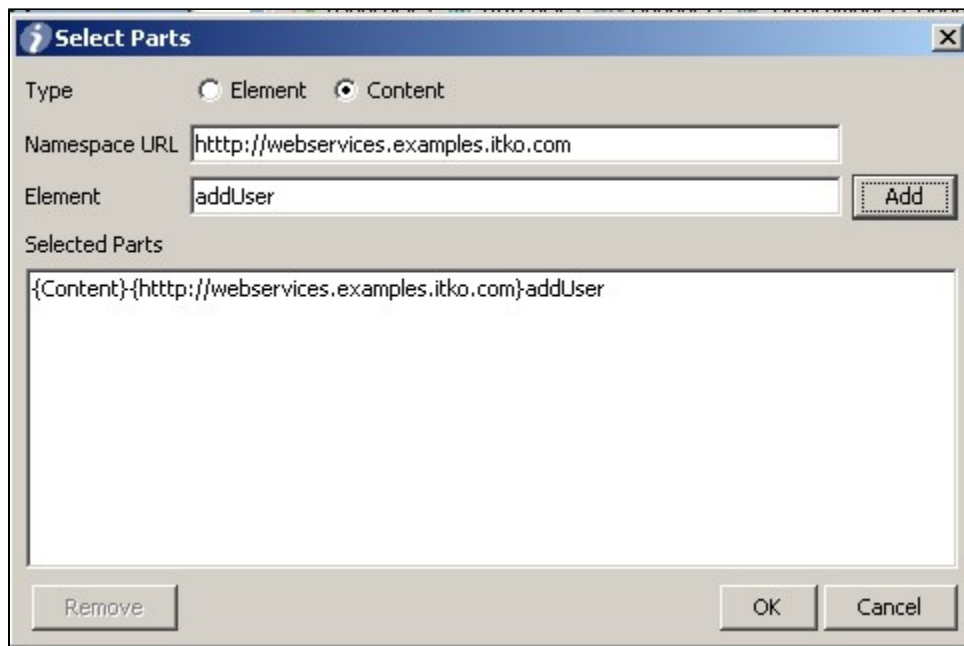


Click the 'Use Encryption' check box

- **Keystore File:** The location of keystore file
- **Keystore Type:** Select Java Key Store (jks) or Personal Information Exchange (PKCS #12)
- **Keystore Password:** Enter the password for the keystore
- **Keystore Alias:** (should be an alias for a public key)
- **Alias Password:** (empty/same as Keystore Password for PKCS #12 files)
- **Key ID Type:** Select the appropriate key ID type from the pull-down menu
- **Algorithm:** Triple DES, AES 128, AED 192, or AES 236
- **Transport:** PKCS#1: RSA Encryption Standard v1.5 is the only option

The default behavior is to encrypt only the SOAP Body contents.

Encrypt Only Parts: If you want to specify different parts to encrypt click **Select** button to identify the parts to be encrypted. In the pop-up screen that appears:



The image shows a 'Select Parts' dialog box with a title bar containing a question mark icon and a close button. Inside the dialog, there are two radio buttons under the 'Type' label: 'Element' (unselected) and 'Content' (selected). Below these are two text input fields: 'Namespace URL' containing 'http://webservices.examples.itko.com' and 'Element' containing 'addUser'. To the right of the 'Element' field is an 'Add' button. Below the input fields is a list box labeled 'Selected Parts' which contains the text '{Content}-{http://webservices.examples.itko.com}addUser'. At the bottom of the dialog are three buttons: 'Remove', 'OK', and 'Cancel'.

Type: select one of the following:

- **Element:** Select if you want to encrypt the element and the content
- **Content:** Select if you want to encrypt just the content
- **Namespace URL:** Enter the value for the element.
- **Element:** enter the name of the element.

You can repeat this for as many elements as you wish by clicking the **Add** button.

You will need to manually add the Body element if you wish it to be included.

Note: If you want to include the Binary Security Token as a part use the Element name 'Token'.

b) Decryption

The parameters required to configure decryption are a subset of those required for encryption:

The screenshot shows the 'Security' tab of a SOAP client interface. On the left, a tree view under 'Send' has 'Encryption' selected. The main area is titled 'Use Encryption' and contains the following fields:

- Keystore File:** A text box with a 'Select...' button.
- Keystore Type:** A dropdown menu set to 'Java Key Store'.
- Keystore Password:** A text box.
- Keystore Alias:** A text box with a 'Select...' button.
- Alias Password:** A text box with a 'Verify...' button.
- Key Id Type:** A dropdown menu set to 'Issuer Serial'.
- Algorithm:** A dropdown menu set to 'Triple DES (Data Encryption Standard)'.
- Transport:** A dropdown menu set to 'PKCS #1: RSA Encryption Standard v1.5'.
- Encrypt Only Parts:** An unchecked checkbox.

At the bottom right, there is a 'Select...' button.

2) XML Signature Token/Signature Verification

The parameters required to configure XML Signature Tokens and Signature Verification are shown below:

a) Signature token

The screenshot shows the 'Security' tab of a SOAP client interface. On the left, a tree view under 'Send' has 'Signature Token' selected. The main area is titled 'Add Signature' and contains the following fields:

- Keystore File:** A text box with a 'Select...' button.
- Keystore Type:** A dropdown menu set to 'Java Key Store'.
- Keystore Password:** A text box.
- Keystore Alias:** A text box with a 'Select...' button.
- Alias Password:** A text box with a 'Verify...' button.
- Key Id Type:** A dropdown menu set to 'Issuer Serial'.
- Algorithm:** A dropdown menu set to 'RSA with SHA-1'.
- Sign Only Parts:** A checked checkbox.

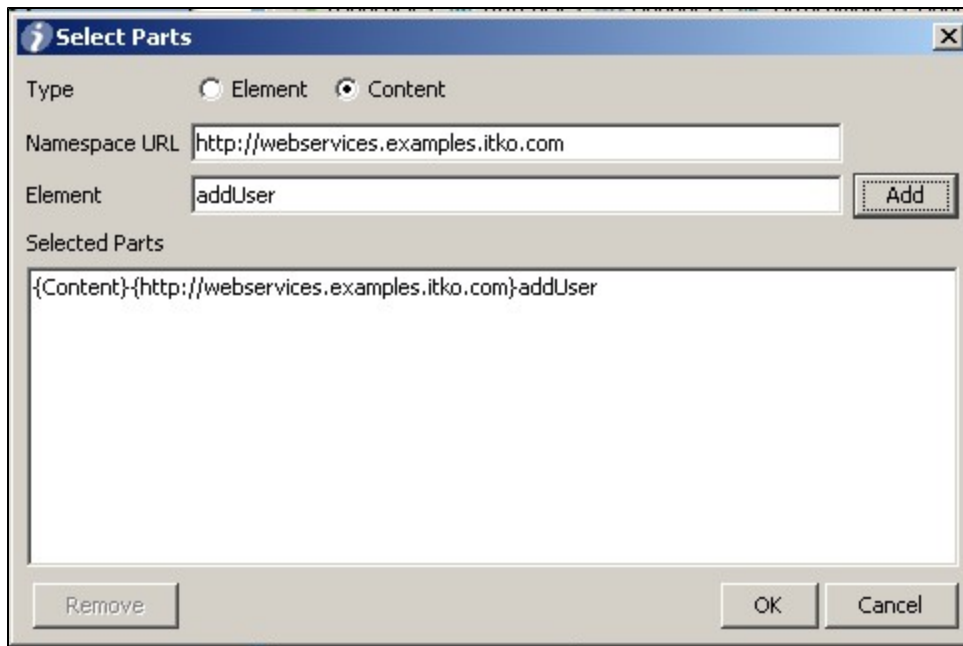
At the bottom right, there is a 'Select...' button.

Click the **Add Signature** box

- **Keystore File:** The location of keystore file
- **Keystore Type:** Select Java Key Store (jks) or Personal Information Exchange (PKCS #12)
- **Keystore Password:** Enter the password for the keystore
- **Keystore alias:** (should be an alias for a private key)
- **Alias Password:** (empty/same as Keystore Password for PKCS #12 files)
- **Key ID type:** Select the appropriate key ID type from the pull-down menu
- **Algorithm:** Select DSA with SHA-1

The default behavior is to sign only the SOAP Body contents.

Sign Only Parts: If you want to specify different parts to sign click the **Select** button to identify the parts to be signed. In the pop-up screen that appears:



The 'Select Parts' dialog box is used to configure the parts to be signed. It features a 'Type' section with radio buttons for 'Element' and 'Content' (selected). Below this are input fields for 'Namespace URL' (containing 'http://webservices.examples.itko.com') and 'Element' (containing 'addUser'), followed by an 'Add' button. A 'Selected Parts' list box contains the entry '{Content}-{http://webservices.examples.itko.com}addUser'. At the bottom are 'Remove', 'OK', and 'Cancel' buttons.

- **Type:** Select one of the following:
- **Element:** Select if you want to sign the element and the content.
- **Content:** Select if you want to sign just the content.
- **Namespace URL:** Enter the value for the element.
- **Element:** enter the name of the element.

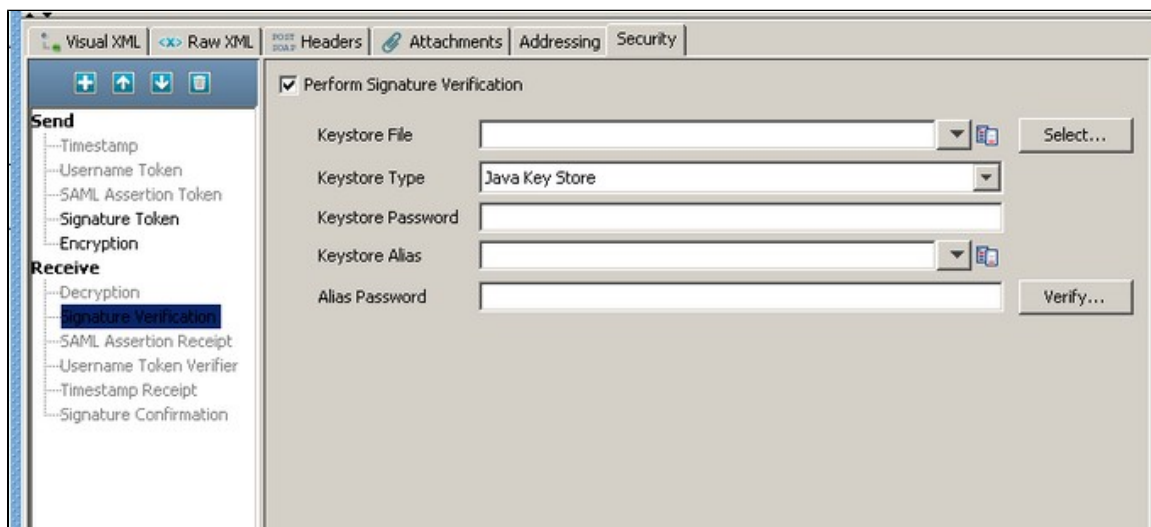
You can repeat this for as many elements as you wish by clicking the **Add** button.

You will need to manually add the Body element if you wish it to be included.

If you want to include the Binary Security Token as a part, use the Element name 'Token'.

b) Signature Verification

The parameters required to configure signature verification are a subset of those required for signing:

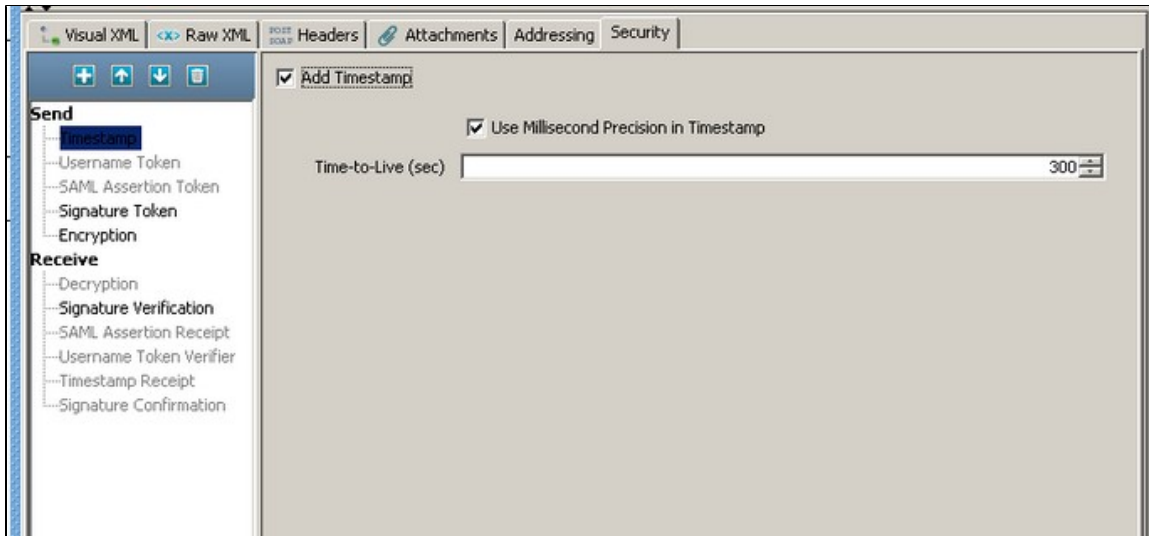


The 'Security' tab of the configuration dialog shows options for signature verification. The 'Perform Signature Verification' checkbox is checked. Below it are fields for 'Keystore File' (with a 'Select...' button), 'Keystore Type' (set to 'Java Key Store'), 'Keystore Password', 'Keystore Alias' (with a file icon button), and 'Alias Password' (with a 'Verify...' button). On the left, a tree view shows 'Send' and 'Receive' sections, with 'Signature Verification' highlighted under 'Receive'.

3) Timestamp/Timestamp Receipt

The parameters required to configure Time stamp and timestamp Receipt are shown below:

a) Timestamp



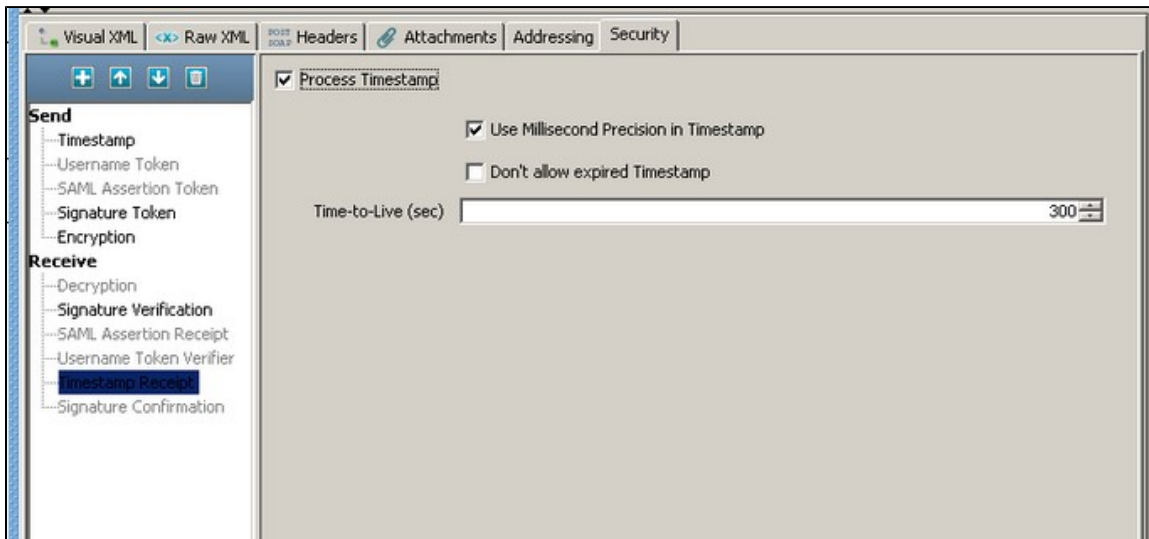
Click the **Add Timestamp** box

- **Time-To-Live (sec):** Enter the lifetime of the message in seconds. Enter 0 to not include an Expires element.

Note : Some Web services, particularly .NET 1.x/2.0 with WSE 2.0, are not compliant with standard time stamp date formatting, and do not allow milliseconds. For these web services:

Use Millisecond Precision in Timestamp: Deselect the checkbox

b) Timestamp Receipt



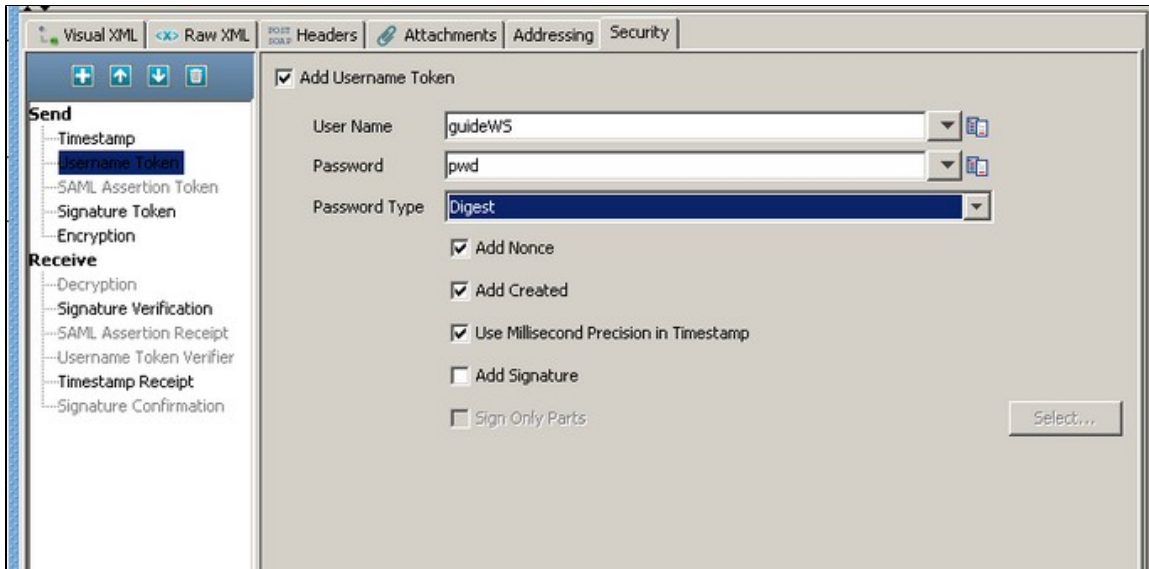
The parameters required for Timestamps Receipt are a super set of those required for Time stamp. The extra parameter:

- **Don't allow expired** Time stamp can be checked if you do not want to allow expired timestamps.

4) Username Token/Username Token Verifier

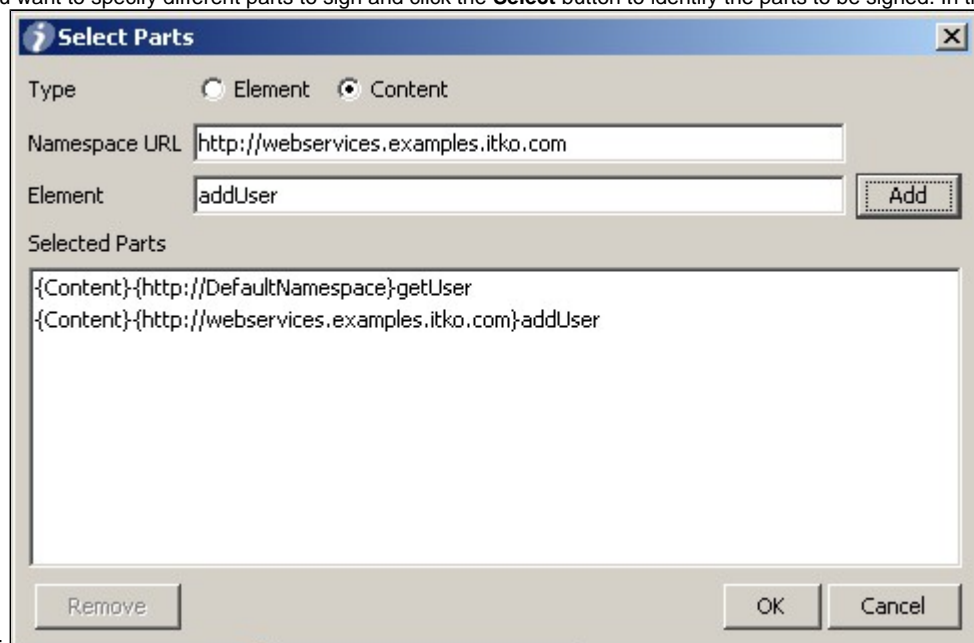
The parameters required to configure Username Token and Username token Verifier are shown below:

a) Username Token



Click the **Add Username Token** box

- **User Name:** Enter the appropriate user name.
- **Password:** Enter the appropriate password.
- **Password Type:** Select the password type from the drop-down menu (Text, Digest, None). None is typically used with the '**Add Signature**' option.
- **Add Nonce:** Click if a Nonce is required – used to protect against replay attacks.
- **Add Created:** Click if a time stamp is required.
- **Use Millisecond Precision in Timestamp:** Select the check box to use millisecond precision. Note: some Web services, particularly .NET 1.x/2.0 with WSE 2.0, are not compliant with standard timestamp date formatting, and do not allow the use of milliseconds.
- **Add Signature:** To add a signature built using a combination of the username and password as the key.
- **Sign Only Parts:** Click if you want to specify different parts to sign and click the **Select** button to identify the parts to be signed. In the



pop-up screen that appears:

Type: select one of the following:

- **Element:** Select if you want to encrypt the element and the content
- **Content:** Select if you want to encrypt just the content
- **Namespace URL:** Enter the value for the element.
- **Element:** enter the name of the element.

You can repeat this for as many elements as you wish by clicking the **Add** button.

You will need to manually add the Body element if you wish it to be included.

If you want to include the Binary Security Token as a part, use the Element name 'Token'.

b) UserName Token Verifier

Visual XML | Raw XML | POST SOAP | Headers | Attachments | Addressing | Security

☒ **Verify Username Token**

User Name

Password

☒ Use Millisecond Precision in Timestamp

☐ Verify Signature

Click the **Verify Username Token** box

- **User Name:** Enter the verification user name.
- **Password:** Enter the verification password.
- **Use Millisecond Precision in Timestamp:** Select the check box to use millisecond precision. Note: some Web services, particularly .NET 1.x/2.0 with WSE 2.0, are not compliant with standard time stamp date formatting, and do not allow the use of milliseconds.
- **Verify Signature:** Click box if Signature verification is required.

5) SAML Assertion Token/SAML Assertion Receipt

The parameters required configuring SAML Assertion Token and SAML Assertion Verifier are shown below:

SAML Assertion Token:

Visual XML | Raw XML | POST SOAP | Headers | Attachments | Addressing | Security

☒ **Add SAML Token**

☒ From Step Result

☐ From Property

☒ Signed Sender Vouches

Keystore File

Keystore Type

Keystore Password

Keystore Alias

Alias Password

Key Id Type

Algorithm

☐ Sign Only Parts

- Click the **Add SAML Token** box
- Select the **From Step Results** radio box: Select the step whose result is an XML SAML Assertion (like a SAML Query Step or a Parse Text Step with xml manually entered), or

- select the **From Property radio box**: Enter the LISA property that contains the XML SAML Assertion
- You can optionally select the **Verify** button to have LISA parse the SAML Assertion XML and build the SAML Assertion object as it would when sending the SOAP request. This is useful to confirm that the SAML Assertion that may have been created manually is a valid SAML Assertion. It will also attempt to verify any signatures associated with the assertion, but it'd likely that LISA will not be able to verify the assertion without configuring a public certificate to verify with. This ability will be added in a future release of LISA.
- **Signed Sender Vouches**: If the assertion needs to be signed by the sender (the sender vouches for it's authenticity vs. the bearer/creator of the SAML Assertion). When selected you'll need to fill in the following information:
- **Keystore File**: The location of keystore file.
- **Keystore Type**: Select Java Key Store (jks) or Personal Information Exchange (PKCS #12).
- **Keystore Password**: Enter the password for the keystore.
- **Keystore alias**: (should be an alias for a private key).
- **Alias Password**: (empty/same as Keystore Password for PKCS #12 files).
- **Key ID type**: Select the appropriate key ID type from the pull-down menu.
- **Algorithm**: Select DSA with SHA-1.

The default behavior is to sign only the SOAP Body contents.

- **Sign Only Parts**: If you want to specify different parts to sign click the **Select** button to identify the parts to be signed. In the pop-up

screen that appears:

- **Type**: select one of the following:
- **Element**: Select if you want to sign the element and the content.
- **Content**: Select if you want to sign just the content.
- **Namespace URL**: Enter the value for the element.
- **Element**: enter the name of the element.

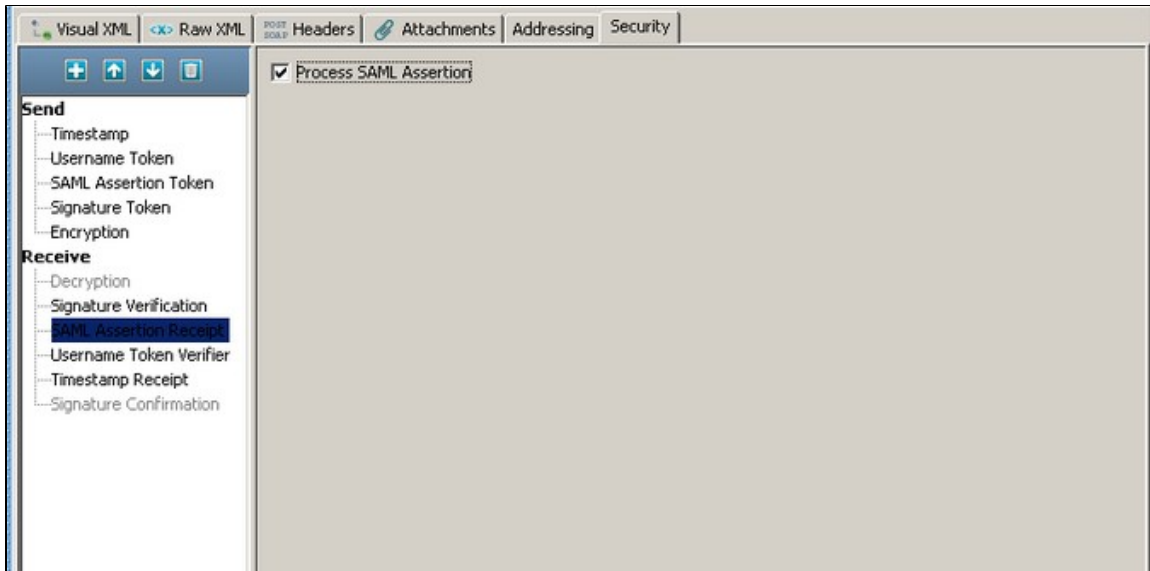
You can repeat this for as many elements as you wish by clicking the **Add** button.

Note: You will need to manually add the Body element if you wish it to be included.

If you want to include the Binary Security Token as a part, use the Element name 'Token'.

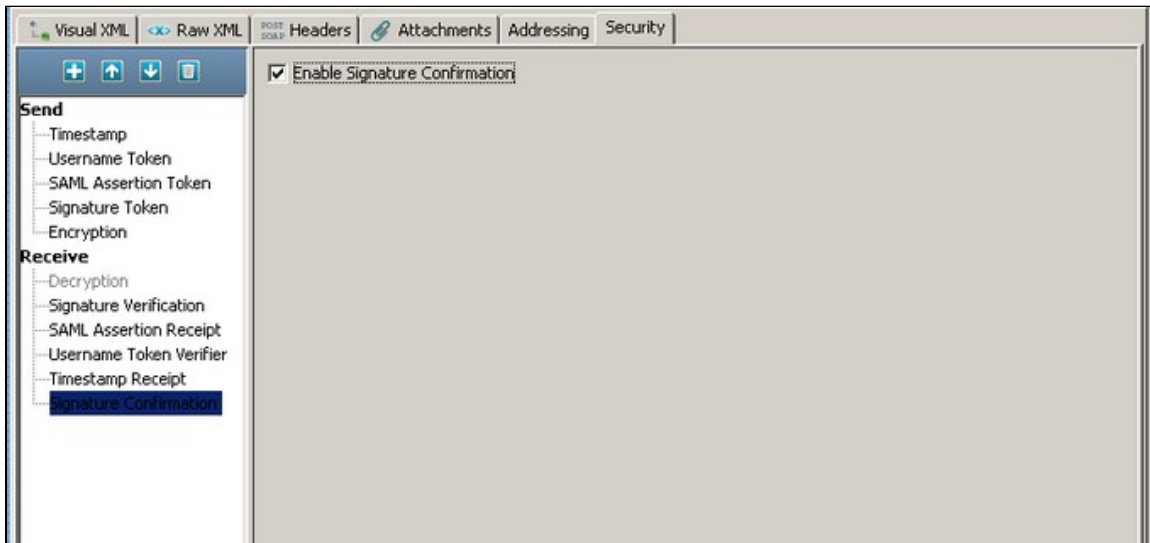
b) SAML Assertion Receipt

Click the **Process SAML Assertion** box if you want LISA to check for a SAML Assertion Receipt header in the response. If you select this option and there is no SAML Assertion Receipt header then an exception occurs.



6) Signature Confirmation

Click the **Signature confirmation** check box if you want LISA to check for a signature confirmation header in the response. If you select this option and there is no confirmation header then an exception occurs.



Using the Keystore Verifier

You can verify your key store settings to make sure you are using the correct format, password, alias and alias password. There is a **Verify** button on the editors for SSL,

Signature, Encryption/Decryption and SAML settings. Clicking **Verify** produces a verification report as shown below (extraneous lines deleted):



Beginning Keystore Verification

File: C:\Lisa\examples\keystores\myout.p12
Keystore size: 1
Provider: BC
Type: pkcs12

Alias: 16c73ab6-b892-458f-abf5-2f875f74882e

Alias Creation Date: Thu Aug 09 01:07:21 CDT 2007
Key Algorithm: RSA
Key Format: PKCS#8
Key:

Certificate #0:

Type: X.509
Certificate Info:
[0] Version: 1
 SerialNumber: 1148383597
 IssuerDN: C=IN,ST=KAr,L=blore,O=adea,OU=adea,CN=vivek
 Start Date: Tue May 23 06:26:37 CDT 2006
 Final Date: Thu May 22 06:26:37 CDT 2008
 SubjectDN: C=IN,ST=KAr,L=blore,O=adea,OU=adea,CN=vivek
 Public Key: RSA Public Key
 modulus: ade0695f66c0715d3aa73cd57dc0ca32810588f96faa4a0f3c889ff022cd432fd6
 public exponent: 10001

Certificate is Self-Signed
Certificate Verified against itself
Found Alias Successfully

SSL verification validates the Keystore password only and confirms that at least one of the keys in the keystore can be loaded using the keystore password.

Alias: test

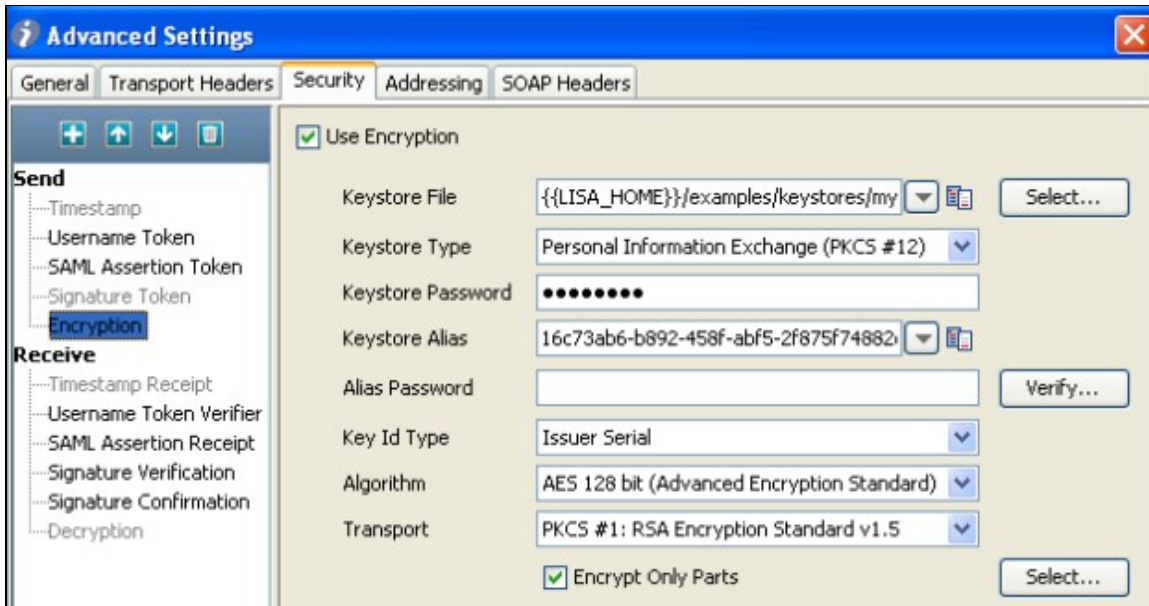
Alias Creation Date: Sat Apr 21 01:37:24 CDT 2007
Successfully loaded key for alias: test using keystore password
Key Algorithm: DSA
Key Format: PKCS#8
Key:

WS-Security verification validates the Keystore password, the alias and the alias password. Correct validation is indicated with a green entry. Any validation errors found will be shown in red. Warnings are shown in orange.

Note: This verification only verifies the keystore parameters. There could still be issues with the web service, such as a mismatch in certificate sets or incorrect choice of algorithm. These issues will need to be validated independently.

Alias Search

If you do not know the expected alias name for a WS-Security setting, you can use the keystore verifier to list all of the aliases in the keystore. Leave the **'Keystore Alias'** and **'Alias Password'** boxes empty and click the **Verify** button:



Advanced Settings

General Transport Headers **Security** Addressing SOAP Headers

Send

- Timestamp
- Username Token
- SAML Assertion Token
- Signature Token
- Encryption**

Receive

- Timestamp Receipt
- Username Token Verifier
- SAML Assertion Receipt
- Signature Verification
- Signature Confirmation
- Decryption

☒ Use Encryption

Keystore File: {{LISA_HOME}}/examples/keystores/my Select...

Keystore Type: Personal Information Exchange (PKCS #12)

Keystore Password:

Keystore Alias: 16c73ab6-b892-458f-abf5-2f875f74882e Select...

Alias Password: Verify...

Key Id Type: Issuer Serial

Algorithm: AES 128 bit (Advanced Encryption Standard)

Transport: PKCS #1: RSA Encryption Standard v1.5

☒ Encrypt Only Parts Select...

Aliases are highlighted with a blue background.



Beginning Keystore Verification

File: C:\Lisa\examples\keystores\myout.p12

Keystore size: 1

Provider: BC

Type: pkcs12

Alias: 16c73ab6-b892-458f-abf5-2f875f74882e

Alias Creation Date: Thu Aug 09 01:11:44 CDT 2007

Certificate #0:

Type: X.509

Note: Verification will fail since the Keystore Alias and Alias Password boxes were left blank.

WS-I Report

While executing WS-I Basic profile Validation ,when you click the **Execute** button on the Object Editor screen, the validation will be run and a report will be generated and saved (in the reports directory in the LISA install directory). You can view the report by pressing the **WS-I Report** tab at the bottom of the screen.

Note: The format of this report is standard, and is dictated by the Web-Services-Interoperability Organization (WS-I). A small example portion of the report is shown below:

WS-I Basic Profile Conformance Report

WS-I
WEB SERVICES
INTEROPERABILITY
ORGANIZATION

Report: WS-I Basic Profile Conformance Report
Timestamp: 2007-08-25T17:45:00.559

Copyright (c) 2003-2004 by The Web Services-Interoperability Organization (WS-I) and Certain of its Members. All Rights Reserved.

Analyzer Tool Information

Version	1.0.0
Release Date	2005-07-04
Implementer Name	WS-I Organization
Location	http://www.ws-i.org

Analyzer Runtime Environment Information

Runtime Name	Java(TM) 2 Runtime Environment, Standard Edition
Runtime Version	1.5.0_05-b05
Operating System Name	Windows XP

1.4 WSDL Validation

Back | Back-Header | Last Request | Last Response | WS-I.org

1.4 WSDL Validation

1.4 WSDL Validation

The WSDL Validation step allows you to load a WSDL and add one or more assertions to validate the WSDL. This step is a little different in that it enables you to load the static WSDL file and perform assertions on it. In most steps the assertions are being made on the response.

The most useful assertions are:

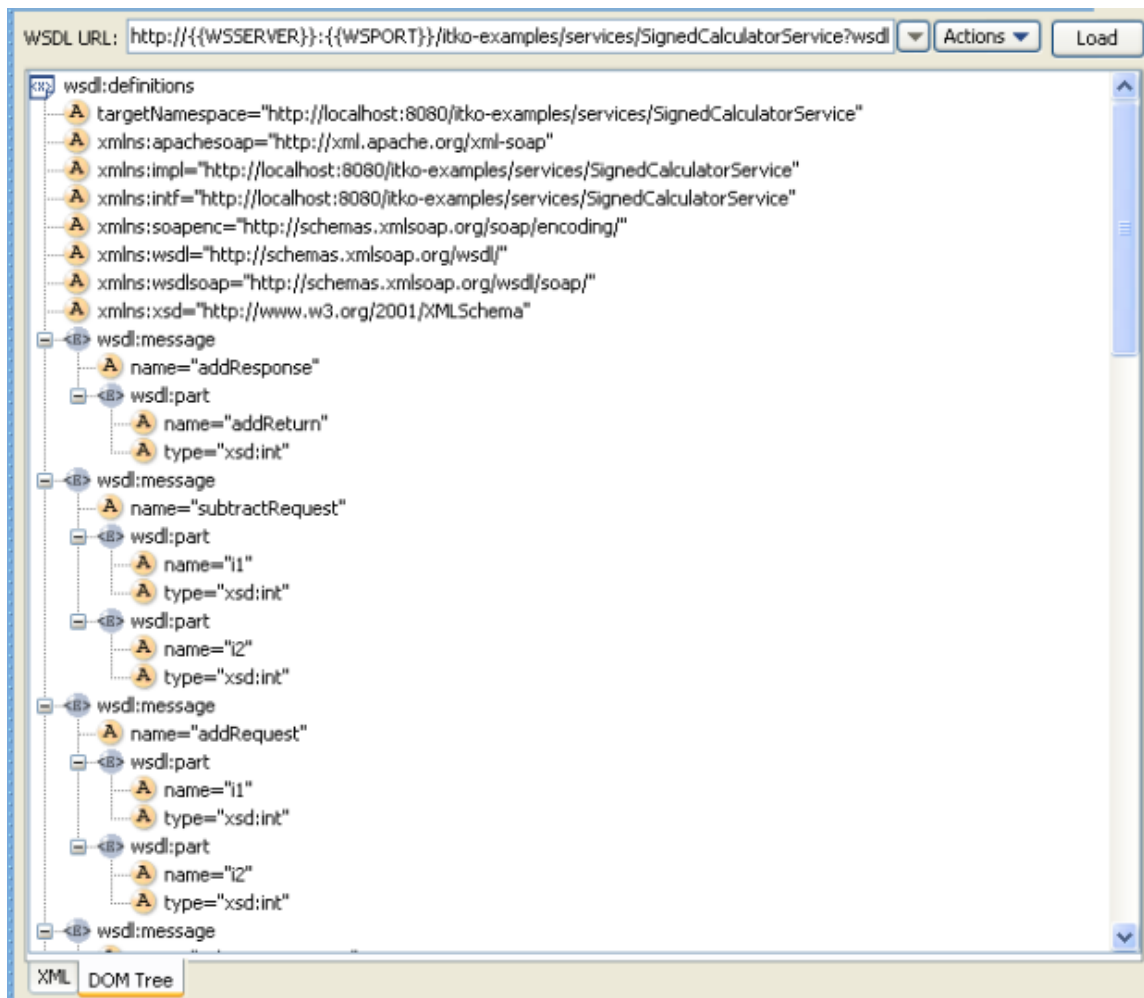
- XML Diff Assertion: Checks that the WSDL hasn't been changed by comparing it to a control copy of the original WSDL.
- XML Validator: Checks for valid XML using Schema or DTD.
- WS-I Basic Profile 1.1 Assertion: Checks for compliance with the WS-I Basic Profile.

These assertions are described in [PART 4 - Assertions](#).

Prerequisites - Familiarity with the 3 assertions named above.

Parameter Requirements - Location of WSDL you want to validate.

To configure the WSDL Validation Step:



The WSDL appears in the editor. You can view it in XML or DOM View.

Enter the following parameters:

- **WSDL URL:** Enter, or select from drop-down menu, the location of the WSDL. You can also click the Actions pull-down to browse the file system, or search a UDDI server.

Click the **Load** button.

You are now ready to add the assertions.

1.5 Raw SOAP Request

1.5 Raw SOAP Request

The Raw SOAP Request step allows you to test a web service by sending a Raw SOAP request (raw xml). It can be used to test legacy SOAP calls or web services that do not have a WSDL. It also allows you to test a web service's reaction to data of an incorrect type (e.g. sending a string when it expects a number) which is not allowed in the Web Service Execution step. Another use for the Raw SOAP request is to reduce LISA overhead during particularly intense load tests. The regular web service step has some additional overhead because it marshals an object into XML to make the request, and unmarshals the SOAP XML response back into an object. The raw soap step avoids this overhead and only deal with the raw SOAP XML. It has less work to do so it executes faster.

The SOAP request can be typed or pasted into the editor, or read from a file, and then parameterized using LISA properties.

There is no support for dynamic WS-Addressing or WS-Security headers. If you want to have these types of headers, they must be statically entered as part of the SOAP request entered in the input area. If your request does contain items like a WS-Security signature token, the signed elements can not be parameterized or the signature will no longer be valid.

Note: This step is not limited to SOAP calls, you can do XML or text POSTs also.

Parameter Requirements - The SOAP request text.

To create a Raw SOAP Request:

The screenshot shows the SOAP Client interface with the following fields and content:

- SOAP Server URL:** `http://{{WSSERVER}}:{{WSPORT}}/itko-examples/services/SignedCalculatorService?wsdl`
- SOAPAction:** (Empty field)
- If environment error:** `Abort the Test`
- Content Type:** ☒ `text/xml; charset=UTF-8` ☐ `application/soap+xml; charset=UTF-8`
- Advanced...** button
- Response** tab (selected)
- SOAP Request XML:**

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
    <LISA:LISAINT2 xmlns:LISA="http://www.itko.com/lisa/lisaint" soapenv:mustUnderstand="1">
      <LISAINT2 forceGC="false" failOnLoggedException="true" parent="false"/>
    </LISA:LISAINT2>
  </soapenv:Header>
  <soapenv:Body>
    <ns1:deleteUser xmlns:ns1="http://webservices.examples.itko.com" soapenv:mustUnderstand="1">
      <user xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">login</user>
    </ns1:deleteUser>
  </soapenv:Body>
</soapenv:Envelope>
```
- Read Request From File...** button
- Test...** button

Enter the following parameters:

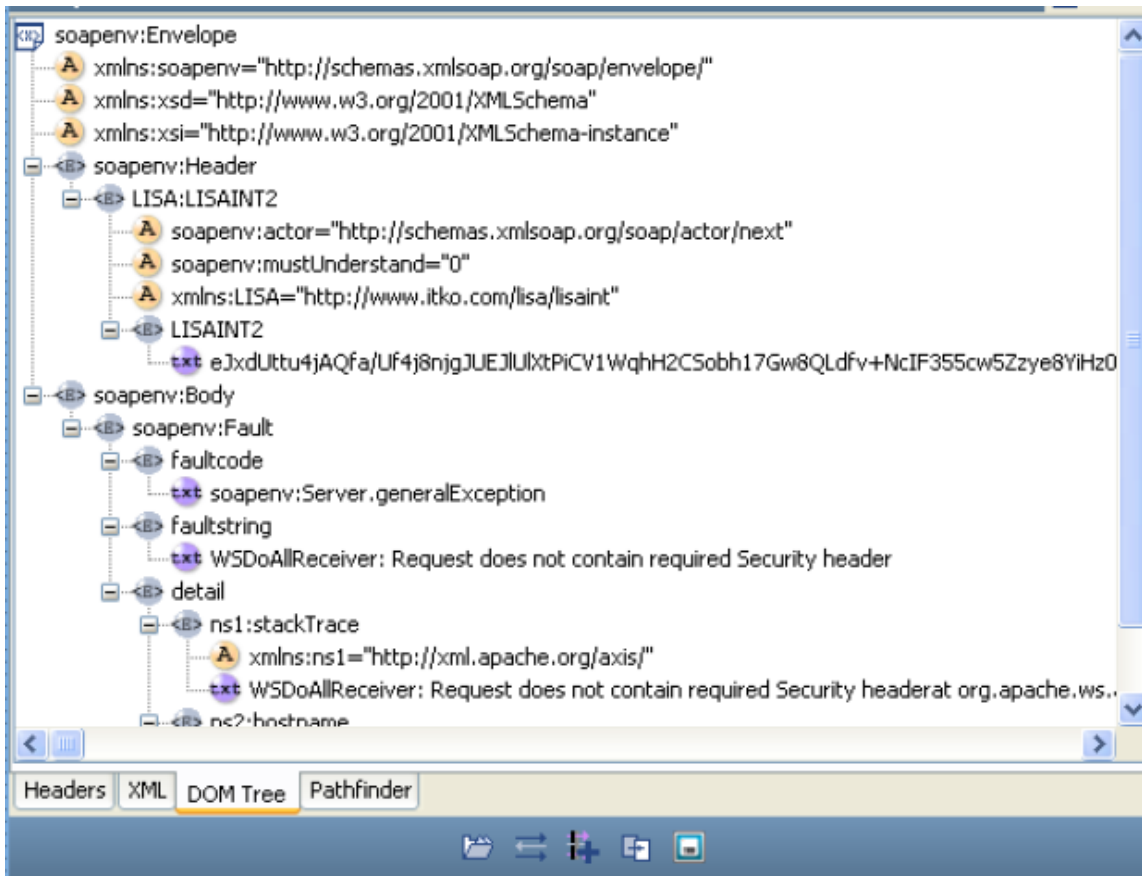
- **SOAP Server URL:** Enter the URL of the Web Service endpoint.
- **SOAP Action:** Fill in the SOAP action as indicated in `<soap: operation>` tag in the WSDL for the method being called. This is required for SOAP 1.1 and usually required to be left blank for SOAP 1.2.
- **Content Type:** Select the Content Type. Use `text/html` for SOAP 1.1, `application/soap+xml` for SOAP 1.2.
- **Advanced button:** Click to add any custom HTTP headers you want to send.

Type or paste the SOAP Request into the editor, or Click the **'Read Request From File'** button and browse to the file containing the SOAP Request.

Now you can parameterize the request, if desired, with properties. In the figure above we are using `login` as the value of the login.

Click the **Test** button to execute the call.

The response can be examined by clicking the **Results** tab:



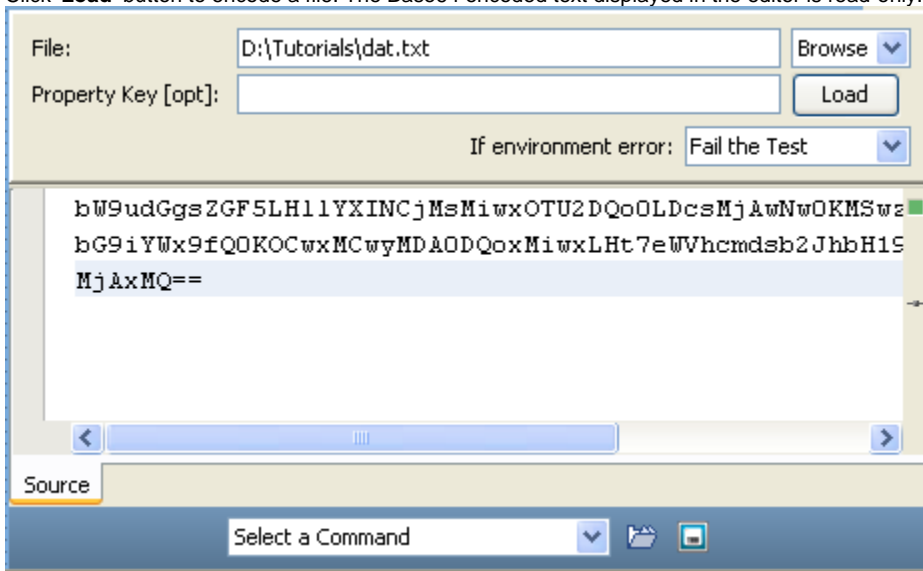
You are now ready to add Filters and Assertions.

1.6 Base64 Encoder



1.6 Base64 Encoder

This step is used to encode a file using the Base-64 encoding algorithm. The result can be stored into a LISA property for use elsewhere in the test case.

The Base64 Encoder step accepts a file as input and encodes the file using Base64 encoding. You can store the encoded file in a LISA property. Click **'Load'** button to encode a file. The Base64 encoded text displayed in the editor is read-only.



Component descriptions

Field	Description
File	Enter the full path and path name, or browse to the file to be encoded.
Property Key [opt]	Optional. The name of the property to store the encoded file.
Load	Click to load and test the encoding of the file. Optionally, store it in the specified property.
Editor	Read only.
Select a Command	After the file is encoded, you can add filters and assertions. The valid options are: <ul style="list-style-type: none"> • Random Selection Filter • Parse Value Filter • XML Xpath Filter • Create HTML Table ResultSet Filter • Make Assert on Selection
Load 	Click to load the file.
Save 	Save the contents of the editor to a new file.

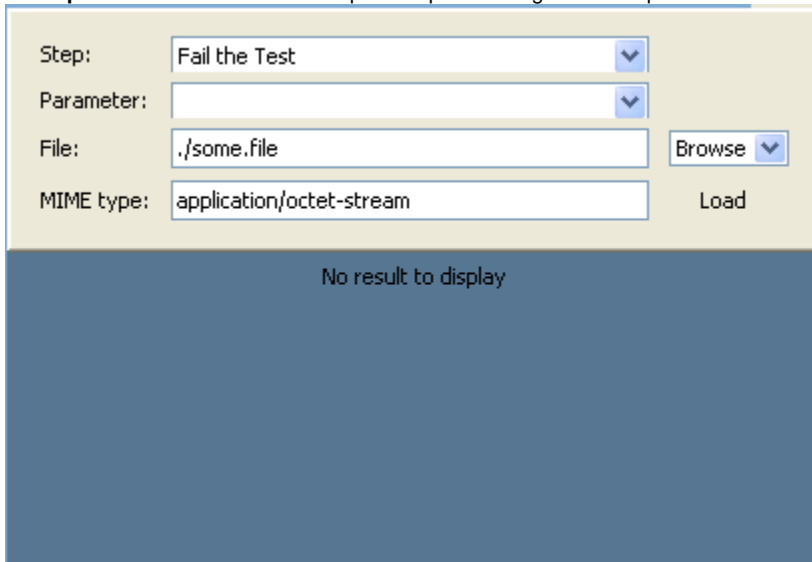
1.7 Multipart MIME (Multipurpose Internet Mail Extensions) Step

1.7 Multipart MIME (Multipurpose Internet Mail Extensions) Step

The Multipart MIME step allows data to be loaded from a file, encoded, and stored in property to be used as a post parameter on an HTTP request. The encoded document will be stored in the LISA property that has been defined previously in an HTTP/HTML Request step.

When LISA records a multipart mime form submit request, it records the contents of the file that was uploaded. Subsequent playback will result in the same content being submitted with "file upload" portion of the form again. The multipart mime step can be used to change what file is uploaded when the test case is played back.

Prerequisite - The HTTP/HTML Request step containing the HTTP parameter must already exist, and it must be before the Multipart MIME step.



Enter the following parameters:

- **Step:** Select the name of the HTTP/HTML Request step, or choose from the pull down menu, the step which will receive the property containing the encoded document.
- **Parameter:** Select the name of the property, chosen from the pull down menu, which is listed in the step named in the Step field above.
- **File:** Enter the pathname or browse to the document to be encoded.
- **MIME Type:** Enter the MIME type expected by the server.

Click **Load** to encode the file.

The screenshot shows a configuration window for a web service step. The 'Step' dropdown is set to 'delete user'. The 'Parameter' dropdown is set to 'name'. The 'File' field contains 'D:\Tutorials\dates.txt' with a 'Browse' button next to it. The 'MIME type' field is set to 'application/octet-stream' with a 'Load' button next to it. Below these fields is a large text area containing a base64-encoded string: 'bW9udGgsZGF5LH11YXINCjMsMiwxOTU2DQoOLDcsMjAwNwOKMSwzLD bG9iYWx9fQOKOCwxMCwyMDA0DQoxMiwxLHt7eWVhcmdsb2JhbH19DQ MjAxMQ=='. A scrollbar is visible below the text area. At the bottom, there is a 'Source' tab and a 'Select a Command' dropdown with icons for file operations.

You can save the contents of the editor to a file using the Save, , icon.

In the example in the figure above, the **httpTest** step has a post parameter **myImage** that contains the encoded version of the file **image.gif**.

1.8 SAML Assertion Query

1.8 SAML Assertion Query

The SAML Assertion Query step allows you to obtain a SAML Assertion from an Identity Provider for later use in a Web Service Execution step that uses a WS-Security SAML 1.x Assertion Token.

Prerequisites – A cursory understanding of what type of SAML Assertion Query you need to perform. This information can be obtained from either the developer of the system that utilizes SAML Assertions as the form of identity security or from the Identity Provider administrator.

Parameter Requirements – At a minimum you will need to know the URL to the Identity Providers SAML Query interface (Endpoint) and the Subject information (who/what you want to obtain a SAML Assertion for), what type of query you want to perform, and some extra information depending on which type of query.

The SAML Assertion Query Editor is shown below:

Connection

Endpoint:

SSL Keystore:

SSL Keystore Password:

SAML Version: ☐ 1.0 ☒ 1.1

Subject

Name:

Name Qualifier:

Format:

Confirmation Methods: ☐ Holder of Key ☐ Sender Vouches ☐ Bearer ☐ Artifact

Response (deprecated)

Respond With	Local Part	Namespace

Query

Type: ☒ Attribute ☐ Authorization ☐ Authorization Decision

Resource:

Attribute Designators	Name	Namespace

Editor | Last Request | Raw Query Result | Last Response

At the bottom you can see a set of four tabs. The 'Editor' tab allows you to configure the query information. After doing so you can test the query using the Test button in the Query section of the Editor. After testing the query you can view the raw request that was sent in the Last Request tab. You can view the raw SOAP response in the Raw Query Result tab (e.g. if it returned more than one assertion you can see that here). And you can view the step response in the Last Response tab. This shows, for example, what will be used in the Web Service WS-Security token.

Connection

This information describes where the SAML Query API server lives and how you want to connect to it.

- **Endpoint:** The URL to the SAML Query API of the Identity Provider.
- **SSL Keystore:** If you need to use client side identification certificates to connect to the Endpoint you can select the keystore file using the Select... button, or you can select a previous item from the pull-down list or enter one manually.
- **SSL Keystore Password:** The password for the SSL Keystore if used.
- **SAML Version:** The SAML version you want to use to query the Identity Provider.

Subject

This information describes who or what you want to request a SAML Assertion for. This could be a user or user group or other entity for which you want to provide an assertion about the subject's current authorization/privileges

- **Name:** The name of the entity (e.g. username).
- **Name Qualifier:** A group or categorization used to qualify the Name (e.g. domain).
- **Format:** This describes what format the name is being sent (e.g. Full Name vs. Username).
- **Confirmation Methods:** Select which confirmation method types you want to include in the query. The query will only return assertions that contain at least one of the specified types. If you leave all types unchecked it will return any assertion no matter what the confirmation method is.

Response (deprecated)

This information describes which assertion statements you want to be returned as part of the SAML Assertion. It has been deprecated as of SAML 1.1.

- **Local Part:** The element name (e.g. AuthenticationStatement, AuthorizationDecisionStatement, and AttributeStatement).
- **Namespace:** The element namespace (e.g. urn:oasis:names:tc:SAML:1.0:assertion).

Use the plus icon button to add more xml elements to the set to be returned. Use the trash icon button to remove any elements you have already added.

Query

A description of which type of query you want to perform. There are three different query types:

- Attribute
- Authorization
- Authorization Decision

Attribute

An attribute query responds with a set of Attribute Statements. For example it may tell you which groups a Subject is a member of.

Query					
Type	<input checked="" type="radio"/> Attribute <input type="radio"/> Authorization <input type="radio"/> Authorization Decision				
Resource	<input type="text"/>				
Attribute Designators	<table border="1"><thead><tr><th>Name</th><th>Namespace</th></tr></thead><tbody><tr><td><input type="text"/></td><td><input type="text"/></td></tr></tbody></table>	Name	Namespace	<input type="text"/>	<input type="text"/>
Name	Namespace				
<input type="text"/>	<input type="text"/>				

- **Resource:** If you want to limit your query to a particular resource (for example, a particular web service, domain, file) you can specify the resource name.
- **Attribute Designators:** Each attribute is identified with a name and namespace (like XML elements). You can filter the set of attribute statements returned by specifying each attribute type you want to be returned. (for example, Name = urn:mace:dir:attribute-def:eduPersonScopedAffiliation, Namespace = urn:mace:shibboleth:1.0:attributeNamespace:uri).

Authorization – Used to request authentication statements related to a specific Subject SAML Assertions

Query	
Type	<input type="radio"/> Attribute <input checked="" type="radio"/> Authorization <input type="radio"/> Authorization Decision
Authentication Method	<input type="text"/>

- **Authorization Method:** If you want to limit your query to return Authorization Statements that are for a particular method of authorization (for example, urn:oasis:names:tc:SAML:1.0:am:X509-PKI, urn:oasis:names:tc:SAML:1.0:am:PGP, urn:oasis:names:tc:SAML:1.0:am:password). A set of predefined authorization methods are available from the pull-down list.

Authorization Decision

Used to request SAML Assertions for particular actions that a subject wants to perform given the evidence.

Query					
Type	<input type="radio"/> Attribute <input type="radio"/> Authorization <input checked="" type="radio"/> Authorization Decision				
Resource	<input type="text"/>				
Actions	<table border="1"><thead><tr><th>Data</th><th>Namespace</th></tr></thead><tbody><tr><td><input type="text"/></td><td><input type="text"/></td></tr></tbody></table>	Data	Namespace	<input type="text"/>	<input type="text"/>
Data	Namespace				
<input type="text"/>	<input type="text"/>				
Evidence (Assertions)	<table border="1"><tr><td>Lisa Property</td></tr><tr><td><input type="text"/></td></tr></table>	Lisa Property	<input type="text"/>		
Lisa Property					
<input type="text"/>					
Evidence (Reference IDs)	<input type="text"/>				

- **Resource:** If you want to limit your query to a particular resource (for example, a particular web service, domain, file) you can specify the resource name.
- **Actions:** You must specify at least one action for which you want to request authorization to perform (for example, login, view, edit) specified with a name (Data) and Namespace (like an XML element).
- **Evidence (Assertions):** Optionally specify one or more SAML Assertion to include with the Authorization Decision Query as advice to the Identity Provider. Specify the Lisa Property which holds the SAML Assertion XML. You can use the for **lisa.<stepname>.rsp** to use the response from a previous step (like another SAML Assertion Query or Parse Text step).
- **Evidence (Reference IDs):** Optionally specify assertion reference ids.

1.9 Web Service Execution (Legacy)

1.9 Web Service Execution (Legacy)

The Web Service Execution step allows you to **construct a web service** (SOAP protocol) test using the information in a Web Service Definition Language file (WSDL).

Once you have identified the WSDL, LISA will construct a client with method calls for each operation specified in the WSDL. You use this web service client, and LISA's Object editor, to test the web service using your test data. Using additional information, not included in the WSDL, you can test complex secure web services easily. The additional parameters are entered using several configuration screens available at the end of the web service execution wizard. These are discussed later in this section.

Prerequisites - Knowledge of LISA's Complex Object Editor is required to manipulate the Web Service Client built by LISA.

For more details, refer to the Chapter "Using the Complex Object Editor" in the [LISA User Guide](#).

Parameter Requirements - At a minimum you must know the location of the WSDL (either from a local file or via the web). For complex web services, and secure web services there are several sets of parameters that you will need to gather before constructing your web service test step. These parameters are known to the web service developer, and are the same parameters required to use the web service in an application. These parameter sets will be specified when each advanced feature is discussed later in this section.

We will start by explaining how to test a simple web service, requiring only a WSDL. Then we will look at the advanced features that you may need to use. Finally we will look at the most complex of the advanced features, configuring WS-Security for a secure web service.

Part A – Testing a Simple Web Service
Part B – Advanced Features
Part C – WS Security

1.9.1 Part A - Testing a Simple Web Service

1.9.1 Part A - Testing a Simple Web Service

On choosing the Web Service Execution step you will be presented with the WSDL selection screen. You will be in the **Editor** tab (see the tab set at the bottom of the following figure).

Web Service Step Wizard WSDL Selection

First we need the URL to the WSDL document for this Web Service.

☐ Load from an existing Web Service Definition...

Web Service Name	WSDL URL
dddd	http://examples.itko.com:80/itko-e...
EJB3UserControlService	http://localhost:8080/itkoExamples...
EncryptedCalculatorService	http://localhost:8080/itko-example...

☒ Create a new Web Service Definition...

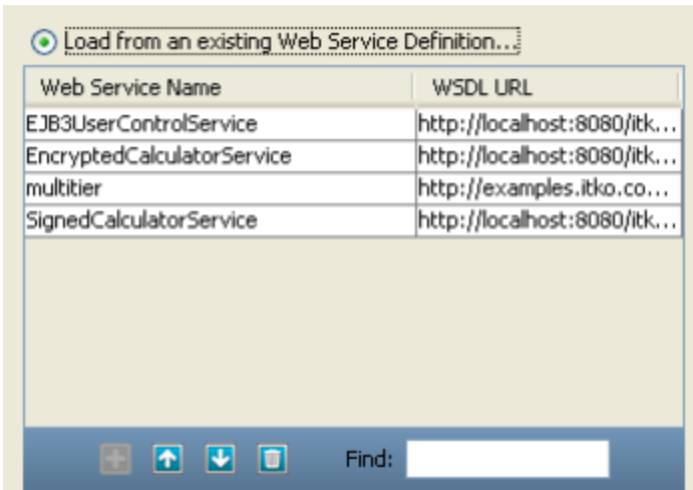
Web Service Name:

WSDL URL:

Editor | Navigator | Last Request/Response | WS-I Report

There are two sections on this screen selectable by the radio button.

The upper section allows you to **choose a WSDL** that LISA already knows about, and has already constructed a web service client that is available to you.



To choose one of these web services:

- Select **Load from an existing Web Service definition** (optional – the radio button is automatically selected if you select a Web Service name from the list).
- Click the Web Service name in the key column.

Click the **Next** button.

The lower section allows you to enter a new WSDL. LISA will build a client corresponding to the contents of this WSDL.

- Select **Create a new Web Service Definition** (optional – the radio button will automatically be selected if you begin entering information in the Web Service Name or WSDL URL fields)
- **Web Service Name:** Enter the name for the new service. This name must be unique. If a group of people plan on sharing test cases it is recommended to use a naming scheme that will ensure unique names for unique web services.
- **WSDL URL:** Enter the URL of the new WSDL, or select it from the pull-down list. You can also click the **Actions** list to browse the file system or search a UDDI server.
- **Advanced:** Opens a window where you can manage your namespaces.

Note: Namespace mapping is described in the next section (Advanced Features).

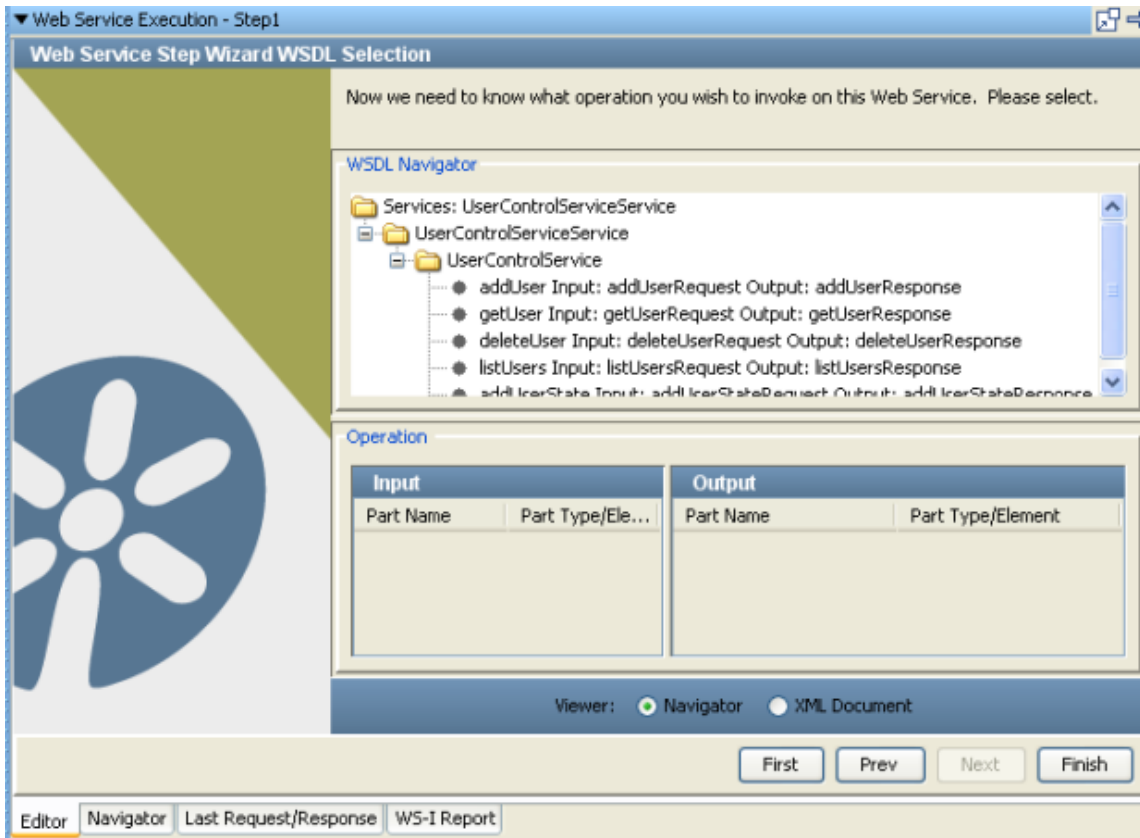
We will use an example that is available on the Demo Server. The WSDL location is:
 {+}http://examples.itko.com:80/itko-examples/services/UserControlService?wsdl+, or
 {+}http://localhost:8080/itko-examples/services/UserControlService?wsdl+

Depending on the Demo Server you are using. We have chosen the latter in the figure above, and we have called the web service "**NewWSDL**".

Note: There should be no space in the Web Service name, else it will give an error.

Click the **Next** button.

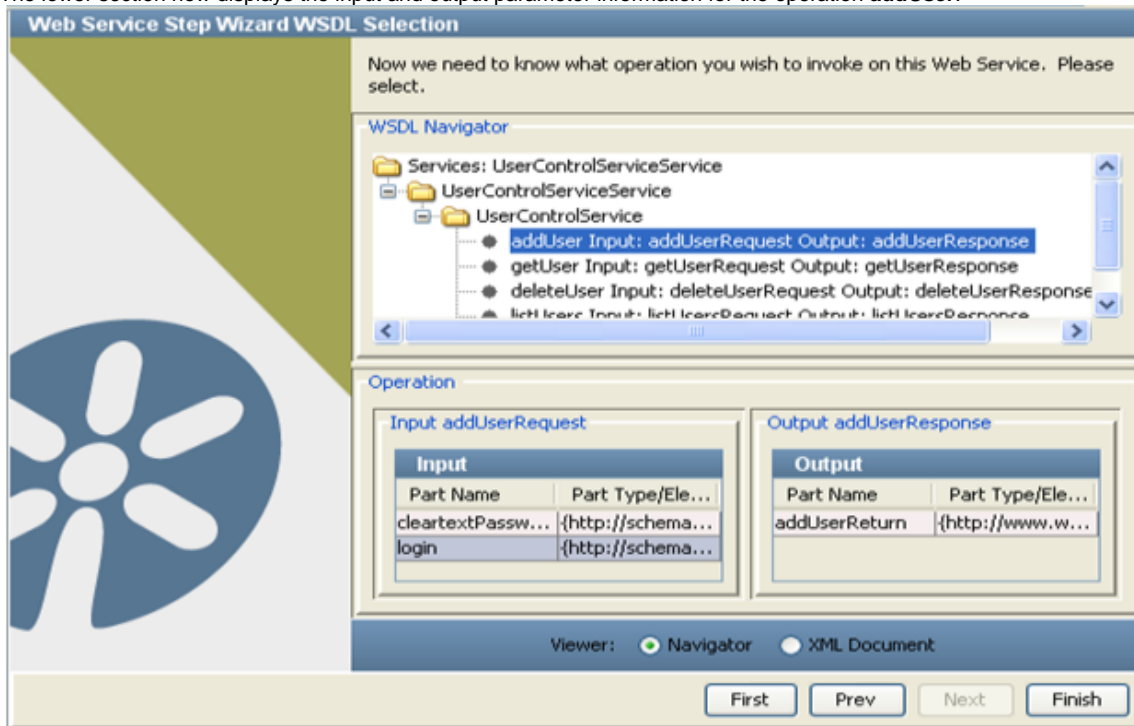
You will now see the WSDL Navigator screen:



The upper section of the Navigator, titled "**WSDL navigator**" shows a hierarchy of folders, the last of which is the web service (User Control Service in our case), listing all the operations available.

- Select an operation to invoke on the Web Service: We have chosen "**addUser**"

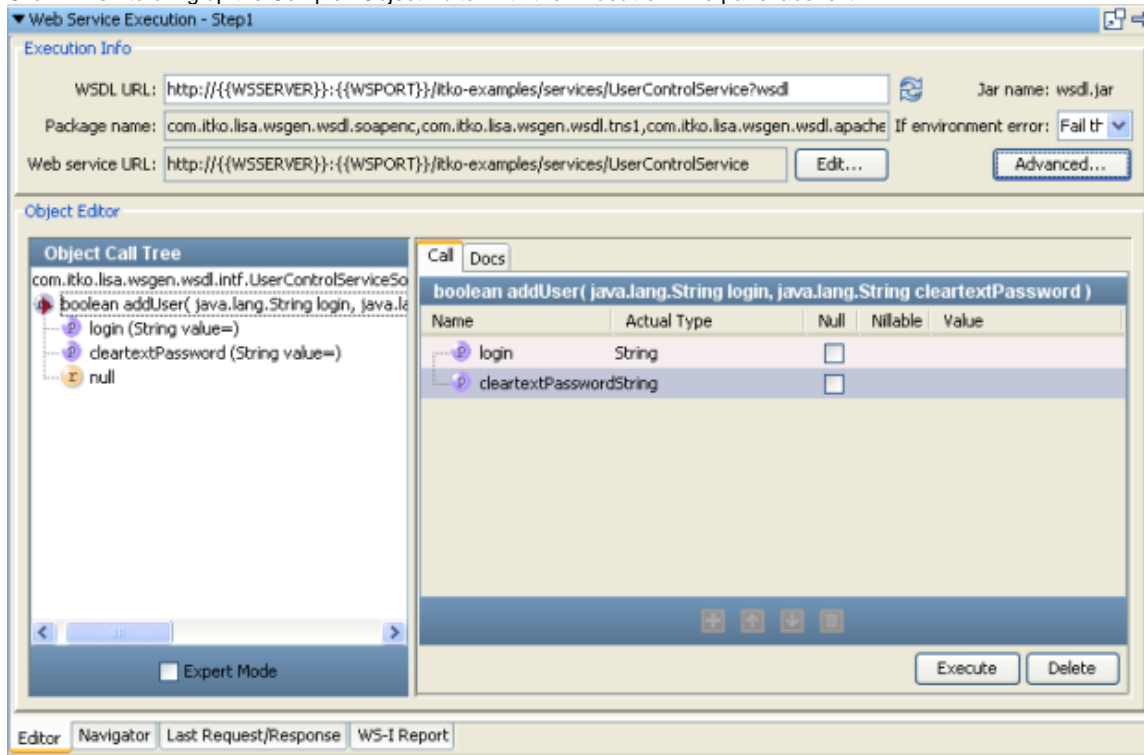
The lower section now displays the input and output parameter information for the operation **addUser**.



Note: Part Name and Part Types are standard Web Service notation for the input and output parameters. These are informational only, the only selection that needs to be made on this screen is web service operation to execute in the WSDL Navigator.

Select the **Viewer** from Navigator or XML Document option. By default it is the Navigator option.

Click **Finish** to bring up the Complex Object Editor with the **Execution Info** panel above it.



The Execution Info panel comes populated with all the necessary fields like WSDL URL, Package name, Web Service URL and the Jar name. Other fields are:

- **If environment error:** Select the next step to execute, normally fail, if an environmental error is returned by the web service. Most SOAP faults will not trigger this workflow change.
- **Edit:** Click the edit button if you wish to change the web service endpoint URL.

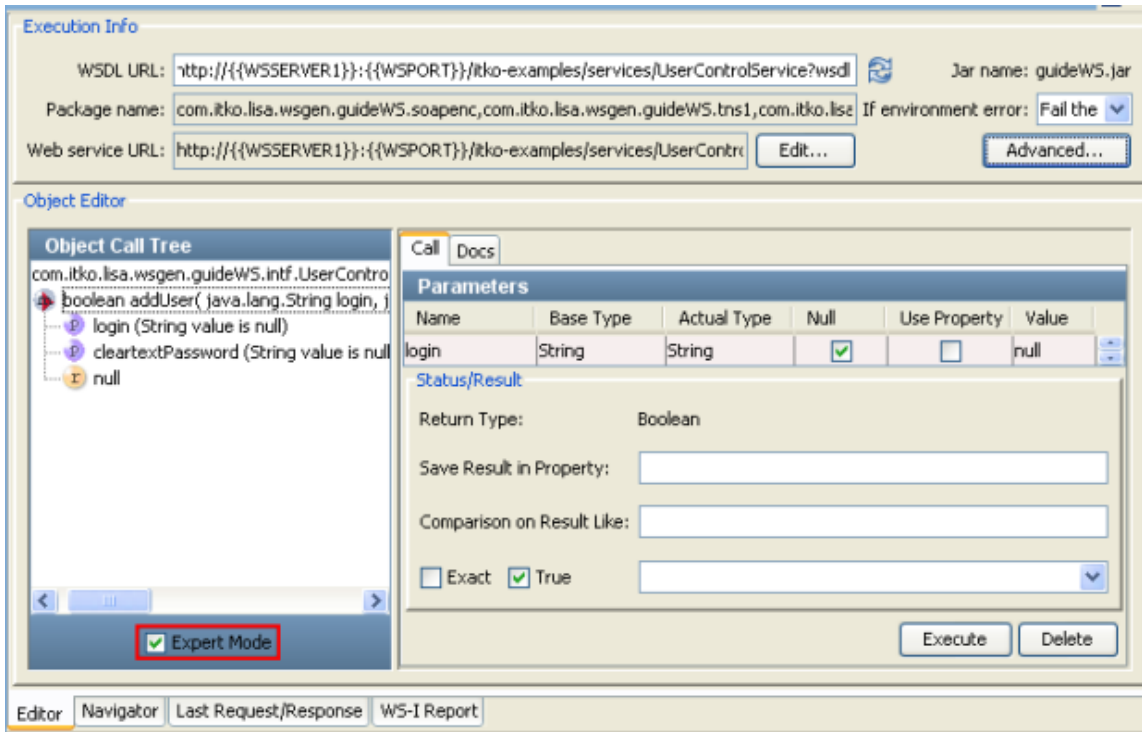
Note: LISA automatically parameterizes the host (WSSERVER) and port (WSPORT) for the WSDL and Web Service URL.

- **Advanced:** Opens a window where you can configure many advanced features associated with your web service.

Note: The advanced features are described in the next section ([Advanced Features](#)). The WS-Security options are discussed in the section after that ([Using WS-Security](#)).

The Object Editor panel uses the standard Object Editor to manipulate the web service client.

The previous figure shows the object editor in the standard mode. Clicking the Expert Mode checkbox reveals the expert mode view:



In Expert Mode you can configure inline filters and assertions. To use an inline filter, enter a property name in the Save Result in Property text box. To add an assertion, configure it using the 'Comparison on Result Like' text box, and the 'Exact' and 'True' checkboxes.

In either mode you can supply the values for each parameter listed. In our example, the **addUser** method takes two parameters, **login** and **cleartextPassword**. These parameter values can be static values, LISA properties, or **NULL**. Enter a static value by filling in the **Value** cell for the parameter. Enter a LISA property by checking the **Use Property** checkbox and filling in the **Value** cell with the property name. Enter **NULL** by checking the **Null** checkbox.

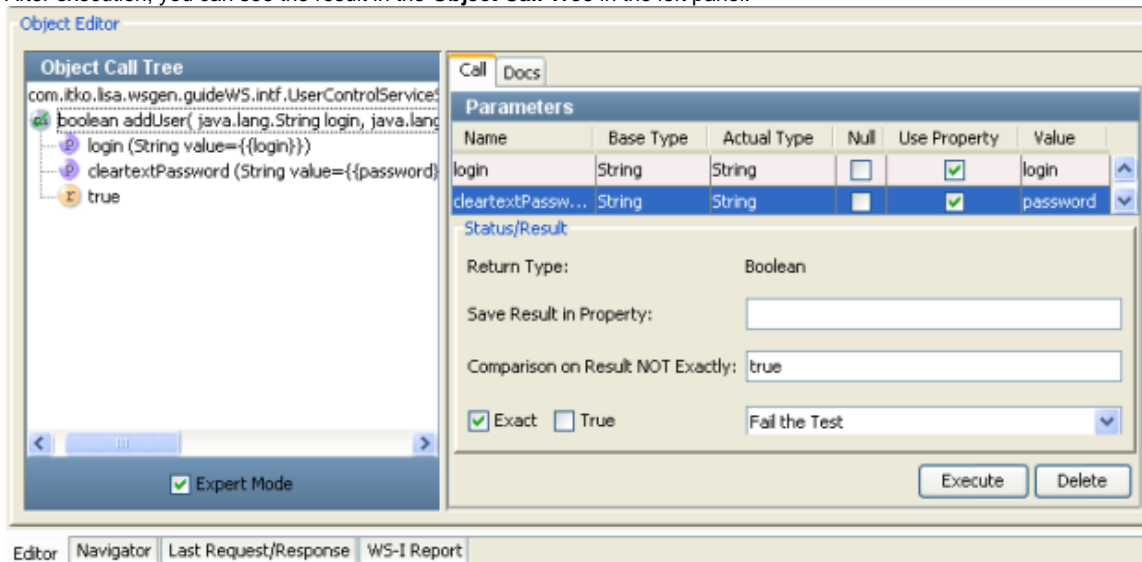
If the parameter type is a complex object, the **Value** cell shows **Ctrl-click to edit**. Press <Ctrl> and click the cell to open the Complex Object Editor can enter the fields of the complex object.

If the **Actual Type** cell displays the name of a Java interface instead of a Java class, you will have to change the Actual Type to be the name of a Java class. An error message is displayed if you try using a Java interface. The developers of the web service should be able to provide you with a class name that can be used in place of the interface name.

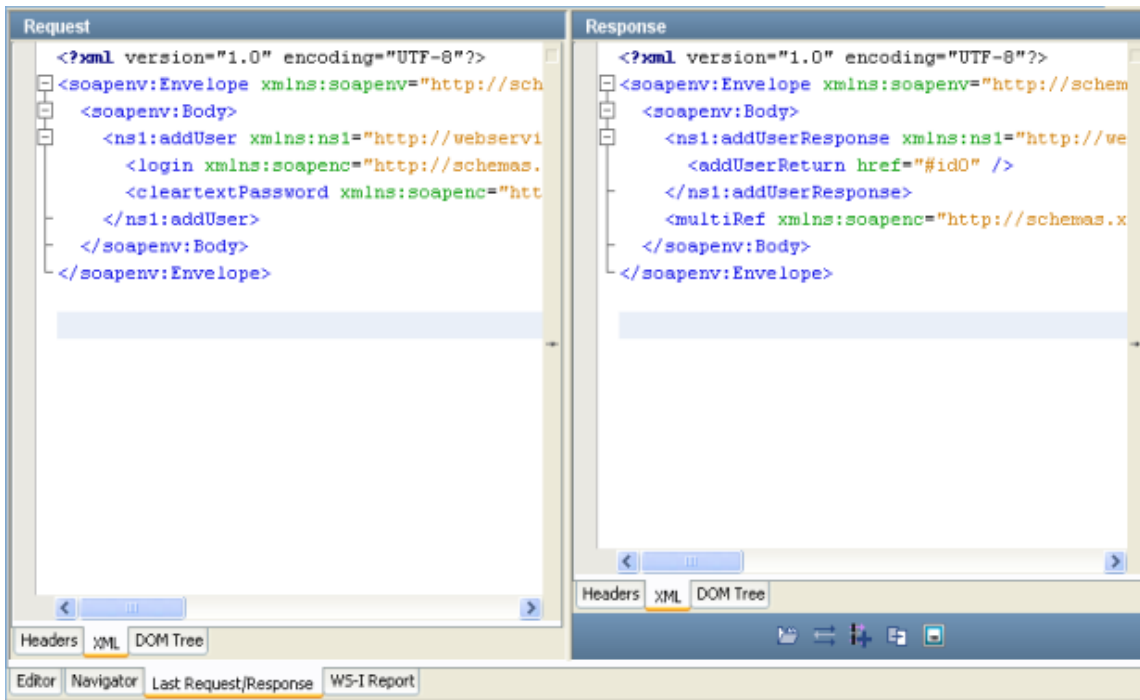
For detailed instructions on the use of the Object editor see the section: Using the Complex Object Editor in the [LISA User Guide](#).

Click **Execute**.

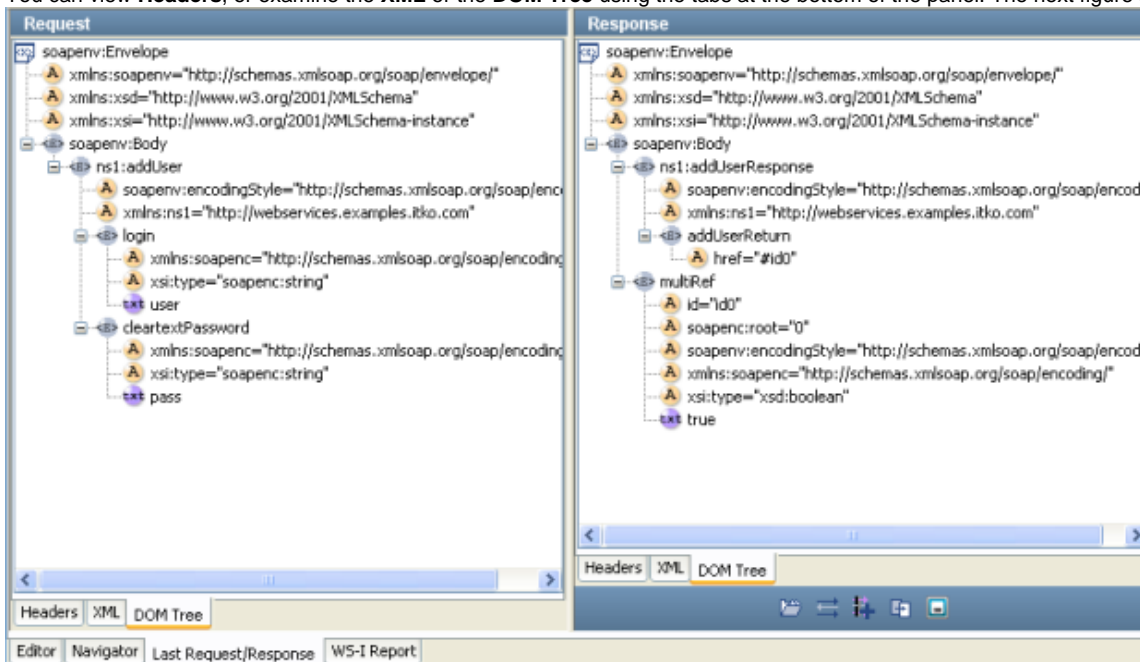
After execution, you can see the result in the **Object Call Tree** in the left panel:




To see the actual SOAP request/ SOAP response, click the **Last Request/Last Response** tab at the bottom of the panel:




You can view **Headers**, or examine the **XML** or the **DOM Tree** using the tabs at the bottom of the panel. The next figure shows the DOM view.



At this point, you are ready to add filters and assertions to the response.

Filters can be added using the "add filter"  icon, or by selecting the filter element for the current step.

Assertion can be added using the "add assertions"  icon, or by selecting the Assertions element for the current step.

Note: Selecting the Filters or Assertions elements will give you a greater choice.

This test step is now complete!

You can go back to the Object Editor, and select additional calls to other procedures in the web service, and execute them in this same step. This is useful if you need to add calls on request or response objects to "drill down" to values. The best practice for authoring test cases is to put each web service call in a separate step. It makes a test case easier to follow and easier to pinpoint which call is causing a problem if there are errors.

1.9.2 Part B - Advanced Features

1.9.2 Part B - Advanced Features

In this section we will describe the advanced features available to you in the Web Service Execution step. Following features are covered:

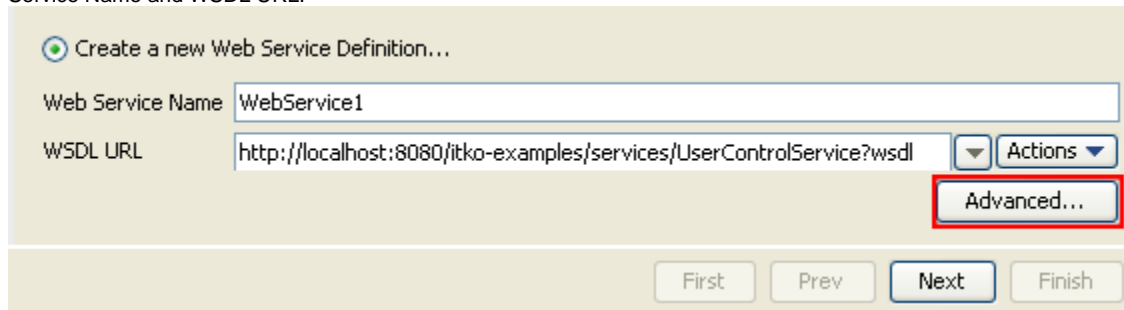
- Namespace mapping
- UDDI access point lookup
- Specific protocol requirement
- Maximum wait time
- Security in transport layer
- WSI basic profile 1.1 validation
- Advanced error handling
- Custom transport headers
- WS Addressing
- Customer SOAP Headers

a) Namespace Mapping

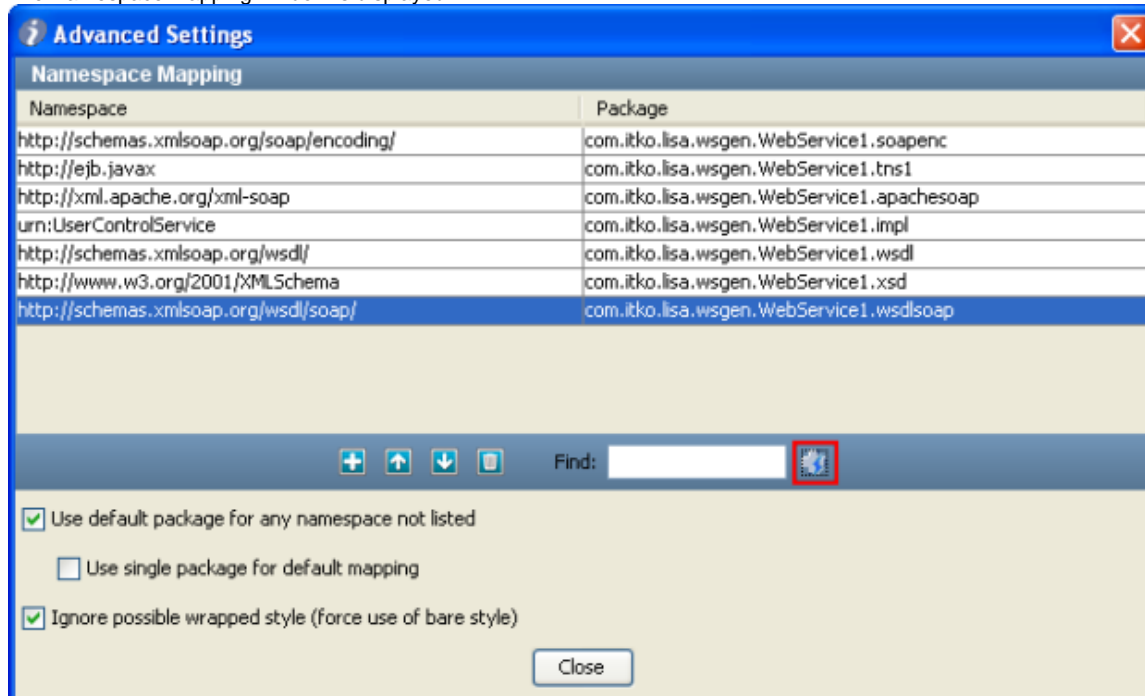
For a nontrivial WSDL with multiple namespaces, or for reuse of client side stubs, you may want to customize the namespace mappings. LISA uses this namespace mapping during the creation of its client classes that it uses to execute the web service call. Its default behavior is to separate out each type defined in the WSDL into a unique package based on its namespace. Use this dialog to change this default behavior. LISA starts with a default namespace of `com.itko.wsgen.<Web service name>`

For each namespace in the WSDL, LISA adds the namespace prefix (or a generated unique prefix if one's not available in the WSDL) to the default namespace to create a unique namespace:
`com.itko.wsgen.<web service name>.<namespace prefix>`

To customize the namespace mappings, select the Advanced... button on the WSDL Selection screen after you have entered your new Web Service Name and WSDL URL:



The Namespace Mapping window is displayed:



Namespace	Package
http://schemas.xmlsoap.org/soap/encoding/	com.itko.lisa.wsgen.WebService1.soapenc
http://ejb.javax	com.itko.lisa.wsgen.WebService1.tns1
http://xml.apache.org/xml-soap	com.itko.lisa.wsgen.WebService1.apachesoap
urn:UserControlService	com.itko.lisa.wsgen.WebService1.impl
http://schemas.xmlsoap.org/wsdl/	com.itko.lisa.wsgen.WebService1.wsdl
http://www.w3.org/2001/XMLSchema	com.itko.lisa.wsgen.WebService1.xsd
http://schemas.xmlsoap.org/wsdl/soap/	com.itko.lisa.wsgen.WebService1.wsdlsoap

Click the Auto Populate icon, to list all the WSDL namespaces and to see what the default LISA mappings will be. You can now add, remove, or edit the namespace mappings. At the bottom of the screen you can select the following options:

- Use default package for any namespace not listed
- Use single package for default mapping. This was the default behavior prior to LISA 3.6.
- Ignore possible wrapped style (force use of bare style) – for every operation in the WSDL, LISA tries to auto-detect if the WSDL has been defined in the wrapped-style and if so it unwraps the set of parameters that are to be sent. For example, if you have an operation defined in the wrapped-style and it takes an OperationRequest type parameter. If you have this option unchecked (default) it will expand the OperationRequest type and specify each of its sub-elements as parameters (e.g. instead of passing an OperationRequest, it would take its individual elements that make up an OperationRequest like param1, param2). If you check this option it will not attempt to detect the wrapped-style and will generate the operation with the single OperationRequest parameter type. Note: this can be useful if you plan on generating parameters using Excel DTOs.

Note: For the rest of the advanced features in this section you will need to click the **Advanced...** button on the Execution Info section of the Web Service Editor. Click the Editor tab at the bottom of the panel to show the Execution Info section:

The screenshot shows the 'Execution Info' section of the Web Service Editor. It contains three text input fields: 'WSDL URL' with the value 'http://{{WSSERVER1}}:{{WSPORT}}/itko-examples/services/UserControlService?wsdl', 'Package name' with the value 'com.itko.lisa.wsgen.WebService1.soapenc,com.itko.lisa.wsgen.WebService1.tns1,com.itko.lisa.wsgen.WebService1', and 'Web service URL' with the value 'http://{{WSSERVER1}}:{{WSPORT}}/itko-examples/services/UserControlService'. To the right of the WSDL URL field is a 'Jar name' label with the value 'WebService1.jar'. Below the Package name field is a label 'If environment error:' followed by a dropdown menu showing 'Fail the Test'. At the bottom right are two buttons: 'Edit...' and 'Advanced...'.

A pop-up window with five tabs is displayed. We will be referring to four of these tabs, General, Transport Headers, Addressing, and SOAP Headers in this section, the fifth, Security is the topic of the next section.

b) UDDI Access Point (Web Service URL) Lookup

Click the **General** tab, and then specify:

- Perform Access Point (Web Service URL) Lookup
- Select the Inquiry URL and Binding Template: use the Search UDDI Server find button to navigate to the correct Binding Template to perform the lookup. Note: when creating the step if you used the UDDI Search function when specifying the Web Service WSDL URL, these values will be automatically filled in. If the Inquiry URL is specified but the Binding Template is not, you may have performed a Model search. In order to locate the Binding Template you need to perform a search at a higher level of the hierarchy and drill down to the TModel thru a particular Binding Template.

c) Specific Protocol Requirements

Click the **General** tab, and then specify:

- HTTP Version: Select version 1.0 or 1.1
- SOAP Version: Select version 1.1 or 1.2

d) Maximum Wait Time

Click the **General** tab, and then specify:

- Call Timeout (ms): The maximum wait time for a response from a web service call (milliseconds). This is especially useful when using SOAP over JMS

e) Security in Transport Layer (SSL – Mutual Authentication)

Click the **General** tab, and then specify:

- SSL Keystore File: The name of the keystore file where the client identity certificate is stored. It can be in jks or pkcs format
- SSL Keystore Password: The password for the keystore

The screenshot shows the SSL configuration fields. The 'SSL Keystore File' field has a dropdown menu showing the path '{{LISA_HOME}}\examples\keystores\keystore.jks' and a 'Select...' button. The 'SSL Keystore Password' field is a text input with asterisks '*****' and a 'Verify...' button.

NOTE: you can specify global certificates properties for SSL in your local.properties file.

- For global certificates (web server, raw SOAP, and web service steps)
 - ssl.client.cert.path = a full path to the keystore
 - ssl.client.cert.pass = password for the keystore (this password will be automatically encrypted when Lisa runs)
 - ssl.client.key.pass = an optional password for the key entry if you are using the JKS keystore and the key has different password from the keystore (this password will be automatically encrypted when Lisa runs)
 - Note: this is currently not an available option to be set in the WS Test step and if required must be set in the local.properties file.
- For web service steps only certificates (not raw soap steps)
 - ws.ssl.client.cert.path = a full path to the keystore
 - ws.ssl.client.cert.pass = password for the keystore (this password will be automatically encrypted when Lisa runs)
 - ws.ssl.client.key.pass = an optional password for the key entry if you are using the JKS keystore and the key has different password from the keystore (this password will be automatically encrypted when Lisa runs)

Note: this is currently not an available option to be set in the WS Test step and if required must be set in the local.properties file.

Note: If you have values in local.properties and in the general tab, the values in the general tab will be used.

e) WS-I Basic Profile 1.1 Validation

Note: For information on WS-I Basic profile 1.1 see www.ws-i.org

Click the **General** tab, and then specify:

WS-I Basic Profile 1.1 Display Only Not Passed Assertions

☐ Validate WSDL

☐ Validate SOAP Message

If WS-I error: Step1

WS-I Basic Profile 1.1: You can choose four different validation levels in the pull down menu.

1. Display All Assertions
2. Display All But Info Assertions
3. Display Only Failed Assertions
4. Display Only Not Passed Assertions

- Validate WSDL: Click to validate the WSDL
- Validate SOAP Message: Click to validate the SOAP message
- If WS-I error: Select the step to redirect to on error

Note: Validation failures are common, but usually should not affect the outcome of the test. It is good practice to set the next step to continue so that you can complete the test.

When you click the **Execute** button on the Object Editor screen, the validation will be run and a report will be generated and saved (in the reports directory in the LISA install directory). You can view the report by pressing the **WS-I Report** tab at the bottom of the screen.

Note: The format of this report is standard, and is dictated by the Web-Services-Interoperability Organization (WS-I). A small example portion of the report is shown below:

WS-I Profile Conformance Report

Report: WS-I Basic Profile Conformance Report
Timestamp: 2007-08-25T17:45:00.559

Copyright (c) 2005-2004 by The Web Services-Interoperability Organization (WS-I) and Certain of its Members. All Rights Reserved.

Analyzer Tool Information

Version	1.0.0
Release Date	2005-07-04
Implementer Name	WS-I Organization
Location	http://www.ws-i.org

Analyzer Runtime Environment Information

Runtime Name	Java(TM) 2 Runtime Environment, Standard Edition
Runtime Version	1.5.0_05-b05
Operating System Name	Windows XP

f) Advanced Error Handling

There are times when a valid xml SOAP response is received, but an exception is thrown when trying to process your results. The following options should allow you to process, by dealing with the xml response directly rather than processing the SOAP response:

Advanced

☐ Allow null non-nillable elements

☐ Do not send request

☐ Do not deserialize response


Allow null non-nillable elements: If there are non-nillable elements in the SOAP request, send the request even if the elements are null.

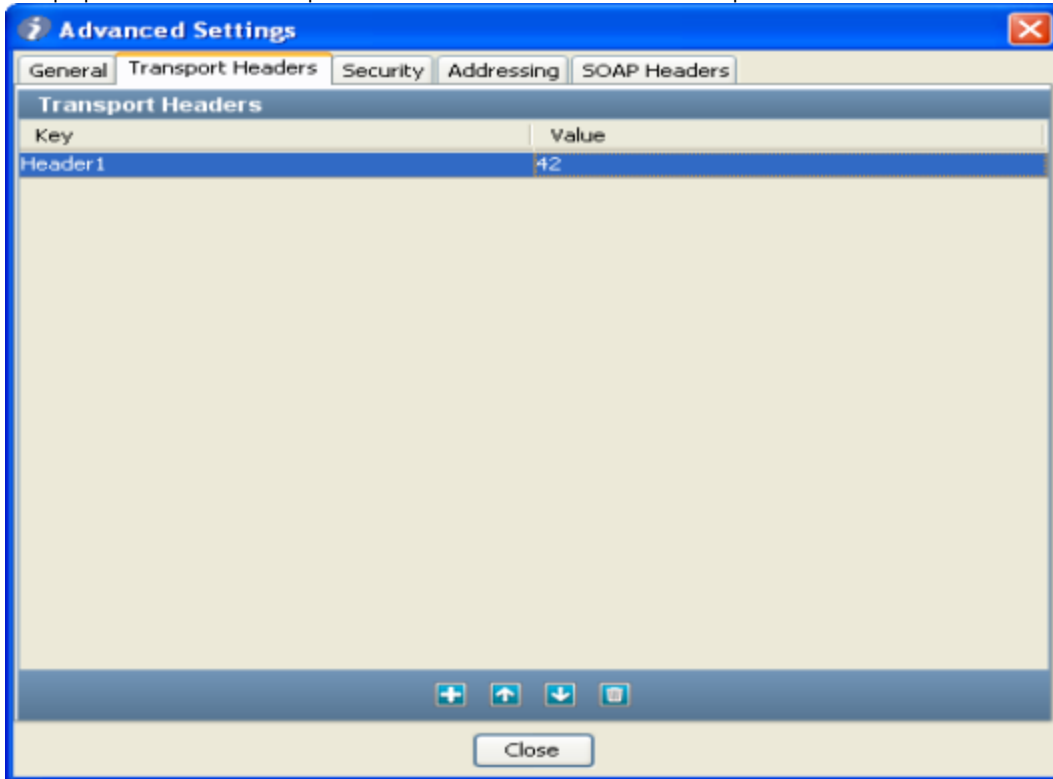
Do not send request: Generate the SOAP request message, but don't send message to Web Service Endpoint. The step response will be the SOAP request message.

Do not deserialize response: Send the SOAP request message and receive a SOAP response, but do not try to process the SOAP response message to generate a Java response object. The step response will not be affected, but the return value of the method invoked in the object editor will be null.

g) Custom Transport Headers

Click the **Transport Headers** tab, then:

Click the Add, , icon and enter the header as a key/value pair. Repeat to add additional headers. These will be sent as HTTP Headers or JMS properties based on which protocol is used to execute the web service operation.



h) WS-Addressing

You can send a WS-Addressing header with your request. The WSDL does not specify if WSAddressing information is required so you must configure it.

Click the Addressing tab, and then specify:

Advanced Settings

General | Transport Headers | Security | **Addressing** | SOAP Headers

☒ Use WS-Addressing

Version: ☒ Final (2005/08)
☐ Submission (2004/08)
☐ Draft (2004/03)
☐ Draft (2003/03)

	Default	Override	
To	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
From	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
Action	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
MessageId	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
ReplyTo	<input type="checkbox"/>	<input type="checkbox"/>	
FaultTo	<input type="checkbox"/>	<input type="checkbox"/>	


☐ Must Understand

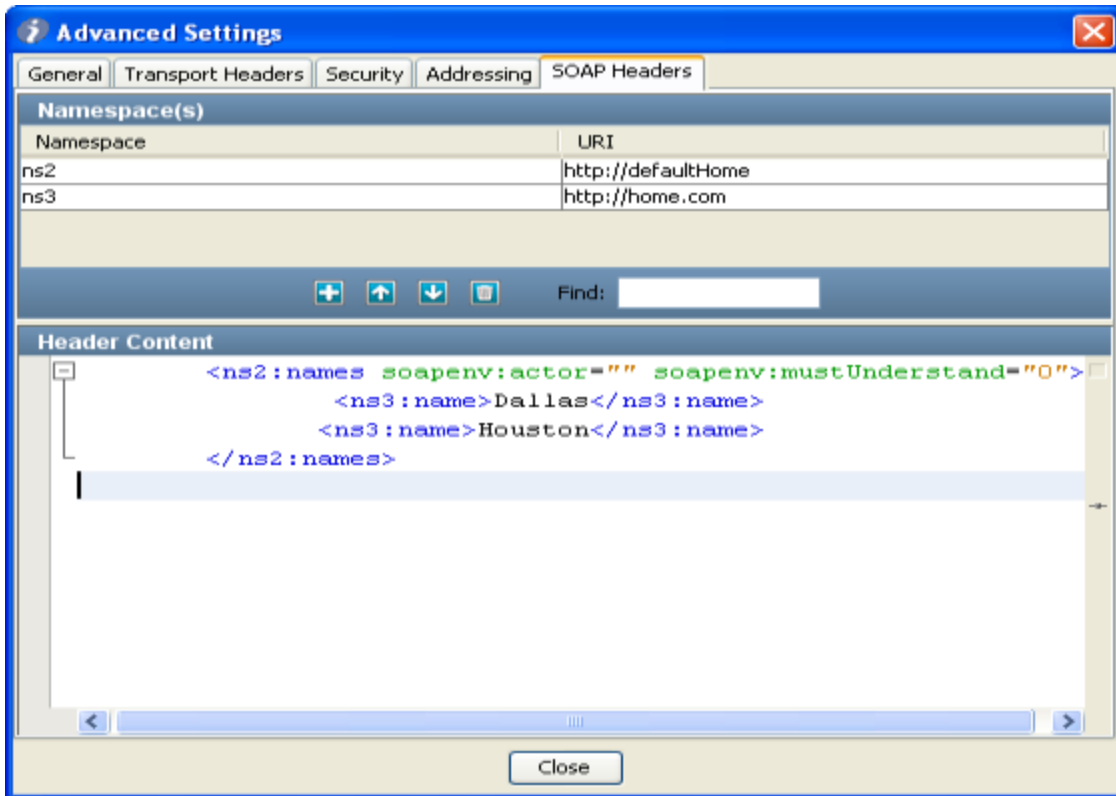
Close

- Use WS-Addressing: Click to use WS-Addressing
- Version: Select appropriate version. Several versions of the WS-Addressing specification are listed as options because some web services platforms (e.g. .Net) still use the older Draft specifications. You will need to determine which your web service platform is using.
- LISA will populate as many values as it can. Then you can choose to use the default value or override it. You can choose not to send some of the default elements by un-checking the default checkbox for that element.
- Click Must Understand checkbox if you want to assure that the web service can understand the WSAddressing header

i) Custom SOAP Headers

Click the **SOAP Headers** tab, then:

Click the Add, , icon in the top panel and enter any namespace prefix(es) and URI(s) used in the custom header content that you want to be defined in the soap envelope element tag. Then enter your custom header as a valid XML fragment as shown in the figure below.



The header will appear in the SOAP request as shown below:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns2="http://defaultHome/http://defaultHome"
xmlns:ns3="http://home.com">
  <soapenv:Header>
    <ns2:names soapenv:actor="" soapenv:mustUnderstand="0">
      <ns3:name>Dallas</ns3:name>
      <ns3:name>Houston</ns3:name>
    </ns2:names>
  </soapenv:Header>
```

1.9.3 Part C - WS-Security

1.9.3 Part C - WS-Security

Prerequisites – Before configuring a secure web service for testing, you should be familiar with the testing of simple web services as described earlier in this section, (Testing a Simple Web Service).

Parameter Requirements – when testing secured web services, there are several sets of parameters that you will need to gather before constructing your web service test step. These parameters are known to the web service developer, and are the same parameters that would be required to call the web service from an application. The parameter sets needed must be made available to you before you can configure your web service for testing. You will need to know which security options are being used by the web service, the configuration parameter values for the options used, and the order in which they are expected in the SOAP headers for the request and the response.

For illustration purposes, when we can, we will use the parameters needed for the security test case available on the Demo Server. This test case called `ws_security`, (located in the example directory), uses a timestamp, encryption and a signature token. The WSDL location is:

{+}http://localhost:8080/itko-examples/services/EncryptedCalculatorService?wsdl+, or
 {+}http://examples.itko.com:80/itko-examples/services/EncryptedCalculatorService?wsdl+ for encryption
 {+}http://localhost:8080/itko-examples/services/SignedCalculatorService?wsdl+, or
 {+}http://examples.itko.com:80/itko-examples/services/SignedCalculatorService?wsdl+ for the signature token, depending on the Demo Server you are using.

The web service names are 'EncryptedCalculatorService' and 'SignedCalculatorService'.


LISA has extensive support for WS-Security including:

Send (request)	Receive (response)
<ul style="list-style-type: none"> Encryption 	<ul style="list-style-type: none"> Decryption


<ul style="list-style-type: none"> • Signature Token 	<ul style="list-style-type: none"> • Signature Verification
<ul style="list-style-type: none"> • Timestamp 	<ul style="list-style-type: none"> • Timestamp Receipt
<ul style="list-style-type: none"> • Username Token 	<ul style="list-style-type: none"> • Username Token Verifier
<ul style="list-style-type: none"> • SAML Assertion Token 	<ul style="list-style-type: none"> • SAML Assertion Receipt
	<ul style="list-style-type: none"> • Signature Confirmation


To add security headers to the web service request:

Click the **Security** tab. Click **Send**.

- Check 'Must Understand' check box (if needed or if you want to ensure that the WS-Security header is processed by the server)
- Enter the Actor/Role name (if needed – most web services do not use multiple Actors/Roles)
- Click the Add,  icon and select the security action type to add. You will be presented with the configuration panel for that security action type. These are discussed below.

Adding security verification to the response is very similar:

Click the **Security** tab. Click the Add, , icon and select **Receive**.

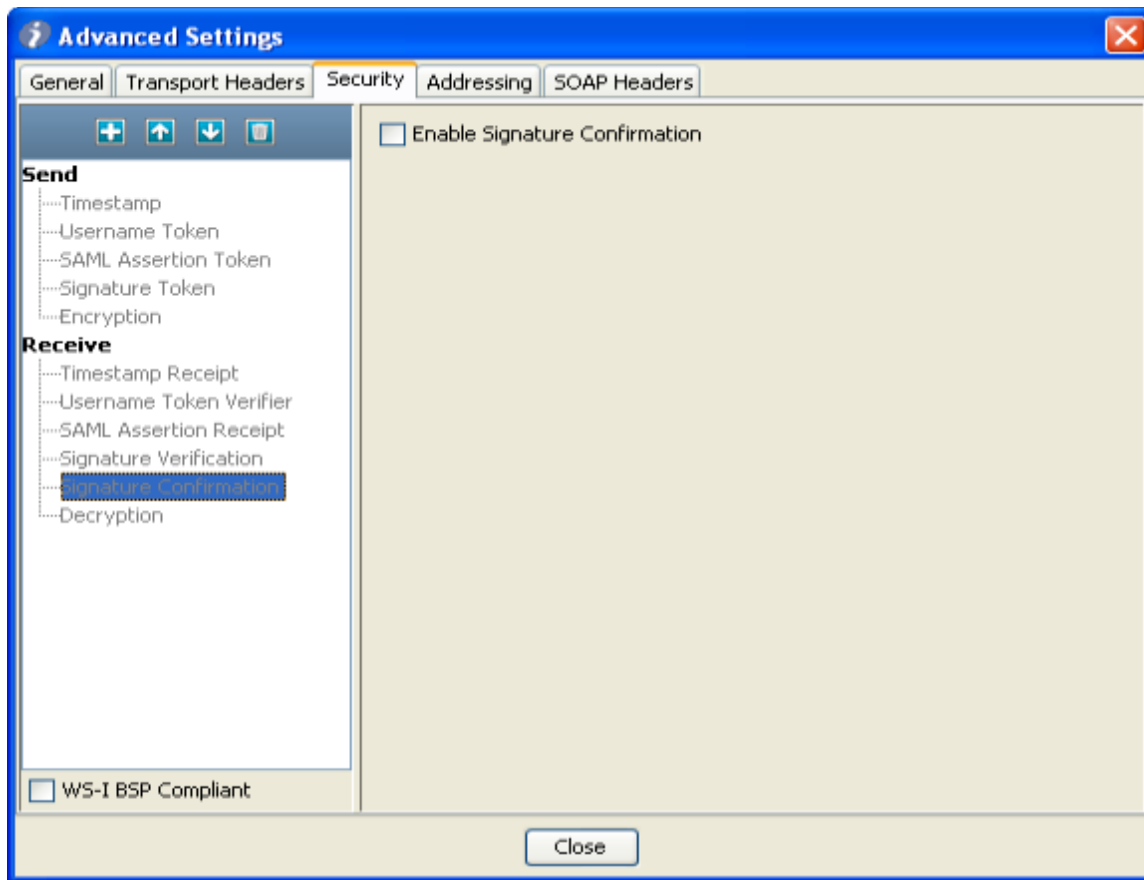
- Enter the Actor/Role name (if needed)
- Click the Add, , icon and select the security action type. You will be presented with the configuration panel for that security action type. These are discussed below.

Note: You can add as many security types as are needed to execute your Web Service.

When a keystore is being used in a security action type configuration (described below for each type), you can verify your keystore settings to make sure you are using the correct format, password, alias and alias password. There is a **Verify** button on the editors for Signature, Encryption/Decryption and SAML Assertion Token. Clicking the **Verify** button will invoke the Keystore Verifier, which will produce a verification report.

If you do not know the expected alias name for a WS-Security setting, you can use the Keystore Verifier to list all of the aliases in the keystore. Leave the Keystore Alias and Alias Password boxes empty and click the **Verify** button. The Keystore Verifier is described at the end of this section on WS-Security.

An example screen with several security types chosen is shown below (disabled actions are shown in gray):

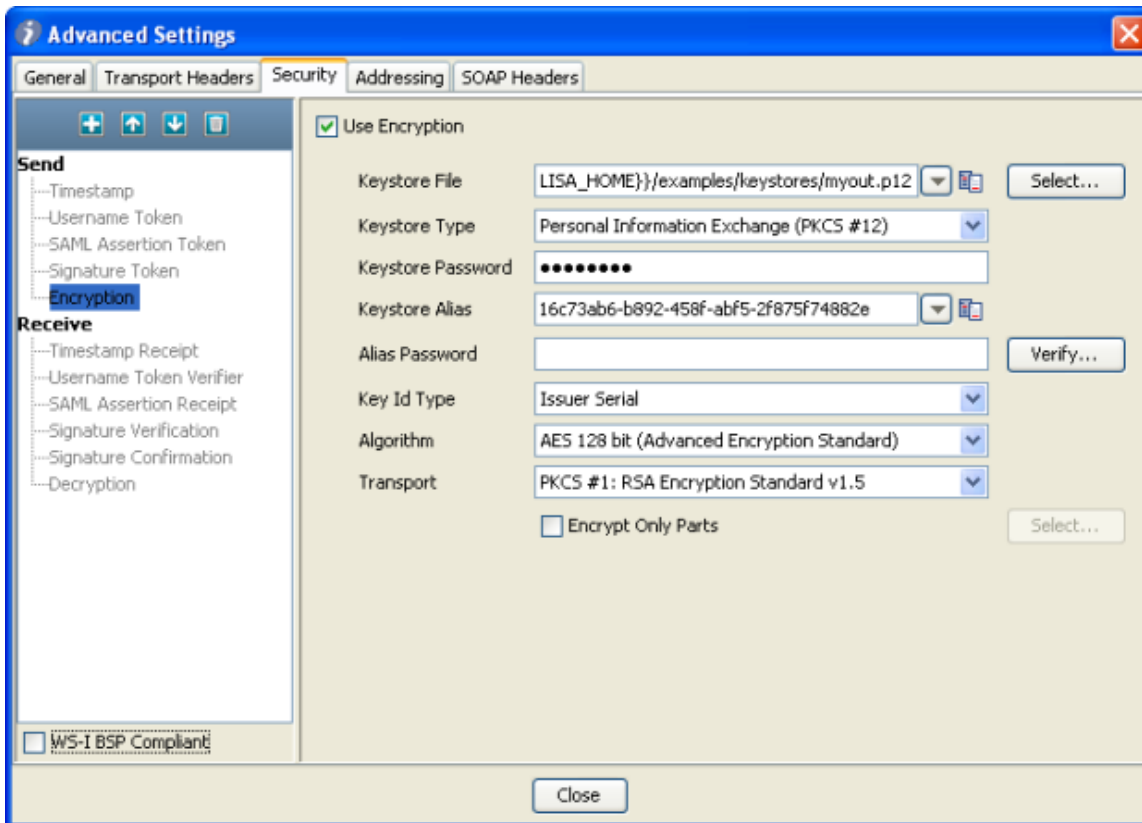


In the following sections we will describe the parameters needed to run the ws-security example

1) XML Encryption/Decryption

The parameters required configuring XML Encryption and Decryption are shown below:

a) Encryption



Advanced Settings

General | Transport Headers | **Security** | Addressing | SOAP Headers

☒ Use Encryption

Send

- Timestamp
- Username Token
- SAML Assertion Token
- Signature Token
- Encryption**

Receive

- Timestamp Receipt
- Username Token Verifier
- SAML Assertion Receipt
- Signature Verification
- Signature Confirmation
- Decryption

☐ WS-I BSP Compliant

Keystore File: LISA_HOME\examples\keystores\myout.p12 [Select...]

Keystore Type: Personal Information Exchange (PKCS #12)

Keystore Password:

Keystore Alias: 16c73ab6-b892-458f-abf5-2f875f74882e [Select...]

Alias Password: [Verify...]

Key Id Type: Issuer Serial

Algorithm: AES 128 bit (Advanced Encryption Standard)

Transport: PKCS #1: RSA Encryption Standard v1.5

☐ Encrypt Only Parts [Select...]

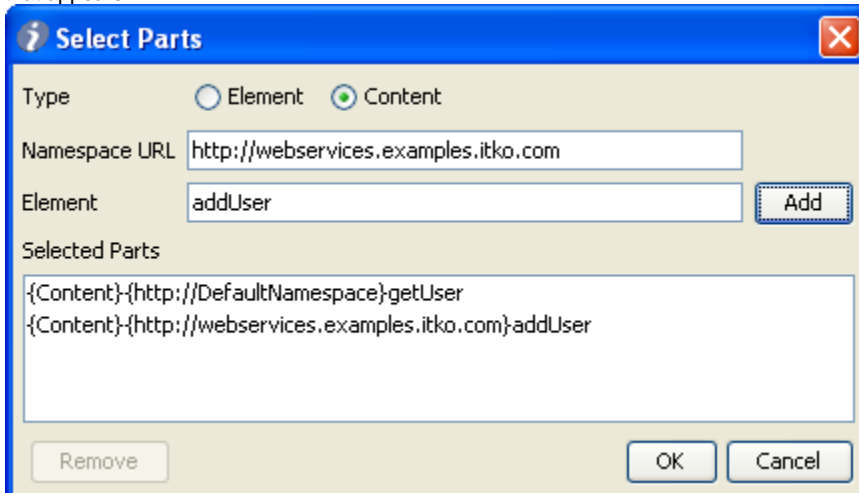
[Close]

Click the 'Use Encryption' check box

- Keystore File: The location of keystore file
- Keystore Type: Select Java Key Store (jks) or Personal Information Exchange (PKCS #12)
- Keystore Password: Enter the password for the keystore
- Keystore Alias: (should be an alias for a public key)
- Alias Password: (empty/same as Keystore Password for PKCS #12 files)
- Key ID Type: Select the appropriate key ID type from the pull-down menu
- Algorithm: Triple DES, AES 128, AED 192, or AES 236
- Transport: PKCS#1: RSA Encryption Standard v1.5 is the only option

The default behavior is to encrypt only the SOAP Body contents.

Encrypt Only Parts: If you want to specify different parts to encrypt click Select... button to identify the parts to be encrypted. In the pop-up screen that appears:



Select Parts

Type: ☐ Element ☒ Content

Namespace URL: http://webservices.examples.itko.com

Element: addUser [Add]

Selected Parts

```
{Content}-{http://DefaultNamespace}-getUser
{Content}-{http://webservices.examples.itko.com}-addUser
```

[Remove] [OK] [Cancel]

Type: select one of the following:

- Element: Select if you want to encrypt the element and the content
- Content: Select if you want to encrypt just the content

- Namespace URL: Enter the value for the element.
- Element: enter the name of the element.

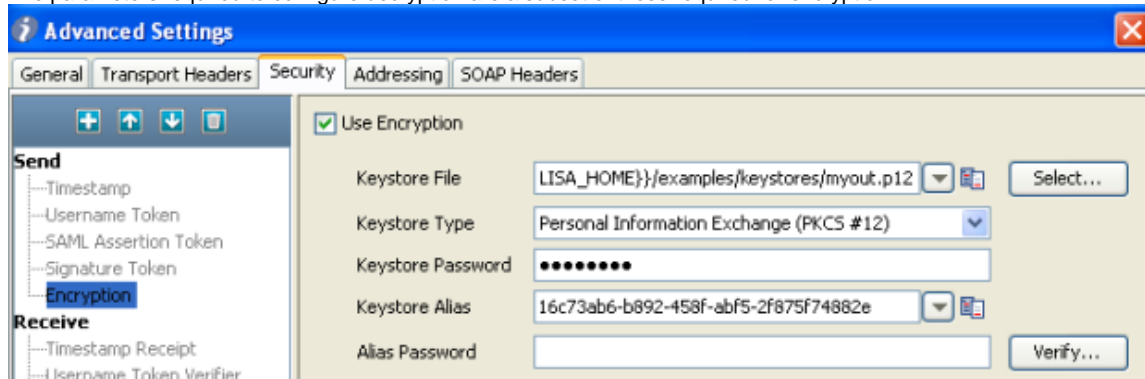
You can repeat this for as many elements as you wish by clicking the **Add** button.

You will need to manually add the Body element if you wish it to be included.

Note: If you want to include the Binary Security Token as a part use the Element name 'Token'.

b) Decryption

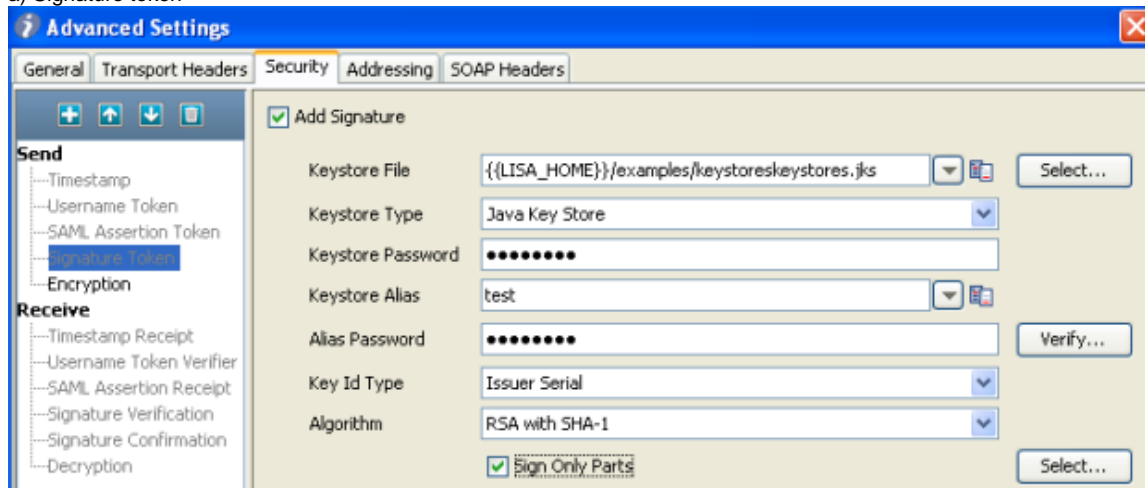
The parameters required to configure decryption are a subset of those required for encryption:



2) XML Signature Token/Signature Verification

The parameters required to configure XML Signature Tokens and Signature Verification are shown below:

a) Signature token

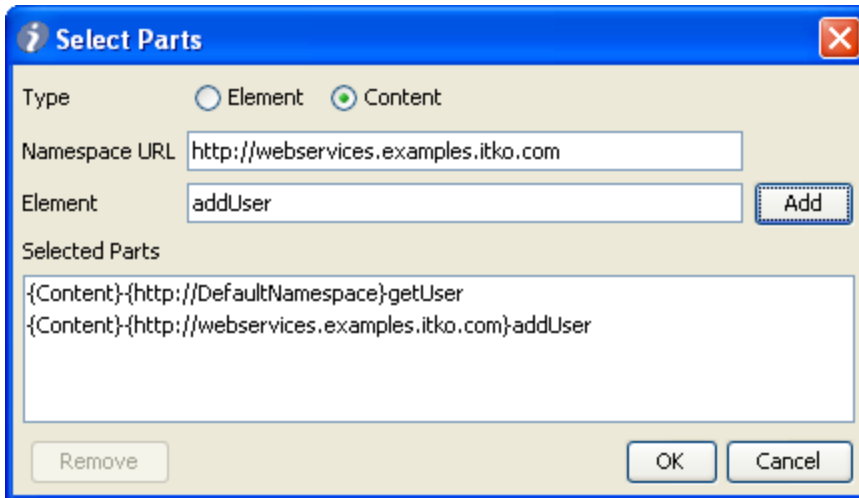


Click the Add Signature box

- **Keystore File:** The location of keystore file
- **Keystore Type:** Select Java Key Store (jks) or Personal Information Exchange (PKCS #12)
- **Keystore Password:** Enter the password for the keystore
- **Keystore alias:** (should be an alias for a private key)
- **Alias Password:** (empty/same as Keystore Password for PKCS #12 files)
- **Key ID type:** Select the appropriate key ID type from the pull-down menu
- **Algorithm:** Select DSA with SHA-1

The default behavior is to sign only the SOAP Body contents.

Sign Only Parts: If you want to specify different parts to sign click the **Select...** button to identify the parts to be signed. In the pop-up screen that appears:



Select Parts

Type: ☐ Element ☒ Content

Namespace URL:

Element:

Selected Parts:

```
{Content}-{http://DefaultNamespace}getUser
{Content}-{http://webservices.examples.itko.com}addUser
```

- **Type:** Select one of the following:
- **Element:** Select if you want to sign the element and the content.
- **Content:** Select if you want to sign just the content.
- **Namespace URL:** Enter the value for the element.
- **Element:** enter the name of the element.

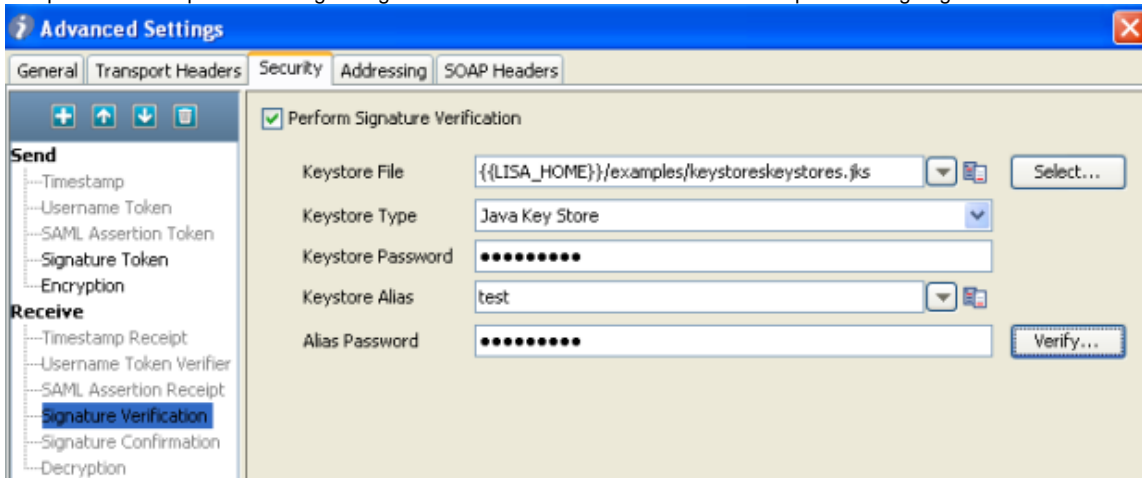
You can repeat this for as many elements as you wish by clicking the Add button.

You will need to manually add the Body element if you wish it to be included.

If you want to include the Binary Security Token as a part, use the Element name 'Token'.

b) Signature Verification

The parameters required to configure signature verification are a subset of those required for signing:



Advanced Settings

General | Transport Headers | **Security** | Addressing | SOAP Headers

☒ Perform Signature Verification

Keystore File:

Keystore Type:

Keystore Password:

Keystore Alias:

Alias Password:

Send

- Timestamp
- Username Token
- SAML Assertion Token
- Signature Token
- Encryption

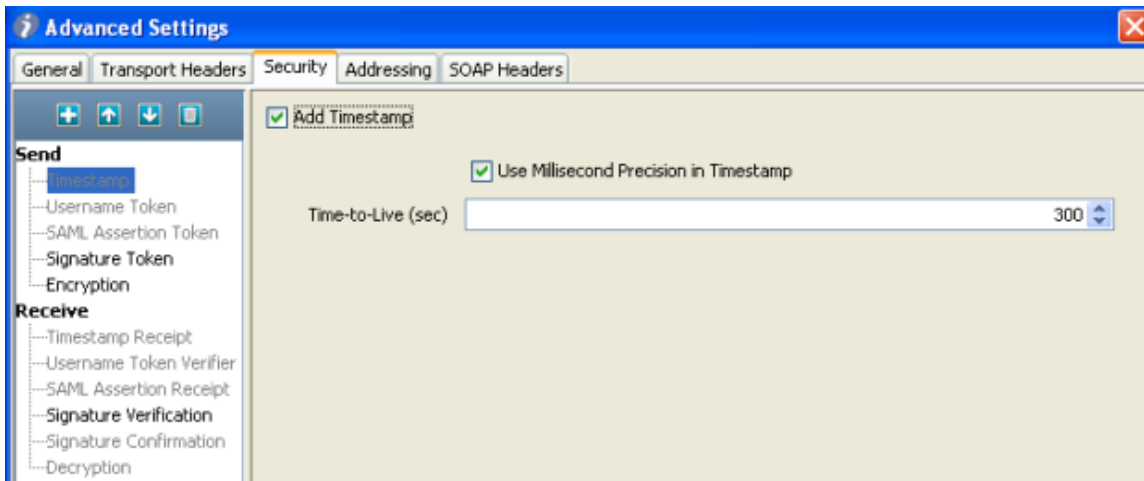
Receive

- Timestamp Receipt
- Username Token Verifier
- SAML Assertion Receipt
- Signature Verification**
- Signature Confirmation
- Decryption

3) Timestamp/Timestamp Receipt

The parameters required to configure Time stamp and timestamp Receipt are shown below:

a) Timestamp



Click the **Add Timestamp** box

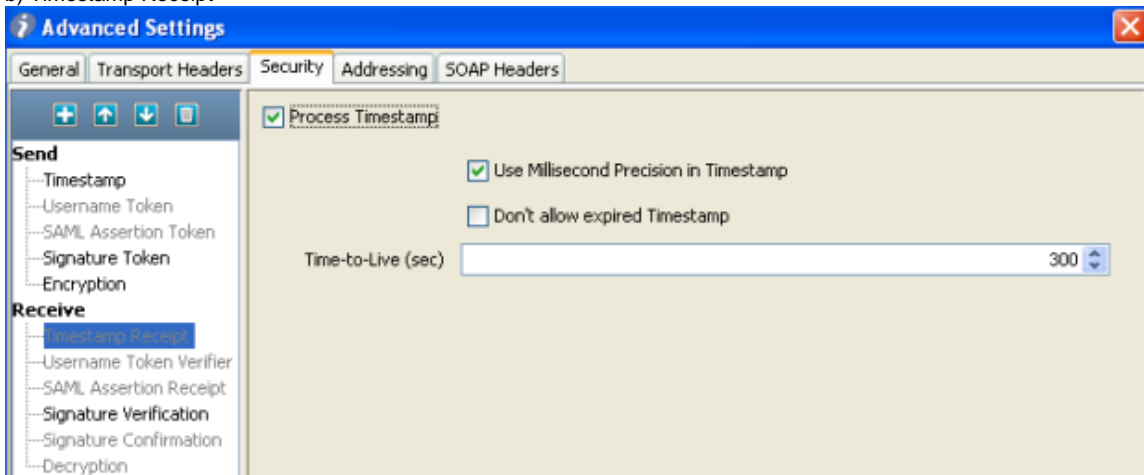
- **Time-To-Live (sec):** Enter the lifetime of the message in seconds. Enter 0 to not include an Expires element.



Some Web services, particularly .NET 1.x/2.0 with WSE 2.0, are not compliant with standard timestamp date formatting, and do not allow milliseconds. For these web services:

- Use Millisecond Precision in Timestamp: Deselect the checkbox

b) Timestamp Receipt



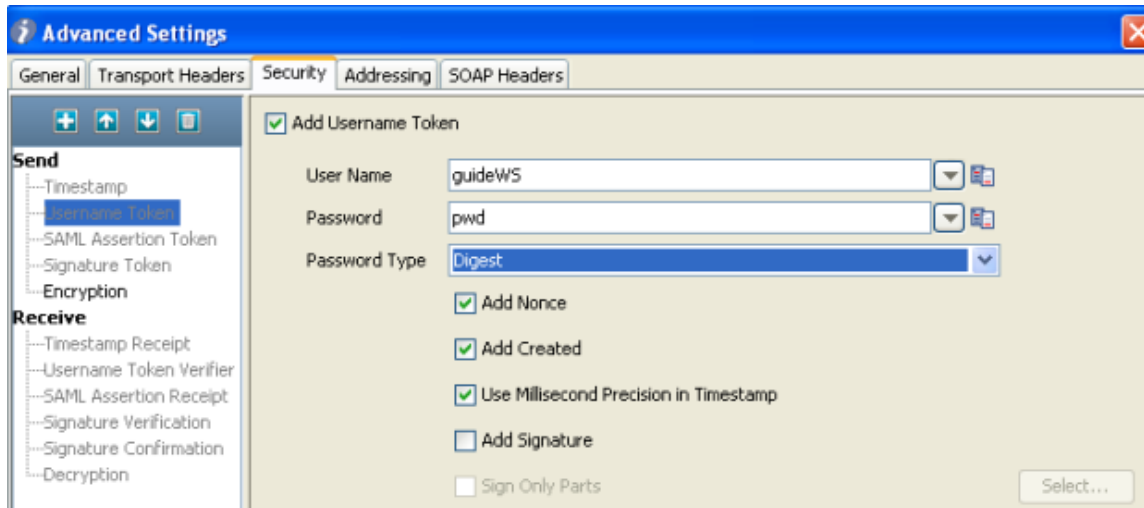
The parameters required for Timestamp Receipt are a superset of those required for Timestamp. The extra parameter:

- **Don't allow expired** Timestamp can be checked if you do not want to allow expired timestamps.

4) Username Token/Username Token Verifier

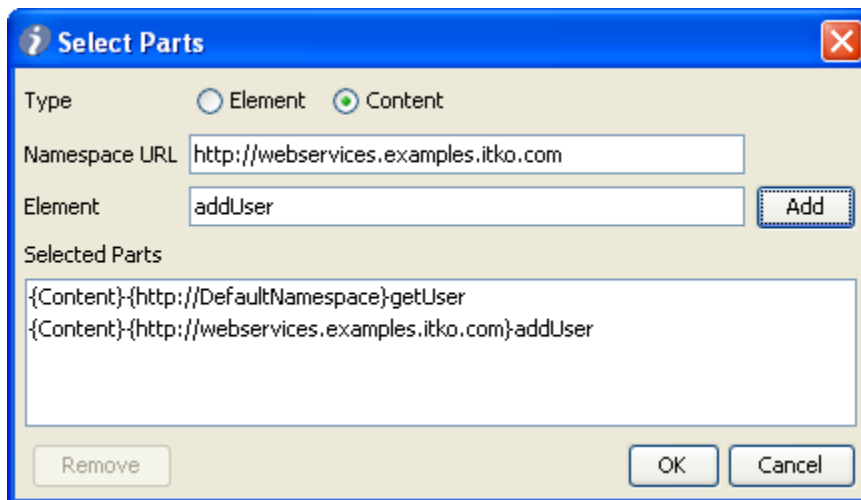
The parameters required to configure Username Token and Username token Verifier are shown below:

a) Username Token



Click the Add Username Token box

- **User Name:** Enter the appropriate user name.
- **Password:** Enter the appropriate password.
- **Password Type:** Select the password type from the drop-down menu (Text, Digest, None). None is typically used with the 'Add Signature' option.
- **Add Nonce:** Click if a Nonce is required – used to protect against replay attacks.
- **Add Created:** Click if a time stamp is required.
- **Use Millisecond Precision in Timestamp:** Select the checkbox to use millisecond precision. Note: some Web services, particularly .NET 1.x/2.0 with WSE 2.0, are not compliant with standard timestamp date formatting, and do not allow the use of milliseconds.
- **Add Signature:** to add a signature built using a combination of the username and password as the key.
- **Sign Only Parts:** Click if you want to specify different parts to sign and click the Select...button to identify the parts to be signed. In the pop-up screen that appears:



Type: select one of the following:

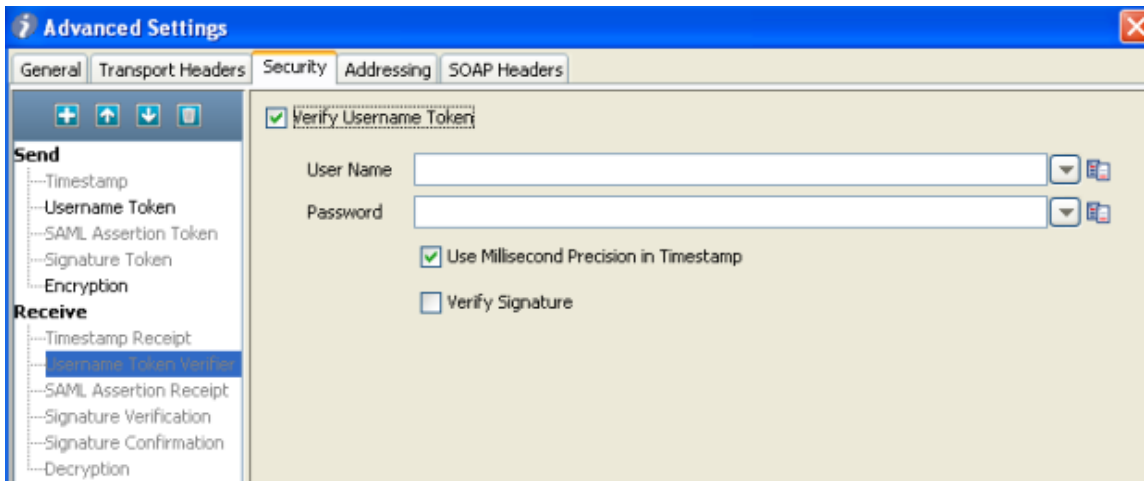
- **Element:** Select if you want to encrypt the element and the content
- **Content:** Select if you want to encrypt just the content
- **Namespace URL:** Enter the value for the element.
- **Element:** enter the name of the element.

You can repeat this for as many elements as you wish by clicking the Add button.

You will need to manually add the Body element if you wish it to be included.

If you want to include the Binary Security Token as a part, use the Element name 'Token'.

b) UserName Token Verifier

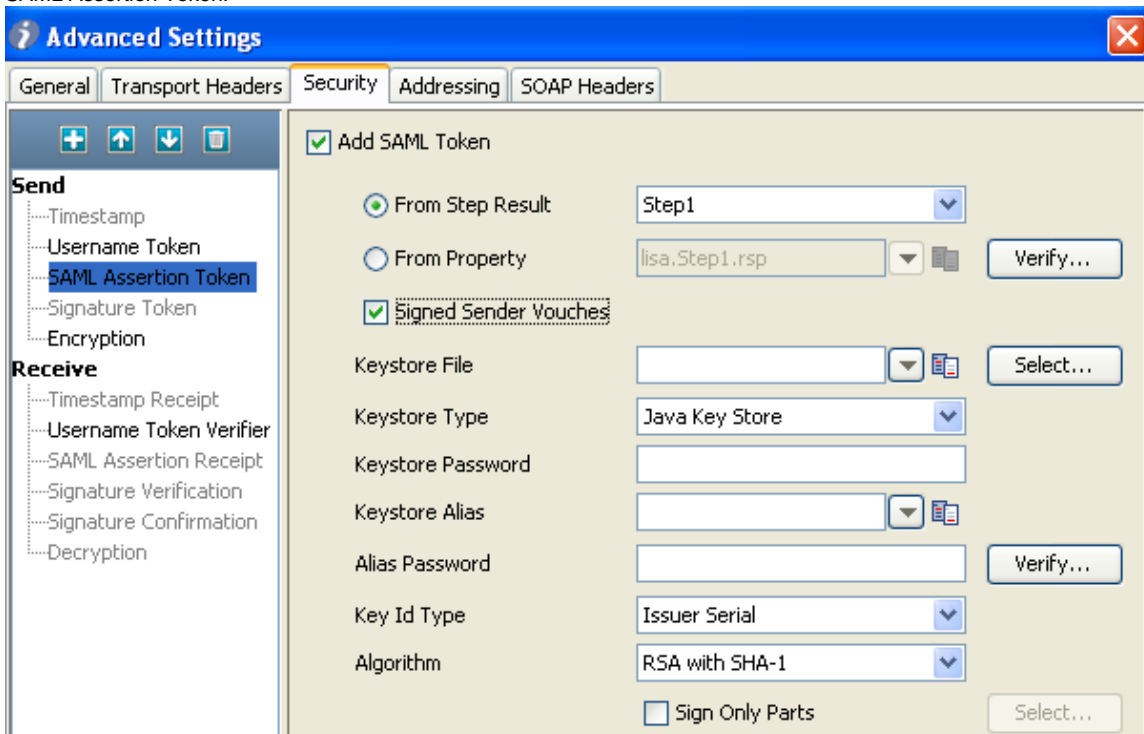


Click the Verify Username Token box

- **User Name:** Enter the verification user name.
- **Password:** Enter the verification password.
- **Use Millisecond Precision in Timestamp:** Select the checkbox to use millisecond precision. Note: some Web services, particularly .NET 1.x/2.0 with WSE 2.0, are not compliant with standard timestamp date formatting, and do not allow the use of milliseconds.
- **Verify Signature:** Click box if Signature verification is required.

5) SAML Assertion Token/SAML Assertion Receipt

The parameters required configuring SAML Assertion Token and SAML Assertion Verifier are shown below:
SAML Assertion Token:



- Click the Add SAML Token box
- Select the From Step Results radio box: Select the step whose result is an XML SAML Assertion (like a SAML Query Step or a Parse Text Step with xml manually entered), or
- select the From Property radio box: Enter the LISA property that contains the XML SAML Assertion
- You can optionally select the Verify... button to have LISA parse the SAML Assertion XML and build the SAML Assertion object as it would when sending the SOAP request. This is useful to confirm that the SAML Assertion that may have been created manually is a valid SAML Assertion. It will also attempt to verify any signatures associated with the assertion, but it'd likely that LISA will not be able to verify the assertion without configuring a public certificate to verify with. This ability will be added in a future release of LISA.
- **Signed Sender Vouches:** If the assertion needs to be signed by the sender (the sender vouches for it's authenticity vs. the bearer/creator of the SAML Assertion). When selected you'll need to fill in the following information:
- **Keystore File:** The location of keystore file.
- **Keystore Type:** Select Java Key Store (jks) or Personal Information Exchange (PKCS #12).

- **Keystore Password:** Enter the password for the keystore.
- **Keystore alias:** (should be an alias for a private key).
- **Alias Password:** (empty/same as Keystore Password for PKCS #12 files).
- **Key ID type:** Select the appropriate key ID type from the pull-down menu.
- **Algorithm:** Select DSA with SHA-1.

The default behavior is to sign only the SOAP Body contents.

- **Sign Only Parts:** If you want to specify different parts to sign click the Select...button to identify the parts to be signed. In the pop-up screen that appears:

- **Type:** select one of the following:
- **Element:** Select if you want to sign the element and the content.
- **Content:** Select if you want to sign just the content.
- **Namespace URL:** Enter the value for the element.
- **Element:** enter the name of the element.

You can repeat this for as many elements as you wish by clicking the **Add** button.

Note: You will need to manually add the Body element if you wish it to be included.

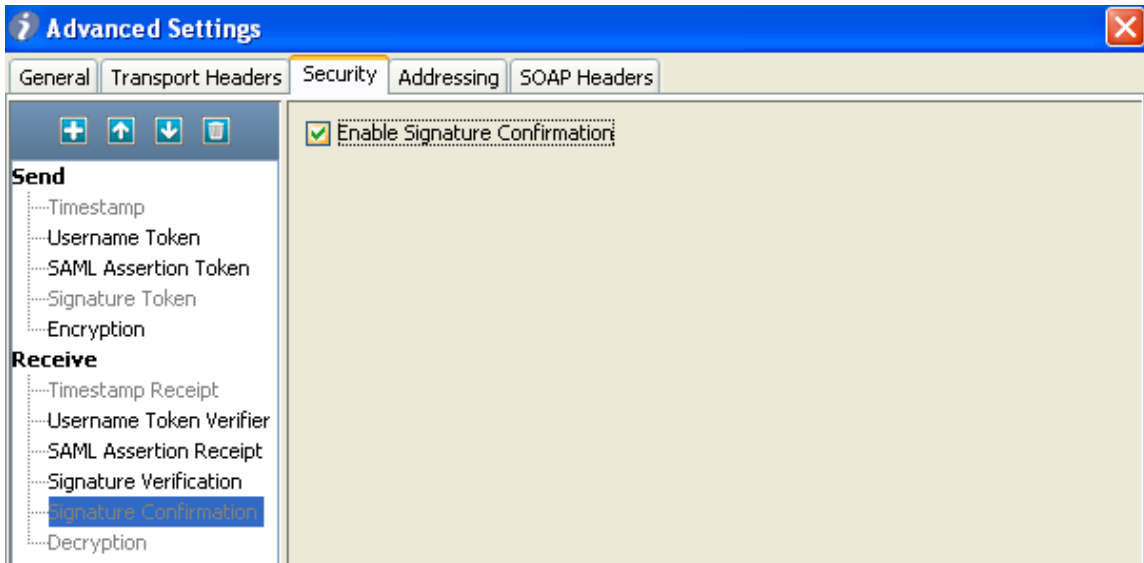
If you want to include the Binary Security Token as a part, use the Element name 'Token'.

b) SAML Assertion Receipt

Click the **Process SAML Assertion** box if you want LISA to check for a SAML Assertion Receipt header in the response. If you select this option and there is no SAML Assertion Receipt header then an exception occurs.

6) Signature Confirmation Completed till here

Click the **Signature confirmation** check box if you want LISA to check for a signature confirmation header in the response. If you select this option and there is no confirmation header then an exception occurs.



Using the Keystore Verifier

You can verify your keystore settings to make sure you are using the correct format, password, alias and alias password. There is a **Verify** button on the editors for SSL, Signature, Encryption/Decryption and SAML settings. Clicking **Verify** produces a verification report as shown below (extraneous lines deleted):



SSL verification validates the Keystore password only and confirms that at least one of the keys in the keystore can be loaded using the keystore password.

Alias: test

Alias Creation Date: Sat Apr 21 01:37:24 CDT 2007

Successfully loaded key for alias: test using keystore password

Key Algorithm: DSA

Key Format: PKCS#8

Key:

WS-Security verification validates the Keystore password, the alias and the alias password. Correct validation is indicated with a green entry. Any validation errors found will be shown in red. Warnings are shown in orange.

Note: This verification only verifies the keystore parameters. There could still be issues with the web service, such as a mismatch in certificate sets or incorrect choice of algorithm. These issues will need to be validated independently.

Alias Search

If you do not know the expected alias name for a WS-Security setting, you can use the keystore verifier to list all of the aliases in the keystore. Leave the 'Keystore Alias' and 'Alias Password' boxes empty and click the Verify button:

Advanced Settings

General Transport Headers **Security** Addressing SOAP Headers

☒ Use Encryption

Keystore File: {{LISA_HOME}}/examples/keystores/my Select...

Keystore Type: Personal Information Exchange (PKCS #12)

Keystore Password:

Keystore Alias: 16c73ab6-b892-458f-abf5-2f875f74882e Select...

Alias Password: Verify...

Key Id Type: Issuer Serial

Algorithm: AES 128 bit (Advanced Encryption Standard)

Transport: PKCS #1: RSA Encryption Standard v1.5

☒ Encrypt Only Parts Select...

Aliases are highlighted with a blue background.

Beginning Keystore Verification

File: C:\Lisa\examples\keystores\myout.p12

Keystore size: 1

Provider: BC

Type: pkcs12

Alias: 16c73ab6-b892-458f-abf5-2f875f74882e

Alias Creation Date: Thu Aug 09 01:11:44 CDT 2007

Certificate #0:

Type: X.509

Note: Verification will fail since the Keystore Alias and Alias Password boxes were left blank.

1.10 Start or Stop Web Server

1.10 Start or Stop Web Server

This step is used to start or stop the embedded web server which is used for certain types of virtual services. When creating a virtual service from a WSDL, the user must first ensure that the embedded web server is started.

Start or Stop Web Server

Web Server: Tomcat 5.0.28

Server State

☒ Start Web Server ☐ Stop Web Server

Port Settings

HTTP Port Number: 9080

HTTPS Port Number: 9443

AJP Port Number: 9009

Proxy Port Number: 9082

JMX Port Number: 9085

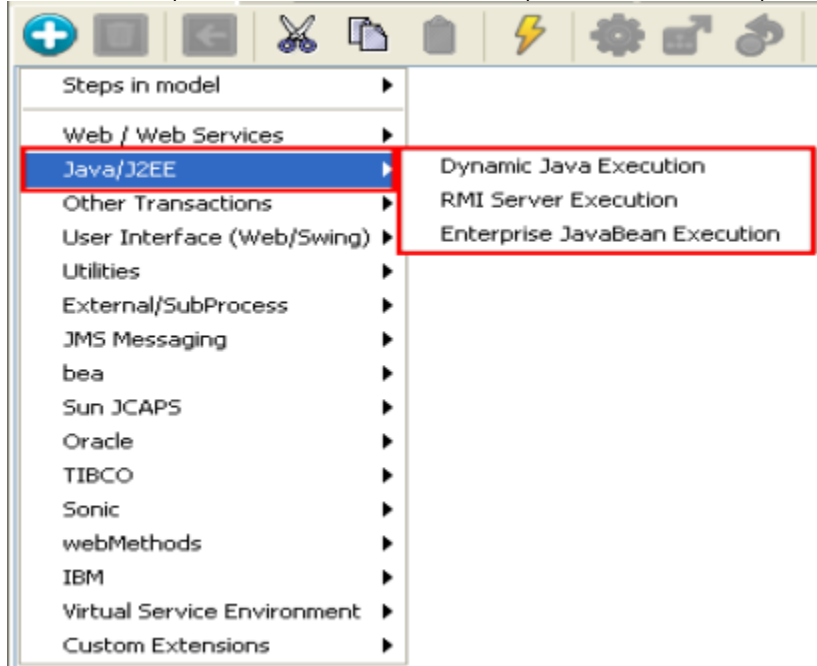
Execute

Editor Results

2. Java_J2EE Steps

2. Java_J2EE Steps

These are the steps available in the Java/J2EE test steps list in the the New Step menu:



This section explains these steps:

- 2.1 Dynamic Java Execution
- 2.2 RMI Server Execution
- 2.3 Enterprise Java Bean Execution

2.1 Dynamic Java Execution

2.1 Dynamic Java Execution

The Dynamic Java Execution step allows you to instantiate and manipulate a Java object. All Java classes on the LISA classpath are available, including the classes in the JRE's classpath. Any user classes can be placed on the classpath by copying them into the hot deploy directory. The class under test is loaded into the LISA complex object editor where it can be manipulated without having to write any java code.

Prerequisites - Knowledge of the LISA complex object editor is assumed. For details see the LISA User Guide. The class under test must be on the LISA classpath.

The Dynamic Java Execution step editor is shown below:

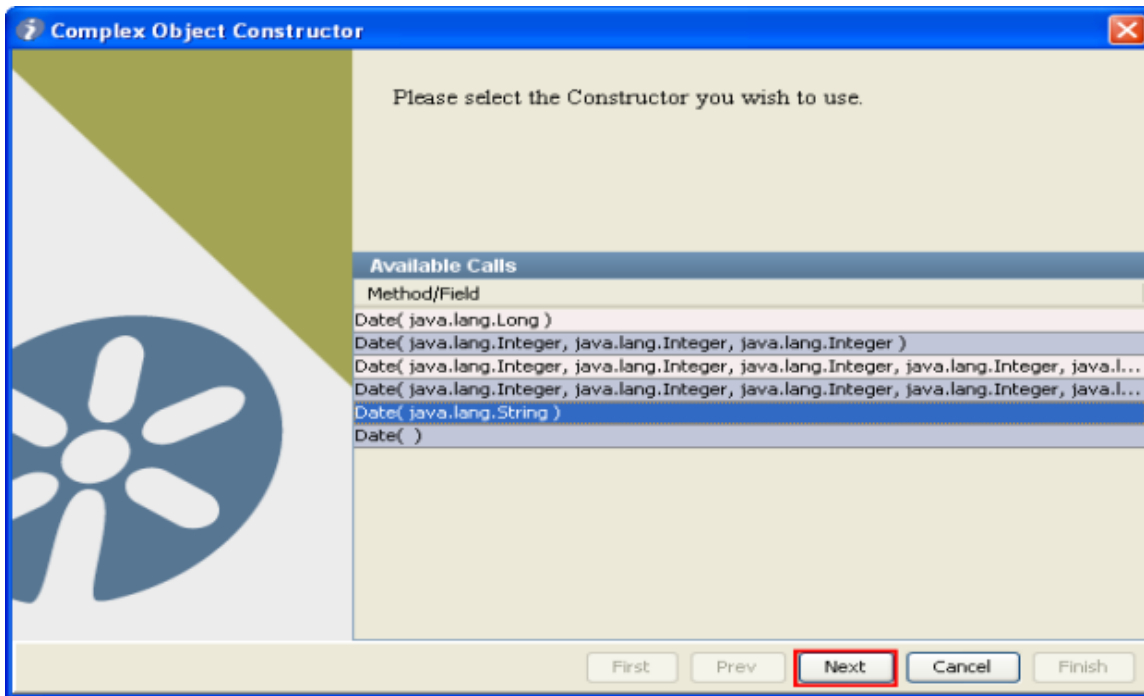
The screenshot shows the 'Class Selection and Error Step' dialog. On the left, under 'Use JVM', the 'Local' radio button is selected. The main area has a 'Local JVM Settings' section. The 'Make New Object Of Class:' radio button is selected, and the text field next to it contains 'java.util.Date'. There is a browse button (three dots) and a refresh button (circular arrow) next to the text field. The 'Load From Property:' radio button is unselected, and its text field is empty. Below this is a dropdown menu for 'If environment error:' with 'Abort the Test' selected. At the bottom of the dialog is a large button labeled 'Construct/Load Object...'.

In our example we are using a Java date instance of class **java.util.Date**.
Enter the following parameters:

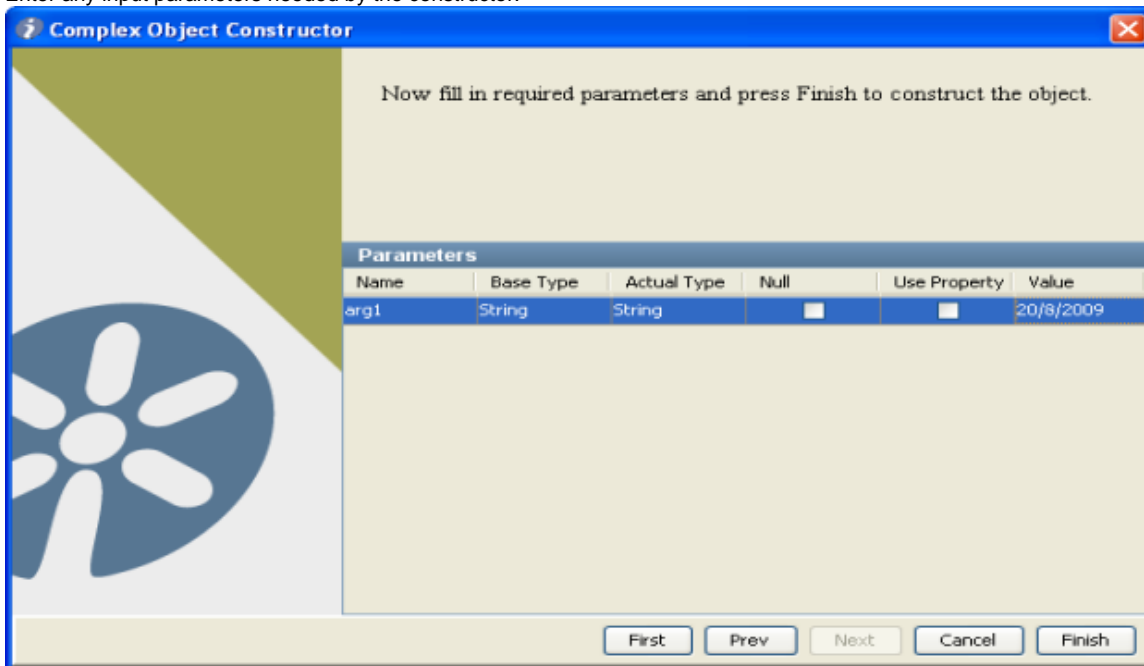
- **Local JVM settings:** Select one of the following radio buttons:
- **Make New Object of Class:** Click the radio button and enter, select or browse to the java class you want to instantiate. This must be the fully qualified class name including the package of the Java class, for example com.example.MyClass.
- **Load from Property:** Click the radio button and enter the name of the property that has the serialized object as its value.
- **If environment error:** Select the step to redirect to if an environment error occurs while trying to create an object.
- **Use JVM:** Select the Local button. It is possible to execute a java object remotely, using In-Container Testing (ICT), by clicking the Remote radio button, but this mode requires some extra set up before it can be used. This is discussed in the LISA for Java Developers.

Note: If you require that the Java object is loaded by its own classloader, you must add the companion "Class Loader Sandbox Companion". Click the **Construct/Load Object** button.

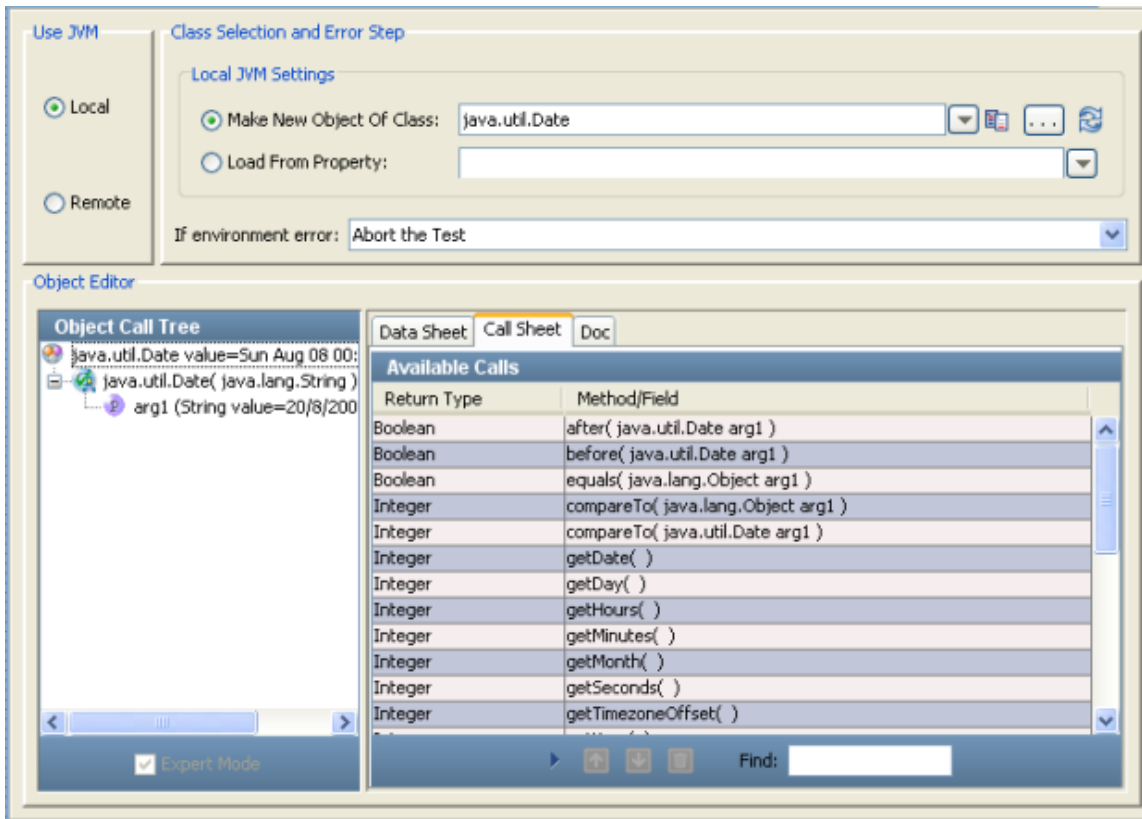
The Complex Object Constructor window is now displayed listing the available constructors for your object:



Select a constructor and click the **Next** button.
Enter any input parameters needed by the constructor:

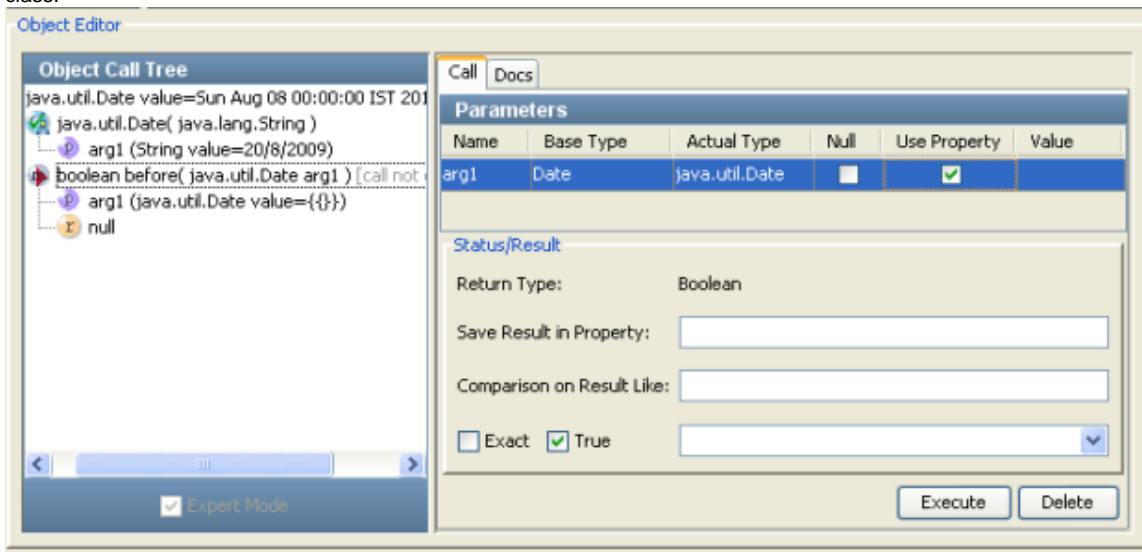


In this example, enter a string representation of a day (20/8/2007). You can enter a value, a property or null.
LISA constructs the object and loads it into the LISA Complex Object Editor:



You can now manipulate the object, and execute methods, using the LISA Complex Object Editor. For details on how to use the Complex Object Editor see the LISA User Guide.

You can also add filters and add assertions, either by using the inline filter/assertion form (which is part of the complex object editor) or manually by selecting filter under your test step in the test case tree. For example, here is a screen just before we execute, the before method on the Date class:



You can see the Status/Result section where you can add an inline filter, in the Save Results in Property textbox, and if desired, an inline assertion, in the Comparison on Result Like textbox.

2.2 RMI Server Execution

2.2 RMI Server Execution

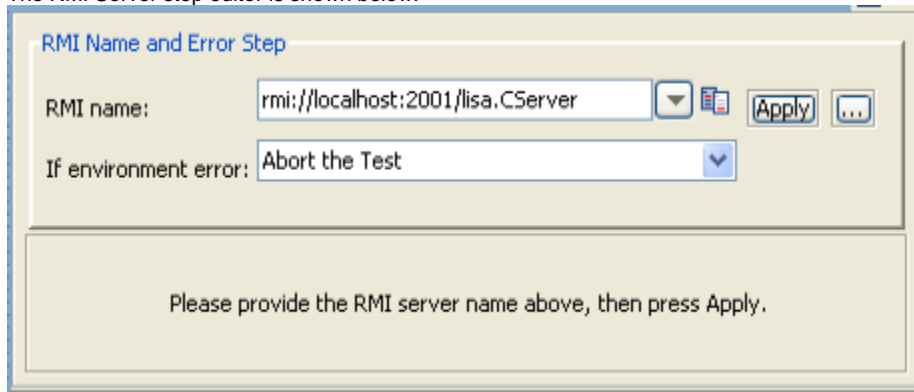
The RMI Server Execution step allows you to acquire a reference to a remote Java object via RMI (Remote method Invocation), and make calls on the Java object.

Prerequisites - Knowledge of the LISA complex object editor is assumed. For details see the LISA User Guide. You also need to copy the interface and stub classes for the remote object into the hot deploy directory. LISA requires these to contact and interact with the remote object.

These classes should be obtainable from the remote object developer.

Parameter Requirement - You will need to know how to connect to the RMI Server, usually a host name and port, and you will need to know the RMI name of the object you want to invoke.

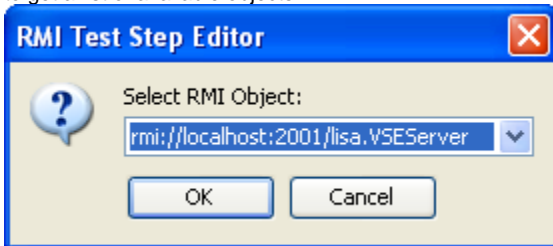
The RMI Server step editor is shown below:



The dialog box is titled "RMI Name and Error Step". It contains two input fields: "RMI name:" with the value "rmi://localhost:2001/lisa.CServer" and a browse button (...), and "If environment error:" with a dropdown menu showing "Abort the Test". There are "Apply" and "..." buttons to the right of the first field. At the bottom, a message says "Please provide the RMI server name above, then press Apply."

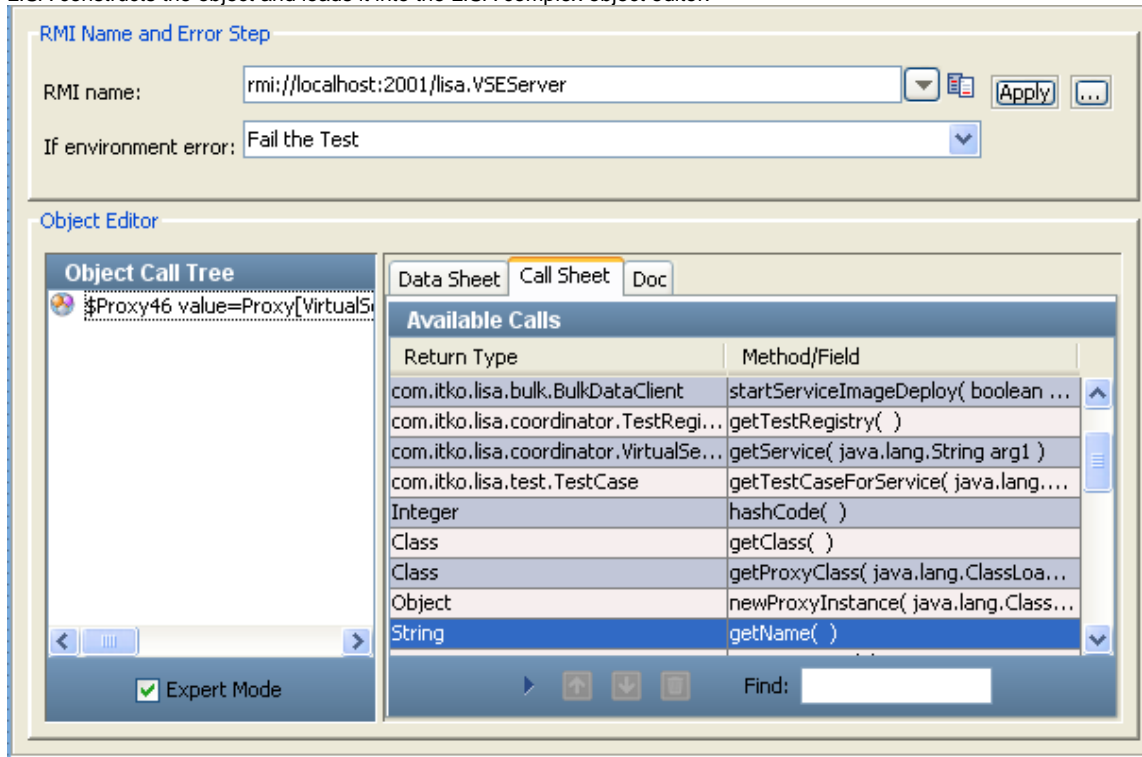
Enter the following parameters:

- RMI Name: Enter or select the full RMI name of the object (as shown above), or enter the RMI Server name and click the Browse button to get a list of available objects:



The dialog box is titled "RMI Test Step Editor". It has a question mark icon and the text "Select RMI Object:". Below this is a dropdown menu showing "rmi://localhost:2001/lisa.VSEServer". At the bottom are "OK" and "Cancel" buttons.

LISA constructs the object and loads it into the LISA complex object editor:



The LISA Complex Object Editor is shown. The top section is "RMI Name and Error Step" with "RMI name:" set to "rmi://localhost:2001/lisa.VSEServer" and "If environment error:" set to "Fail the Test". The bottom section is "Object Editor". On the left is the "Object Call Tree" showing "\$Proxy46 value=Proxy[VirtualSe...". On the right is the "Available Calls" table.

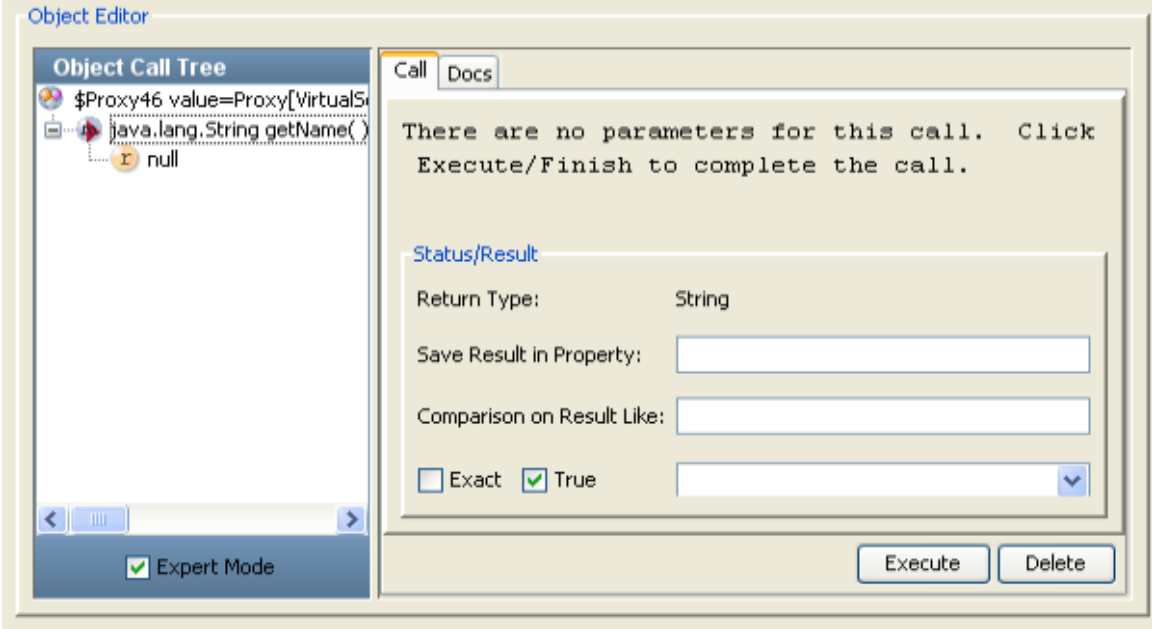
Return Type	Method/Field
com.itko.lisa.bulk.BulkDataClient	startServiceImageDeploy(boolean ...
com.itko.lisa.coordinator.TestRegi...	getTestRegistry()
com.itko.lisa.coordinator.VirtualSe...	getService(java.lang.String arg1)
com.itko.lisa.test.TestCase	getTestCaseForService(java.lang....
Integer	hashCode()
Class	getClass()
Class	getProxyClass(java.lang.ClassLoa...
Object	newProxyInstance(java.lang.Class...
String	getName()

At the bottom of the Object Editor, there is a "Find:" text box and a "Execute" button. The "Expert Mode" checkbox is checked.

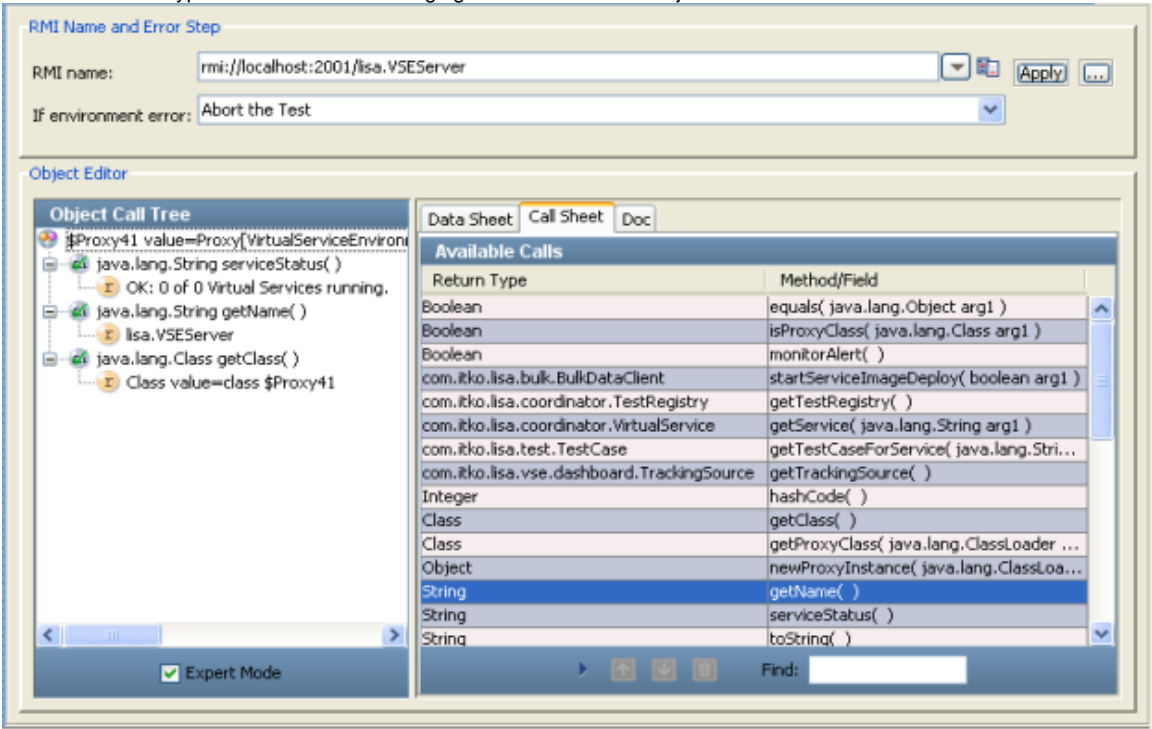
You can now manipulate the object, and execute methods, using the LISA Complex Object Editor.

In the figure above we have selected the getName method. If we double-click it, it gets added in the Object Call Tree, and then it can be executed using the Execute button which appears in the dialog, along with any method arguments that need to be passed to the method, and information

on how to process the result.



The above figure shows null as the return value because the method is not yet executed. Once **Execute** button is clicked, it gets executed and the correct return type is shown. The following figure shows how the Object Call Tree tracks the results of execution of several methods.



You can also add filters and add assertions, either by using the inline filter/assertion form (see below) or manually by selecting filter under your test step in the test case tree. You can see the **Status/Result** section where you can add an inline filter, in the **Save Results in Property** textbox, and if desired, an inline assertion, in the **Comparison on Result Like** textbox.

Note: If you have multiple network cards, using localhost in the RMI name can cause errors. You may need to use the IP address, or the host name that corresponds to the particular IP address.

2.3 Enterprise Java Bean Execution

2.3 Enterprise Java Bean Execution

The Enterprise Java Bean Execution step allows you to acquire a reference to and make calls on an Enterprise Java Bean (EJB) running in a J2EE application server.

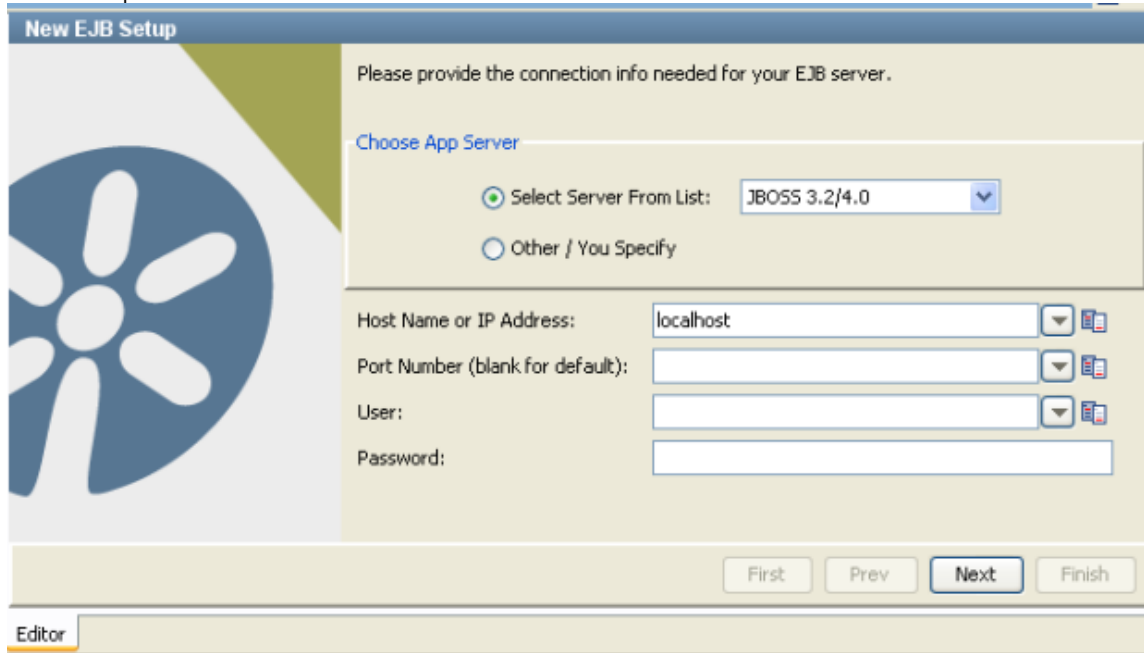
Testing an EJB is similar to testing a Java object. LISA will dynamically connect to the EJB using the Home EJB interface, and then from it create an instance of an EJB object. This process is a little different for EJB as they do not require a home interface. The EJB under test is loaded into the LISA Complex Object Editor where it can be manipulated without having to write any java code.

Prerequisites - Knowledge of the LISA Complex Object Editor is assumed. For details see the LISA User Guide. The application client jar and client EJB jar must be in the LISA Class path. Both of these jar files are usually copied into the hot deploy directory. LISA ships with the jboss-all-client.jar file for the JBOSS application server and the LISA examples jar file in the hot deploy directory so that you can run the EJB examples immediately.

Note: When using an IBM WebSphere server, you will need to run LISA with the IBM JRE and JDK. The RMI and IIOP implementations are different for Sun and IBM.

Parameter Requirements - Server connection information (JNDI connection) and userid and password (if required). The global JNDI lookup name of your EJB home interface. This information should be provided by the EJB deployer.

The EJB step editor is shown below:



In our example, we are using the iTKO example server, a JBoss server. The host name is **example.itko.com**, with a default port. To use your local Demo Server use **localhost** as the host name.

Enter the following parameters:

- **Choose App Server:** Select your application server from the list. If your application server is not on the list then click the **Other/You Specify** radio button.

The lower section of the editor will change depending on your selection. The figure above shows the configuration panel for JBoss. The next figure shows the panel that is displayed if you select **Other/You Specify**.

For the JBoss panel (above), enter the following parameters:

- Host Name or IP Address: Enter hostname or IP address of your application server.
- Port Number (blank for default): Enter port number if different from the default (1099).
- User: Enter if a User ID is required for the application server.
- Password: Enter if a password is required for the application server.

Click the **Next** button.

For the Other/You Specify panel (below):

New EJB Setup

Please provide the connection info needed for your EJB server.

Choose App Server

☐ Select Server From List: JBOSS 3.2/4.0

☒ Other / You Specify

JNDI Factory:

 JNDI Server URL:

 User:

 Password:

First Prev **Next** Finish

Editor

Enter the following parameters:

- JNDI factory: Enter or select the fully qualified JNDI factory class name for your application server.
- URL: Enter or select the JNDI server name.
- User: Enter if a User ID is required for the application server.
- Password: Enter if a password is required for the application server.

Click the **Next** button.

You are now presented with the New EJB Setup window showing a list of all the JNDI names registered with the application server:

New EJB Setup

Now we need the name and class of the Home EJB interface (if using EJB 2.x or EJB3 with a defined home) or just the remote interface if you are using EJB3.

Selected JNDI Name

EJB JNDI Name: com.itko.examples.ejb.UserControlBean

Remote Local

- com.itko.examples.ejb
 - SimpleEchoBean
 - IntUserControlLocal
 - EJB3AccountControlBean
 - remote
 - UserTransaction
 - com.itko.examples.ejb.UserControlBean
 - HTTPConnectionFactory
- local
 - CalculatorLocalEJB@18942714
 - UIL2XAConnectionFactory
- invokers
 - Istanbul
 - ilop
 - UILConnectionFactory

Refresh List

First Prev **Next** Finish

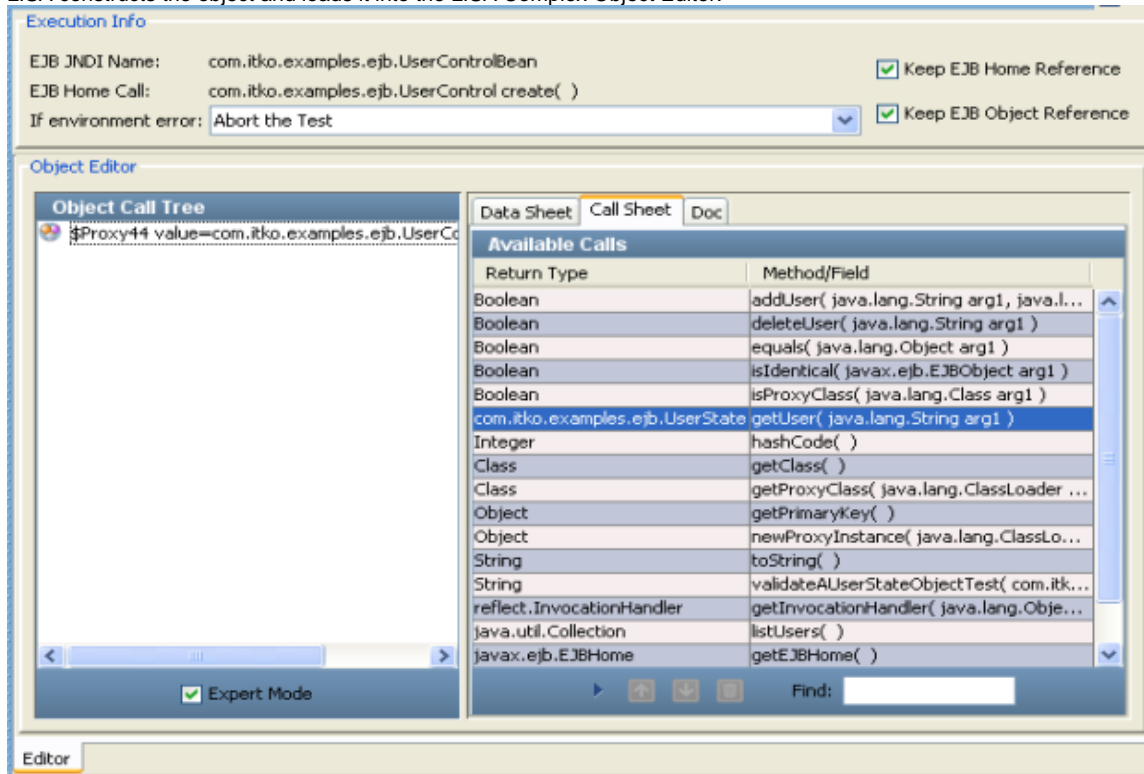
Editor

Select the name of the appropriate EJB home interface. In our example, the JNDI name is: **com.itko.examples.ejb.UserControlBean**.

The EJB3 specification allows stateful and stateless beans to bind to the JNDI tree directly and not require a home interface. If this is the case, the bean can be selected directly and LISA will not need to create a new instance.

Click the **Next** button.

LISA constructs the object and loads it into the LISA Complex Object Editor:



In the **Execution Info** you will see the current EJB information displayed. If you plan to reuse this EJB you can keep the references to the EJB object and the EJB Home by clicking the **Keep EJB Home Reference**, and **Keep EJB Object Reference** checkboxes on the right. If the bean is an EJB3 bean without a Home interface, the **Keep EJB Home Reference** checkbox will be disabled. Set the **If Exception** to the step to redirect to if an exception occurs.

You can now manipulate the object, and execute methods, using the LISA Complex Object Editor. The usage is the same as is seen in the [RMI Server Execution](#) step.

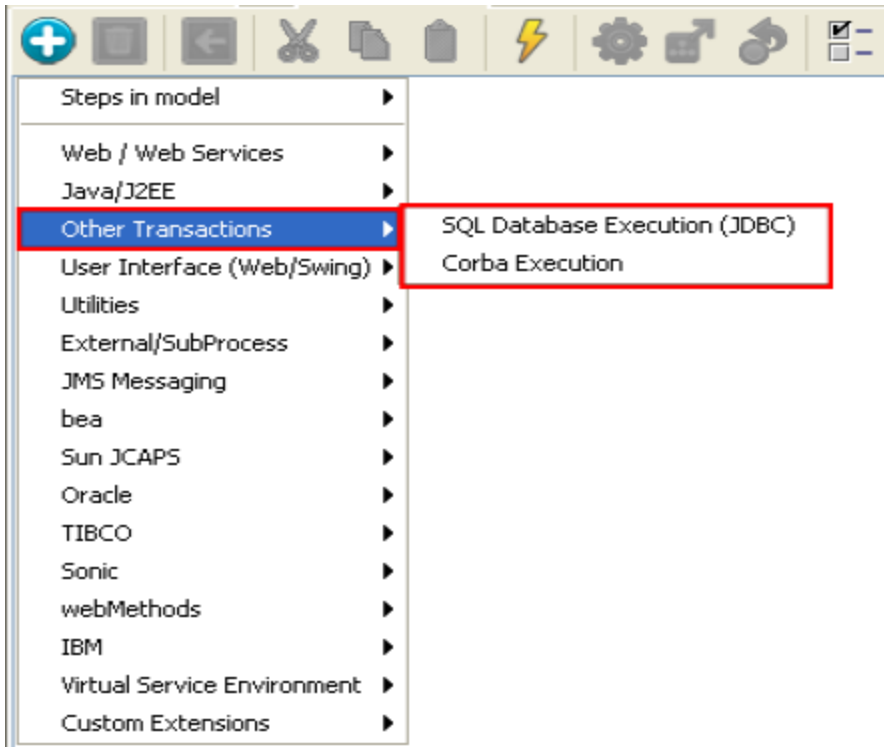
You can also add filters and add assertions, either by using the inline filter/assertion or manually by selecting filter under your test step in the test case tree. You can see the Status/Result section where you can add an inline filter, in the **Save Results in Property** textbox, and if desired, an inline assertion, in the **Comparison on Result Like** textbox.

3. Other Transaction Steps

3. Other Transaction Steps

These are the steps available in the "Other Transactions" test steps list in the New Step menu:

- SQL Database Execution
- Corba Execution



This section explains these steps:

- [3.1 SQL Database Execution \(JDBC\)](#)
- [3.2 Corba Execution](#)

3.1 SQL Database Execution (JDBC)

3.1 SQL Database Execution (JDBC)

The SQL Database Execution step allows you to connect to a database using JDBC (Java Database Connectivity) and make SQL queries on the database.

Full SQL syntax is supported, but LISA does not validate the SQL. It is passed through to the database where it will be validated. Make sure that the SQL being used is valid for the database manager you are using.

Prerequisites - The JDBC driver appropriate for your database must be on the LISA classpath. You can place the driver jar file in the hot deploy directory. LISA includes the Derby client driver in its classpath so it's not needed to be added again.

Parameter Requirements - You will need to have the name of the JDBC driver class, the JDBC URL for your database, and if required, a user ID and password for the database. You will also need to know the schemas for the tables in the database in order to construct your SQL queries.

The SQL Database step editor is shown below with the base tab selected (at the bottom of the screen).

The screenshot shows the JDBC Step Editor dialog box with the following configuration:

- Connection Info:**
 - JDBC Driver: `org.gjt.mm.mysql.Driver`
 - Connect String: `jdbc:mysql://examples.itko.com/itko_examples`
 - Max Rows to Fetch: `-1`
- Execution Info:**
 - User ID: `itko_examples`
 - Password: `.....`
 - Keep Connection Open: ☐
 - Returns Result Set: ☒
 - If SQL error: `Fail the Test`
- SQL Statement:**

```
SELECT * FROM users
```
- Buttons:** `Test Connection`, `Test/Execute SQL`
- Tabs:** `Base`, `Result Set`

Enter the following parameters:

Connection Info

- **JDBC Driver:** Enter or select the full package name of the appropriate driver class. Standard driver classes are available in the pull down menu. You can also use the browse... button to browse the LISA class path for the driver class.
- **Connect String:** This is the standard JDBC URL for your database. Enter or select the URL. JDBC URL templates for common database managers are available in the pull down menu.
- **Max Rows to Fetch:** Enter the maximum number of rows you want returned in the result set. This is a required field, use -1 for unlimited rows.

Execution Info

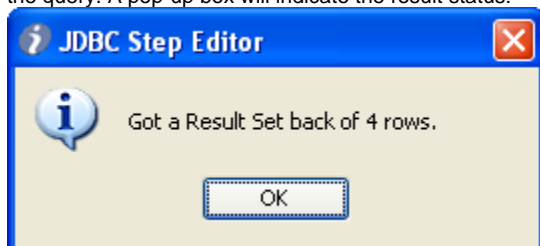
- **User ID:** Enter a User ID (if it is required by the database)
- **Password:** Enter a password (if it is required by the database)
- **Keep Connection Open:** If selected the database connection opened the first time the step executes is cached, and is closed when the step is garbage collected. If Keep Connection Open is not selected, the connection is closed each time the step executes.
- **Returns Result Set:** Click this checkbox if your query will result in a Result Set being returned, i.e. a SELECT type query. Leave unchecked for an UPDATE, INSERT or DELETE. Your query will cause an error if this checkbox is set incorrectly.
- **If SQL error:** Select the step to redirect to if an error occurs.

The values in the figure above are appropriate for the LISA examples database on the iTKO server (password is examples). For the local DemoServer use:

- **JDBC Driver:** `org.apache.derby.jdbc.ClientDriver`
- **Connect String:** `jdbc:derby://localhost:1527/reports/lisa-reports.db`
- **User ID:** `sa`
- **Password:** `sa`

Once you have entered the database connection information; including the User ID and password (if required) you can use the Test Connection button to test your connection. If the information is correct you will get a success message in a pop-up window. Otherwise you will get an error message indicating what the problem might be.

You are now ready to enter your SQL statement in the lower window. Properties can be used in your SQL. LISA will make the parameter substitution before passing the SQL string to the database. Once you have created the SQL query, use the **Test/Execute SQL** button to execute the query. A pop-up box will indicate the result status:



Click **OK**, and your results will be displayed under the **Result Set** tab as shown below:

Result Set					
fname	lname	login	pwd	email	type
Sara	Bellum	sbellum	sbpwd	sbellum@mycompan...	1
Warren	Piece	wpiece	wppwd	wpiece@mycompan...	1
Amanda	Reckonwith	areck	apwd	areck@mycompany...	1
		itko	test		0

You are now ready to create filters and assertions on the result set.

The 3 icons on the bottom of the Result Set panel give you easy access to the following filters and assertions:

- Get value for another value in a Result Set Row. You select a search field cell, a value field filter and enter a property name. If LISA finds the cell value in the search field, the value in the value field in that row will be set as the value of the property that is entered.
 - Parse Result for Value filter. The value in the selected cell will be set as the value of the property that is entered.
 - Result Set Contents Assertion. The values in the chosen field (column) will be compared to the regular expression that is entered.

For more details on these and other filters and assertions appropriate for result sets can be found in [PART 3 - Filters](#) and [PART 4 - Assertions](#).

3.2 Corba Execution

3.2 Corba Execution

This step is used to make CORBA calls using Java RMI-IIOP library.

User is expected to provide appropriate skeleton classes.

Before starting execution, user need to copy 'corbaserver.jar' file (this jar is available in CORBA server 'lib' directory) to LISA 'lib' directory .


Prerequisites: Corba Server available.

Object IOR: This field contains the raw IOR string for the object or the name service.

This string can be taken from the output (generated IOR) of running nameserver.sh batch file.

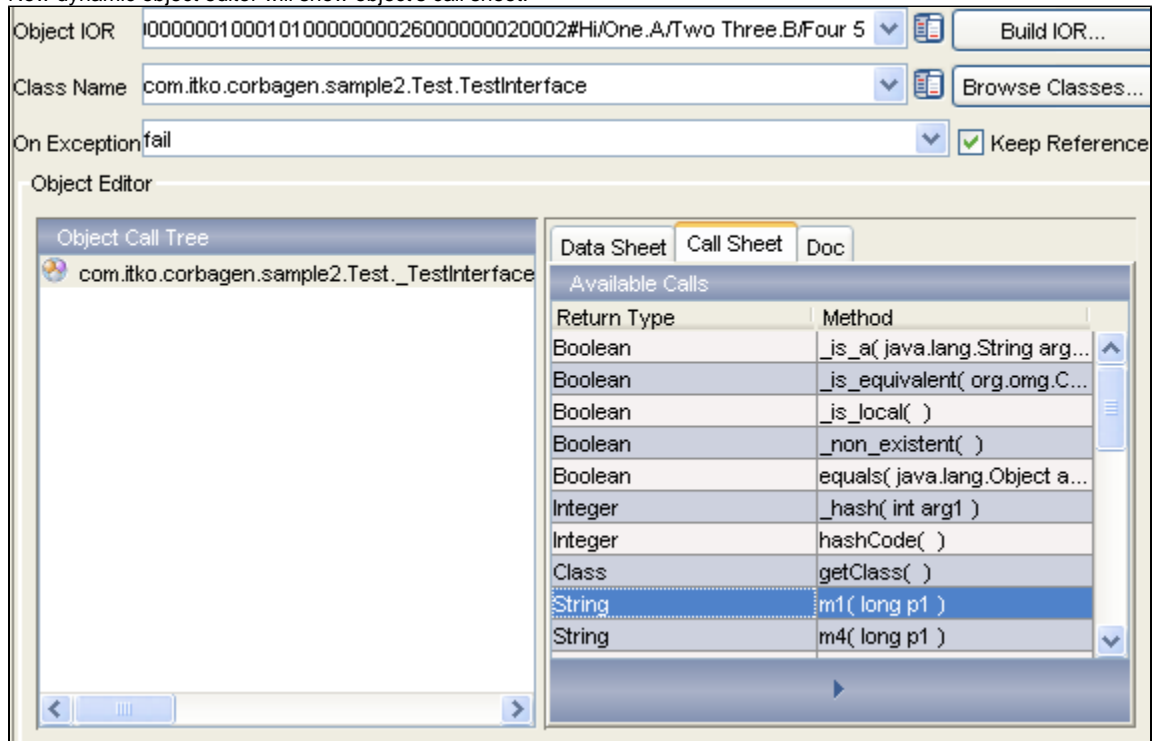
Only precaution that needs to be taken is the entire string should not content any spaces in between which may happen if it's directly copy & pasted from nameserver printed output to Object IOR field.

Class Name: This is the object class that IOR references.

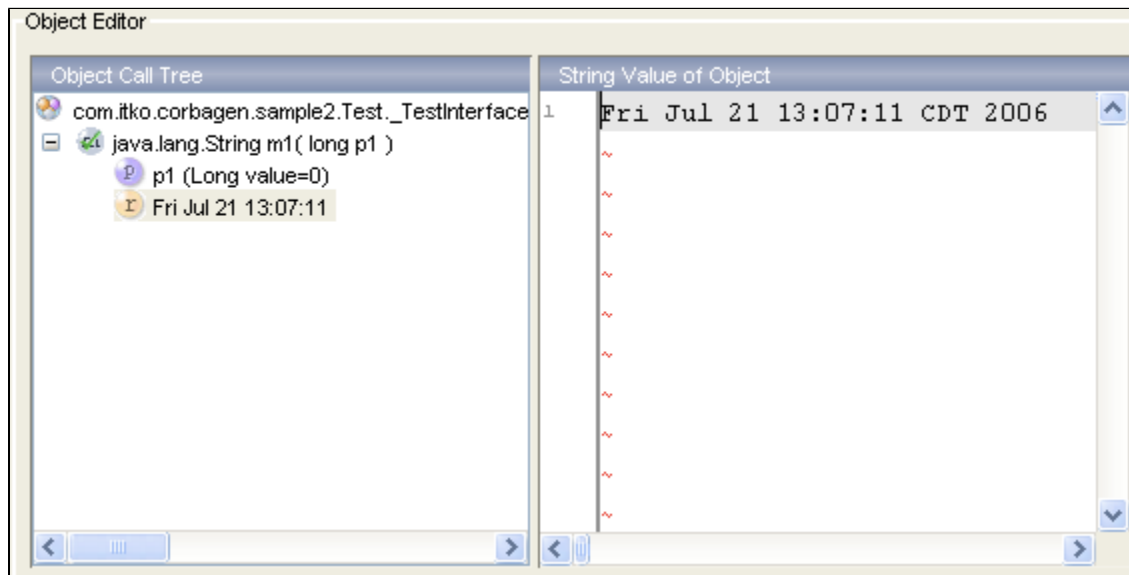
Object IOR	IOR:0000000000000002b49444c3a6f6d672e6f72672f436f734e616d696e67: v		Build IOR...
------------	--	---	--------------

You can also use the IOR construction dialog. When open it will parse any IOR string entered and fill in the individual parts. Don't worry about the strange looking key. IORs store the key in a byte format so not every byte can be displayed properly. Just don't edit the field if you want to use the parsed version of a raw IOR, it will still work.

Now dynamic object editor will show object's call sheet.



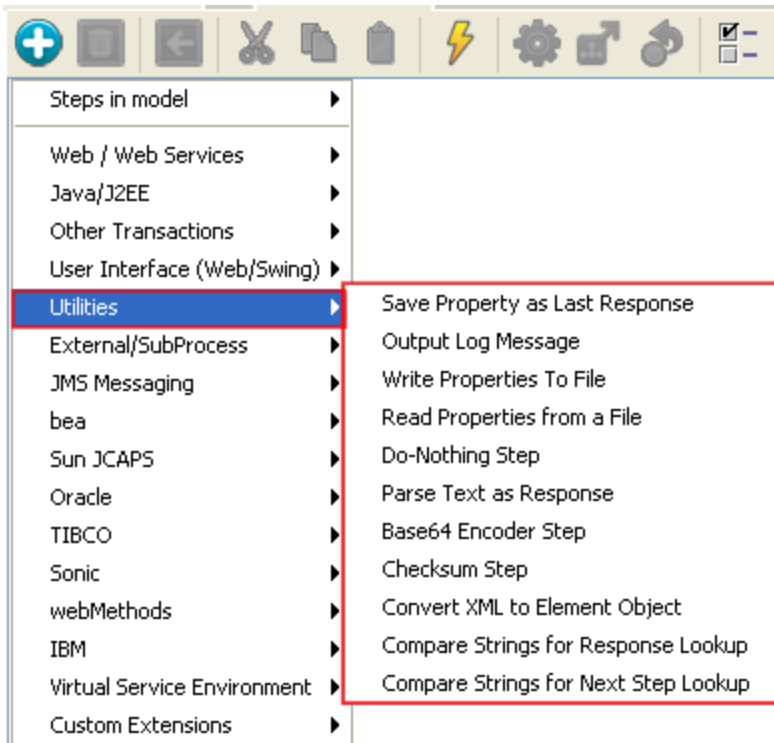
Now select the method you like to call & execute it.



4. Utilities Steps

4. Utilities Steps

These are the steps available in the Utilities test steps list in the New Step menu:



This section explains these steps:

- 4.1 Save Property as Last Response
- 4.2 Output Log Message
- 4.3 Write Properties to File
- 4.4 Read Properties from a File
- 4.5 Do-Nothing Step
- 4.6 Parse Text as Response
- 4.7 Base64 Encoder Step
- 4.8 Checksum Step
- 4.9 Convert XML to Element Object
- 4.10 Compare Strings for Response Lookup step
- 4.11 Compare Strings for Next Step Lookup step

4.1 Save Property as Last Response

4.1 Save Property as Last Response

The Save Property as Last Response step allows you to save the value of a LISA property as the last response.

Enter the property name of an existing LISA property or select it from the pull-down menu. The value of the property will be loaded as the last response (and the step response). It can then be accessed immediately as the last response, or later in the test case using the property **lisa.thisStepName.rsp**, where **thisStepName** is the name of current step.

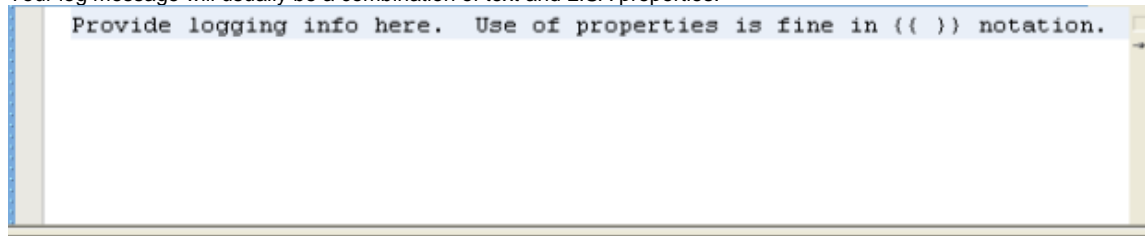
Recall that each step in a LISA test case has a response associated with it, and when that step is executed, its response is automatically saved in two LISA properties: LASTRESPONSE and **lisa.thisStepName.rsp**. You use this step often so that you can have filters and assertions apply to a property value rather than the real response of the step.

4.2 Output Log Message

4.2 Output Log Message

The Output Log Message step allows you to output a text message to the log. This is useful for tracking and logging the test case as it executes.

Your log message will usually be a combination of text and LISA properties.



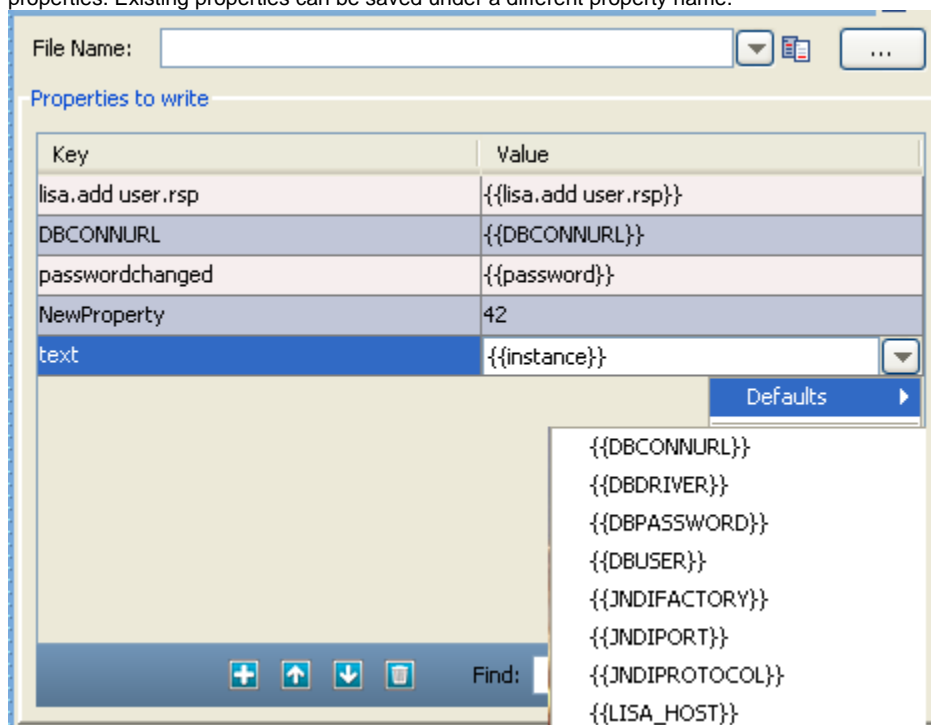
Provide logging info here. Use of properties is fine in {{ }} notation.



Enter your log message into the editor as shown above. This log message will appear when this step executes in the Interactive Test Run (ITR), and will be logged when during a test run.

4.3 Write Properties to File


4.3 Write Properties to File





The Write Properties to File step allows you to save a list of LISA properties in a text file. The properties can be existing properties or new properties. Existing properties can be saved under a different property name.




File Name:  

Properties to write


Key	Value
lisa.add user.rsp	{{lisa.add user.rsp}}
DBCONNURL	{{DBCONNURL}}
passwordchanged	{{password}}
NewProperty	42
text	{{instance}} 

    Find:

Defaults 

- {{DBCONNURL}}
- {{DBDRIVER}}
- {{DBPASSWORD}}
- {{DBUSER}}
- {{JNDIFACTORY}}
- {{JNDIIPORT}}
- {{JNDIPROTOCOL}}
- {{LISA_HOST}}

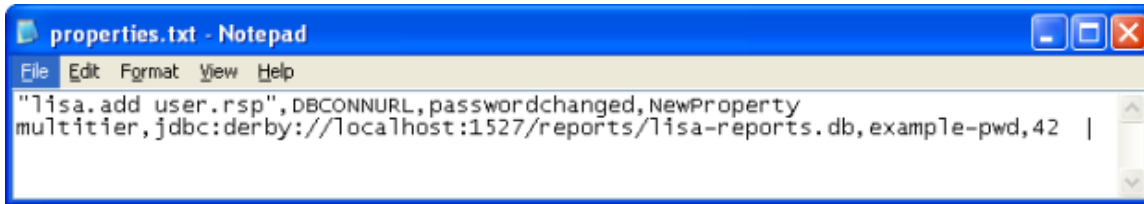
Enter the following:

- File Name: Enter path name of file, or browse to file using the browse (...) button.
- Properties to write: Use the Add, , button to add a row. Then enter the properties (Key) and corresponding values (Value) you want to save. Your list of properties may contain existing or new properties. When specifying existing properties (Key), you may override its current value by specifying a new value, or you may use its existing value by selecting it from the drop-down list as shown above.

In the example above:

- The first 2 properties were stored without change.
- The third property was stored under a new name (key).
- The fourth property is new.
- The fifth entry is in progress showing the pull down menu in the Value column.

The properties are stored in a comma delimited text file where the first line in the file is the set of property names being saved, and the second line contains the values corresponding to each of the properties. The figure below shows the file produced using the four properties in the figure above.



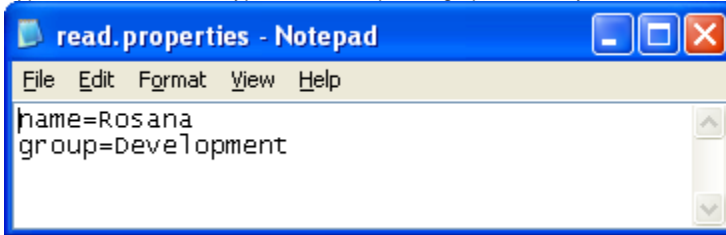
4.4 Read Properties from a File

4.4 Read Properties from a File

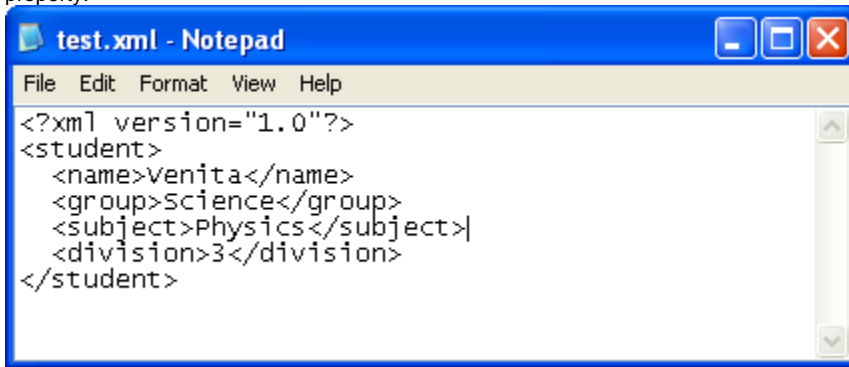
This step is used to read the properties from an external file. You can read the properties in two ways; name-value-pairs or in xml tags.

Enter the following parameters:

- File Name: Enter the file name or browse to the file containing the properties.
- Type of File: Select the type of a file, depending upon the way it has stored the properties.



In the above figure you can see the **Name-Value-Pair** type of a properties file, where name is a property whereas Rosana is a value of the name property.



In the above figure you can see the **XML Tags** type of a properties file, where name is a property whereas Venita is a value of the name property.

4.5 Do-Nothing Step

4.5 Do-Nothing Step

The Do-Nothing step does not take any parameters, nor does it have any functionality by itself. However it is useful in certain situations, as you can add assertions to the step.

For example, you can use the scripted assertion to add a quick custom assertion, to do a comparison of numbers or dates, or some numerical comparison test (greater than or less than).

4.6 Parse Text as Response

4.6 Parse Text as Response

The Parse Text as Response step allows you to enter textual content from a file that can be saved as the last response. The content can be stored in a LISA property (optional). You can type or paste the text into the editor, or load it from a file.

Property Key (opt)

```
<xml>
  <name>Rosanna</name>
  <group>Development</group>
  <message>This is correct</message>
</xml>
```

Enter the following parameters:

- Property Key [opt]: The name of the property to store the content (optional)
- Load from File: Click to browse to the file. Otherwise type or paste the text into the editor.

The content is now available for you to parameterize, filter and add assertions.

Click the **Test** button to show the resulting text.

```
<xml>
  <name>Rosanna</name>
  <group>Development</group>
  <message>This is correct</message>
</xml>
```

4.7 Base64 Encoder Step

4.7 Base64 Encoder Step

The Checksum step calculates the checksum of a file and optionally saves the value in a LISA property. For more information refer to [1.6 Base64 Encoder](#) in [1. Web_Web Services Steps](#).

4.8 Checksum Step

4.8 Checksum Step

The Checksum step calculates the checksum of a file and optionally saves the value in a LISA property.

File:

Property Key [opt]:

If environment error:

2075036343

Source

Enter the following parameters:

- **File:** Enter the pathname or browse to the file on which you want the checksum calculated.
- **Property key [opt]:** The name of the property that will store the checksum value.

Click the **Load** button.

The checksum will be displayed as the response, and if a property name was entered the value will be stored in that property.

The checksum value is now available for you to filter and add assertions.

4.9 Convert XML to Element Object

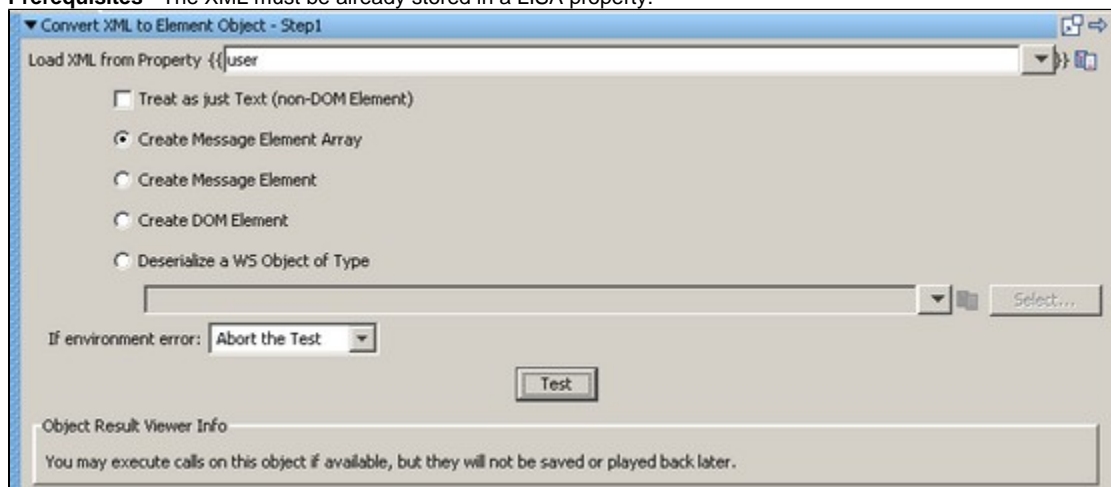
4.9 Convert XML to Element Object

The Convert XML to Element Object step converts a raw XML into an object of one of the following types:

- Message Element Array
- Message Element
- DOM Element

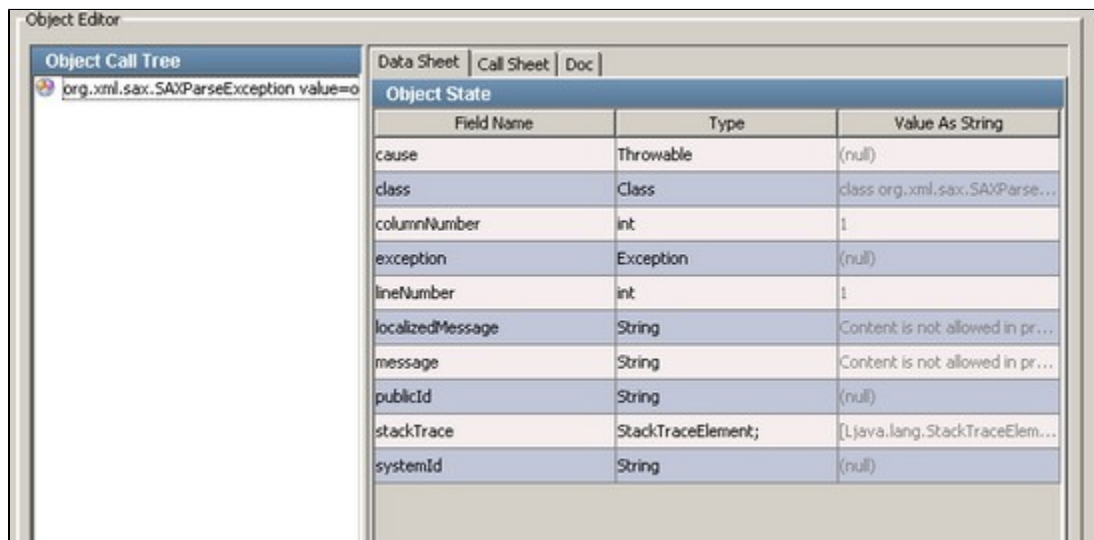
This is useful when you have a web service API that takes any type using strict processing. This type of WSDL element will require a Message Element Array as an input parameter. You can capture the raw XML from a previous step (such as Read from File or Parse Text as Response) and store it in a LISA property. That property becomes an input parameter for this step.

Prerequisites - The XML must be already stored in a LISA property.



Enter the following parameters:

- **Load XML from Property:** Enter the property that contains the XML. This can be a user defined property or a built in LISA property as shown above.
- Check the **Treat as just Text** box if you need to use plain text as your input rather than XML. This will result in a message element that contains plain text.
- Select the type of object you want from the available types by clicking the respective radio button.
- Click the **Test** button to do the conversion.



Use the response from this step when this object is required as a parameter in another step.

You can use the **Save Step Response as a Property** filter to save the response in a filter, or you can refer to it as **lisa.<stepname>.rsp**.

4.10 Compare Strings for Response Lookup step

4.10 Compare Strings for Response Lookup step

This step is used to look at an incoming request and determine the appropriate response. This step is text based. You can match the incoming requests using partial text match, regular expression, among others.

Text to match:

Range to match: Start: End:

If no match found:

If environment error:

☐ Store responses in a compressed form in the test case file.

Case Response Entries					
Enabled	Name	Delay Spec	Criteria	Compare Type	Response
<input checked="" type="checkbox"/>	str-compare	0	Simpson	Find in string	Simpson is found

Find:

Criteria	Response
<div></div>	<div></div>

The step components are:

Field	Description
Text to match	Enter the text against which criteria should be matched. This is typically a property reference, such as LASTRESPONSE .
Range to match	Enter the Start and End of the range.
If no match found	From the list, select the step to go to if no match is found.
If environment error	From the list, select the step to go to if the test fails.
Store responses in a compressed form...	Selected by default. This option compresses the responses in the test case file.
Case Response Entries	Add, move, and delete entries.
Enabled	Selected by default when you add an entry. Deselect to ignore an entry.
Name	Enter a unique name for the case response entry.
Delay Spec	Enter the delay specification range. The default is 1000-10000 , which indicates to use a randomly selected delay time between 1000 and 10000 milliseconds. (The syntax is the same format as Think Time specifications.)
Criteria	This area provides the string to compare against the Text to match field. To edit the criteria, in the Case Response Entries area select the appropriate row, and then select a different setting from the Criteria list.

Compare Type	Select an option from the list: <ol style="list-style-type: none"> 1. Find in string (default) 2. Regular expression 3. Starts with 4. Ends with 5. Exactly equals
Response	This area provides the response of this step if the entry matches the Text to match field. To edit the response, in the Case Response Entries area select the appropriate row, and then select a different setting from the Response list.
Criteria	Allows for the update of the criteria string for an entry.
Response	Allows for the update of the step response for an entry.

4.11 Compare Strings for Next Step Lookup step

4.11 Compare Strings for Next Step Lookup step

This step is used to look at an incoming request and determine the appropriate next step. You can match incoming requests using partial text match, regular expression, among others.

Each matching criterion specifies the name of the step to which to transfer if the match succeeds.

Text to match:

Range to match: Start:
End:

If no match found:
If environment error:

Next Step Entries					
Enabled	Name	Delay Spec	Criteria	Compare Type	Next Step
<input checked="" type="checkbox"/>	str-compare	0	Simpson	Find in string	FTP Step

+

↑

↓

✕

Find:

Criteria

Simpson

The step components are:

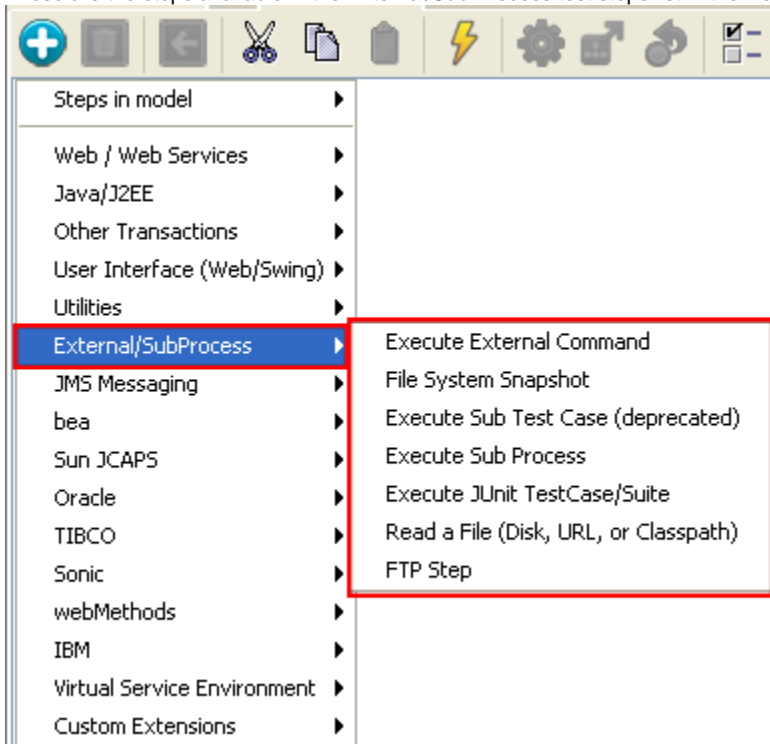
Field	Description
Text to match	Enter the text against which criteria should be matched. This is typically a property reference, such as LASTRESPONSE .
Range to match	Enter the Start and End of the range.
If no match found	From the list, select the step to go to if no match is found.
If environment error	From the list, select the step to go to if the test fails.
Next Step Entries	Add, move, and delete entries.

Enabled	Selected by default when you add an entry. Deselect to ignore an entry.
Name	Enter a unique name for the next step entry.
Delay Spec	Enter the delay specification range. The default is 1000-10000 , which indicates to use a randomly selected delay time between 1000 and 10000 milliseconds. (The syntax is the same format as Think Time specifications.)
Criteria	This area provides the string to compare against the Text to match field. To edit the criteria, in the Next Step Entries area select the appropriate row, and then select a different setting from the Criteria list.
Compare Type	Select an option from the list: <ul style="list-style-type: none"> 1. Find in string (default) 2. Regular expression 3. Starts with 4. Ends with 5. Exactly equals
Next Step	From the list, select the step to go to if the match is found.
Criteria	Allows for the update of the criteria string for an entry.

5. External_Sub Process Steps

5. External_Sub Process Steps

These are the steps available in the External/Sub Process test steps list in the New Step menu:



The following topics are available in this chapter.

- 5.1 Execute External Command
- 5.2 File System Snapshot
- 5.3 Execute Sub Test Case
- 5.4 Execute Sub Process
- 5.5 Execute JUnit Test Case_Suite
- 5.6 Read a File (Disk, URL or Class Path)
- 5.7 FTP Step

5.1 Execute External Command

5.1 Execute External Command

The Execute External Command step allows you to execute an external program, such as an OS script, an OS command, or an executable, and capture its contents for filtering or making assertions. The external program syntax will depend on your platform.

The external command editor is shown below:

Basic Info

Execute from directory:

Time Out (Seconds): If timeout: If environment error:

☐ Allow Properties: ☒ Wait For Completion ☐ Kill at Test End ☐ Spawn Process ☐ Exec Shell

☐ Append to Environment

Command Line

{command}

Environment Variables (empty for default)

Key	Value
-----	-------

Find:

Exit Codes (csv list, use -1 in last row as a catch-all)

Exit Codes (csv list, use -1 ...	Next Step
----------------------------------	-----------

Find:

Editor Results

Enter the following parameters:

- Execute from directory: The directory that will be considered current when the external command is executed. If the directory does not exist on the machine that is running the test, the directory will be created, subject to file system permissions. If the directory does not exist and cannot be created the step will fail.
- Time Out (Seconds): How long LISA should wait before transferring to the step defined by On Time Out Execute.
- If timeout: The step to execute if the external command does not complete execution before the given time out value.
- If environment error: The step to execute when a LISA environment error occurs.
- Allow Properties: This checkbox determines whether Properties are allowed for the next four parameters. It changes the look of the command editor interface.

1. With the Allow Properties check box unchecked you will see four check boxes as shown in the figure above. Here your only choice is to select the parameter or not.

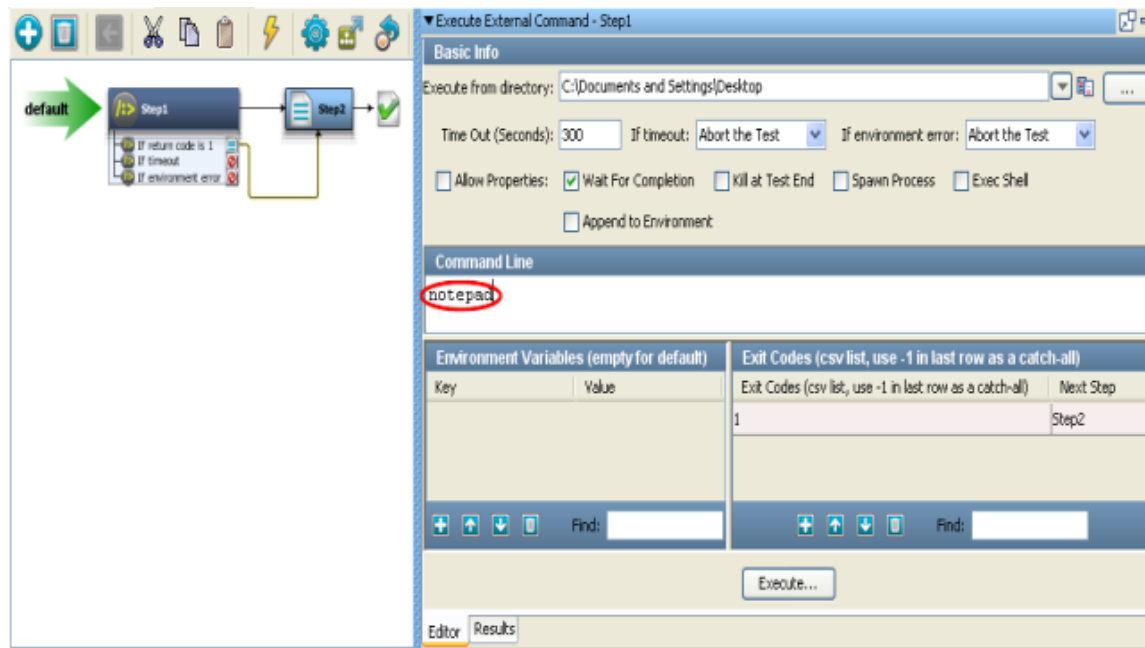
- Wait for Completion: The step waits until the execution has completed, and then the results can be filtered or asserted upon. If this checkbox is not selected filters and assertions will not be executed.
- Kill at Test End: can be used to kill the process once the test case has completed, if the Wait for Completion box is not checked. This allows a process to run while a test case executes and then be shutdown. A LISA property will contain the process ID of the started command.
- Spawn Process: creates a new process in the operating system to run the command in. This is helpful if you want to have a long running background process or need to make sure that a new set of environment variables is set and nothing set by LISA is in your environment.
- Exec Shell: allows you to run the contents of the Command Line within a system shell. This is required if you need to use the features of a shell process such as redirecting (pipe) output streams to files or other commands. Depending upon your system this option may be required for you to run system commands like dir or ls. Windows operating systems need this to be checked.
- Append to Environment: allows you to run the contents of the Command Line within a system shell. This is required if you need to use the features of a shell process such as redirecting (pipe) output streams to files or other commands. Depending upon your system this option may be required for you to run system commands like dir or ls. Windows operating system needs this to be checked.

With the Allow properties check box checked you will see pull down menus which have the same functionality as above but each parameter can

now be a property as shown in the figure below:

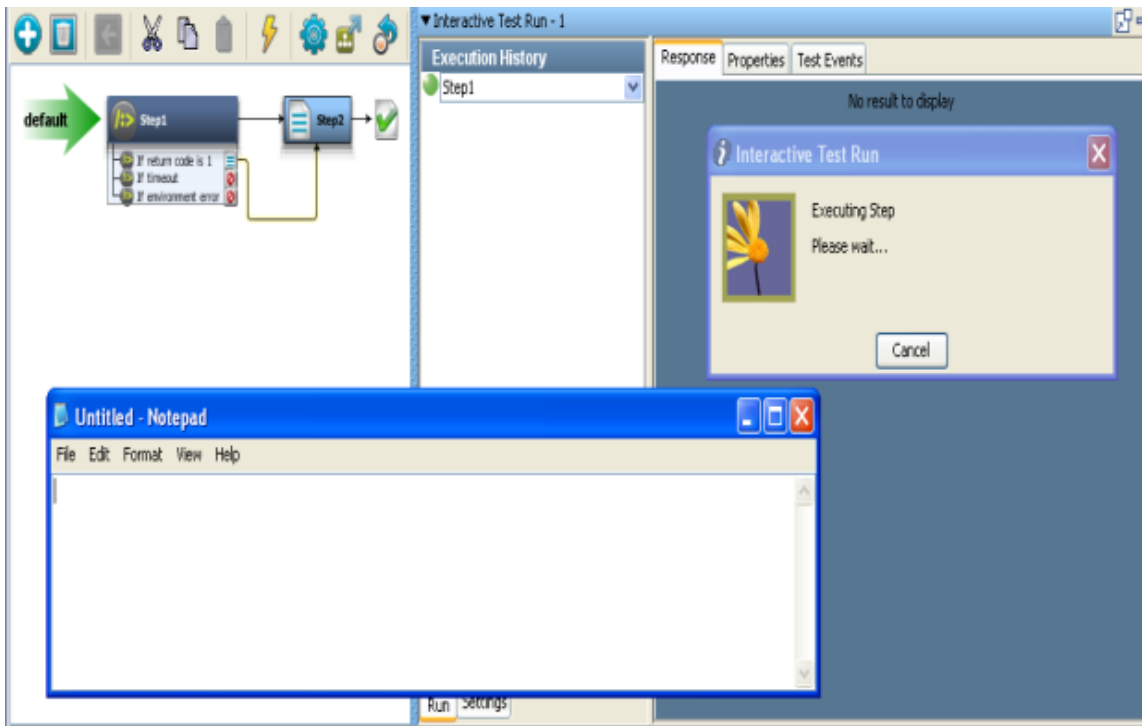
<input checked="" type="checkbox"/> Allow Properties:	Wait: true	Kill: false
	Spawn: false	Shell: false

- Wait: Wait for Completion
- Spawn: Spawn Process
- Kill: Kill at Test End
- Shell: Exec Shell



- Command Line: The external command is generally just one command written as a shell script or batch file. It is possible to execute multiple commands when the Exec Shell option is also selected. The command string must be valid for the platform that you are running LISA upon.
- Environment Variables: allows existing environment variables to be overridden with new environment variables. If nothing is entered the existing environment variables are used for the command. If an environment variable is specified, the new variables sets are used; the environment variables that were used to start LISA, is not used.
- Exit Codes: allows you to change the outcome of the test based upon the exit code of the executed process. You may enter a comma-delimited string of exit codes along with a corresponding step to execute when the process exits with this code.

To test your command, press the **Execute** button. Here is a sample of its output:

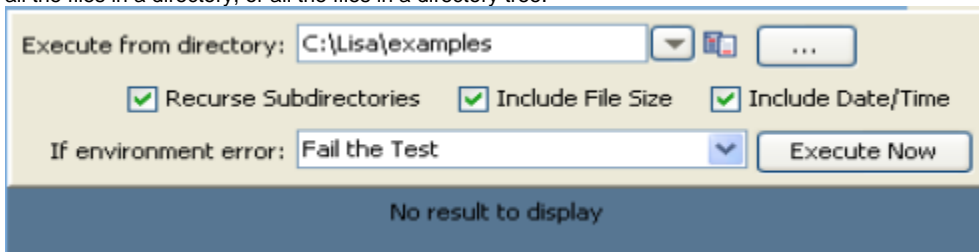


The content is now available for you to filter and add assertions.

5.2 File System Snapshot

5.2 File System Snapshot

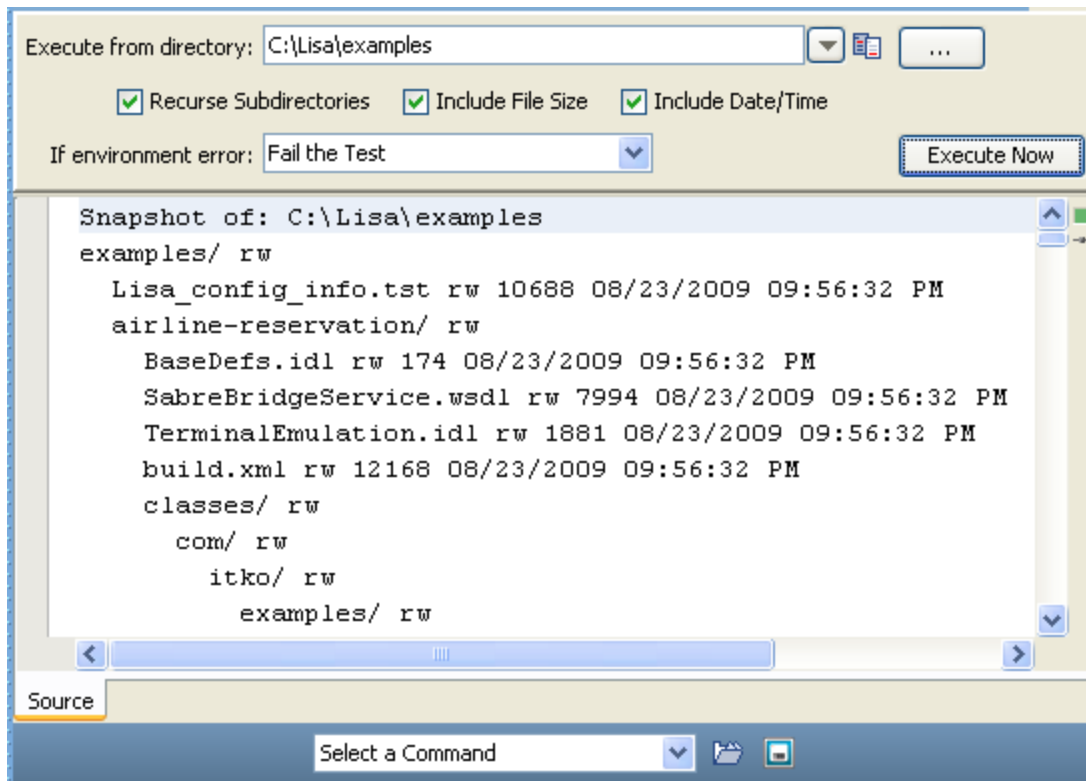
The File System Snapshot step allows you to list the files in a directory in a format that is operating system independent. You can list a single file, all the files in a directory, or all the files in a directory tree:



Enter the following parameters:

- Execute from directory: Enter the path name or browse to the file or directory.
- Recurse Subdirectories: Check if you want the complete directory tree.
- Include File Size: Check if you want the file sizes to be listed.
- Include Date/Time: Check if you want the last modified date to be listed.
- If environment error: Select an action to be carried out when there is an environment error.

Click **Execute Now** to list the files.



The content is now available for you to filter and add assertions.

5.3 Execute Sub Test Case

5.3 Execute Sub Test Case


This step has been deprecated and its use is discouraged. Use the Execute Sub Process step instead.

The Execute Sub Test Case step allows you to run part of another test case as a single step in the current test case.

Prerequisite – A knowledge of the external test case to be used as the sub-test.

The figure below shows the configuration screen for the sub-test:

Sub Test Case Info

File Name:  Browse

Start Step:

Execute: ☒ Just This Step ☐ From Here to End of Sub Test

☐ Read Events from Sub Into This Test






☐ Send this Test State to Sub

☐ Update this Test State from Sub

If environment error:

Parameters

Key	Value
unique_user	{{UNIQUE_USER}}


    Find: 

Enter the following parameters:

- File Name: Enter the name or browse to the external test case to be used as the sub-test.
- Open: Click to open the sub-test specified in the File Name text box.

Click the **Refresh** button.

- Start Step: Select the step to start from in the sub-test case
- Execute: Choose whether to run a single step or run to the end of the sub-test case
- Read Events from Sub Into This Test: Click if desired.
- Send this Test State to Sub: Click if desired.
- Update this Test State form Sub: Click if desired.
- If environment error: Select step to redirect to on failure of the sub test.

In the Parameters panel use the Add,  , button to add any LISA properties that you want to pass to the sub-test.

5.4 Execute Sub Process

5.4 Execute Sub Process

The Execute Sub Process step allows you to execute a sub Process as a single step. For more information on Sub Processes, and how to create them, see Chapter 15 Building Sub Processes in the [LISA User Guide](#).

Parameter Requirements - Knowledge of the input and output requirements needed to execute the Sub Process.

Sub Process: C:\Lisa\examples\multi-tier-combo.tst Browse Open

Configuration: localhost

Options

☐ Fully Expand Props

☐ Send Configuration

HTTP Cookies

☐ Send HTTP Cookies

☐ Get HTTP Cookies

Event Filter: No Filter

If environment error: Fail the T

Documentation:

Test Case

multi-tier-combo.tst

Parameters to Sub Process

Key	Description	Value
unique_user	unique_user	multitier-1519410445

Result Properties

☐ DBCONNURL

☐ DBDRIVER

☐ DBPASSWORD

☐ DBUSER

☐ JNDIFACTORY

☐ JNDIPORT


☐ JNDIPROTOCOL

☐ PORT

☐ SERVER

☐ WSPORT

Find:



Enter the following parameters:

- Sub Process: Enter the name, select from the pull-down menu, or browse to the LISA test case that is the sub process.
- Open: The open button can be used to open the subprocess test case. The subprocess test case is opened under a new project
- Configuration: Enter the name of an external configuration file if you want to use one. If not, see the Send Configuration option below.
- Options
- Fully Expand Props: when parameters have nested properties fully expand the properties before sending to the sub process.
- Send Configuration: Check, if you want the sub-process to use the configuration file that is active in your test case (the one you are building). If this is unchecked, the configuration in the sub-process will be used (unless an external file was entered in the textbox above)
- HTTP Cookies
- Send HTTP Cookies: Check if you want to forward Cookies to the sub-process
- Get HTTP Cookies: Get HTTP Cookies from the sub process.
- Event Filter: Select one of the event filter sets in the pull down menu. This filter set will apply to the events in the sub-process.
- If environment error: Select the step to redirect to if the sub-process fails.

Click the Refresh,  button at the bottom of the panel to refresh the parameters.

In the Parameters to Sub Process panel you will see a list of the parameters required by the sub-process. These keys and values must be present in your current test case. Edit the values column as needed to supply the correct values.

On the right of the Sub Process information screen, LISA displays the documentation that was entered into the Documentation of the sub-process test case.

LISA also lists all of the properties produced in the sub-process in Result Properties. You should select the properties you want returned from the sub-process. These will be used in your test case. Note that you are not limited to a single return value.

When this step is executed it will appear to run as a single step. In the Interactive Test Run (ITR) you will be able to see the events fired during the execution of the step. The short name of the events will be a combination of the name of the current step plus the name of the sub-process step where the event was fired.

An example of the Test Events list in the ITR is shown below:

Response Properties Test Events			
EventID	Timestamp	Short	Long
Step started	Wed Sep 02 12:58:35 IST 2009	Subprocess step	
Subprocess ran	Wed Sep 02 12:58:35 IST 2009	Subprocess step	TestNode name Subprocess step and type co...
Property set	Wed Sep 02 12:58:35 IST 2009	lisa.subprocess.C:\lisa\examples\multi-tier-combo.tst.xml...	com.itko.lisa.test.TestCaseDocument@622948
Step request	Wed Sep 02 12:58:41 IST 2009	Subprocess step	C:\lisa\examples\multi-tier-combo.tst
Property set	Wed Sep 02 12:58:41 IST 2009	Subprocess step-unique_user	multitier-65623833373231662D336530612D3...
Data set read	Wed Sep 02 12:58:41 IST 2009	Subprocess step-unique_user	unique_user {unique_user=multitier-6562383...
Property set	Wed Sep 02 12:58:41 IST 2009	Subprocess step-lisa.hidden.LisaFilterBase.processedObject	false
Property set	Wed Sep 02 12:58:41 IST 2009	Subprocess step-lisa.http.persistentHeader	
Property set	Wed Sep 02 12:58:41 IST 2009	Subprocess step-lisa.hidden.lisaint.support	true
Info message	Wed Sep 02 12:58:42 IST 2009	Subprocess step-start	Starting LISA example test case multi-tier-co...
Assert evaluated	Wed Sep 02 12:58:42 IST 2009	Subprocess step-start [SimpleWebAssertion1]	Assert of type [Simple Web Assertion] evalua...
Log message	Wed Sep 02 12:58:42 IST 2009	Subprocess step-Will execute the default next step	
Step started	Wed Sep 02 12:58:42 IST 2009	Subprocess step-new user	
Property set	Wed Sep 02 12:58:42 IST 2009	Subprocess step-lisa.hidden.LisaFilterBase.processedObject	false
Property set	Wed Sep 02 12:58:42 IST 2009	Subprocess step-lisa.hidden.lisaint.support	true
Log message	Wed Sep 02 12:58:42 IST 2009	Subprocess step-new user	THINK for 1325
Property set	Wed Sep 02 12:58:43 IST 2009	Subprocess step-lisa.new user.http.responseCode	200
Property set	Wed Sep 02 12:58:43 IST 2009	Subprocess step-lisa.new user.http.headers	Server=Apache-Coyote/1.1&X-Powered-By=...
Step request	Wed Sep 02 12:58:43 IST 2009	Subprocess step-new user	GET /itko-examples/user-manage.jsp?cmd=n...
Step target	Wed Sep 02 12:58:43 IST 2009	Subprocess step-new user	http://localhost:8080/itko-examples/user-ma...

Notice the new user notation in the Short column.

5.5 Execute JUnit Test Case_Suite

5.5 Execute JUnit Test Case_Suite

The Execute JUnit Test Case/Suite allows you to run a JUnit test case or a JUnit test suite in a LISA step. If the JUnit test pass, then so does the LISA step. On failure you can redirect to another LISA step.

Prerequisite - Your JUnit test must be on the classpath. Drop them into the hot deploy directory.

Test class:
...

If environment error:

Load
Execute

Suite Results

com.itko.examples.junit.ComboE:

```

.E
Time: 16.188
There was 1 error:
1) testExecuteIt (com.itko.examplo
    at com.itko.examples.junit.C
    at sun.reflect.NativeMethodA
    at sun.reflect.NativeMethodA
    at sun.reflect.DelegatingMet
    at org.junit.internal.runner:
    at junit.framework.JUnit4Tes
    at com.itko.lisa.junit.ExecJ
    at com.itko.lisa.junit.ExecJ
    at com.itko.lisa.junit.ExecJ

```

Enter the following parameters:

- Test Class: Enter the package name of the JUnit test class or test suite class. You can browse the classpath using the browse button. This will also confirm that your class is on your classpath.
- IF environment error: Select the step to redirect to if the JUnit test fail.

Click the **Load** button to load the class files. The class tree is displayed in the left panel.

Click the **Execute** button to execute the JUnit test. The standard JUnit results are displayed in the right panel.

The figure above shows the editor after the test case has been executed. The text in the right hand panel is set as the response for this step and is available for you to add filters and assertions.

5.6 Read a File (Disk, URL or Class Path)

5.6 Read a File (Disk, URL or Class Path)

The Read a File step reads a file from your file system, a URL, or the classpath. You can read a text file or a binary file. The contents of the file can optionally be stored in a LISA property.

File:

Property Key [opt]:

☐ Load as byte[]

If environment error:

The properties are stored in a comma delimited t

Source

Enter the following parameters:

- File: Enter the pathname, a URL, or a class path, or, browse to the file using the Browse button.
- Property Key [opt]: Enter the name of the property to store the file contents (optional).
- Load as Byte []: Check this checkbox to load contents as a byte array. (useful when loading a file to be used as a binaryData type in a web service execution parameter):
- If environment error: Select the step to redirect to if the Read a File test fail.
- Display as characters: Contents will be displayed as hexadecimal encoded bytes unless you check this checkbox. This checkbox is only visible if the Load as Byte [] box is checked.

Click the **Load** button to load and display the file. The content is now ready for you to filter and add assertions.

The figures below show a small binary file displayed as bytes and characters respectively:

File:

Property Key [opt]:

☒ Load as byte[]

If environment error:

☐ Display as characters

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
54	68	65	20	70	72	6F	70	65	72	74	69	65	73	20	61	72	65	20	73	74	6F	72	65	64	20	69	6E	20	61	20	63
6F	6D	6D	61	20	64	65	6C	69	6D	69	74	65	64	20	74	65	78	74	20	66	69	6C	65	20	77	68	65	72	65	20	74
68	65	20	66	69	72	73	74	20	6C	69	6E	65	20	69	6E	20	74	68	65	20	66	69	6C	65	20	69	73	20	74	68	65
20	73	65	74	20	6F	66	20	70	72	6F	70	65	72	74	79	20	6E	61	6D	65	73	20	62	65	69	6E	67	20	73	61	76
65	64	2C	20	61	6E	64	20	74	68	65	20	73	65	63	6F	6E	64	20	6C	69	6E	65	20	63	6F	6E	74	61	69	6E	73
20	74	68	65	20	76	61	6C	75	65	73	20	63	6F	72	72	65	73	70	6F	6E	64	69	6E	67	20	74	6F	20	65	61	63

File:

Property Key [opt]:

☒ Load as byte[]

If environment error:

☒ Display as characters

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
T	h	e		p	r	o	p	e	r	t	i	e	s		a	r	e		s	t	o	r	e	d		i	n		a		c
o	m	m	a	d	e	l	i	m	i	t	e	d		t	e	x	t		f	i	l	e		w	h	e	r	e		t	
h	e		f	i	r	s	t		l	i	n	e		i	n		t	h	e		f	i	l	e		i	s		t	h	e
s	e	t		o	f		p	r	o	p	e	r	t	y		n	a	m	e	s		b	e	i	n	g		s	a	v	
e	d	,		a	n	d		t	h	e		s	e	c	o	n	d		l	i	n	e		c	o	n	t	a	i	n	s
t	h	e		v	a	l	u	e	s		c	o	r	r	e	s	p	o	n	d		i	n	g		t	o		e	a	c

Note: If a binary file is loaded but you do not choose to Load as byte[], LISA will convert the binary data into characters, many of which will be unreadable, and the step response will be a String.

Note: In previous versions of LISA, this step was known as "Read Result from Stream".

5.7 FTP Step

5.7 FTP Step

The FTP Step allows you to send or receive a file using ftp protocol. After entering the ftp server name, user name, and user password you can either upload a file or download a file.

User:

Password:

Host:

Source Path:

Destination Path:

Transfer Type: If environment error: ☐ Upload

No result to display

Enter the following parameters:

- User: Enter the user name for ftp server access.
- Password: Enter the password for ftp server access.
- Host: Enter the host name of the ftp server (without the protocol).
- Source Path: Enter the path to the source file (either on ftp server or local machine).
- Destination Path: Enter the path for the destination file (either on ftp server or local machine).
- Transfer Type: Select the file transfer type; Binary or ASCII.
- If environment error: Select the step to redirect to in the advent of an error.
- Upload: Check this box for an ftp upload, leave unchecked for an ftp download.

Click the **Execute Now** button to initiate the send/receive action.

The content is now available for you to filter and add assertions.

The response from a download is the file itself. The response from a successful upload is the string **success**.

Note: If a file by the same name already exists, it will be overwritten without a warning message.

The figure above shows an example of receiving a file from an ftp server. The figure below shows an example of uploading a file via ftp.

User:

Password:

Host:

Source Path:

Destination Path:

Transfer Type: If environment error: ☒ Upload

No result to display

6. JMS Messaging Steps

6. JMS Messaging Steps

The following topics are available in this chapter.

6.1 JMS Messaging (JNDI)
6.2 Message Consumer

6.1 JMS Messaging (JNDI)

6.1 JMS Messaging (JNDI)

The JMS Messaging (JNDI) step allows you to send messages to, and receive messages from topics and queues. You can also receive, modify and forward an existing message.

LISA supports all the common message types including Empty, Text, Object, Bytes, Message and Mapped (Extended).

The JMS Messaging (JNDI) step is configured regardless of the messaging requirements. Input options will vary on the messaging requirements. The editor will only allow valid configurations, so when you enable certain features others may become inactive.

Prerequisites - You will be required to supply the necessary jar files for your chosen configuration and connection. They will typically need to be provided under the hotDeploy directory.

Parameter Requirements - You need the connection parameters and the Queue/Topic names used in an application under test. There may be other parameters required, depending on your environment. These should be obtained in advance from the developers of the application. In most cases you can browse for server resources to obtain some of these needed parameters.

Messaging Using JNDI and JMS

An example test case, **jms.tst**, is included in the LISA examples directory. You are encouraged to look at that test case to illustrate what is being described in this section. We will use the inputs required by that test case in our descriptions here.

The example test uses a publish/subscribe JMS step to send a message and listen on a temporary queue. The message is handled by a Message Driven Bean (MDB) on the server which drops the message onto the temporary queue. The message type is text. It is an XML payload that is created by inserting LISA properties dynamically into the XML elements. The properties come from a dataset named **order_data**.

After the response message is received, the XML from the JMS message is put into a LISA property. The next step does an assertion validating the product ID. After this check asserts true we modify the existing message object, and finally, we send the message to another JMS destination.

This is a good example of how LISA could listen in and intercept messages as they move through a multi-point messaging service backbone. You can run this test case against the JBoss Demo Server on your machine. The application backend is available there.

The Messaging step editor is shown below:

There are four tabs available at the bottom of the editor.

- The **Base** tab is where you define your connection and messaging parameters.
- The **Selector Query** tab allows you to specify a selector query to be run when listening for a message on a queue.
- The **Send Message Data** tab is where you will create your message content.
- The **Response Message** tab is where your response messages will be posted.

Base tab

The Base tab view is shown in the figure above. It is divided into 5 major sections: Server Connection Info, Subscriber Info, Publisher Info, ReplyTo Info, and Error Handling and Test.

The **Server Connection Info**, **Error Handling and Test**, **Publisher Info** are always active. The **Subscriber Info** and **ReplyTo Info** sections can be enabled or disabled using the enable checkbox in the top left corner of each section. Using these checkboxes you can configure the step to be a publish step, a subscribe step, or both. You can also choose to include a "replyto" component in the step. When you have completely configured your test step, use the Test button in the Error Handling and Test section to test your configuration settings.

Server Connection Info

Here you enter the JNDI information.

These values should be parameterized with properties that are in your configuration, making it easy to change the application under test. An

example of this can be seen in the figure below:

Server Connection Info

JNDI Factory Class:

JNDI Server URL:

JMS ConnectionFactory:

User:

Password:

☐ Share Sessions

☐ Share Publishers

The first three parameters should be available to you for the system under test.

- JNDI Factory Class
- JNDI Server URL
- JMS Connection Factory: You can use the magnifying glass, , to browse available resources on the server. Select or type in a Connection Factory per the JMS specification to use for this Step execution.

The pull-down menus contain common examples or templates for these values.

The user and password may or may not be needed

- User
- Password
- Share Sessions and Share Publishers checkboxes are used to share JMS Sessions and Publishers throughout the test case. This can lower overhead, but does not always provide a realistic simulation since typically JMS clients want to release resources. Note: If you check Share Publishers, the Share Sessions checkbox is also checked for you as you cannot share publishers without sharing sessions.
- The **Stop All** button allows you to stop any listeners at design time now. This is to resolve issues during test case creation where some listeners can get orphaned, but will still consume messages making it difficult or sometimes impossible to finish test case creation.
- The **Advanced** button displays a pop-up panel where you can add custom properties that will be sent with the connection information.

Publisher Info

☒ enable ☐ use transaction


Name:


Type:

Message:

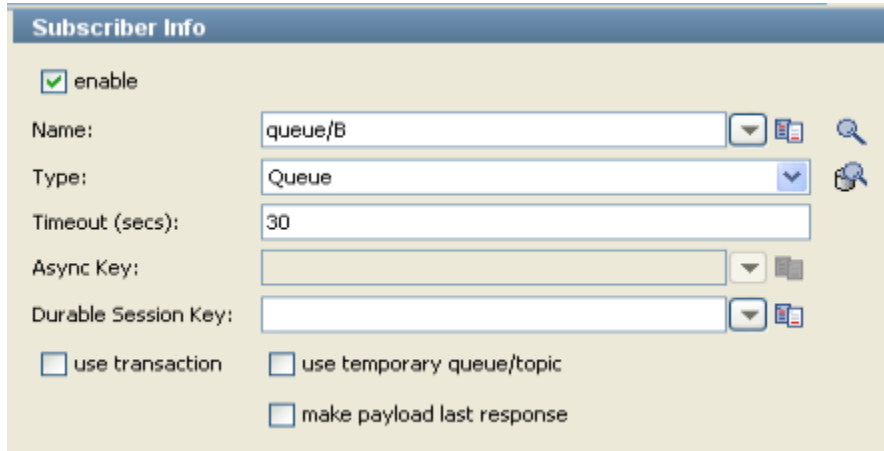
Check the enable checkbox to set up the ability to send (publish) messages. Click the use transaction checkbox to execute a commit when the message is sent.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.

- Type: Select whether you are using a topic or queue. The browse icon,  , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).
- Message: Select the type of message you are sending from the pull-down menu. The supported messages are: Empty, Text, Object, Bytes, Message, and Mapped (Extended).
- Advanced button: Displays a pop-up panel where you can add custom message properties to be sent with the message.



Subscriber Info



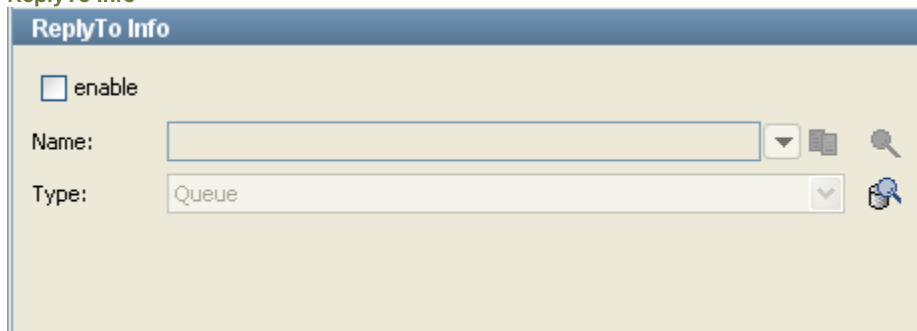
The Subscriber Info form has a title bar 'Subscriber Info'. It contains an 'enable' checkbox which is checked. Below it are five fields: 'Name' with value 'queue/B', 'Type' with value 'Queue', 'Timeout (secs)' with value '30', 'Async Key' (empty), and 'Durable Session Key' (empty). Each of these fields has a browse icon to its right. At the bottom are three checkboxes: 'use transaction' (unchecked), 'use temporary queue/topic' (unchecked), and 'make payload last response' (unchecked).

Check the enable checkbox to set up to enable the ability to receive (subscribe to) messages.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass,  , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue, and whether you want to listen in synchronous or asynchronous mode. For asynchronous mode you will also have to have an entry in the Async key field (see below). The browse icon,  , to the right of this field can be used to see what messages are waiting to be consumed from a queue.
- Timeout (secs): Enter the period to wait before LISA interrupts waiting for a message. (Use 0 for no timeout)
- Async Key: Enter the value needed to identify asynchronous messages. This is only needed in asynchronous mode. It will be used in a subsequent Message Consumer step to retrieve asynchronous messages.
- Durable Session Key: By entering a name here you are requesting a durable session. You are also providing a key for that session. A durable session allows you to receive all of your messages from a topic even if you logoff and then logon again.
- Use transaction checkbox: Click the use transaction checkbox to execute a Commit when a message is received.
- Use temporary queue/topic checkbox: Click the use temporary queue/topic checkbox if you want the JMS provider to set up a temporary queue/topic on your behalf. When a temporary queue/topic is used, LISA will automatically set the JMS ReplyTo parameter of the message you send to the temporary queue/topic. The temporary queue/topic feature must always be used in conjunction with a publisher so that a reply can be sent. If you use a temporary queue/topic the ReplyTo section is disabled by LISA.
- Make Payload Last Response: Click Make Payload Last Response check box if you want to make the payload response as a last response.

ReplyTo Info





The ReplyTo Info form has a title bar 'ReplyTo Info'. It contains an 'enable' checkbox which is unchecked. Below it are two fields: 'Name' (empty) and 'Type' with value 'Queue'. Each field has a browse icon to its right.

Check the enable checkbox to set up a destination queue/topic.

If your application needs a destination, it is setup in this section.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue. The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).

Error Handling and Test

Error Handling and Test

If environment error:

Error Handling and Test section allows you to redirect to a step if an error occurs.

- If environment error: Select the step to redirect to if an environment error occurs.

Click the Test button to test your step configuration settings.

Selector Query Tab

The Selector Query tab view is shown below:

JMS Selector Query

JMSCorrelationID LIKE '({lisa.jms.correlation.id})%'

You can enter a JMS selector query in this editor. The syntax closely follows SQL. It is a subset of SQL92. For further details see the JMS 1.1 specification, section 3.8. A JMS selector query can be specified when listening for a message on a queue that is a response to a published message. The example above shows a specific query looking for a JMSCorrelationID that matches one set in a LISA property lisa.jms.correlation.id as sent with the original message.

Note: LISA has a built in mechanism for allowing a test creator to set the JMSCorrelationID for a message before sending it. Anytime before the message is sent you can set the correlation ID by setting the following LISA property: lisa.jms.correlation.id. LISA will detect a non-zero value and set the message JMSCorrelationID property before the message is sent.

Send Message Data Tab

If your step is configured to publish, this is where you compose your message. The Send Message Data tab view is shown below for a text message:

Message Data

```

<order>
  <id>{{order_id}}</id>
  <name>{{order_user}}</name>
  <product>{{order_product}}</product>
</order>

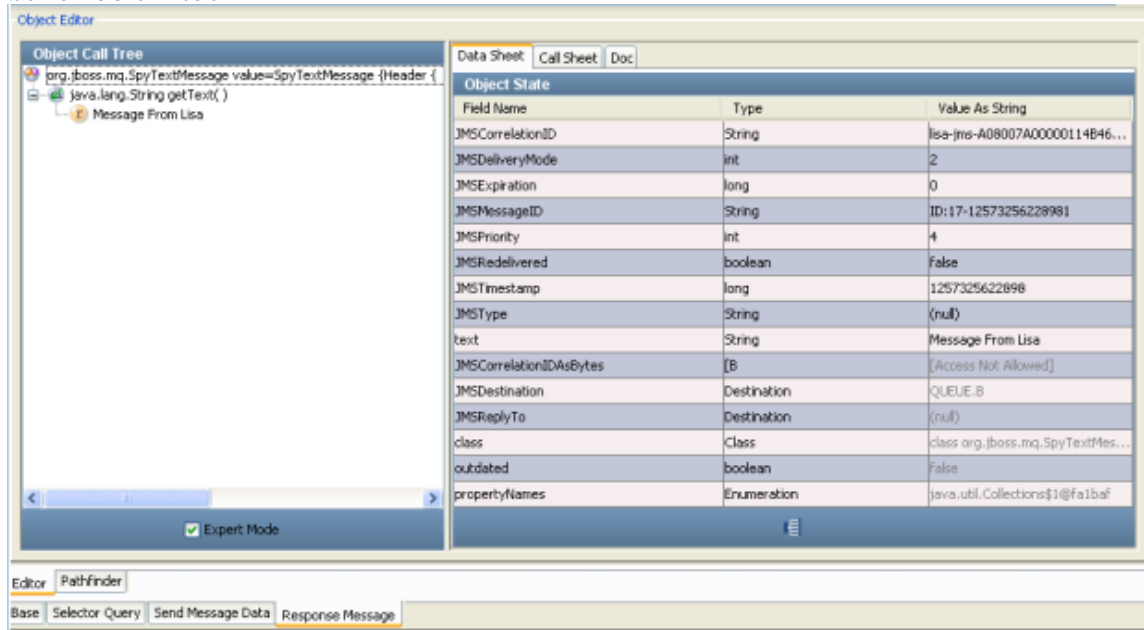
```

This particular example shows an XML fragment with LISA properties being used. The text can be typed in or it can be read from a file using the **Read Message from the File...** button in the bottom right corner, or it can be stored in a LISA property, in which case you would just place the LISA property in the editor i.e. something like `LISA_PROP`.

Notice that LISA properties are used in the message XML allowing the message to be created dynamically during the test run.

Response Message Tab

If your step is configured to subscribe, your response will be shown here after clicking the Test button in the Base View. The Response Message tab view is shown below:



Here you can see two tabs Editor and Pathfinder.

Editor: The tab that is open by default and shows the Complex Object Editor for the returned object. The object that is returned will vary with application server type. You have access to all the JMS parameters returned in addition to the message itself. The object will be loaded into LISA's Complex Object Editor where it can be manipulated like any other Java object. In the example above we interrogate a JBoss object to get the text response.

Pathfinder: Displays diagrammatically the calls made as a workflow. When you click on a node, the information of that node is displayed in the bottom portion of the editor. Pathfinder functionality is explained in more detail in the [LISA User Guide](#).

You are now ready to use filters, assertions and data sets to complete your step.

6.2 Message Consumer

6.2 Message Consumer

The Message Consumer step allows you to consume asynchronous messages in a test case. This step can attach to a known queue/topic and get messages posted for this subscriber. You identify yourself via a unique key. You must have already subscribed to the queue/topic and the messages have been pushed to that destination.

Prerequisite – Before executing the example test case you must have the demo server running.

Parameter Requirements - Knowledge of the queue/topic being used in the application under test.










An example test case, **async-consumer-jms.tst**, is included in the LISA examples directory. You are encouraged to look at that test case to illustrate what is being described in this section.

The create Message Consumer step subscribes to asynchronous message (topic/testTopic) using the Async Key (**EXAMPLE-ASYNC-WRAPPER**). The send-message step publishes message to a queue (queue/C). Number of messages to be published is controlled using the data set (counterA). Message consumer step has an Async queue (**EXAMPLE-ASYNC-WRAPPER**) using which message subscribed by the create-consumer step is consumed. Number of messages to be consumed is controlled using the data set (DatSetB).

Note: The Async Key specified in create-consumer and consumer step must match.

The figure below shows, an example of the subscriber section of a step.

Subscriber Info







☒ enable
Name:   
Type:  
Timeout (secs):
Async Key:  
Durable Session Key:  
☐ use transaction ☐ use temporary queue/topic
☐ make payload last response

Check the enable checkbox to set up, to enable the ability to listen to (subscribe to) messages.

Notice that there is an asynchronous topic specified in the Type field, and an Async Key parameter defined. This key will be needed as an input in the current step.



The figure below shows, an example of the publisher section of a step.

Publisher Info

☒ enable ☐ use transaction
Name:   
Type:  
Message: 

Check the enable checkbox to set up the ability to send (publish) messages. Click the use transaction checkbox to execute a commit when the message is sent.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue. The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).
- Message: Select the type of message you are sending from the pull-down menu. The supported messages are: Empty, Text, Object, Bytes, Message, and Mapped (Extended).
- Advanced button: Displays a pop-up panel where you can add custom message properties to be sent with the message.

The step editor is shown below:

Wrapper Queue Properties

Async Queue:

Wait Timeout (Seconds):

If environment error:

Wrapper Status: Current Wrapper Depth: 4 Total Wrappers: 1

☒ make payload last response

The next message on the Queue/Topic will show here when you execute.

Enter the following parameters:

- Async Queue: Enter or select the **Async Key** parameter that was named in a preceding subscriber step (**EXAMPLE-ASYNC-WRAPPER**). These names must match.
- Wait Timeout (Seconds): Enter the time, in seconds, that LISA should wait for the next message.
- If environment error: Select the step to redirect to if an environment error occurs.

Wrapper Status contains 2 output status values:

- Current Wrapper Depth: Number of messages left to be read in current wrapper
- Total Wrappers: Number of wrappers (destinations).
- Make payload last response: Select this option if you want to make the payload as the last response in the step.

If there are messages waiting they can be read by clicking the **Next Message** button. This will normally show the message in the LISA Complex Object Editor.

Wrapper Queue Properties

Async Queue:

Wait Timeout (Seconds):

If environment error:

Wrapper Status: Current Wrapper Depth: 0 Total Wrappers: 1

☒ make payload last response

Object Editor

Object Call Tree:

Data Sheet | Call Sheet | Doc

Field Name	Type	Value As String
JMSCorrelationID	String	(null)
JMSDeliveryMode	int	2
JMSExpiration	long	0
JMSMessageID	String	ID:25-12524133986131
JMSPriority	int	4
JMSRedelivered	boolean	false
JMSTimestamp	long	1252413398613
JMSType	String	(null)
JMSCorrelationIDAsBytes	[B	[Access Not Allowed]
JMSDestination	Destination	QUEUE.B

☒ Expert Mode

You are now ready to manipulate this object.

A wrapper is a FIFO list for holding responses from asynchronous topics and queues. It provides a place for the application to put responses for later consumption. Messages wait in this list for subsequent LISA processing (in this Message Consumer step).

7. Bea Steps

7. Bea Steps

The following steps are included in this chapter.

7.2 Weblogic JMS (JNDI)
7.3 Message Consumer
7.4 Read a File (Disk, URL, or Classpath)
7.5 Next Generation Web Service Execution (alpha)
7.6 Web Service Execution
7.7 Raw Soap Request
7.8 FTP Step

7.2 Weblogic JMS (JNDI)

7.2 Weblogic JMS (JNDI)

Weblogic JMS (JNDI) supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

The Weblogic JMS (JNDI) step allows you to send messages to, and receive messages from topics and queues. You can also receive, modify and forward an existing message.

The Weblogic JMS (JNDI) step is configured using a single LISA editor regardless the messaging requirements. Input options will vary on the messaging requirements. The editor will only allow valid configurations, so when you enable certain features others may become inactive.

Prerequisites - You will be required to supply the necessary jar files for your chosen configuration and connection.

Weblogic requires that the weblogic jar file be added to your CLASSPATH. You can put it in the hot deploy directory. There maybe other jar files required if you are using security or JMX. Check the weblogic documentation for more information on the jar files you might need. The jar files can be found in the **lib** directory of the Weblogic installation.

Parameter Requirements - You need the connection parameters and the subject names used in the application under test. The parameters needed are listed below in the Server Connection Info section. There may be other parameters required, depending on your environment. These should be obtained in advance from the developers of the application.

The messaging step editor for Weblogic JMS (JNDI) is shown below:

The screenshot shows the Weblogic JMS (JNDI) messaging step editor. It is divided into four main sections: Server Connection Info, Subscriber Info, ReplyTo Info, and Publisher Info. The Server Connection Info section includes fields for JNDI Factory Class (weblogic.indi.WLInitialContextFactory), JNDI Server URL (t3://{{WLS_SERVER}}:7001), JMS ConnectionFactory (QueueFactory), User, and Password. It also has checkboxes for Share Sessions and Share Publishers, and buttons for Stop All and Advanced. The Subscriber Info section has an enable checkbox, Name (LISATestQueue), Type (Queue), Timeout (secs) (30), Async Key, Durable Session Key, and checkboxes for use transaction, use temporary queue/topic, and make payload last response. The ReplyTo Info section has an enable checkbox, Name, and Type (Queue). The Publisher Info section has an enable checkbox, use transaction checkbox, Name (LISATestQueue), Type (Queue), and Message (Empty). At the bottom, there is an Error Handling and Test section with a dropdown for 'If environment error' (Fail the Test) and a Test... button. The bottom of the editor shows four tabs: Base, Selector Query, Send Message Data, and Response Message.

There are four tabs available at the bottom of the editor.

- The **Base** tab is where you define your connection and messaging parameters.
- The **Selector Query** tab allows you to specify a selector query to be run when listening for a message on a queue.
- The **Send Message Data** tab is where you will create your message content.

- The [Response Message](#) tab is where your response messages will be posted.

Base tab

The Base tab view is shown in the figure above. It is divided into 5 major sections:

- [Server Connection Info](#)
- [Subscriber Info](#)
- [Publisher Info](#)
- [ReplyTo Info](#)
- [Error Handling and Test](#)

The [Server Connection Info](#), [Error Handling and Test](#), [Publisher Info](#) are always active. The [Subscriber Info](#) and, [ReplyTo Info](#) sections can be enabled or disabled using the enable checkbox in the top left corner of each section.

Using these checkboxes you can configure the step to be a publish step, a subscribe step, or both. You can also choose to include a "replyto" component in the step. When you have completely configured your test step, use the Test button in the Error Handling and Test section to test your configuration settings.


Server Connection Info

The screenshot shows the 'Server Connection Info' configuration window. It has a title bar with the text 'Server Connection Info'. Below the title bar, there are several input fields and checkboxes. The 'JNDI Factory Class' field contains 'weblogic.jndi.WLInitialContextFactory'. The 'JNDI Server URL' field contains 't3://{{WLS_SERVER}}:7001'. The 'JMS ConnectionFactory' field contains 'QueueFactory'. There are empty fields for 'User' and 'Password'. Below these fields are two checkboxes: 'Share Sessions' and 'Share Publishers', both of which are unchecked. At the bottom right, there are two buttons: 'Stop All' and 'Advanced'.

Enter the JNDI information.

These values should be parameterized with properties that are in your configuration, making it easy to change the application under test. LISA, by default uses the `WLS_SERVER` property in the JNDI Server URL. This property must be added to your Configuration if you plan to use it.


The five parameters should be available to you for the system under test. The pull-down menus contain common examples or templates for these values.


- JNDI Factory Class: LISA pre-populates this field with the default values.
- JNDI Server URL: LISA pre-populates this field with the default values.
- JMS Connection Factory: You can use the magnifying glass, , to browse available resources on the server. Select or type in a Connection Factory per the JMS specification to use for this Step execution.
- User
- Password
- Share Sessions and Share Publishers checkboxes are used to share JMS Sessions and Publishers throughout the test case. This can lower overhead, but does not always provide a realistic simulation since typically JMS clients want to release resources. Note: If you check Share Publishers, the Share Sessions checkbox is also checked for you as you cannot share publishers without sharing sessions.
- The Stop All button allows you to stop any listeners at design time now. This is to resolve issues during test case creation where some listeners can get orphaned, but will still consume messages making it difficult or sometimes impossible to finish test case creation.
- The Advanced button displays a pop-up panel where you can add custom properties that will be sent with the connection information.

Publisher Info

Publisher Info

☒ enable ☐ use transaction



Name: 

Type: 

Message:

Check the enable checkbox to set up the ability to send (publish) messages. Click the use transaction checkbox to execute a commit when the message is sent.


Enter the following parameters:


- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue. The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).
- Message: Select the type of message you are sending from the pull-down menu. The supported messages are: Empty, Text, Object, Bytes, Message, and Mapped (Extended).
- Advanced button: Displays a pop-up panel where you can add custom message properties to be sent with the message.

Subscriber Info


Subscriber Info


☒ enable

Name: 

Type: 

Timeout (secs):

Async Key: 



Durable Session Key: 

☐ use transaction ☐ use temporary queue/topic

☒ make payload last response

Check the enable checkbox to set up to enable the ability to receive (subscribe to) messages.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue, and whether you want to listen in synchronous or asynchronous mode. For asynchronous mode you will also have to have an entry in the Async key field (see below). The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).
- Timeout (secs): Enter the period to wait before LISA interrupts waiting for a message (this field can be left blank for no timeout).
- Async Key: Enter the value needed to identify asynchronous messages. This is only needed in asynchronous mode. It will be used in a subsequent Message Consumer step to retrieve asynchronous messages.
- Durable Session key: By entering a name here you are requesting a durable session. You are also providing a key for that session. A durable session allows you to receive all of your messages from a topic even if you logoff, and then logon again.
- use transaction checkbox: Click the use transaction checkbox to execute a Commit when a message is received.
- use temporary queue/topic checkbox: Click the use temporary queue/topic checkbox if you want the JMS.provider to set up a temporary queue/topic on your behalf. When a temporary queue/topic is used, LISA will automatically set the JMS ReplyTo parameter of the message you send to the temporary queue/topic. The temporary queue/topic feature must always be used in conjunction with a publisher so that a reply can be sent. If you use a temporary queue/topic the ReplyTo section is disabled by LISA.

- make payload last response: Check this option if you want to make payload as the response of this step.

ReplyTo Info





The 'ReplyTo Info' dialog box has a title bar 'ReplyTo Info'. Inside, there is an 'enable' checkbox. Below it, there are two fields: 'Name:' with a text input box and a magnifying glass icon to its right, and 'Type:' with a dropdown menu showing 'Queue' and a browse icon to its right.

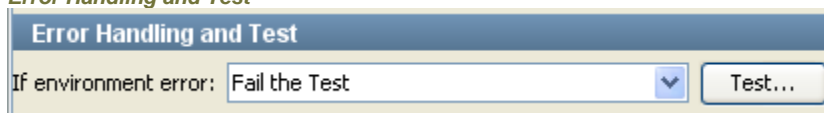
Check the enable checkbox to set up a destination queue/topic.

If your application needs a destination, it is setup in this section.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue. The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).

Error Handling and Test



The 'Error Handling and Test' dialog box has a title bar 'Error Handling and Test'. Inside, there is a label 'If environment error:' followed by a dropdown menu showing 'Fail the Test' and a 'Test...' button.

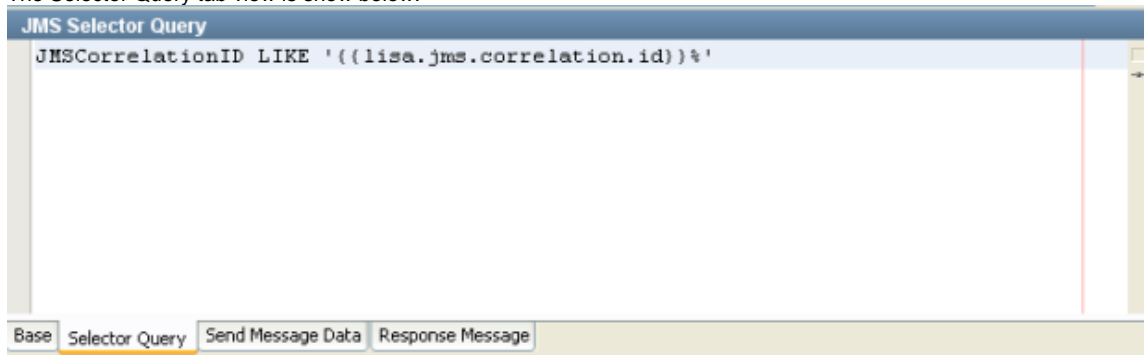
Error Handling and Test allows you to redirect to a step if an exception occurs.

- If environment error: Select the step to redirect to if an error occurs.

Click the Test button to test your step configuration settings.

Selector Query Tab

The Selector Query tab view is show below:



The 'JMS Selector Query' editor has a title bar 'JMS Selector Query'. The main area contains a text input field with the query: `JMSCorrelationID LIKE '({lisa.jms.correlation.id})%'`. At the bottom, there are four tabs: 'Base', 'Selector Query' (which is selected), 'Send Message Data', and 'Response Message'.

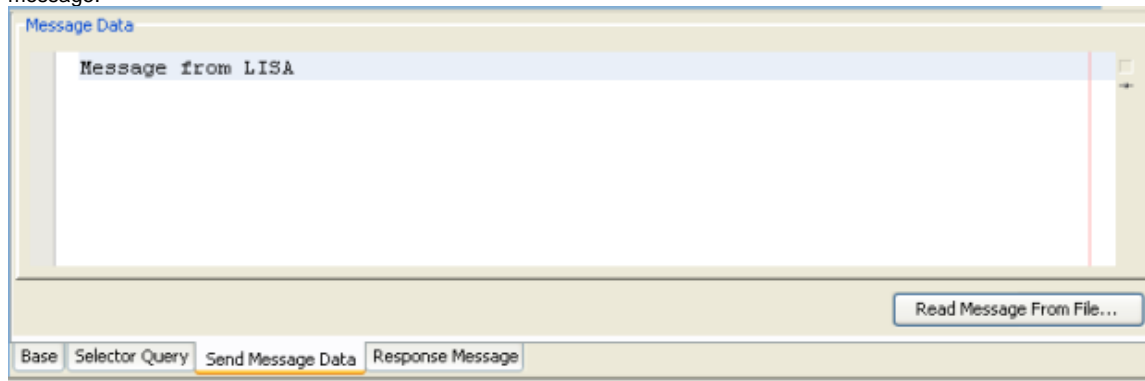
You can enter a JMS selector query in this editor. The syntax closely follows SQL. It is a subset of SQL92. For further details see the JMS 1.1 specification, section 3.8. A JMS selector query can be specified when listening for a message on a queue that is a response to a published message. The example above shows a specific query looking for a JMSCorrelationID that matches one set in a LISA property as sent with the original message.

LISA has a built in mechanism for allowing a test creator to set the JMSCorrelationID for a message before sending it. Anytime before the message is sent you can set the correlation ID by setting the following LISA property:
lisa.jms.correlation.id

LISA will detect a non-zero value and set the message JMSCorrelationID property before the message is sent.

Send Message Data tab

If your step is configured to publish, this is where you compose your message. The Send Message Data tab view is shown below for a text message:



This particular example shows an XML fragment with LISA properties being used. The text can be typed in or it can be read from a file using the **Read Message from the File** button in the bottom-right corner, or it can be stored in a LISA property, in which case you would just place the LISA property in the editor, for example `LISA_PROP`.

Notice that LISA properties are used in the message XML allowing the message to be created dynamically during the test run.

Response Message Tab

If your step is configured to subscribe, your response will be shown here. For more information refer to [6.1 JMS Messaging \(JNDI\)](#).

7.3 Message Consumer

7.3 Message Consumer

For detailed information regarding this step, please refer to [6.2 Message Consumer](#).

7.4 Read a File (Disk, URL, or Classpath)

7.4 Read a File (Disk, URL, or Classpath)

For detailed information regarding this step, please refer to [5.6 Read a File \(Disk, URL or Class Path\)](#).

7.5 Next Generation Web Service Execution (alpha)

7.5 Next Generation Web Service Execution (alpha)

For detailed information regarding this step, please refer to [1.3 Web Service Execution \(XML\)](#).

7.6 Web Service Execution

7.6 Web Service Execution

For detailed information regarding this step, please refer to [1.9 Web Service Execution \(Legacy\)](#).

7.7 Raw Soap Request

7.7 Raw Soap Request

For detailed information regarding this step, please refer to [1.5 Raw SOAP Request](#).

7.8 FTP Step

7.8 FTP Step

For detailed information regarding this step, please refer to [5.7 FTP Step](#).

8. Sun JCAPS Steps

8. Sun JCAPS Steps

The following steps are included in this chapter.

- 8.1 JCaps Messaging (Native)
- 8.2 JCaps Messaging (JNDI)
- 8.3 Message Consumer
- 8.4 Read a File (Disk, URL, or Classpath)
- 8.5 Next Generation Web Service Execution (alpha)
- 8.6 Web Service Execution
- 8.7 Raw Soap Request
- 8.8 SQL Database Execution
- 8.9 FTP Step

8.1 JCaps Messaging (Native)

8.1 JCaps Messaging (Native)

The JCaps Messaging (Native) step allows you to send messages to, and receive messages from topics and queues. You can also receive, modify and forward an existing message.

JCaps Messaging (Native) supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

The JCaps Messaging (Native) step is configured using a single LISA editor regardless the messaging requirements. Input options will vary on the messaging requirements. The editor will only allow valid configurations, so when you enable certain features others may become inactive.

Prerequisites - You will be required to supply the necessary jar files for your chosen configuration and connection.

JCaps messaging requires that several jar files be added to your CLASSPATH. You can put them in the hot deploy directory. Check the JCaps documentation for more information on the jar files you might need. The jar files can be found in the lib directory of the JCaps installation.

Parameter Requirements - You need the connection parameters and the subject names used in the application under test. The parameters needed are listed below in the Server Connection Info section. There may be other parameters required, depending on your environment. These should be obtained in advance from the developers of the application.

The messaging step editor for JCaps Messaging (Native) is shown below:

The screenshot shows the 'Base' tab of the editor, which is divided into five main sections:

- Server Connection Info:** Host: localhost, Port: 18007. Includes an 'Advanced' button.
- Subscriber Info:** ☒ enable. Name: LISATestQueue, Type: Queue, Timeout (secs): 30, Async Key: (empty), Durable Session Key: (empty). Checkboxes: ☐ use transaction, ☐ use temporary queue/topic, ☒ make payload last response.
- Publisher Info:** ☒ enable, ☐ use transaction. Name: LISATestQueue, Type: Queue, Message: Empty. Includes an 'Advanced' button.
- ReplyTo Info:** ☐ enable. Name: (empty), Type: Queue.
- Error Handling and Test:** If environment error: Fail the Test, Test... button.

At the bottom, there are four tabs: Base (selected), Selector Query, Send Message Data, and Response Message.

There are four tabs available at the bottom of the editor.

- The Base tab is where you define your connection and messaging parameters.
- The Selector Query tab allows you to specify a selector query to be run when listening for a message on a queue.
- The Send Message Data tab is where you will create your message content.
- The Response Message tab is where your response messages will be posted.

Base Information (Base tab)

The Base tab view is shown in the figure above. It is divided into 5 major sections:

1. Server Connection Info
2. Subscriber Info
3. Publisher Info
4. ReplyTo Info
5. Error Handling and Test

The Server Connection Info and Error Handling and Test sections are always active. The Subscriber Info, Publisher Info, and, ReplyTo Info sections can be enabled or disabled using the enable checkbox in the top left corner of each section. Using these checkboxes you can configure the step to be a publish step, a subscribe step, or both. You can also choose to include a "replyto" component in the step. When you have completely configured your test step, use the Test button in the Error Handling and Test section to test your configuration settings.

Server Connection Info

This screenshot shows the 'Server Connection Info' section with the following fields:

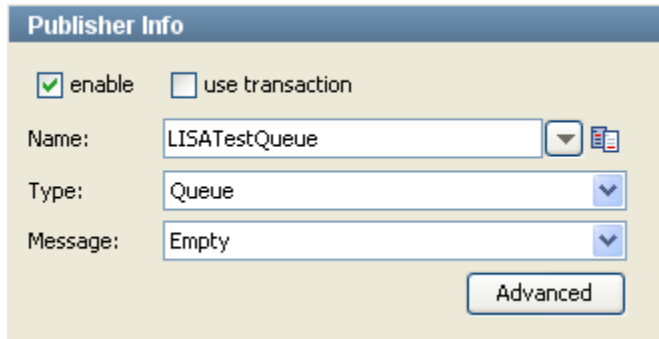
- Host: localhost
- Port: 18007
- An 'Advanced' button.

Here you enter the server connection information.

The two parameters should be available to you for the system under test.

- Host: The Name of the JMS server.
- Port: The Port number the JMS server is running on.
- The Advanced button displays a pop-up panel where you can add custom properties that will be sent with the connection information.

Publisher Info



The **Publisher Info** dialog box contains the following fields and controls:

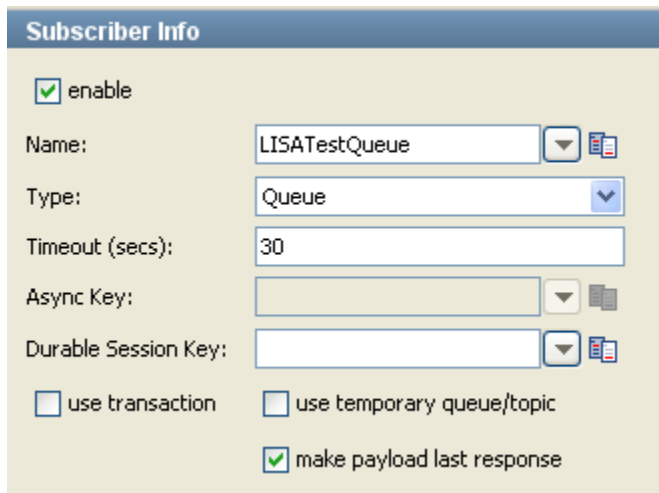
- ☒ enable ☐ use transaction
- Name: LISATestQueue (with a dropdown arrow and a document icon)
- Type: Queue (with a dropdown arrow)
- Message: Empty (with a dropdown arrow)
- Advanced (button)

Check the enable checkbox to set up the ability to send (publish) messages. Click the use transaction checkbox to execute a commit when the message is sent.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use.
- Type: Select whether you are using a topic or queue.
- Message: Select the type of message you are sending from the pull-down menu. The supported messages are: Empty, Text, Object, Bytes, Message, and Mapped (Extended).
- Advanced button: Displays a pop-up panel where you can add custom message properties to be sent with the message.

Subscriber Info



The **Subscriber Info** dialog box contains the following fields and controls:

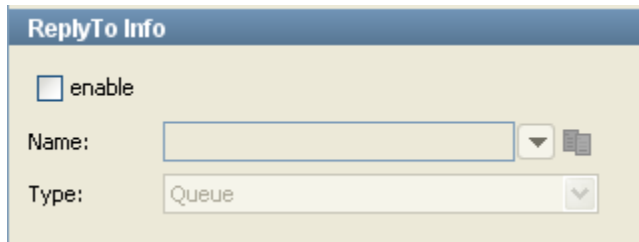
- ☒ enable
- Name: LISATestQueue (with a dropdown arrow and a document icon)
- Type: Queue (with a dropdown arrow)
- Timeout (secs): 30
- Async Key: (with a dropdown arrow and a document icon)
- Durable Session Key: (with a dropdown arrow and a document icon)
- ☐ use transaction ☐ use temporary queue/topic
- ☒ make payload last response

Check the enable checkbox to set up to enable the ability to receive (subscribe to) messages.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use.
- Type: Select whether you are using a topic or queue, and whether you want to listen in synchronous or asynchronous mode. For asynchronous mode you will also have to have an entry in the Async key field (see below).
- Timeout (secs): Enter the period to wait before LISA interrupts waiting for a message (this field can be left blank for no timeout).
- Async Key: Enter the value needed to identify asynchronous messages. This is only needed in asynchronous mode. It will be used in a subsequent Message Consumer step to retrieve asynchronous messages.
- Durable Session key: By entering a name here you are requesting a durable session. You are also providing a key for that session. A durable session allows you to receive all of your messages from a topic even if you logoff and then logon again.
- use transaction checkbox: Click the use transaction checkbox to execute a Commit when a message is received.
- use temporary que/topic checkbox: Click the use temporary queue/topic checkbox if you want the JMS provider to set up a temporary queue/topic on your behalf. When a temporary queue/topic is used, LISA will automatically set the JMS ReplyTo parameter of the message you send to the temporary queue/topic. The temporary queue/topic feature must always be used in conjunction with a publisher so that a reply can be sent. If you use a temporary queue/topic the ReplyTo section is disabled by LISA.
- make payload last response: Check this option if you want to make payload as the step response.

ReplyTo Info



The 'ReplyTo Info' dialog box has a title bar with the same name. It contains an 'enable' checkbox. Below it, there is a 'Name:' label followed by a text input field and a small icon. Below that, there is a 'Type:' label followed by a dropdown menu currently showing 'Queue'.

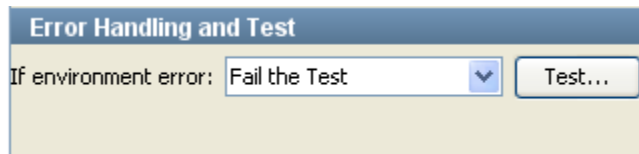
Check the enable checkbox to set up a destination queue/topic.

If your application needs a destination, it is setup in this section.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use.
- Type: Select whether you are using a topic or queue.

Error Handling and Test



The 'Error Handling and Test' dialog box has a title bar with the same name. It contains a label 'If environment error:' followed by a dropdown menu showing 'Fail the Test' and a 'Test...' button.

Error Handling and Test allows you to redirect to a step if an error occurs.

If environment error: Select the step to redirect to if an error occurs.

Click the Test button to test your step configuration settings.

Selector Query Tab

The Selector Query tab view is show below:



The 'JMS Selector Query' editor has a title bar with the same name. It contains a large text area with the query: `JMSCorrelationID LIKE '{{lisa.jms.correlation.id}}%'`. At the bottom, there are four tabs: 'Base', 'Selector Query' (which is selected), 'Send Message Data', and 'Response Message'.

You can enter a JMS selector query in this editor. The syntax closely follows SQL. It is a subset of SQL92. For further details see the JMS 1.1 specification, section 3.8. A JMS selector query can be specified when listening for a message on a queue that is a response to a published message. The example above shows a specific query looking for a JMSCorrelationID that matches one set in a LISA property as sent with the original message.

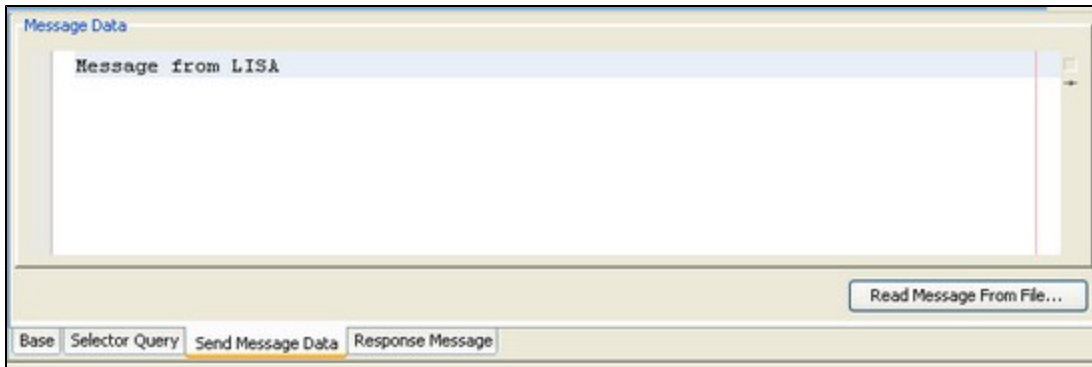
NOTE: LISA has a built in mechanism for allowing a test creator to set the JMSCorrelationID for a message before sending it. Anytime before the message is sent you can set the correlation ID by setting the following LISA property:

`lisa.jms.correlation.id`

LISA will detect a non-zero value and set the message JMSCorrelationID property before the message is sent.

Send Message Data

If your step is configured to publish, this is where you compose your message. The Send Message Data tab view is shown below for a text message:



This particular example shows an XML fragment with LISA properties being used. The text can be typed in or it can be read from a file using the **Read Message from the File...** button in the bottom right corner, or it can be stored in a LISA property, in which case you would just place the LISA property in the editor i.e. something like `LISA_PROP`.

Notice that LISA properties are used in the message XML allowing the message to be created dynamically during the test run.

Response Message

If your step is configured to subscribe, your response will be shown. For more information refer to [6.1 JMS Messaging \(JNDI\)](#)

8.2 JCaps Messaging (JNDI)

8.2 JCaps Messaging (JNDI)

The JCaps Messaging (JNDI) step allows you to send messages to, and receive messages from topics and queues. You can also receive, modify and forward an existing message.

JCaps Messaging (JNDI) supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

The JCaps Messaging (JNDI) step is configured using a single LISA editor regardless the messaging requirements. Input options will vary on the messaging requirements. The editor will only allow valid configurations, so when you enable certain features others may become inactive.

Prerequisites - You will be required to supply the necessary jar files for your chosen configuration and connection.

JCaps messaging requires that several jar files be added to your CLASSPATH. You can put them in the hot deploy directory. Check the JCaps documentation for more information on the jar files you might need. The jar files can be found in the lib directory of the JCaps installation.

Parameter Requirements - You need the connection parameters and the subject names used in the application under test. The parameters needed are listed below in the Server Connection Info section. There may be other parameters required, depending on your environment. These should be obtained in advance from the developers of the application.

The messaging step editor for JCaps Messaging (JNDI) is shown below:

The screenshot shows a configuration window with four tabs: **Server Connection Info**, **Subscriber Info**, **ReplyTo Info**, and **Publisher Info**. The **Base** tab is selected at the bottom.

- Server Connection Info:**
 - JNDI Factory Class: `com.stc.jms.indispi.InitialContextFactory`
 - JNDI Server URL: `stcms://{JCAPS_SERVER}:18007`
 - JMS ConnectionFactory: `QueueFactory`
 - User: (empty)
 - Password: (empty)
 - ☐ Share Sessions
 - ☐ Share Publishers
 - Buttons: **Stop All**, **Advanced**
- Subscriber Info:**
 - ☒ enable
 - Name: `LISATestQueue`
 - Type: `Queue`
 - Timeout (secs): `30`
 - Async Key: (empty)
 - Durable Session Key: (empty)
 - ☐ use transaction
 - ☐ use temporary queue/topic
 - ☒ make payload last response
- ReplyTo Info:**
 - ☐ enable
 - Name: (empty)
 - Type: `Queue`
- Publisher Info:**
 - ☒ enable
 - ☐ use transaction
 - Name: `LISATestQueue`
 - Type: `Queue`
 - Message: `Empty`
 - Button: **Advanced**
- Error Handling and Test:**
 - If environment error: `Fail the Test`
 - Button: **Test...**

At the bottom, there are four tabs: **Base**, **Selector Query**, **Send Message Data**, and **Response Message**.

There are four tabs available at the bottom of the editor.

- The **Base** tab is where you define your connection and messaging parameters.
- The **Selector Query** tab allows you to specify a selector query to be run when listening for a message on a queue.
- The **Send Message Data** tab is where you will create your message content.
- The **Response Message** tab is where your response messages will be posted.

Base Tab

The Base tab view is shown in the figure above. It is divided into 5 major sections:

- **Server Connection Info**
- **Subscriber Info**
- **Publisher Info**
- **ReplyTo Info**
- **Error Handling and Test**

The Server Connection Info and Error Handling and Test sections are always active. The Subscriber Info, Publisher Info, and, ReplyTo Info sections can be enabled or disabled using the enable checkbox in the top left corner of each section. Using these checkboxes you can configure the step to be a publish step, a subscribe step, or both. You can also choose to include a "replyto" component in the step. When you have completely configured your test step, use the Test button in the Error Handling and Test section to test your configuration settings

Server Connection Info

Server Connection Info

JNDI Factory Class:

JNDI Server URL:

JMS ConnectionFactory:

User:

Password:


☐ Share Sessions

☐ Share Publishers

Enter the JNDI information.

These values should be parameterized with properties that are in your configuration, making it easy to change the application under test. LISA, by default uses the `JCAPS_SERVER` property in the JNDI Server URL. This property must be added to your Configuration if you plan to use it.

The five parameters should be available to you for the system under test. The pull-down menus contain common examples or templates for these values.

- JNDI Factory Class: LISA pre-populates this field with default values.
- JNDI Server URL: LISA pre-populates this field with default values.
- JMS Connection Factory: You can use the magnifying glass, , to browse available resources on the server. Select or type in a Connection Factory per the JMS specification to use for this Step execution.
- User
- Password
- Share Services and Share Publishers checkboxes are used to share JMS Sessions and Publishers throughout the test case. This can lower overhead, but does not always provide a realistic simulation since typically JMS clients want to release resources. Note: If you check Share Publishers, the Share Sessions checkbox is also checked for you as you cannot share publishers without sharing sessions.
- The **Stop All** button allows you to stop any listeners at design time now. This is to resolve issues during test case creation where some listeners can get orphaned, but will still consume messages making it difficult or sometimes impossible to finish test case creation.
- The **Advanced** button displays a pop-up panel where you can add custom properties that will be sent with the connection information.

Publisher Info

☒ enable ☐ use transaction



Name:

Type:

Message:

Check the enable checkbox to set up the ability to send (publish) messages. Click the use transaction checkbox to execute a commit when the message is sent.



Enter the following parameters:



- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue. The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).
- Message: Select the type of message you are sending from the pull-down menu. The supported messages are: Empty, Text, Object, Bytes, Message, and Mapped (Extended).
- Advanced button: Displays a pop-up panel where you can add custom message properties to be sent with the message.

Subscriber Info



Subscriber Info



☒ enable

Name:  

Type:  

Timeout (secs):

Async Key:  



Durable Session Key:  

☐ use transaction ☐ use temporary queue/topic

☒ make payload last response

Check the enable checkbox to set up to enable the ability to receive (subscribe to) messages.


Enter the following parameters:


- Name: Enter the name of the topic or queue to use. The magnifying glass,  , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue, and whether you want to listen in synchronous or asynchronous mode. For asynchronous mode you will also have to have an entry in the Async key field. The browse icon,  , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).
- Timeout (secs): Enter the period to wait before LISA interrupts waiting for a message (this field can be left blank for no timeout).
- Async Key: Enter the value needed to identify asynchronous messages. This is only needed in asynchronous mode. It will be used in a subsequent Message Consumer step to retrieve asynchronous messages.
- Durable Session key: By entering a name here you are requesting a durable session. You are also providing a key for that session. A durable session allows you to receive all of your messages from a topic even if you logoff and then logon again.
- Use transaction checkbox: Click the use transaction checkbox to execute a Commit when a message is received.
- use temporary que/topic checkbox: Click the use temporary queue/topic checkbox if you want the JMS.provider to set up a temporary queue/topic on your behalf. When a temporary queue/topic is used, LISA will automatically set the JMS ReplyTo parameter of the message you send to the temporary queue/topic. The temporary queue/topic feature must always be used in conjunction with a publisher so that a reply can be sent. If you use a temporary queue/topic the ReplyTo section is disabled by LISA.
- make payload last response: Check this option if you want to make payload as the last response.

ReplyTo Info

ReplyTo Info

☐ enable



Name:  

Type:  

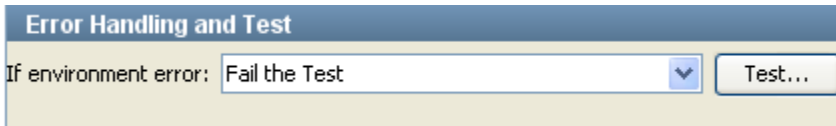
Check the enable checkbox to set up a destination queue/topic.

If your application needs a destination, it is setup in this section.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass,  , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue. The browse icon,  , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).

Error Handling and Test



Error Handling and Test allows you to redirect to a step if an error occurs.

- If environment error: Select the step to redirect to if an error occurs.

Click the **Test** button to test your step configuration settings.

Selector Query Tab

The Selector Query tab view is show below:

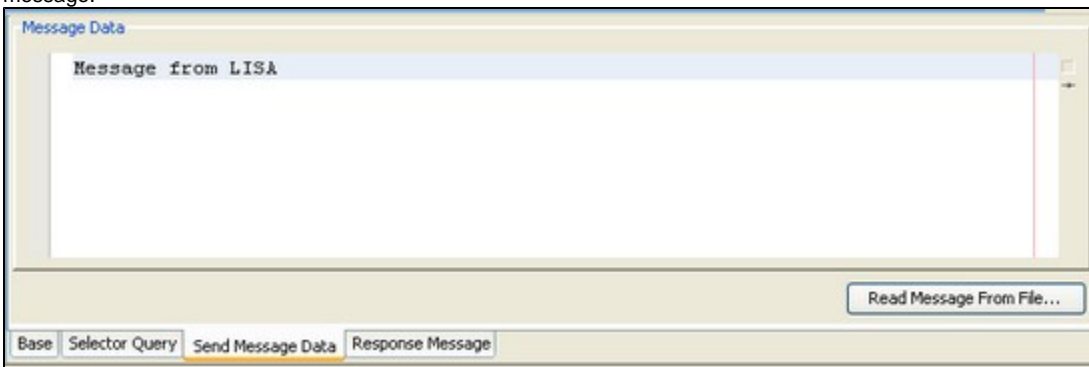


You can enter a JMS selector query in this editor. The syntax closely follows SQL. It is a subset of SQL92. For further details see the JMS 1.1 specification, section 3.8. A JMS selector query can be specified when listening for a message on a queue that is a response to a published message. The example above shows a specific query looking for a JMSCorrelationID that matches one set in a LISA property as sent with the original message. Note: LISA has a built in mechanism for allowing a test creator to set the JMSCorrelationID for a message before sending it. Anytime before the message is sent you can set the correlation ID by setting the following LISA property: `lisa.jms.correlation.id`

LISA will detect a non-zero value and set the message JMSCorrelationID property before the message is sent.

Send Message Data Tab

If your step is configured to publish, this is where you compose your message. The Send Message Data tab view is shown below for a text message:



This particular example shows an XML fragment with LISA properties being used. The text can be typed in or it can be read from a file using the Read Message from the File... button in the bottom right corner, or it can be stored in a LISA property, in which case you would just place the LISA property in the editor i.e. something like `LISA_PROP`.

Notice that LISA properties are used in the message XML allowing the message to be created dynamically during the test run.

Response Message Tab

If your step is configured to subscribe, your response will be shown. For more information refer to [6.1 JMS Messaging \(JNDI\)](#)

8.3 Message Consumer

8.3 Message Consumer

For detailed information regarding this step, please refer to [6.2 Message Consumer](#).

8.4 Read a File (Disk, URL, or Classpath)

8.4 Read a File (Disk, URL, or Classpath)

For detailed information regarding this step, please refer to [5.6 Read a File \(Disk, URL or Class Path\)](#).

8.5 Next Generation Web Service Execution (alpha)

8.5 Next Generation Web Service Execution (alpha)

For detailed information regarding this step, please refer to [1.3 Web Service Execution \(XML\)](#).

8.6 Web Service Execution

8.6 Web Service Execution

For detailed information regarding this step, please refer to [1.9 Web Service Execution \(Legacy\)](#).

8.7 Raw Soap Request

8.7 Raw Soap Request

For detailed information regarding this step, please refer to [1.5 Raw SOAP Request](#).

8.8 SQL Database Execution

8.8 SQL Database Execution

For detailed information regarding this step, please refer to [3.1 SQL Database Execution \(JDBC\)](#).

8.9 FTP Step

8.9 FTP Step

For detailed information regarding this step, please refer to [5.7 FTP Step](#).

9. Oracle Steps

9. Oracle Steps

The following steps are available in this chapter.

- [9.1 Oracle OC4J \(JNDI\)](#)
- [9.2 Oracle AQ \(JMS\)](#)
- [9.3 Oracle AQ \(JPUB\)](#)
- [9.4 Message Consumer](#)
- [9.5 Read a File \(Disk, URL, or Classpath\)](#)
- [9.6 Next Generation Web Service Execution \(alpha\)](#)
- [9.7 Web Service Execution](#)
- [9.8 Raw Soap Request](#)
- [9.9 SQL Database Execution](#)
- [9.10 FTP Step](#)

9.1 Oracle OC4J (JNDI)

9.1 Oracle OC4J (JNDI)

The Oracle OC4J (JNDI) step allows you to send messages to, and receive messages from topics and queues. You can also receive, modify and forward an existing message.

Oracle OC4J (JNDI) supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

The Oracle OC4J (JNDI) step is configured using a single LISA editor regardless the messaging requirements. Input options will vary on the messaging requirements. The editor will only allow valid configurations, so when you enable certain features others may become inactive.

Prerequisites - You will be required to supply the necessary jar files for your chosen configuration and connection.

OC4J messaging requires that the several jar files be added to your CLASSPATH. You can put them in the hot deploy directory. Check the OC4J documentation for more information on the jar files you might need. The jar files can be found in the lib directory of the OC4J installation.

Parameter Requirements - You need the connection parameters and the subject names used in the application under test. The parameters needed are listed below in the Server Connection Info section. There may be other parameters required, depending on your environment. These should be obtained in advance from the developers of the application.

The messaging step editor for Oracle OC4J (JNDI) is shown below:

The screenshot shows the Oracle OC4J (JNDI) messaging step editor. It is divided into four main sections: Server Connection Info, Subscriber Info, ReplyTo Info, and Publisher Info. The Base tab is selected at the bottom. The Server Connection Info section includes fields for JNDI Factory Class, JNDI Server URL, JMS ConnectionFactory, User, and Password, along with checkboxes for Share Sessions and Share Publishers. The Subscriber Info section includes an enable checkbox, Name, Type, Timeout (secs), Async Key, Durable Session Key, and checkboxes for use transaction, use temporary queue/topic, and make payload last response. The ReplyTo Info section includes an enable checkbox, Name, and Type. The Publisher Info section includes an enable checkbox, use transaction checkbox, Name, Type, and Message. The Error Handling and Test section includes a dropdown for 'If environment error' and a 'Test...' button. At the bottom, there are four tabs: Base, Selector Query, Send Message Data, and Response Message.

There are four tabs available at the bottom of the editor.

- The **Base** tab is where you define your connection and messaging parameters.
- The **Selector Query** tab allows you to specify a selector query to be run when listening for a message on a queue.
- The **Send Message Data** tab is where you will create your message content.
- The **Response Message** tab is where your response messages will be posted.

Base Information (Base tab)

The Base tab view is shown in the figure above. It is divided into 5 major sections:

- Server Connection Info.
- Subscriber Info.
- Publisher Info.
- ReplyTo Info.
- Error Handling and Test.

The Server Connection Info and Error Handling and Test sections are always active. The Subscriber Info, Publisher Info, and, ReplyTo Info sections can be enabled or disabled using the enable checkbox in the top left corner of each section. Using these checkboxes you can configure the step to be a publish step, a subscribe step, or both. You can also choose to include a "replyto" component in the step. When you have completely configured your test step, use the Test button in the Error Handling and Test section to test your configuration settings

Server Connection Info

Server Connection Info

JNDI Factory Class:

JNDI Server URL:

JMS ConnectionFactory:

User:

Password:


☐ Share Sessions

☐ Share Publishers

Here you enter the JNDI information.

These values should be parameterized with properties that are in your configuration, making it easy to change the application under test. LISA, by default uses the OC4J_SERVER property in the JNDI Server URL. This property must be added to your Configuration if you plan to use it.

The five parameters should be available to you for the system under test. The pull-down menus contain common examples or templates for these values.

- JNDI Factory Class: LISA pre-populates this field with default values.
- JNDI Server URL: LISA pre-populates this field with default values.
- JMS Connection Factory: You can use the magnifying glass, , to browse available resources on the server. Select or type in a Connection Factory per the JMS specification to use for this Step execution.
- User.
- Password.
- Share Sessions and Share Publishers checkboxes are used to share JMS Sessions and Publishers throughout the test case. This can lower overhead, but does not always provide a realistic simulation since typically JMS clients want to release resources. Note: If you check Share Publishers, the Share Sessions checkbox is also checked for you as you cannot share publishers without sharing sessions.
- The Stop All button allows you to stop any listeners at design time now. This is to resolve issues during test case creation where some listeners can get orphaned, but will still consume messages making it difficult or sometimes impossible to finish test case creation.
- The Advanced button displays a pop-up panel where you can add custom properties that will be sent with the connection information.

Publisher Info

Publisher Info

☒ enable ☐ use transaction



Name:

Type:

Message:

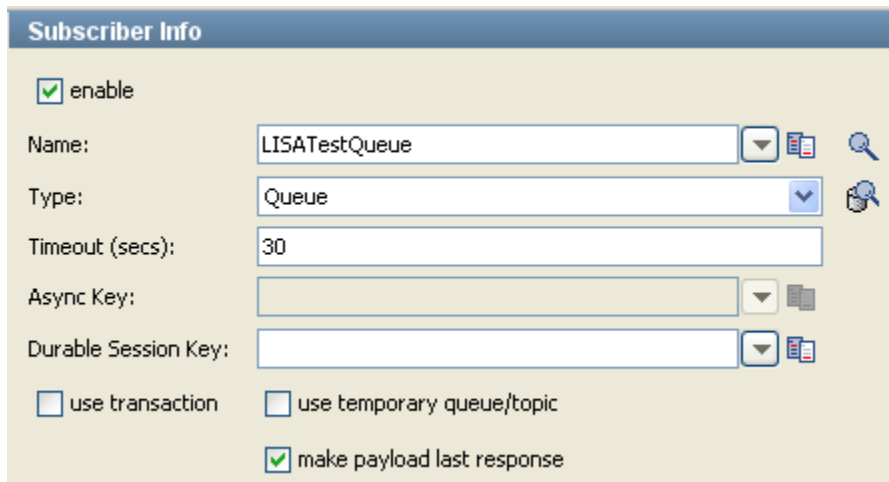
Check the enable checkbox to set up the ability to send (publish) messages. Click the use transaction checkbox to execute a commit when the message is sent.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue. The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).
- Message: Select the type of message you are sending from the pull-down menu. The supported messages are: Empty, Text, Object, Bytes, Message, and Mapped (Extended).

- Advanced button: Displays a pop-up panel where you can add custom message properties to be sent with the message.

Subscriber Info





The Subscriber Info form has a title bar 'Subscriber Info'. It contains the following fields and controls:

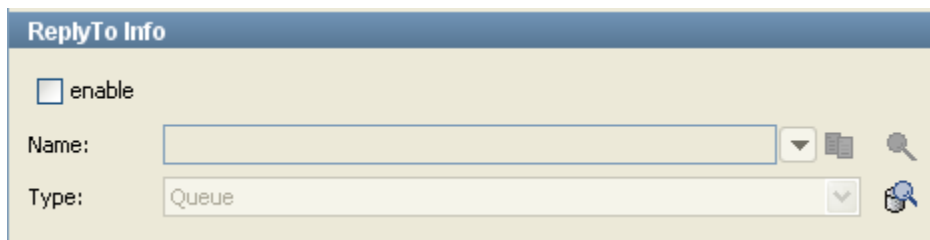
- ☒ enable
- Name: [dropdown] [browse icon]
- Type: [dropdown] [browse icon]
- Timeout (secs):
- Async Key: [dropdown] [document icon]
- Durable Session Key: [dropdown] [document icon]
- ☐ use transaction
- ☐ use temporary queue/topic
- ☒ make payload last response

Check the enable checkbox to set up to enable the ability to receive (subscribe to) messages.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue, and whether you want to listen in synchronous or asynchronous mode. For asynchronous mode you will also have to have an entry in the Async key field (see below). The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).
- Timeout (secs): Enter the period to wait before LISA interrupts waiting for a message (this field can be left blank for no timeout).
- Async Key: Enter the value needed to identify asynchronous messages. This is only needed in asynchronous mode. It will be used in a subsequent Message Consumer step to retrieve asynchronous messages.
- Durable Session key: By entering a name here you are requesting a durable session. You are also providing a key for that session. A durable session allows you to receive all of your messages from a topic even if you logoff, and then logon again.
- use transaction checkbox: Click the use transaction checkbox to execute a Commit when a message is received.
- use temporary que/topic checkbox: Click the use temporary queue/topic checkbox if you want the JMS.provider to set up a temporary queue/topic on your behalf. When a temporary queue/topic is used, LISA will automatically set the JMS ReplyTo parameter of the message you send to the temporary queue/topic. The temporary queue/topic feature must always be used in conjunction with a publisher so that a reply can be sent. If you use a temporary queue/topic the ReplyTo section is disabled by LISA.
- make payload last response: Check this option if you want to make payload as the last response.

ReplyTo Info





The ReplyTo Info form has a title bar 'ReplyTo Info'. It contains the following fields and controls:

- ☐ enable
- Name: [dropdown] [document icon]
- Type: [dropdown] [browse icon]

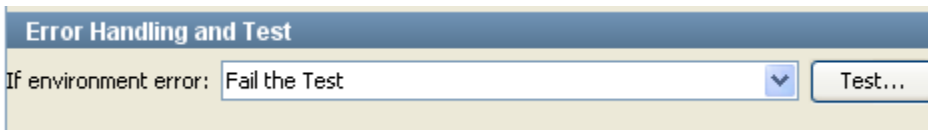
Check the enable checkbox to set up a destination queue/topic.

If your application needs a destination, it is setup in this section.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue. The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).

Error Handling and Test



Error Handling and Test allows you to redirect to a step if an error occurs.

- If environment error: Select the step to redirect to if an error occurs.

Click the Test button to test your step configuration settings.

Selector Query Tab

The Selector Query tab view is show below:



You can enter a JMS selector query in this editor. The syntax closely follows SQL. It is a subset of SQL92. For further details see the JMS 1.1 specification, section 3.8. A JMS selector query can be specified when listening for a message on a queue that is a response to a published message. The example above shows a specific query looking for a JMSCorrelationID that matches one set in a LISA property as sent with the original message.

NOTE: LISA has a built in mechanism for allowing a test creator to set the JMSCorrelationID for a message before sending it. Anytime before the message is sent you can set the correlation ID by setting the following LISA property:

`lisa.jms.correlation.id`

LISA will detect a non-zero value and set the message JMSCorrelationID property before the message is sent.

Send Message Data Tab

If your step is configured to publish, this is where you compose your message. The Send Message Data tab view is shown below for a text message:



This particular example shows an XML fragment with LISA properties being used. The text can be typed in or it can be read from a file using the Read Message from the File... button in the bottom right corner, or it can be stored in a LISA property, in which case you would just place the LISA property in the editor i.e. something like `LISA_PROP`.

Notice that LISA properties are used in the message XML allowing the message to be created dynamically during the test run.

Response Message Tab

If your step is configured to subscribe, your response will be shown here. For more information refer to [6.1 JMS Messaging \(JNDI\)](#)

9.2 Oracle AQ (JMS)

9.2 Oracle AQ (JMS)

Oracle Active Queuing (AQ) is a messaging provider built into the Oracle Database itself. It's used as the default JMS provider for many Oracle products, such as Oracle Enterprise Service Bus.

One of the two distinct ways to utilize AQ is JMS.

▼ Oracle AQ (JMS) - Step2

Server Connection Info	Subscriber Info
JDBC URL: jdbc:oracle:thin:@localhost:1521:orcl	<input type="checkbox"/> enable
JDBC Driver: oracle.jdbc.driver.OracleDriver	Schema:
User:	Name:
Password:	Type: Queue
<input type="checkbox"/> Share Sessions	Timeout (secs): 30
<input type="checkbox"/> Share Publishers	Async Key:
Stop All	Durable Session Key:
	<input type="checkbox"/> use transaction <input type="checkbox"/> use temporary queue/topic
	<input checked="" type="checkbox"/> make payload last response

ReplyTo Info	Publisher Info
<input type="checkbox"/> enable	<input checked="" type="checkbox"/> enable <input type="checkbox"/> use transaction
Schema:	Schema:
Name:	Name:
Type: Queue	Type: Queue
	Message: Empty
	Advanced

Using their JMS library it works like any other normal JMS provider, with a few notable differences:

1. The JMS connection is not made through JNDI; it's made using a JDBC connection. This means that it involves entering a JDBC URL, driver class name, username, and password. This information goes under 'Server Connection Info' of LISA Oracle AQ(JMS).
2. Queues and Topics are tied to schemas in the database. This means that in order to send to or receive from a queue you need to give both the queue name and queue schema.
3. Each Queue or Topic is restricted to a particular type of JMS Message. This means that if a Queue normally transports JMS Text Messages then you can't use that same Queue to transport JMS Object Messages, or JMS Byte Messages, etc.

9.3 Oracle AQ (JPUB)

9.3 Oracle AQ (JPUB)

As mentioned previously, two distinct ways to utilize AQ, one being JMS, the other way is JPub.

Oracle AQ (JPub) - Step1

Server Connection Info

JDBC URL: jdbc:oracle:thin:@localhost:1521:orcl

JDBC Driver: oracle.jdbc.driver.OracleDriver

User:

Password:

☐ Share Sessions

Stop All

Subscriber Info

☐ enable

Schema:

Name:

Type: Queue

Timeout (secs): 30

Async Key:

Generate JPub Classes

Payload Class Name:

Advanced

☒ make payload last response

Error Handling and Test

If environment error: Abort the Test

Test...

Publisher Info

☒ enable

Schema:

Name:

Generate JPub Classes

Payload Class Name:

Advanced

Base Condition Send Message Data Response Message

Oracle AQ's JMS API is a layer built on top of a lower-level AQ API. This lower-level API is much more difficult to deal with; it acts almost nothing like JMS.

- The principal distinction is the message format. Low-level AQ messages contain a **payload**, which can be any type defined in the database. It could be a varchar, or a clob, but usually it's a user-defined structured database type. Similar to AQ JMS Queues, each AQ low-level Queue can only handle one payload type.
- **Oracle provides a utility, called 'JPub'**, that can generate Java objects that can deal with these user-defined structured types, in the same way that Axis generates Java objects that utilize web services. Our low-level AQ step, called 'Oracle AQ JPub', can automatically use this utility to generate the client classes based on the queue information. The user then fills in their payload object using a standard COE.
- There is no distinction between Queues or Topics. A client can either remove the next message from the AQ queue, making it essentially a queue, or read the next message from the AQ queue without removing it, making it essentially a Topic.
- Setting up low-level AQ Queues is again done through stored procedures. There is the possible additional step of creating your own user-defined structured type in the database before creating an AQ queue around it.
- Technically, you can interact with AQ JMS Queues using the low-level API. The JMS Queues simply have a specific payload type that's structured like a standard JMS message. However, you cannot use the AQ JMS API to interact with low-level AQ Queues, i.e. those that don't use a JMS payload type.

9.4 Message Consumer

9.4 Message Consumer

For details please see [6.2 Message Consumer](#).

9.5 Read a File (Disk, URL, or Classpath)

9.5 Read a File (Disk, URL, or Classpath)

For details please see [5.6 Read a File \(Disk, URL or Class Path\)](#).

9.6 Next Generation Web Service Execution (alpha)

9.6 Next Generation Web Service Execution (alpha)

For detailed information regarding this step, please refer to [1.3 Web Service Execution \(XML\)](#).

9.7 Web Service Execution

9.7 Web Service Execution

For detailed information regarding this step, please refer to [1.9 Web Service Execution \(Legacy\)](#).

9.8 Raw Soap Request

9.8 Raw Soap Request

For detailed information regarding this step, please refer to [1.5 Raw SOAP Request](#).

9.9 SQL Database Execution

9.9 SQL Database Execution

For detailed information regarding this step, please refer to [3.1 SQL Database Execution \(JDBC\)](#).

9.10 FTP Step

9.10 FTP Step

For detailed information regarding this step, please refer to [5.7 FTP Step](#).

10. TIBCO Steps

10. TIBCO Steps

The following Steps are available in this chapter.

[10.1 TIBCO Rendezvous Messaging](#)
[10.2 TIBCO Direct JMS](#)
[10.3 Message Consumer](#)
[10.4 Read a File \(Disk, URL, or Classpath\)](#)
[10.5 Next Generation Web Service Execution \(alpha\)](#)
[10.6 Web Service Execution](#)
[10.7 Raw Soap Request](#)
[10.8 SQL Database Execution](#)
[10.9 FTP Step](#)

10.1 TIBCO Rendezvous Messaging

10.1 TIBCO Rendezvous Messaging

The TIBCO Rendezvous Messaging step allows you to send messages to, and receive messages from Rendezvous "Subjects" using Native Rendezvous protocol. You can also receive, modify and forward an existing message.

The TIBCO Rendezvous Messaging step is configured using a single LISA editor regardless the messaging requirements. Input options will vary on the messaging requirements. The step editor will only allow valid configurations, so when you enable certain features others may become inactive.

Prerequisites - You will be required to supply the necessary jar files for your chosen configuration and connection.

The TIBCO Rendezvous "bin" directory must be added to your "PATH" environment variable.

TIBCO Rendezvous requires that several TIBCO RV jar files in your CLASSPATH. The jar files can be found in the **lib** directory of the TIBCO installation:

- tibjms.jar
- tibjmsadmin.jar
- tibjmsapps.jar

- tibrvjms.jar

TIBCO classes require access to the system class loader, so it is suggested that you create a LISA_PRE_CLASSPATH environment variable in your OS that list the TIBCO RV jar files.

Parameter Requirements - You need the connection parameters and the subject names used in the application under test. The parameters needed are listed below in the Server Connection Info section. There may be other parameters required, depending on your environment. These should be obtained in advance from the developers of the application.

The messaging step editor for TIBCO Rendezvous Messaging is shown below:

The screenshot shows the TIBCO Rendezvous Messaging step editor. It has a tabbed interface at the bottom with three tabs: 'Base', 'Send Message Data', and 'Response Message'. The 'Base' tab is currently selected. The main area is divided into six sections, each with an 'enable' checkbox in the top left corner:

- Server Connection Info:** Contains fields for Service (7522:7524), Network, Daemon, and Client Mode (Native Client). There is a 'Stop All' button.
- Subscriber Info:** Contains an 'enable' checkbox, Subject, Timeout (secs) (30), and Async Key.
- Certified Transport Info:** Contains an 'enable' checkbox, Sender Name, Advisory Subject (_RV.INFO.RVCM.DELIVERY.CONFIRM.>), and Time Limit.
- Publisher Info:** Contains an 'enable' checkbox, 'Enable Inbox Type', Subject, Message (Empty), and Send Field.
- ReplyTo Info:** Contains an 'enable' checkbox, Subject, and an 'Inbox Timeout' field.
- Error Handling and Test:** Contains an 'If environment error' dropdown (Fail the Test) and a 'Test...' button.

There are three tabs available at the bottom of the editor.

- The Base tab is where you define your connection and messaging parameters.
- The Send Message Data tab is where you will create your message content.
- The Response Message tab is where your response messages will be posted.

Base Information (Base tab)

The Base tab is shown in the figure above. It is divided into 5 major sections:

1. Server Connection Info.
2. Subscriber Info.
3. Certified Transport Info.
4. Publisher Info.
5. ReplyTo Info.
6. Error Handling and Test.

The Server Connection Info and Error Handling and Test sections are always active. The Subscriber Info, Publisher Info, and ReplyTo Info sections can be enabled or disabled using the enable checkbox in the top left corner of each section. Using these checkboxes you can configure the step to be a publish step, a subscribe step, or both. You can also choose to include a "replyto" component in the step. When you have completely configured your test step, use the Test button in the Error Handling and Test section to test your configuration settings.

Server Connection Info

Server Connection Info

Service: 7522:7524

Network:

Daemon:

Client Mode: Native Client

Stop All

Here you enter the Connection information specific to Rendezvous: information.

The four parameters should be available to you for the system under test.

- Service, Network, and Daemon: These are parameters to enable connection to the RV network you wish to communicate on.
- Client Mode: Choose between Rendezvous' Native client and Java Client mode. Usually you will want to use the more versatile Client mode.

These values should be parameterized with properties that are in your Configuration, making it easy to change for a different system under test.

Publisher Info

Publisher Info

☒ enable ☐ Enable Inbox Type

Subject:

Message: Empty

Send Field:

Inbox Timeout:

☐ Enable sendReply:

Check the enable checkbox to set up the ability to send (publish) messages.

Enter the following parameters:

- Subject: Enter the name of the subject to use. You can define your own subjects.
- Message: Select the type of message you are sending from the pull-down menu. The supported messages are: Empty, Text, Object, Bytes, Message, and Mapped (Extended).
- Send Field: RV messages are actually maps of fields and values. LISA uses this field to enable quick single field messages. When you enter a value here the "Send Message" data is put into the value of a field with this name. This is overridden with "Mapped (Extended)" type messages as they will allow you to do multiple fields and values in a single message.

Check the Enable Inbox Type checkbox to (TODO)

Enable sendReply: (TODO)

Subscriber Info

Subscriber Info

☒ enable

Subject:

Timeout (secs): 30

Async Key:

Need to write about the link between Enable Inbox Type and Subscriber Info (TODO)

Check the enable checkbox to set up to enable the ability to receive (subscribe to) messages.

Enter the following parameters:

- Subject: Enter the name of the subject to use. You can define your own subjects.
- Timeout (secs): Enter the period to wait before LISA interrupts waiting for a message (this field can be left blank for no timeout).
- Async Key: Enter the value needed to identify asynchronous messages. This is only needed in asynchronous mode. It will be used in a subsequent Message Consumer step to retrieve asynchronous messages.

ReplyTo Info

The dialog box has a title bar 'ReplyTo Info'. Inside, there is a checkbox labeled 'enable' which is checked. Below it is a text field labeled 'Subject:' followed by a dropdown menu and a button with a document icon.

Check the enable checkbox to set up a destination subject.

If your application needs a destination, it is setup in this section.

Enter the following parameters:

- Subject: Enter the name of the subject to use.

Error Handling and Test

The dialog box has a title bar 'Error Handling and Test'. Inside, there is a label 'If environment error:' followed by a dropdown menu showing 'Fail the Test' and a 'Test...' button.

Error Handling and Test allows you to redirect to a step if an error occurs.

- If environment error: Select the step to redirect to if an error occurs.

Click the **Test** button to test your step configuration settings.

Send Message Data

If your step is configured to publish, this is where you compose your message. The Send Message Data tab view is shown below for a text message:

Unable to render embedded object: File (worddavbb519a89f2fdca7504643676a59737b.png) not found.

This particular example shows an XML fragment with LISA properties being used. The text can be typed in or it can be read from a file using the **Read Message from the File** button in the bottom right corner, or it can be stored in a LISA property, in which case you would just place the LISA property in the editor, for example, `LISA_PROP`.

Notice that LISA properties are used in the message XML allowing the message to be created dynamically during the test run.

Response Message

If your step is configured to subscribe, your response will be shown. For more information refer to [JMS Messaging \(JNDI\)](#).

TIBCO EMS Messaging

The TIBCO EMS Messaging step allows you to send messages to, and receive messages from topics and queues. You can also receive, modify and forward an existing message.

LISA supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

Prerequisites - You will be required to supply the necessary jar files for your chosen configuration and connection.

The TIBCO Rendezvous **bin** directory must be added to your **PATH** environment variable.

TIBCO Rendezvous requires that several TIBCO RV jar files in your CLASSPATH. The jar files can be found in the **lib** directory of the TIBCO installation:

- tibjms.jar
- tibjmsadmin.jar
- tibjmsapps.jar
- tibrvms.jar

TIBCO classes require access to the system class loader, so it is suggested that you create a **LISA_PRE_CLASSPATH** environment variable in your OS that list the TIBCO RV jar files.

Parameter Requirements - You need the connection parameters and the subject names used in the application under test. The parameters needed are listed below in the Server Connection Info section. There may be other parameters required, depending on your environment. These should be obtained in advance from the developers of the application.

The messaging step editor for TIBCO EMS Messaging is shown below:

The screenshot shows the TIBCO EMS Messaging step editor. It has four main sections: **Server Connection Info**, **Subscriber Info**, **ReplyTo Info**, and **Publisher Info**. The **Server Connection Info** section includes fields for JNDI Factory Class, JNDI Server URL, JMS ConnectionFactory, User, and Password, along with checkboxes for Share Sessions and Share Publishers. The **Subscriber Info** section includes an enable checkbox, Name, Type, Timeout (secs), Async Key, Durable Session Key, and checkboxes for use transaction, use temporary queue/topic, and make payload last response. The **ReplyTo Info** section includes an enable checkbox, Name, and Type. The **Publisher Info** section includes an enable checkbox, use transaction checkbox, Name, Type, and Message. At the bottom, there is an **Error Handling and Test** section with a dropdown for 'If environment error' and a 'Test...' button. The bottom of the editor shows four tabs: Base, Selector Query, Send Message Data, and Response Message.

There are four tabs available at the bottom of the editor.

- The Base tab is where you define your connection and messaging parameters.
- The Selector Query tab allows you to specify a selector query to be run when listening for a message on a queue.
- The Send Message Data tab is where you will create your message content.
- The Response Message tab is where your response messages will be posted.

Base Information (Base tab)

The Base tab view is shown in the figure above. It is divided into 5 major sections:

1. Server Connection Info.
2. Subscriber Info.
3. Publisher Info.
4. ReplyTo Info.
5. Error Handling and Test.

The Server Connection Info and Error Handling and Test sections are always active. The Subscriber Info, Publisher Info, and, ReplyTo Info sections can be enabled or disabled using the enable checkbox in the top left corner of each section. Using these checkboxes you can configure the step to be a publish step, a subscribe step, or both. You can also choose to include a "replyto" component in the step. When you have completely configured your test step, use the Test button in the Error Handling and Test section to test your configuration settings.


Server Connection Info

The screenshot shows the **Server Connection Info** section of the TIBCO EMS Messaging step editor. It includes fields for JNDI Factory Class, JNDI Server URL, JMS ConnectionFactory, User, and Password, along with checkboxes for Share Sessions and Share Publishers. At the bottom are 'Stop All' and 'Advanced' buttons.

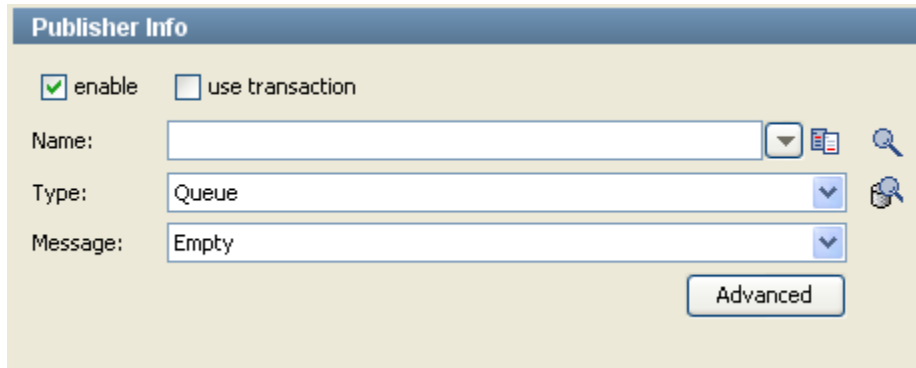
Here you enter the JNDI information.

These values should be parameterized with properties that are in your configuration, making it easy to change the application under test. LISA, by default uses the TIBCO_SERVER property in the JNDI Server URL. This property must be added to your Configuration if you plan to use it.

The five parameters should be available to you for the system under test. The pull-down menus contain common examples or templates for these values.

- JNDI Factory Class: LISA pre-populates this field with default values.
- JNDI Server URL: LISA pre-populates this field with default values.
- JMS Connection Factory: You can use the magnifying glass, , to browse available resources on the server. Select or type in a Connection Factory per the JMS specification to use for this Step execution. The EMS implementation in LISA uses TIBCO administrative APIs to gather more information about these resources than a typical JMS/JNDI view. This does however; require a login to the EMS server that is able to view resources. LISA will ask you for the login ID and Password to retrieve the available resources.
- User.
- Password.
- Share Sessions and Share Publishers checkboxes are used to share JMS Sessions and Publishers throughout the test case. This can lower overhead, but does not always provide a realistic simulation since typically JMS clients want to release resources. Note: If you check Share Publishers, the Share Sessions checkbox is also checked for you as you cannot share publishers without sharing sessions.
- The Stop All button allows you to stop any listeners at design time now. This is to resolve issues during test case creation where some listeners can get orphaned, but will still consume messages making it difficult or sometimes impossible to finish test case creation.
- The Advanced button displays a pop-up panel where you can add custom properties that will be sent with the connection information.



Publisher Info



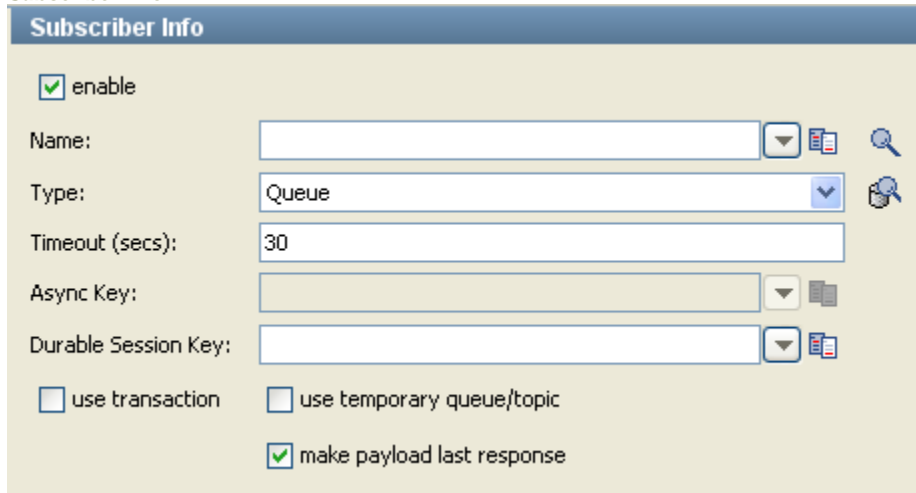
The Publisher Info dialog box has a title bar 'Publisher Info'. It contains two checkboxes: 'enable' (checked) and 'use transaction' (unchecked). Below these are three fields: 'Name:' with a text box and a magnifying glass icon, 'Type:' with a dropdown menu showing 'Queue' and a browse icon, and 'Message:' with a dropdown menu showing 'Empty' and a browse icon. At the bottom right is an 'Advanced' button.

Check the enable checkbox to set up the ability to send (publish) messages. Click the use transaction checkbox to execute a commit when the message is sent.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue. The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).
- Message: Select the type of message you are sending from the pull-down menu. The supported messages are: Empty, Text, Object, Bytes, Message, and Mapped (Extended).
- Advanced button: Displays a pop-up panel where you can add custom message properties to be sent with the message.



Subscriber Info



The Subscriber Info dialog box has a title bar 'Subscriber Info'. It contains an 'enable' checkbox (checked). Below it are five fields: 'Name:' with a text box and a magnifying glass icon, 'Type:' with a dropdown menu showing 'Queue' and a browse icon, 'Timeout (secs):' with a text box showing '30', 'Async Key:' with a text box and a magnifying glass icon, and 'Durable Session Key:' with a text box and a magnifying glass icon. At the bottom are three checkboxes: 'use transaction' (unchecked), 'use temporary queue/topic' (unchecked), and 'make payload last response' (checked).

Check the enable checkbox to set up to enable the ability to receive (subscribe to) messages.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue, and whether you want to listen in synchronous or asynchronous mode. For asynchronous mode you will also have to have an entry in the Async key field (see below). The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).
- Timeout (secs): Enter the period to wait before LISA interrupts waiting for a message (this field can be left blank for no timeout).
- Async Key: Enter the value needed to identify asynchronous messages. This is only needed in asynchronous mode. It will be used in a subsequent Message Consumer step to retrieve asynchronous messages.
- Durable Session key: By entering a name here you are requesting a durable session. You are also providing a key for that session. A durable session allows you to receive all of your messages from a topic even if you logoff and then logon again.
- use transaction checkbox: Click the use transaction checkbox to execute a Commit when a message is received.
- use temporary que/topic checkbox: Click the use temporary queue/topic checkbox if you want the JMS.provider to set up a temporary queue/topic on your behalf. When a temporary queue/topic is used, LISA will automatically set the JMS ReplyTo parameter of the message you send to the temporary queue/topic. The temporary queue/topic feature must always be used in conjunction with a publisher so that a reply can be sent. If you use a temporary queue/topic the ReplyTo section is disabled by LISA.
- make payload last response: Check this option if you want to make payload as the last response.

ReplyTo Info





The 'ReplyTo Info' dialog box has a title bar 'ReplyTo Info'. Inside, there is an 'enable' checkbox. Below it, the 'Name:' field is empty with a magnifying glass icon to its right. The 'Type:' field is set to 'Queue' with a browse icon to its right.

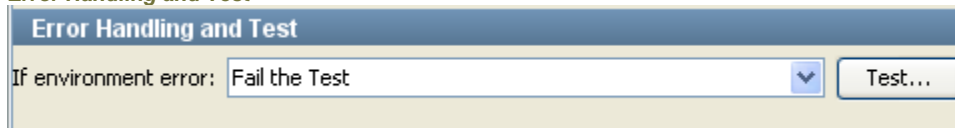
Check the enable checkbox to set up a destination queue/topic.

If your application needs a destination, it is setup in this section.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue. The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).

Error Handling and Test



The 'Error Handling and Test' dialog box has a title bar 'Error Handling and Test'. Inside, there is a dropdown menu labeled 'If environment error:' with 'Fail the Test' selected. To the right of the dropdown is a 'Test...' button.

Error Handling and Test allows you to redirect to a step if an exception occurs.

- On Exception Execute Step: Select the step to redirect to if an exception was thrown.

Click the **Test** button to test your step configuration settings.

Selector Query Tab

The Selector Query tab view is show below:

Unable to render embedded object: File (worddav3d642571722d831ecad0ac55a583e593.png) not found.

You can enter a JMS selector query in this editor. The syntax closely follows SQL. It is a subset of SQL92. For further details see the JMS 1.1 specification, section 3.8. A JMS selector query can be specified when listening for a message on a queue that is a response to a published message. The example above shows a specific query looking for a JMSCorrelationID that matches one set in a LISA property as sent with the original message.

LISA has a built in mechanism for allowing a test creator to set the JMSCorrelationID for a message before sending it. Anytime before the message is sent you can set the correlation ID by setting the following LISA property:
lisa.jms.correlation.id

LISA will detect a non-zero value and set the message JMSCorrelationID property before the message is sent.

Send Message Data

If your step is configured to publish, this is where you compose your message. The Send Message Data tab view is shown below for a text message:

Unable to render embedded object: File (worddavbb519a89f2fdfca7504643676a59737b.png) not found.

This particular example shows an XML fragment with LISA properties being used. The text can be typed in or it can be read from a file using the Read Message from the File button in the bottom-right corner, or it can be stored in a LISA property, in which case you would just place the LISA property in the editor, for example, `LISA_PROP`.

Notice that LISA properties are used in the message XML allowing the message to be created dynamically during the test run.

Response Message

If your step is configured to subscribe, your response will be shown here. For more information refer to [JMS Messaging \(JNDI\)](#).

10.2 TIBCO Direct JMS

10.2 TIBCO Direct JMS

The TIBCO Direct JMS step allows you to send messages to, and receive messages from topics and queues. You can also receive, modify and forward an existing message.

TIBCO Direct JMS supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

The TIBCO Direct JMS step is configured using a single LISA editor regardless of the messaging requirements. Input options will vary on the messaging requirements. The editor will only allow valid configurations, so when you enable certain features others may become inactive.

Prerequisites - You will be required to supply the necessary jar files for your chosen configuration and connection.

TIBCO Direct JMS requires that several jar files be added to your CLASSPATH. Check the **JCaps** documentation for more information on the jar files you might need. These are normally found in the TIBCO install directory in **lib** for the product you are using. It is suggested to create that your **LISA_PRE_CLASSPATH** environment variable in your operating system and restart LISA. The Tibco classes require access to the system class loader.

Parameter Requirements - You need the connection parameters and the subject names used in the application under test. The parameters needed are listed below in the **Server Connection Info** section. There may be other parameters required, depending on your environment. These should be obtained in advance from the developers of the application.

The messaging step editor for TIBCO Direct JMS is shown below:

Server Connection Info		Subscriber Info	
Server URL:	tcp://{{TIBCO_SERVER}}:7222	<input checked="" type="checkbox"/> enable	
User:		Name:	LISATestQueue
Password:		Type:	Queue
<input type="checkbox"/> Share Sessions		Timeout (secs):	30
<input type="checkbox"/> Share Publishers		Async Key:	
<input type="button" value="Stop All"/> <input type="button" value="Advanced"/>		Durable Session Key:	
		<input type="checkbox"/> use transaction	<input type="checkbox"/> use temporary queue/topic
		<input checked="" type="checkbox"/> make payload last response	
ReplyTo Info		Publisher Info	
<input type="checkbox"/> enable		<input checked="" type="checkbox"/> enable	<input type="checkbox"/> use transaction
Name:		Name:	LISATestQueue
Type:	Queue	Type:	Queue
		Message:	Empty
		<input type="button" value="Advanced"/>	
Error Handling and Test			
If environment error:		Fail the Test	<input type="button" value="Test..."/>
<div> <div>Base</div> <div>Selector Query</div> <div>Send Message Data</div> <div>Response Message</div> </div>			

There are four tabs available at the bottom of the editor.

- The **Base** tab is where you define your connection and messaging parameters.
- The **Selector Query** tab allows you to specify a selector query to be run when listening for a message on a queue.
- The **Send Message Data** tab is where you will create your message content.
- The **Response Message** tab is where your response messages will be posted.

Base Information (Base tab)



The Base tab is shown in the figure above. It is divided into 5 major sections:



- Server Connection Info.
- Subscriber Info.
- Publisher Info.
- ReplyTo Info.
- Error Handling and Test.

The Server Connection Info, Error Handling and Test sections are always active. The Subscriber Info, Publisher Info, and ReplyTo Info sections can be enabled or disabled using the enable checkbox in the top left corner of each section. Using these checkboxes you can configure the step to be a publish step, a subscribe step, or both. You can also choose to include a "replyto" component in the step. When you have completely configured your test step, use the Test button in the Error Handling and Test section to test your configuration settings.

Server Connection Info

Server Connection Info

Server URL:  

User:  

Password:

☐ Share Sessions

☐ Share Publishers

Here you enter the server connection information.

These values should be parameterized with properties that are in your configuration, making it easy to change the application under test. LISA, by default uses the `TIBCO_SERVER` property in the JNDI Server URL. This property must be added to your Configuration if you plan to use it.




The three parameters should be available to you for the system under test. The pull-down menus contain common examples or templates for these values.



- Server URL: The URL of the TIBCO server.
- User.
- Password.
- Share Sessions and Share Publishers checkboxes are used to share JMS Sessions and Publishers throughout the test case. This can lower overhead, but does not always provide a realistic simulation since typically JMS clients want to release resources. Note: If you check Share Publishers, the Share Sessions checkbox is also checked for you as you cannot share publishers without sharing sessions.
- The Stop All button allows you to stop any listeners at design time now. This is to resolve issues during test case creation where some listeners can get orphaned, but will still consume messages making it difficult or sometimes impossible to finish test case creation.
- The Advanced button displays a pop-up panel where you can add custom properties that will be sent with the connection information.


Publisher Info

Publisher Info

☒ enable ☐ use transaction



Name:   

Type:  

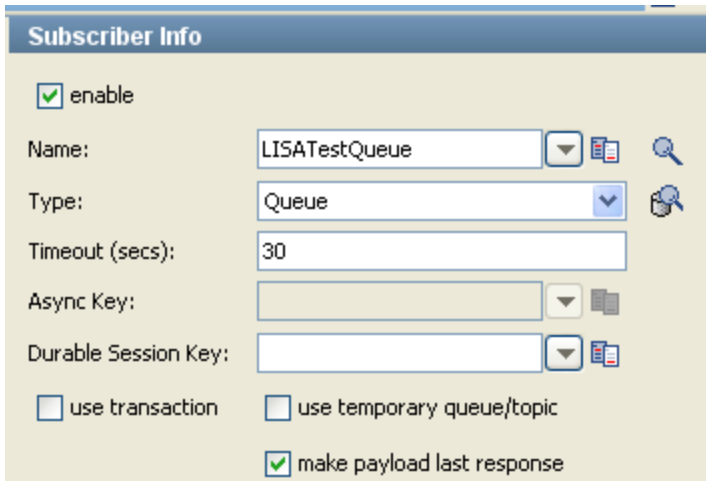
Message: 

Check the **enable** checkbox to set up the ability to send (publish) messages. Click the **use transaction** checkbox to execute a commit when the message is sent.

Enter the following parameters:



- Name: Enter the name of the topic or queue to use. The magnifying glass,  , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue. The browse icon,  , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).
- Message: Select the type of message you are sending from the pull-down menu. The supported messages are: Empty, Text, Object, Bytes, Message, and Mapped (Extended).
- Advanced button: Displays a pop-up panel where you can add custom message properties to be sent with the message.


Subscriber Info




Subscriber Info


☒ enable

Name:  

Type: 

Timeout (secs):

Async Key: 



Durable Session Key: 

☐ use transaction ☐ use temporary queue/topic

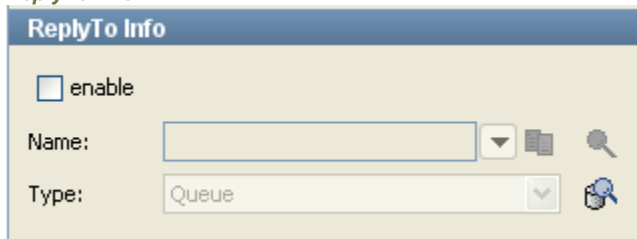
☒ make payload last response

Check the enable checkbox to set up to enable the ability to receive (subscribe to) messages.

Enter the following parameters:



- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue, and whether you want to listen in synchronous or asynchronous mode. For asynchronous mode you will also have to have an entry in the Async key field. The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue only.
- Timeout (secs): Enter the period to wait before LISA interrupts waiting for a message (this field can be left blank for no timeout).
- Async Key: Enter the value needed to identify asynchronous messages. This is only needed in asynchronous mode. It will be used in a subsequent Message Consumer step to retrieve asynchronous messages.
- Durable Session key: By entering a name here you are requesting a durable session. You are also providing a key for that session. A durable session allows you to receive all of your messages from a topic even if you logoff and then logon again.
- use transaction checkbox: Click the use transaction checkbox to execute a Commit when a message is received.
- use temporary que/topic checkbox: Click the use temporary queue/topic checkbox if you want the JMS.provider to set up a temporary queue/topic on your behalf. When a temporary queue/topic is used, LISA will automatically set the JMS ReplyTo parameter of the message you send to the temporary queue/topic. The temporary queue/topic feature must always be used in conjunction with a publisher so that a reply can be sent. If you use a temporary queue/topic the ReplyTo section is disabled by LISA.
- make payload last response: Check this option if you want to make payload as the last response.


ReplyTo Info



ReplyTo Info

☐ enable



Name:  

Type: 

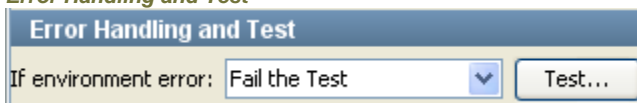
Check the enable checkbox to set up a destination queue/topic.

If your application needs a destination, it is setup in this section.


Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue. The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).

Error Handling and Test



Error Handling and Test

If environment error: 

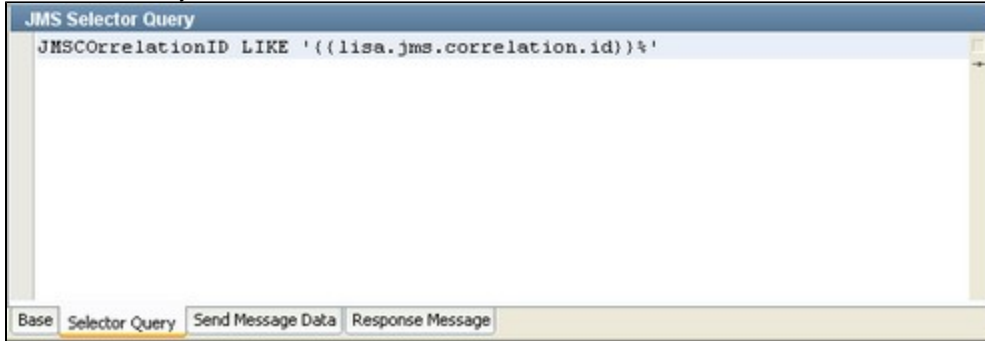
Error Handling and Test allows you to redirect to a step if an error occurs.

- If environment error: Select the step to redirect to if an error occurs.

Click the **Test** button to test your step configuration settings.

Selector Query Tab

The Selector Query tab view is shown below:



You can enter a JMS selector query in this editor. The syntax closely follows SQL. It is a subset of SQL92. For further details see the JMS 1.1 specification, section 3.8. A JMS selector query can be specified when listening for a message on a queue that is a response to a published message. The example above shows a specific query looking for a JMSCorrelationID that matches one set in a LISA property as sent with the original message.

LISA has a built in mechanism for allowing a test creator to set the JMSCorrelationID for a message before sending it. Anytime before the message is sent you can set the correlation ID by setting the following LISA property:
lisa.jms.correlation.id

LISA will detect a non-zero value and set the message JMSCorrelationID property before the message is sent.

Send Message Data Tab

If your step is configured to publish, this is where you compose your message. The Send Message Data tab view is shown below for a text message:



This particular example shows an XML fragment with LISA properties being used. The text can be typed in or it can be read from a file using the **Read Message from the File** button in the bottom-right corner, or it can be stored in a LISA property, in which case you would just place the LISA property in the editor, for example, `LISA_PROP`.

Notice that LISA properties are used in the message XML allowing the message to be created dynamically during the test run.

Response Message Tab

If your step is configured to subscribe, your response will be shown here. For more information refer to [6.1 JMS Messaging \(JNDI\)](#)

10.3 Message Consumer

10.3 Message Consumer

For detailed information regarding this step, please refer to [6.2 Message Consumer](#).

10.4 Read a File (Disk, URL, or Classpath)

10.4 Read a File (Disk, URL, or Classpath)

For detailed information regarding this step, please refer to [5.6 Read a File \(Disk, URL or Class Path\)](#).

10.5 Next Generation Web Service Execution (alpha)

10.5 Next Generation Web Service Execution (alpha)

For detailed information regarding this step, please refer to [1.3 Web Service Execution \(XML\)](#).

10.6 Web Service Execution

10.6 Web Service Execution

For detailed information regarding this step, please refer to [1.9 Web Service Execution \(Legacy\)](#).

10.7 Raw Soap Request

10.7 Raw Soap Request

For detailed information regarding this step, please refer to [1.5 Raw SOAP Request](#).

10.8 SQL Database Execution

10.8 SQL Database Execution

For detailed information regarding this step, please refer to [3.1 SQL Database Execution \(JDBC\)](#).

10.9 FTP Step

10.9 FTP Step

For detailed information regarding this step, please refer to [5.7 FTP Step](#).

11. Sonic Steps

11. Sonic Steps

The following Test Steps are available in this chapter.

- [11.1 SonicMQ Messaging \(Native\)](#)
- [11.2 SonicMQ Messaging \(JNDI\)](#)
- [11.3 Message Consumer](#)
- [11.4 Read a File \(Disk, URL, or Classpath\)](#)
- [11.5 Next Generation Web Service Execution \(alpha\)](#)
- [11.6 Web Service Execution](#)
- [11.7 Raw Soap Request](#)
- [11.8 SQL Database Execution](#)
- [11.9 FTP Step](#)

11.1 SonicMQ Messaging (Native)

11.1 SonicMQ Messaging (Native)

SonicMQ Messaging (Native) supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

The SonicMQ Messaging (Native) step allows you to send messages to, and receive messages from topics and queues, using native Sonic

protocol. You can also receive, modify and forward an existing message.

The SonicMQ Messaging (Native) step is configured using a single LISA editor regardless the messaging requirements. Input options will vary on the messaging requirements. The step editor will only allow valid configurations, so when you enable certain features others may become inactive.

Prerequisites - You will be required to supply the necessary jar files for your chosen configuration and connection.

Sonic requires several jar files to be added to your CLASSPATH. You can put them in the **hotdeploy** directory. The jar files can be found in the lib directory of the Sonic installation:

- Sonic_XA.jar
- mfcontext.jar
- sonic_Client.jar

Parameter Requirements - You need the connection parameters and the subject names used in the application under test. The parameters needed are listed below in the Server Connection Info section. There may be other parameters required, depending on your environment. These should be obtained in advance from the developers of the application.

The messaging step editor for SonicMQ Messaging (Native) is shown below:

The screenshot shows the SonicMQ Messaging (Native) step editor. It is a web-based form with a light beige background and blue headers. The form is organized into four main sections, each with a blue header: **Server Connection Info**, **Subscriber Info**, **ReplyTo Info**, and **Publisher Info**.
 - **Server Connection Info**: Contains fields for Broker Host (xpp-sonicmq), Broker Port (2506), User, and Password.
 - **Subscriber Info**: Contains a checkbox for 'enable' (checked), Name (SimpleQ1), Type (Queue), Timeout (secs) (30), Async Key, Durable Session Key, and checkboxes for 'use transaction', 'use temporary queue/topic', and 'make payload last response' (checked).
 - **ReplyTo Info**: Contains a checkbox for 'enable' (unchecked), Name, and Type (Queue).
 - **Publisher Info**: Contains a checkbox for 'enable' (checked), a checkbox for 'use transaction' (unchecked), Name (SimpleQ1), Type (Queue), and Message (Text).
 - **Error Handling and Test**: Located at the bottom of the main form area, it includes a dropdown for 'If environment error' (Fail the Test) and a 'Test...' button.
 - **Bottom Tabs**: A row of four tabs: 'Base' (selected), 'Selector Query', 'Send Message Data', and 'Response Message'.

These are the tabs available at the bottom of the editor.

- The **Base** tab is where you define your connection and messaging parameters.
- The **Selector Query** tab allows you to specify a selector query to be run when listening for a message on a queue.
- The **Send Message Data** tab is where you will create your message content.
- The **Response Message** tab is where your response messages will be posted.

Base Information (Base tab)

The Base tab view is shown in the figure above. It is divided into 5 major sections:

- Server Connection Info.
- Subscriber Info.

- Publisher Info.
- ReplyTo Info.
- Error Handling and Test.

The Server Connection Info and Error Handling and Test sections are always active. The Subscriber Info, Publisher Info, and, ReplyTo Info sections can be enabled or disabled using the enable checkbox in the top left corner of each section. Using these checkboxes you can configure the step to be a publish step, a subscribe step, or both. You can also choose to include a "replyto" component in the step. When you have completely configured your test step, use the Test button in the Error Handling and Test section to test your configuration settings.

Server Connection Info

Here you enter the connection information specific to native SonicMQ.

The four parameters should be available to you for the system under test.

- Broker Host.
- Broker Port.
- User.
- Password.

These values should be parameterized with properties that are in your Configuration, making it easy to change for a different system under test.

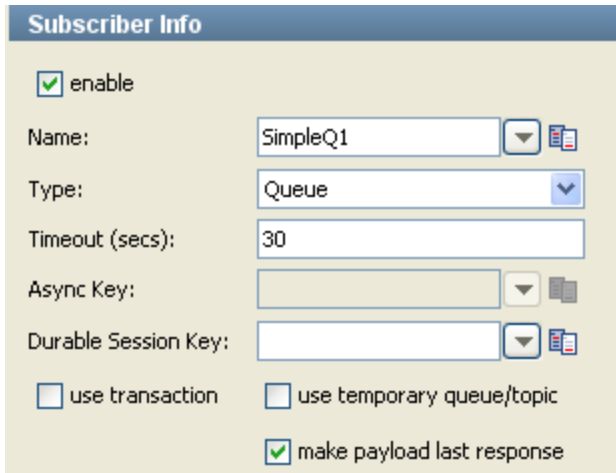
Publisher Info

Check the enable checkbox to set up the ability to send (publish) messages.

Enter the following parameters:



- Name: Enter the name of the topic or queue to use.
- Type: Select whether you are using a topic or queue.
- Message: Select the type of message you are sending from the pull-down menu. The supported messages are: Empty, Text, Object, Bytes, Message, and Mapped (Extended).
- Advanced button: Displays a pop-up panel where you can add custom message properties to be sent with the message.


Subscriber Info





Subscriber Info



☒ enable

Name:  

Type: 

Timeout (secs):

Async Key:  

Durable Session Key:  

☐ use transaction ☐ use temporary queue/topic

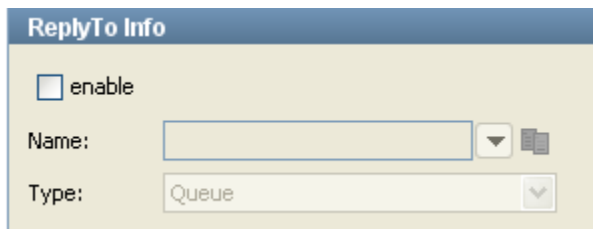
☒ make payload last response

Check the enable checkbox to set up to enable the ability to receive (subscribe to) messages.



Enter the following parameters:


- Name: Enter the name of the topic or queue to use.
- Type: Select whether you are using a topic or queue, and whether you want to listen in synchronous or asynchronous mode. For asynchronous mode you will also have to have an entry in the Async key field (see below).
- Timeout (secs): Enter the period to wait before LISA interrupts waiting for a message (this field can be left blank for no timeout).
- Async Key: Enter the value needed to identify asynchronous messages. This is only needed in asynchronous mode. It will be used in a subsequent Message Consumer step to retrieve asynchronous messages.
- Durable Session key: By entering a name here you are requesting a durable session. You are also providing a key for that session. A durable session allows you to receive all of your messages from a topic even if you logoff, and then logon again.
- use transaction checkbox: Click the use transaction checkbox to execute a Commit when a message is received.
- use temporary que/topic checkbox: Click the use temporary queue/topic checkbox if you want the JMS.provider to set up a temporary queue/topic on your behalf. When a temporary queue/topic is used, LISA will automatically set the JMS ReplyTo parameter of the message you send to the temporary queue/topic. The temporary queue/topic feature must always be used in conjunction with a publisher so that a reply can be sent. If you use a temporary queue/topic the ReplyTo section is disabled by LISA.
- make payload last response: Check this option if you want to make payload as the last response.

ReplyTo Info



☐ enable

Name:  

Type: 

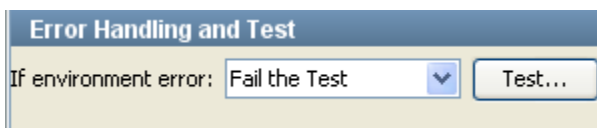
Check the enable checkbox to set up a destination subject.

If your application needs a destination, it is setup in this section.


Enter the following parameters:

- Name: Enter the name of the topic or queue to use.
- Type: Select whether you are using a topic or queue.

Error Handling and Test



Error Handling and Test

If environment error: 

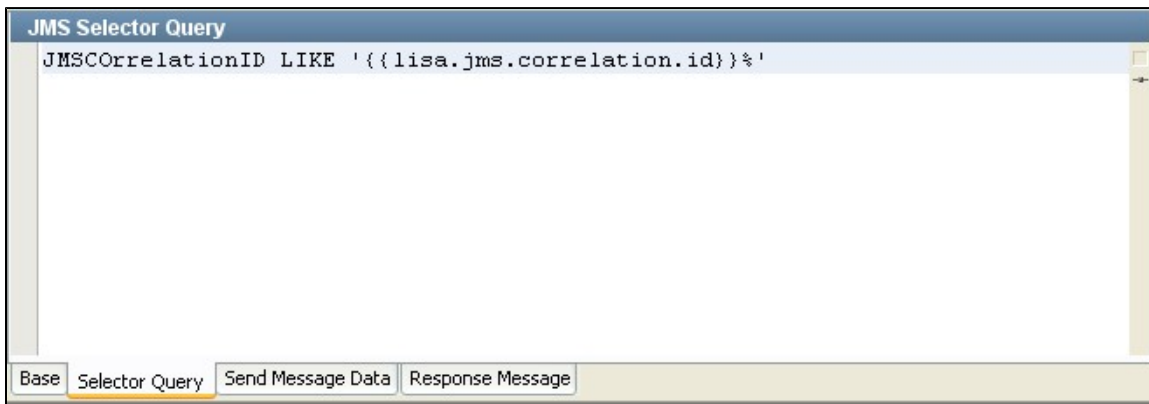
Error Handling and Test allows you to redirect to a step if an error occurs.

- If environment error: Select the step to redirect to if an error occurs.

Click the Test button to test your step configuration settings.

Selector Query Tab

The Selector Query tab view is shown below:



You can enter a JMS selector query in this editor. The syntax closely follows SQL. It is a subset of SQL92. For further details see the JMS 1.1 specification, section 3.8. A JMS selector query can be specified when listening for a message on a queue that is a response to a published message. The example above shows a specific query looking for a JMSCorrelationID that matches one set in a LISA property as sent with the original message.

LISA has a built in mechanism for allowing a test creator to set the JMSCorrelationID for a message before sending it. Anytime before the message is sent you can set the correlation ID by setting the following LISA property:
lisa.jms.correlation.id

LISA will detect a non-zero value and set the message JMSCorrelationID property before the message is sent.

Send Message Data Tab

If your step is configured to publish, this is where you compose your message. The Send Message Data tab view is shown below for a text message:



This particular example shows an XML fragment with LISA properties being used. The text can be typed in or it can be read from a file using the **Read Message from the File** button in the bottom right corner, or it can be stored in a LISA property, in which case you would just place the LISA property in the editor, for example, `LISA_PROP`.

Notice that LISA properties are used in the message XML allowing the message to be created dynamically during the test run.

Response Message Tab

If your step is configured to subscribe, your response will be shown here. For more information refer to [6.1 JMS Messaging \(JNDI\)](#)

11.2 SonicMQ Messaging (JNDI)

11.2 SonicMQ Messaging (JNDI)

SonicMQ Messaging (JNDI) supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

The SonicMQ Messaging (JNDI) step allows you to send messages to, and receive messages from topics and queues. You can also receive, modify and forward an existing message.

The SonicMQ Messaging (JNDI) step is configured using a single LISA editor regardless of the messaging requirements. Input options will vary on the messaging requirements. The editor will only allow valid configurations, so when you enable certain features others may become inactive.

Prerequisites - You will be required to supply the necessary jar files for your chosen configuration and connection.

Sonic requires several jar files be added to your CLASSPATH. You can put them in the hot deploy directory. The jar files can be found in the lib directory of the Sonic installation:

- Sonic_XA.jar
- mfcontext.jar
- sonic_Client.jar

Parameter Requirements - You need the connection parameters and the subject names used in the application under test. The parameters needed are listed below in the **Server Connection Info** section. There may be other parameters required, depending on your environment. These should be obtained in advance from the developers of the application.

The messaging step editor for SonicMQ Messaging (JNDI) is shown below:

The screenshot shows the SonicMQ Messaging (JNDI) configuration editor. The editor is divided into four main sections: Server Connection Info, Subscriber Info, ReplyTo Info, and Publisher Info. The Base tab is selected at the bottom. The Server Connection Info section includes fields for JNDI Factory Class, JNDI Server URL, JMS ConnectionFactory, User, and Password, along with checkboxes for Share Sessions and Share Publishers. The Subscriber Info section includes an enable checkbox, Name, Type, Timeout (secs), Async Key, Durable Session Key, and checkboxes for use transaction, use temporary queue/topic, and make payload last response. The ReplyTo Info section includes an enable checkbox, Name, and Type. The Publisher Info section includes an enable checkbox, use transaction checkbox, Name, Type, and Message. The Error Handling and Test section includes a dropdown for 'If environment error' and a 'Test...' button. The bottom of the editor shows four tabs: Base, Selector Query, Send Message Data, and Response Message.

There are four tabs available at the bottom of the editor.

- The **Base** tab is where you define your connection and messaging parameters.
- The **Selector Query** tab allows you to specify a selector query to be run when listening for a message on a queue.
- The **Send Message Data** tab is where you will create your message content.
- The **Response Message** tab is where your response messages will be posted.

Base Information (Base tab)

The Base tab view is shown in the figure above. It is divided into 5 major sections:

- Server Connection Info.
- Subscriber Info.
- Publisher Info.
- ReplyTo Info.
- Error Handling and Test.

The Server Connection Info and Error Handling and Test sections are always active. The Subscriber Info, Publisher Info, and, ReplyTo Info sections can be enabled or disabled using the enable checkbox in the top left corner of each section. Using these checkboxes you can configure the step to be a publish step, a subscribe step, or both. You can also choose to include a "replyto" component in the step. When you have completely configured your test step, use the Test button in the Error Handling and Test section to test your configuration settings.

Server Connection Info

Server Connection Info

JNDI Factory Class:

JNDI Server URL:

JMS ConnectionFactory:

User:

Password:


☐ Share Sessions

☐ Share Publishers

Enter the JNDI information.

These values should be parameterized with properties that are in your configuration, making it easy to change the application under test. LISA, by default uses the SONICMQ_SERVER property in the JNDI Server URL. This property must be added to your Configuration if you plan to use it.

The five parameters should be available to you for the system under test. The pull-down menus contain common examples or templates for these values.

- JNDI Factory Class: LISA pre-populates this field with default values.
- JNDI Server URL: LISA pre-populates this field with default values.
- JMS Connection Factory: You can use the magnifying glass, , to browse available resources on the server. Select or type in a Connection Factory per the JMS specification to use for this Step execution.
- User
- Password
- Share Sessions and Share Publishers checkboxes are used to share JMS Sessions and Publishers throughout the test case. This can lower overhead, but does not always provide a realistic simulation since typically JMS clients want to release resources. Note: If you check Share Publishers, the Share Sessions checkbox is also checked for you as you cannot share publishers without sharing sessions.
- The Stop All button allows you to stop any listeners at design time now. This is to resolve issues during test case creation where some listeners can get orphaned, but will still consume messages making it difficult or sometimes impossible to finish test case creation.
- The Advanced button displays a pop-up panel where you can add custom properties that will be sent with the connection information.

Publisher Info

☒ enable ☐ use transaction



Name:

Type:

Message:

Check the enable checkbox to set up the ability to send (publish) messages. Click the use transaction checkbox to execute a commit when the message is sent.



Enter the following parameters:


- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue. The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).
- Message: Select the type of message you are sending from the pull-down menu. The supported messages are: Empty, Text, Object, Bytes, Message, and Mapped (Extended).
- Advanced button: Displays a pop-up panel where you can add custom message properties to be sent with the message.

Subscriber Info



Subscriber Info



☒ enable

Name:  

Type: 

Timeout (secs):

Async Key:  



Durable Session Key:  

☐ use transaction ☐ use temporary queue/topic

☒ make payload last response

Check the enable checkbox to set up to enable the ability to receive (subscribe to) messages.

Enter the following parameters:


- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue, and whether you want to listen in synchronous or asynchronous mode. For asynchronous mode you will also have to have an entry in the Async key field (see below). The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).
- Timeout (secs): Enter the period to wait before LISA interrupts waiting for a message (this field can be left blank for no timeout).
- Async Key: Enter the value needed to identify asynchronous messages. This is only needed in asynchronous mode. It will be used in a subsequent Message Consumer step to retrieve asynchronous messages.
- Durable Session key: By entering a name here you are requesting a durable session. You are also providing a key for that session. A durable session allows you to receive all of your messages from a topic even if you logoff and then logon again.
- use transaction checkbox: Click the use transaction checkbox to execute a Commit when a message is received.
- use temporary que/topic checkbox: Click the use temporary queue/topic checkbox if you want the JMS.provider to set up a temporary queue/topic on your behalf. When a temporary queue/topic is used, LISA will automatically set the JMS ReplyTo parameter of the message you send to the temporary queue/topic. The temporary queue/topic feature must always be used in conjunction with a publisher so that a reply can be sent. If you use a temporary queue/topic the ReplyTo section is disabled by LISA.

ReplyTo Info

ReplyTo Info

☐ enable



Name:  

Type: 

Check the enable checkbox to set up a destination queue/topic.


If your application needs a destination, it is setup in this section.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue. The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).

Error Handling and Test

Error Handling and Test

If environment error: 

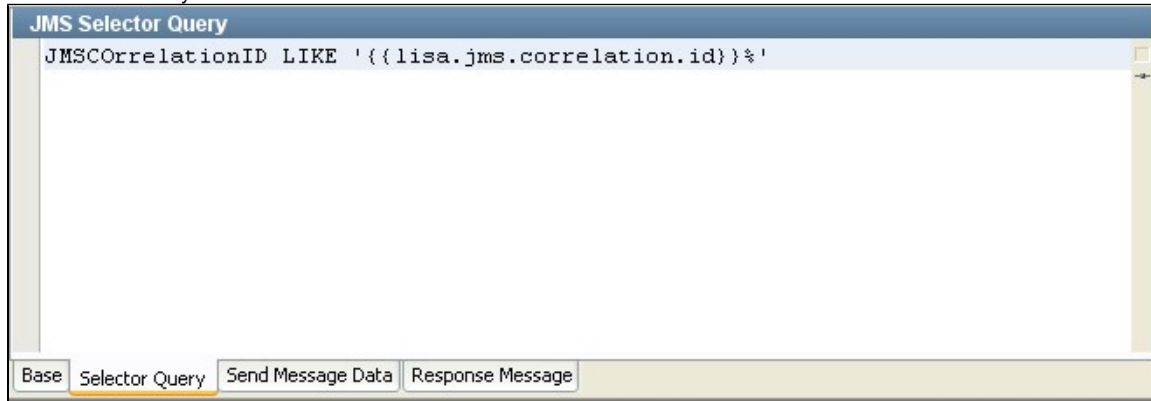
Error Handling and Test allows you to redirect to a step if an error occurs.

- If environment error: Select the step to redirect to if an error occurs.

Click the **Test** button to test your step configuration settings.

Selector Query Tab

The Selector Query tab view is shown below:



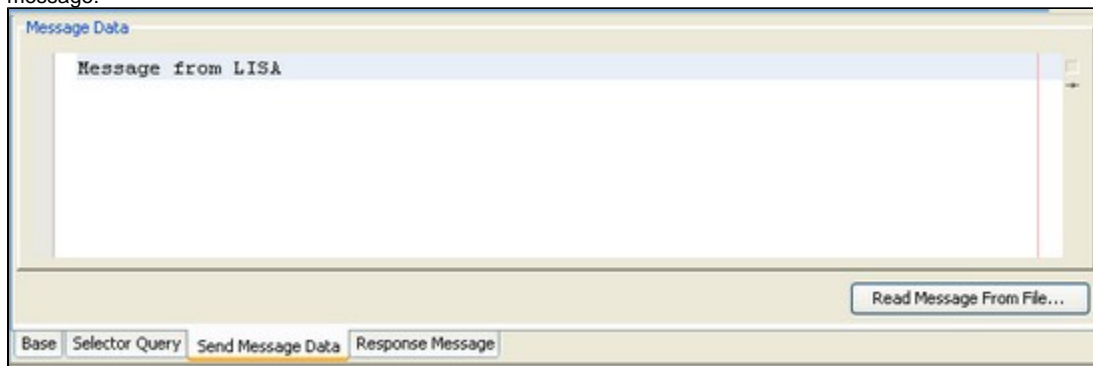
You can enter a JMS selector query in this editor. The syntax closely follows SQL. It is a subset of SQL92. For further details see the JMS 1.1 specification, section 3.8. A JMS selector query can be specified when listening for a message on a queue that is a response to a published message. The example above shows a specific query looking for a JMSCorrelationID that matches one set in a LISA property as sent with the original message.

LISA has a built in mechanism for allowing a test creator to set the JMSCorrelationID for a message before sending it. Anytime before the message is sent you can set the correlation ID by setting the following LISA property:
lisa.jms.correlation.id

LISA will detect a non-zero value and set the message JMSCorrelationID property before the message is sent.

Send Message Data Tab

If your step is configured to publish, this is where you compose your message. The **Send Message Data** tab view is shown below for a text message:



This particular example shows an XML fragment with LISA properties being used. The text can be typed in or it can be read from a file using the **Read Message from the File** button in the bottom right corner, or it can be stored in a LISA property, in which case you would just place the LISA property in the editor i.e. something like `LISA_PROP`.

Notice that LISA properties are used in the message XML allowing the message to be created dynamically during the test run.

Response Message Tab

If your step is configured to subscribe, your response will be shown here. For more information refer to [6.1 JMS Messaging \(JNDI\)](#).

11.3 Message Consumer

11.3 Message Consumer

For detailed information regarding this step, please refer to [6.2 Message Consumer](#).

11.4 Read a File (Disk, URL, or Classpath)

11.4 Read a File (Disk, URL, or Classpath)

For detailed information regarding this step, please refer to [5.6 Read a File \(Disk, URL or Class Path\)](#).

11.5 Next Generation Web Service Execution (alpha)

11.5 Next Generation Web Service Execution (alpha)

For detailed information regarding this step, please refer to [1.3 Web Service Execution \(XML\)](#).

11.6 Web Service Execution

11.6 Web Service Execution

For detailed information regarding this step, please refer to [1.9 Web Service Execution \(Legacy\)](#).

11.7 Raw Soap Request

11.7 Raw Soap Request

For detailed information regarding this step, please refer to [1.5 Raw SOAP Request](#).

11.8 SQL Database Execution

11.8 SQL Database Execution

For detailed information regarding this step, please refer to [3.1 SQL Database Execution \(JDBC\)](#).

11.9 FTP Step

11.9 FTP Step

For detailed information regarding this step, please refer to [5.7 FTP Step](#).

12. WebMethods Steps

12. WebMethods Steps

The following Test Steps are available in this chapter.

- [12.1 WebMethods Broker](#)
- [12.2 webMethods Integration Server Services](#)
- [12.3 Message Consumer](#)
- [12.4 Read a File \(Disk, URL, or Classpath\)](#)
- [12.5 Next Generation Web Service Execution \(alpha\)](#)
- [12.6 Web Service Execution](#)
- [12.7 Raw Soap Request](#)
- [12.8 SQL Database Execution](#)
- [12.9 FTP Step](#)

12.1 WebMethods Broker

12.1 WebMethods Broker

WebMethods Broker supports Mapped (Extended) messages that will create Broker Events.

The webMethods Broker step allows you to send messages to, and receive messages from the Broker. You can also receive, modify and forward an existing Broker Events/ Messages.

The webMethods Broker step is configured using a single LISA editor regardless the messaging requirements. Input options will vary on the messaging requirements. The step editor will only allow valid configurations, so when you enable certain features others may become inactive.

Prerequisites - You will be required to supply the necessary jar files for your chosen configuration and connection.

The webMethods Broker requires that you put several jar files in your CLASSPATH. The jar files can be found in the **lib** directory of the webMethods installation:

- wm-client.jar
- wm-enttoolkit.jar
- wmbrokerclient.jar
- wmjmsadmin.jar
- wmjmsclient.jar
- wmjmsnaming.jar

Parameter Requirements - You need the connection parameters and the subject names used in the application under test. The parameters needed are listed below in the Server Connection Info section. There may be other parameters required, depending on your environment. These should be obtained in advance from the developers of the application.

The messaging step editor for webMethods Broker is shown below:

The screenshot shows the webMethods Broker step editor interface. It is divided into four main sections: **Server Connection Info**, **Subscriber Info**, **ReplyTo Info**, and **Publisher Info**. At the bottom, there is an **Error Handling and Test** section and a tabbed interface with three tabs: **Base**, **Send Message Data**, and **Response Message**.

Server Connection Info:

- Broker Host: localhost
- Host Port: 6849
- Broker Name:
- Client ID:
- Client Group:
- App Name: LISA

Subscriber Info:

- ☐ enable
- docType:
- Timeout (secs): 30
- Async Key:
- ☐ Auto convert to: ☒ string ☐ xml

ReplyTo Info:

- ☐ enable
- Name:

Publisher Info:

- ☒ enable
- docType:
- Message: webMethods Broker
- ☐ Force Document Pre-fill
- ☐ Deliver Enabled
- Deliver Client ID:
- Envelope Tag:

Error Handling and Test:

- If environment error: Fail the Test
- Test...

There are three tabs available at the bottom of the editor.

- The **Base** tab is where you define your connection and messaging parameters.
- The **Send Message Data** tab is where you will create your message content.
- The **Response Message** tab is where your response messages will be posted.

Base Tab

The Base tab view is shown in the figure above. It is divided into 5 major sections:

- Server Connection Info.

- Subscriber Info.
- Publisher Info.
- ReplyTo Info.
- Error Handling and Test.

The Server Connection Info and Error Handling and Test sections are always active. The Subscriber Info, Publisher Info, and ReplyTo Info sections can be enabled or disabled using the enable checkbox in the top left corner of each section. Using these checkboxes you can configure the step to be a publish step, a subscribe step, or both. You can also choose to include a "replyto" component in the step. When you have completely configured your test step, use the Test button in the Error Handling and Test section to test your configuration settings

Server Connection Info

Server Connection Info	
Broker Host:	localhost
Host Port:	6849
Broker Name:	
Client ID:	
Client Group:	
App Name:	LISA

Here you enter the connection information specific to webMethods Broker:

The four parameters should be available to you for the system under test.

- Broker Host.
- Host Port.
- Broker Name.
- Client ID.
- Client Group: This is the Client group able to see the Broker destinations you wish to use.
- App Name: You can specify the application using the Broker here. This is an optional parameter and mostly used in server logs for debugging. We default to LISA for this purpose. It is a good practice to do this, but if you need to use something else for application logic you can.

These values should be parameterized with properties that are in your Configuration, making it easy to change for a different system under test.

Publisher Info

Publisher Info	
<input checked="" type="checkbox"/> enable	
docType:	
Message:	webMethods Broker
<input type="checkbox"/> Force Document Pre-fill	
<input type="checkbox"/> Deliver Enabled	
Deliver Client ID:	
Envelope Tag:	

Check the enable checkbox to set up the ability to send (publish) messages.

Enter the following parameters:

- docType: Enter the name of the docType to use.
- Message: Select the type of message you are sending from the pull-down menu. The supported messages are: webMethods Broker, Object, Message, and Mapped (Extended).
- Force Document Pre-fill: LISA inspects the selected docType and can pre-load the message with the required fields. This check box allows you to have made modifications and decide you want any missing fields re-added. It will not write over existing fields with the same name. This property is only a design time effect and does nothing at test runtime.
- Deliver Enabled : (TODO)
- Deliver Client ID : (TODO)
- Envelope Tag :This allows the user to set the _env.tag property on a broker event message.

Subscriber Info



The 'Subscriber Info' form has a title bar 'Subscriber Info'. It contains an 'enable' checkbox. Below it are three fields: 'docType:' with a text input, a dropdown arrow, a document icon, and a magnifying glass icon; 'Timeout (secs):' with a text input containing '30'; and 'Async Key:' with a text input, a dropdown arrow, a document icon, and a magnifying glass icon. At the bottom is an 'Auto convert to:' section with a checkbox, a radio button selected for 'string', and a radio button for 'xml'.

Check the enable checkbox to set up to enable the ability to receive (subscribe to) messages.

Enter the following parameters:

- docType: Enter the name of the docType to use.
- Timeout (secs): Enter the period to wait before LISA interrupts waiting for a message (this field can be left blank for no timeout).
- Async Key: Enter the value needed to identify asynchronous messages. This is only needed in asynchronous mode. It will be used in a subsequent Message Consumer step to retrieve asynchronous messages.
- Auto convert to : (TODO)

ReplyTo Info



The 'ReplyTo Info' form has a title bar 'ReplyTo Info'. It contains an 'enable' checkbox. Below it is a 'Name:' field with a text input, a dropdown arrow, a document icon, and a magnifying glass icon.

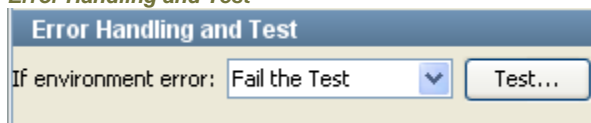
Check the enable checkbox to set up a destination queue/topic.

If your application needs a destination, it is setup in this section.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass, !magnifying glass.jpg|border=1! , can be used to browse the JNDI server for the topic or queue name.

Error Handling and Test



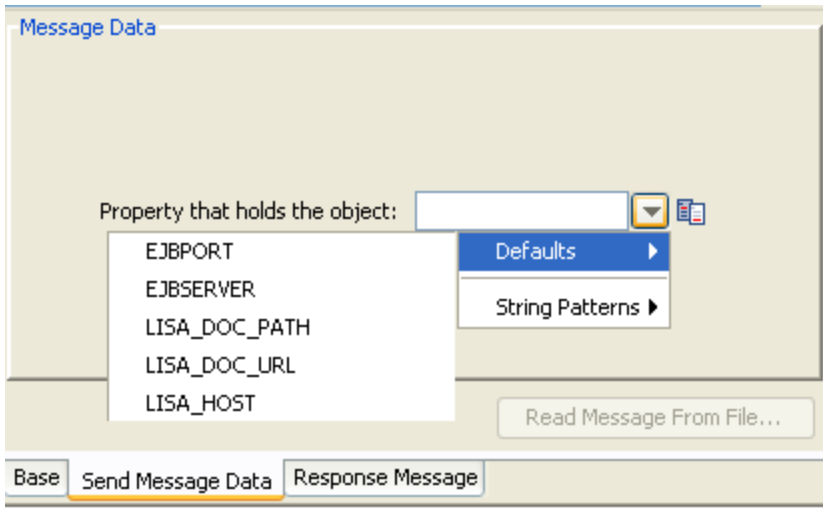
The 'Error Handling and Test' form has a title bar 'Error Handling and Test'. It contains a label 'If environment error:' followed by a dropdown menu showing 'Fail the Test' and a 'Test...' button.

Error Handling and Test allows you to redirect to a step if an error occurs.

- If environment error: Select the step to redirect to if an error occurs.

Click the **Test** button to test your step configuration settings.

Send Message Data Tab



This particular example shows how to select an object for the message. The message is stored in a LISA property.

Response Message Tab

If your step is configured to subscribe, your response will be shown here. For more information refer to [6.1 JMS Messaging \(JNDI\)](#)

12.2 webMethods Integration Server Services

12.2 webMethods Integration Server Services

The webMethods Integration Server Services step allows you to execute Integration Server services through the native Java APIs. This is done using IData objects so it works with services not exposed via HTTP transports.

Prerequisites - You will be required to supply the necessary jar files for your chosen configuration and connection. See the list of jar files in the previous topic– [webMethods Broker](#).

Parameter Requirements - You need the connection parameters and the subject names used in the application under test. The parameters needed are listed below in the Server Connection Info section. There may be other parameters required, depending on your environment. These should be obtained in advance from the developers of the application.

The messaging step editor for webMethods Integration Server Services is shown below:

Server Connection Info

Enter the following parameters:

- Host: Enter the Host name.
- User: Enter the User name
- Password: Enter the Password
- Package: This is the package the service is located in.
- Service: The name of the actual service you want to call.
- Input Type : (TODO)
- Output Type : (TODO)
- If environment error: Select the step to redirect to if an error occurs.

Click **Execute** to connect. You will see an object response. Export this object into a Java execution Step to pull the payload or other properties from the response which is an IData object itself. This can easily be done by creating a new Java Step in LISA and loading from property specifying the step name pattern for a last response. This is **lisa.<stepName>.rsp**.

12.3 Message Consumer

12.3 Message Consumer

For detailed information regarding this step, please refer to [6.2 Message Consumer](#).

12.4 Read a File (Disk, URL, or Classpath)

12.4 Read a File (Disk, URL, or Classpath)

For detailed information regarding this step, please refer to [5.6 Read a File \(Disk, URL or Class Path\)](#).

12.5 Next Generation Web Service Execution (alpha)

12.5 Next Generation Web Service Execution (alpha)

For detailed information regarding this step, please refer to [1.3 Web Service Execution \(XML\)](#).

12.6 Web Service Execution

12.6 Web Service Execution

For detailed information regarding this step, please refer to [1.9 Web Service Execution \(Legacy\)](#).

12.7 Raw Soap Request

12.7 Raw Soap Request

For detailed information regarding this step, please refer to [1.5 Raw SOAP Request](#).

12.8 SQL Database Execution

12.8 SQL Database Execution

For detailed information regarding this step, please refer to [3.1 SQL Database Execution \(JDBC\)](#).

12.9 FTP Step

12.9 FTP Step

For detailed information regarding this step, please refer to [5.7 FTP Step](#).

13. IBM Steps

13. IBM Steps

The following Test Steps are available in this chapter.

13.1 IBM Websphere MQ

13.1 IBM Websphere MQ

IBM Websphere MQ supports all the common message types including Empty, Text, Object, Bytes and Message and Mapped (Extended).

The IBM Websphere MQ step allows you to send messages to, and receive messages from topics and queues. You can also receive, modify and forward an existing message.

The IBM Websphere MQ step is configured using a single LISA editor regardless the messaging requirements. Input options will vary on the messaging requirements. The editor will only allow valid configurations, so when you enable certain features others may become inactive.

Prerequisites - To use Websphere MQ you will need to add several IBM JMS jar files to the **LISA_HOME/lib** folder (or put them on the CLASSPATH some other way). The jar files you will need are:

- com.ibm.mqjms.jar
- com.ibm.mqbind.jar
- com.ibm.mq.pcf.jar
- com.ibm.mq.jar

These can be found in your Websphere MQ installation.

Note: The jar file names may be different depending on the version of Websphere you are using. The ones shown above are for version 5.2.

Parameter Requirements – You will need to have the connection parameters for your system under test. The parameters needed are listed below in the Server Connection Info section.

The messaging step editor for Websphere MQ is shown below:

The screenshot shows the IBM Websphere MQ messaging step editor. It is divided into four main sections: **Server Connection Info**, **Subscriber Info**, **ReplyTo Info**, and **Publisher Info**. At the bottom, there is an **Error Handling and Test** section and a tabbed interface with four tabs: **Base**, **Selector Query**, **Send Message Data**, and **Response Message**.

Server Connection Info: Fields include Host Name, TCP/IP Port, Channel, Queue Manager, CCID, User, Password, and Client Mode (set to JMS). There are buttons for Advanced and Stop All, and a checkbox for Share Sessions.

Subscriber Info: Fields include Name, Type (set to Queue), Timeout (secs) (set to 30), Queue Model, Async Key, and Durable Session Key. There are checkboxes for use transaction, use temporary queue/topic, make payload last response (checked), and use correlation ID for subscribe.

ReplyTo Info: Fields include Name, Type (set to Queue), and Queue Manager. There is a checkbox for enable.

Publisher Info: Fields include Name, Type (set to Queue), Message (set to Empty), and Alt QManager. There is a checkbox for enable and use transaction.

Error Handling and Test: Fields include If environment error (set to Abort the Test) and a Test... button.

There are four tabs available at the bottom of the editor.

- The **Base** tab is where you define your connection and messaging parameters.
- The **Selector Query** tab allows you to specify a selector query to be run when listening for a message on a queue.
- The **Send Message Data** tab is where you will create your message content.
- The **Response Message** tab is where your response messages will be posted.

Base Information (Base tab)

The Base tab view is shown in the figure above. It is divided into 5 major sections: Server Connection Info, Subscriber Info, Publisher Info, ReplyTo Info, and Error Handling and Test.

The Server Connection Info and Error Handling and Test sections are always active.

The Subscriber Info, Publisher Info, and ReplyTo Info sections can be enabled or disabled using the enable checkbox in the top left corner of each section. Using these checkboxes you can configure the step to be a publish step, a subscribe step, or both. You can also choose to include a "replyto" component in the step. When you have completely configured your test step, use the Test button in the Error Handling and Test section to test your configuration settings.

Server Connection Info

The screenshot shows a configuration window titled "Server Connection Info". It contains the following fields and controls:

- Host Name: Text input field with a dropdown arrow and a document icon.
- TCP/IP Port: Text input field with a dropdown arrow and a document icon.
- Channel: Text input field with a dropdown arrow and a document icon.
- Queue Manager: Text input field with a dropdown arrow and a document icon.
- CCID: Text input field with a dropdown arrow and a document icon.
- User: Text input field with a dropdown arrow and a document icon.
- Password: Text input field.
- Client Mode: Dropdown menu currently showing "JMS".
- Buttons: "Advanced" and "Stop All".
- Checkbox: "Share Sessions" (unchecked).

Here you enter the JNDI information. These parameters should be available to you for the system under test.




To connect to Websphere MQ you need to enter the following information:



- Host Name.
- TCP/IP Port.
- Channel: Familiar to Websphere MQ users, a connection property that used for routing and management in the message bus.
- Queue Manager: Familiar to Websphere MQ users, a connection property that used for routing and management in.
- CCID: Optional for connections and will only apply if Character transformation needs to occur between the client (LISA) and server.
- User.
- Password.
- Client Mode: Select from JMS, Native Client or Bindings. Allows the user to select how they wish to interact with the MQ server.
- Native Client - A pure java implementation using IBM specific APIs.
- JMS – A pure java implementation based on the JMS specification.
- Bindings - Requires access to the native libraries from a WebsphereMQ client installation. You must make sure these libraries are accessible by the LISA application runtime. In most cases having these available in the PATH environment is good enough.


Publisher Info



Publisher Info

☒ enable ☐ use transaction

Name:   



Type:  

Message: 

Alt QManager:  

Check the enable checkbox to set up the ability to send (publish) messages. Click the use transaction checkbox to execute a commit when the message is sent.




Enter the following parameters:



- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue. The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).
- Message: Select the type of message you are sending from the pull-down menu. The supported messages are: Empty, Text, Object, Bytes, Message, and Mapped (Extended).
- Alt Qmanager : (TODO)

Subscriber Info



Subscriber Info



☐ enable



Name:   

Type:  

Timeout (secs):

Queue Model:  

Async Key:  

Durable Session Key:  



☐ use transaction ☐ use temporary queue/topic

☒ make payload last response

☐ use correlation ID for subscribe

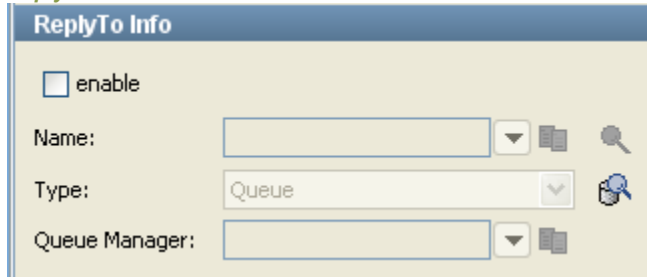
Enable - Select the enable checkbox to set up to enable the ability to receive (subscribe to) messages.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue, and whether you want to listen in synchronous or asynchronous mode. For asynchronous mode you will also have to have an entry in the Async key field (see below). The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).
- Timeout (secs): Enter the period to wait before LISA interrupts waiting for a message (this field can be left blank for no timeout).
- Queue Model: This is required by MQ to create temporary destinations. It is configured on the MQ server. It is only active when "use temporary queue/topic" is checked. In this case the ReplyTo Info section is disabled.
- Async Key: Enter the value needed to identify asynchronous messages. This is only needed in asynchronous mode. It will be used in a subsequent Message Consumer step to retrieve asynchronous messages.
- Durable Session key: By entering a name here you are requesting a durable session. You are also providing a key for that session. A

- durable session allows you to receive all of your messages from a topic even if you logoff, and then logon again.
- use transaction checkbox: Click the use transaction checkbox to execute a Commit when a message is received.
- use temporary que/topic checkbox: Click the use temporary queue/topic checkbox if you want the JMS.provider to set up a temporary queue/topic on your behalf. When a temporary queue/topic is used, LISA will automatically set the JMS ReplyTo parameter of the message you send to the temporary queue/topic. The temporary queue/topic feature must always be used in conjunction with a publisher so that a reply can be sent. If you use a temporary queue/topic the ReplyTo section is disabled by LISA.
- make payload last response: Check this option if you want to make payload as the last response.
- use correlation ID for subscribe: (TODO).

ReplyTo Info





The 'ReplyTo Info' dialog box contains the following fields:

- ☐ enable
- Name: [text field] [dropdown] [magnifying glass icon]
- Type: [Queue] [dropdown] [browse icon]
- Queue Manager: [text field] [dropdown] [magnifying glass icon]

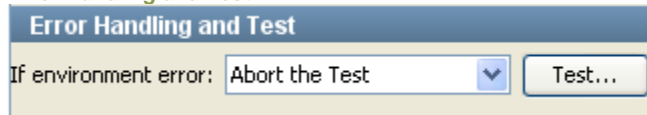
Select the enable checkbox to set up a destination queue/topic.

If your application needs a destination, it is setup in this section.

Enter the following parameters:

- Name: Enter the name of the topic or queue to use. The magnifying glass, , can be used to browse the JNDI server for the topic or queue name.
- Type: Select whether you are using a topic or queue. The browse icon, , to the right of this field can be used to see what messages are waiting to be consumed from a queue (only).
- Queue Manager Allows the replyTo to be on a different Queue Manager than the Publisher (in this step).

Error Handling and Test



The 'Error Handling and Test' dialog box contains the following fields:

- If environment error: [Abort the Test] [dropdown] [Test... button]

Error Handling and Test allows you to redirect to a step if an error occurs.

- If environment error: Select the step to redirect to if an error occurs.

Click the **Test** button to test your step configuration settings.

Selector Query Tab

The Selector Query tab view is show below:



The 'JMS Selector Query' editor shows a text area with the following query:

```
JMSCorrelationID LIKE '({lisa.jms.correlation.id}){'
```

At the bottom, there are tabs: Base, **Selector Query**, Send Message Data, and Response Message.

You can enter a JMS selector query in this editor. The syntax closely follows SQL. It is a subset of SQL92. For further details see the JMS 1.1 specification, section 3.8. A JMS selector query can be specified when listening for a message on a queue that is a response to a published message. The example above shows a specific query looking for a JMSCorrelationID that matches one set in a LISA property as sent with the original message.

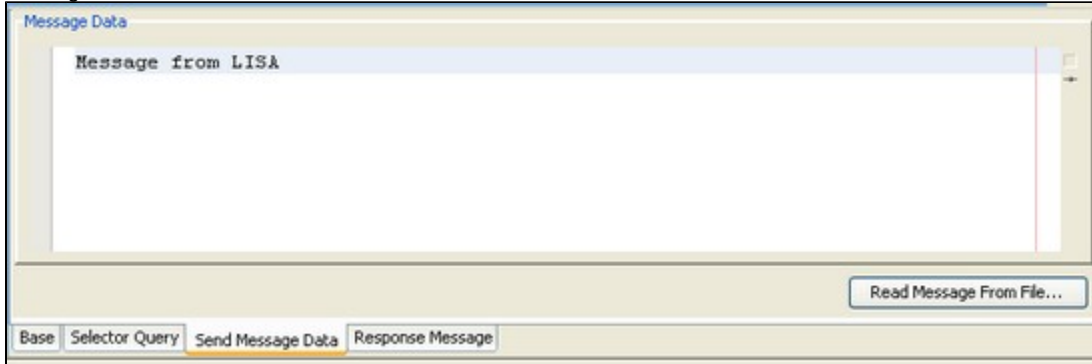
NOTE: LISA has a built in mechanism for allowing a test creator to set the JMSCorrelationID for a message before sending it. Anytime before the

message is sent you can set the correlation ID by setting the following LISA property:
lisa.jms.correlation.id

LISA will detect a non-zero value and set the message JMSCorrelationID property before the message is sent.

Send Message Data Tab

If your step is configured to publish, this is where you compose your message. The Send Message Data tab view is shown below for a text message:



This particular example shows an XML fragment with LISA properties being used. The text can be typed in or it can be read from a file using the Read Message from the File... button in the bottom right corner, or it can be stored in a LISA property, in which case you would just place the LISA property in the editor i.e. something like `LISA_PROP`.

Notice that LISA properties are used in the message XML allowing the message to be created dynamically during the test run.

Response Message Tab

If your step is configured to subscribe, your response will be shown here. For more information refer to [6.1 JMS Messaging \(JNDI\)](#)

13.2 Message Consumer

13.2 Message Consumer

For detailed information regarding this step, please refer to [6.2 Message Consumer](#).

14. Virtual Service Environment Steps

14. Virtual Service Environment Steps

The following Test Steps are available in this chapter.

- 14.1 Virtual Service Router
- 14.2 Virtual Service Tracker
- 14.3 Virtual Conversational_Stateless Response Selector
- 14.4 Virtual HTTP_S Listener
- 14.5 Virtual HTTP_S Live Invocation
- 14.6 Virtual HTTP_S Responder
- 14.7 Virtual JDBC Listener
- 14.8 Virtual JDBC Responder
- 14.9 Socket Server Emulator
- 14.10 Messaging Virtualization Marker
- 14.11 Socket Server Emulator
- 14.12 Compare Strings for Response Lookup
- 14.13 Compare Strings for Next Step Lookup

14.1 Virtual Service Router

14.1 Virtual Service Router

This step is used to route a request from a VS listen step to the response selector step and/or the protocol specific live invocation step. The decision is made based on the current execution mode for the running model.

Virtual Service Execution Router - Step1

Live invocation step: Step1

If environment error: Abort the Test

When in dynamic mode, determine real mode using: Subprocess Script

Test

```
// This script must return either an enum ent
// a string that is the name of an enum entry
// not be returned. It will be executed for
// only.
import com.itko.lisa.vse.ExecutionMode;

return ExecutionMode.EFFICIENT;
```

Name	Type	Value
AGENT_BROKER...	java.lang.String	tcp://localhost:2...
AGENT_UPDATE...	java.lang.Integer	1000
DBCONNURL	java.lang.String	jdbc:derby://loc...
DBDRIVER	java.lang.String	org.apache.derb...
DBNAME	java.lang.String	itko_examples
DBPASSWORD	java.lang.String	sa
DBPORT	java.lang.Integer	3306
DBUSER	java.lang.String	sa
EJBPORT	java.lang.Integer	1099
EJBSERVER	java.lang.String	localhost
EJBSERVER1	java.lang.String	examples.itko.com
ENABLE_CAPTURE	java.lang.Boolean	true
EXAMPLES_HOME	java.lang.String	{{LISA_HOME}}/...
GRE_HOME	java.lang.String	C:\Lisa\bin\xulr...
JMSCONNECTIO...	java.lang.String	ConnectionFactory
JMS_INVOKE_TI...	java.lang.Integer	1000
JMS_POLL_INT	java.lang.Integer	5000
JMS_XML_SERIA...	java.lang.Boolean	true
JNDIFACTORY	java.lang.String	org.jnp.interfac...
JNDIPOINT	java.lang.Integer	1099
JNDIPROTOCOL	java.lang.String	jnp
Key1	java.lang.String	Value
LEK_LOG_CALLS	java.lang.Boolean	true
LISA_BROKER	java.lang.Boolean	true
LISA_CONSOLE	java.lang.Boolean	true
LISA_DOC_PATH	java.lang.String	C:\Lisa\example...
LISA_DOC_URL	java.lang.String	file:/C:/Lisa/exa...
LISA_HOME	java.lang.String	C:\Lisa\
LISA_HOST	java.lang.String	hardwar
LISA_JMX_ITKO...	java.lang.String	{{LISA_HOME}}/...
LISA_JMX_JBOS...	java.lang.String	{{LISA_HOME}}/...
LISA_JMX_JSES	java.lang.String	
LISA_JMX_JSR1...	java.lang.String	{{LISA_HOME}}/...
LISA_JMX_OC4J	java.lang.String	{{LISA_HOME}}/...
LISA_JMX_TOM...	java.lang.String	{{LISA_HOME}}/...
LISA_JMX_WAS...	java.lang.String	{{LISA_HOME}}/...
LISA_JMX_WLS6...	java.lang.String	{{LISA_HOME}}/...
LISA_JMX_WLS9	java.lang.String	{{LISA_HOME}}/...
LISA_LAST_STEP	java.lang.String	
LISA_LOG	java.lang.String	tmanager_log.log
LISA_PROJ_NAME	java.lang.String	examples
LISA_PROJ_ROOT	java.lang.String	C:\Lisa\examples
LISA_TC_PATH	java.lang.String	C:\Lisa\example...
LISA_TC_URL	java.lang.String	file:/C:/Lisa/exa...
LISA_USER	java.lang.String	snehalg
PORT	java.lang.Integer	8080
PORT1	java.lang.Integer	80

14.2 Virtual Service Tracker

14.2 Virtual Service Tracker

This step is used to track responses in a running virtual service and, optionally, validate against a live system. This allows for easier service model debugging and service model “healing”.

Virtual Service Execution Tracker

Image response: lisa.vse.image.response

Live response: lisa.vse.live.response

If environment error: End the Test

14.3 Virtual Conversational_Stateless Response Selector

14.3 Virtual Conversational_Stateless Response Selector

This step is responsible for deciding on an appropriate virtual response for a given request, and can thus be seen as the main step in any VSM. It does that by looking at a service image that is set into it. It is possible that there can be more than one response for a request; therefore, the responses are always emitted as a list. It is typically created by recording and virtualizing some form of service traffic.

Conversational Transaction Response Selector - Step1

Service Image to use: Set to Selected

Request property name: lisa.vse.request

☒ Format step response as XML

If environment error: Abort the Test

ID	Name	Created	Last Modified	Description
----	------	---------	---------------	-------------

View/Edit Selected Service Image Refresh Service Image List

Enter the data for the fields as described:

Field	Description
Service Image to use	From the Service Images List area, select the service image you need. Click Set to Selected . If the service image list is empty and you are not going to record one, go to System > View Virtual Service Images and import one.
Request property name	Set the property name to define the property to look in for the inbound request. Required. Note: This is usually the response of the previous step.
Format step response as XML	By default the step response is formatted as XML. The VSE framework expects Respond steps to accept either a response object, a list of response objects, or an XML document that represents either. If this field is unchecked, the step outputs a list of response objects, even if the list contains only one response.
If environment error	Select the step to go to if an exception occurs. The default is fail.
Service Images List	Lists all the available service images. Select one from the list to view, edit, or set as the service image the step is to use by then clicking Set to Selected .
View/Edit Selected Service Image	After you select a service image from the list, click the View/Edit button to open the Service Images Editor. For more information, see Service Image Editor .

Refresh Service Image List	Click to refresh the list of service images to display changes.
----------------------------	---

14.4 Virtual HTTP_S Listener

14.4 Virtual HTTP_S Listener

This step is used to simulate an HTTP server, including SSL support. It listens for incoming HTTP requests and converts them to a standard virtual request format.

Enter the data for the fields as described:

Field	Description
Listen port	Enter the port on which LISA listens for the HTTP/S traffic.
Bind address	Enter the local IP address on which connections can come in. By default with no bind address specified, the listen step accepts connections on the specified port regardless of the NIC (or IP address) it comes in on.
Bind only	Check this box to acquire the network resource and move to the next step. A second Listen step is required that does not use the Bind only option. This option allows the model to listen on a port (requests are queued until a listen step consumes them) and perform setup tasks before dropping into the wait/process/respond loop. For example, Step 1 of the model acquires the listening port (using Bind only) and step 2 triggers external software that sends requests.
Use SSL	Check this box if you want to simulate a secure HTTPS Web site. Then supply the SSL keystore information.
SSL keystore file	Click Select to browse to your SSL keystore file. The same keystore file must be available to the VSE server to which the VS model is deployed.
Keystore password	Enter the keystore password, and click Verify .
Base path	Identify the HTTP requested resource URIs that the listen step is to process. When the request comes in, the list of queue names is scanned for a name (base path) that starts the URI on the request. The queue name that matches is the one into which the request is placed and the listen step that is associated with the queue (by base path) processes the request.
Format step response as XML	By default the step response is formatted as XML. The VSE framework expects Respond steps to accept either a response object, a list of response objects, or an XML document that represents either. If this field is unchecked, the step outputs a list of response objects, even if the list contains only one response.

14.5 Virtual HTTP_S Live Invocation

14.5 Virtual HTTP_S Live Invocation

This step is used to make a real HTTP call to real server within the context of a virtualized HTTP service. It is typically created by recording and virtualizing some form of HTTP traffic. It will perform the real request based on the current VSE request in play.



HTTP/S Protocol Live Invocation

Target server:

Target port:

☒ Format step response as XML

If environment error:

Target Server: Name of the server to which request is made.

Target port: Name of the port on which request will be made.

Format step response as XML: This field if checked, formats response of the step as XML.

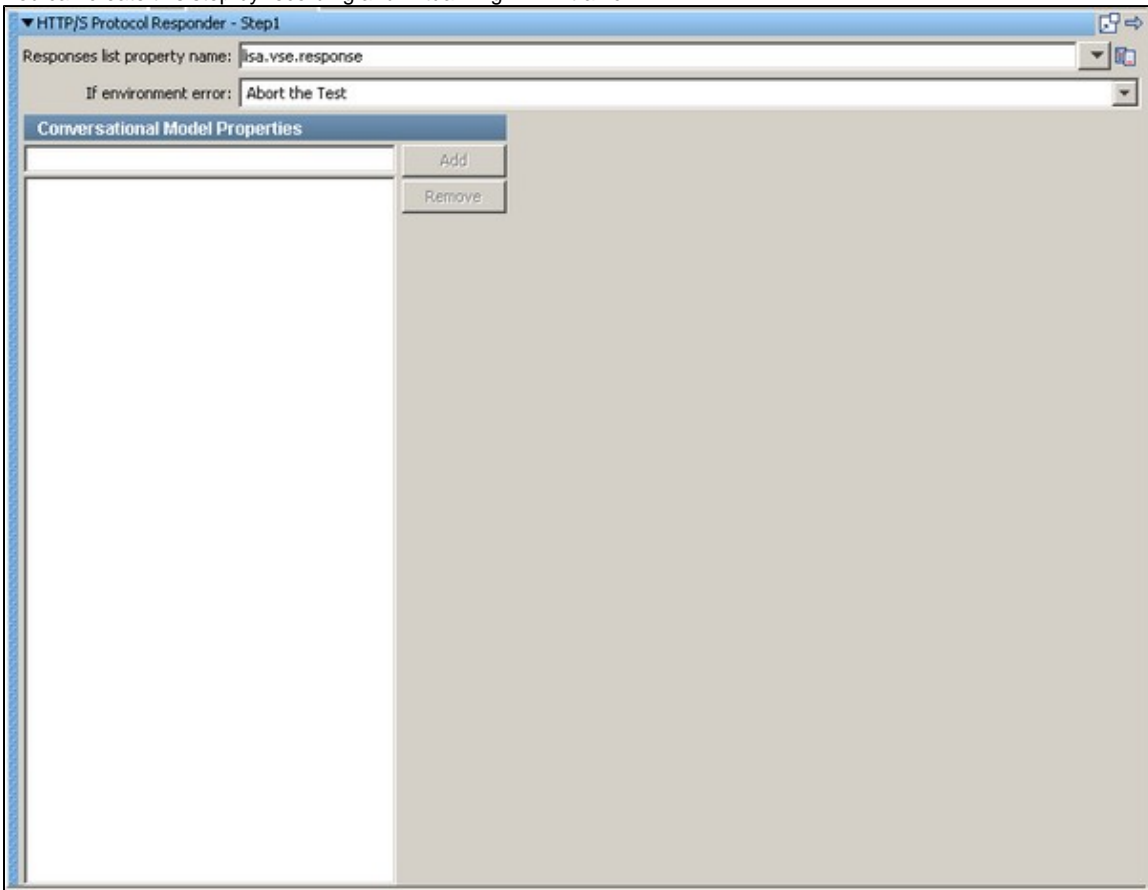
If environment error: This field indicates action to be taken in case of environment error.

14.6 Virtual HTTP_S Responder

14.6 Virtual HTTP_S Responder

This step is used in conjunction with the **Virtual HTTP/S Listener** step to transmit responses to HTTP requests produced by the listener. It takes a virtual response and uses it as the reply to the corresponding request using the HTTP/S protocol.

You can create this step by recording and virtualizing HTTP traffic.



HTTP/S Protocol Responder - Step1

Responses list property name:

If environment error:

Conversational Model Properties

Enter the data for the fields as described:

Field	Description
Responses list property name	The name of the property to look in for the response to send.
If environment error	Select the step to go to if an exception occurs. The default is blank.

Conversational Model Properties	Enter a property and click Add . To delete a property, select it from the list and click Remove . The properties listed here are associated with the current conversation session, which allows their values to be available to downstream conversational requests.
---------------------------------	---

14.7 Virtual JDBC Listener

14.7 Virtual JDBC Listener

This step is used to control the simulation of JDBC database traffic. It manages the communication with the simulation driver that is embedded in the database client.

Enter the data for the fields as described:

Field	Description
JDBC simulation driver host	Enter the host name or IP address of the database client where the simulation driver is running.
JDBC simulation driver port	Enter the port number for the JDBC simulation driver. The default is 2999 .
Format step response as XML	By default the step response is formatted as XML. The VSE framework expects Respond steps to accept either a response object, a list of response objects, or an XML document that represents either. If this field is unchecked, the step outputs a list of response objects, even if the list contains only one response.
If environment error	Select the step to go to if an exception occurs. The default is fail .
Connect/Disconnect	Click Connect to connect to the JDBC simulator. If connected, click Disconnect to end the connection. You can use this feature to validate the connection information.
Installed and Initialized JDBC Drivers	Displays the JDBC drivers installed and initialized in the database client.
Current SQL Activity	Displays the current SQL activity within the database client.

14.8 Virtual JDBC Responder

14.8 Virtual JDBC Responder

This step is used to send the result of a JDBC data call as selected by a conversational response selection step to the simulation driver that is embedded in the database client.

▼ JDBC Protocol Responder - Step1

Responses list property name:

If environment error:

Conversational Model Properties

<input type="text"/>	<input type="button" value="Add"/>
	<input type="button" value="Remove"/>

Enter the data for the fields as described:

Field	Description
Responses list property name	The name of the property to look in for the response to send.
If environment error	Select the step to go to if an exception occurs. The default is blank.
Conversational Model Properties	Enter a property and click Add . To delete a property, select it from the list and click Remove. The properties listed here are associated with the current conversation session, which allows their values to be available to downstream conversational requests.

14.9 Socket Server Emulator

14.9 Socket Server Emulator

This step can be used to simulate any text-based (typically HTTP) server socket. It supports listening, responding, and binding.

Virtual Socket Service - Step1

Virtual HTTP/Socket Setup Info

Process mode: Full Process

Listen port: 8080 Bind address: localhost ☐ Close immediately

☐ Use SSL SSL keystore file: Select...
Keystore password: Verify...

Base path: /

If environment error: Abort the Test Record terminator: ☐ ☒ Ensure proper HTTP response format

Listener status: Not running

Response to Send

```

<html>
  <head>Test</head>
  <body>
    <h1>Test</h1>
  </body>
</html>

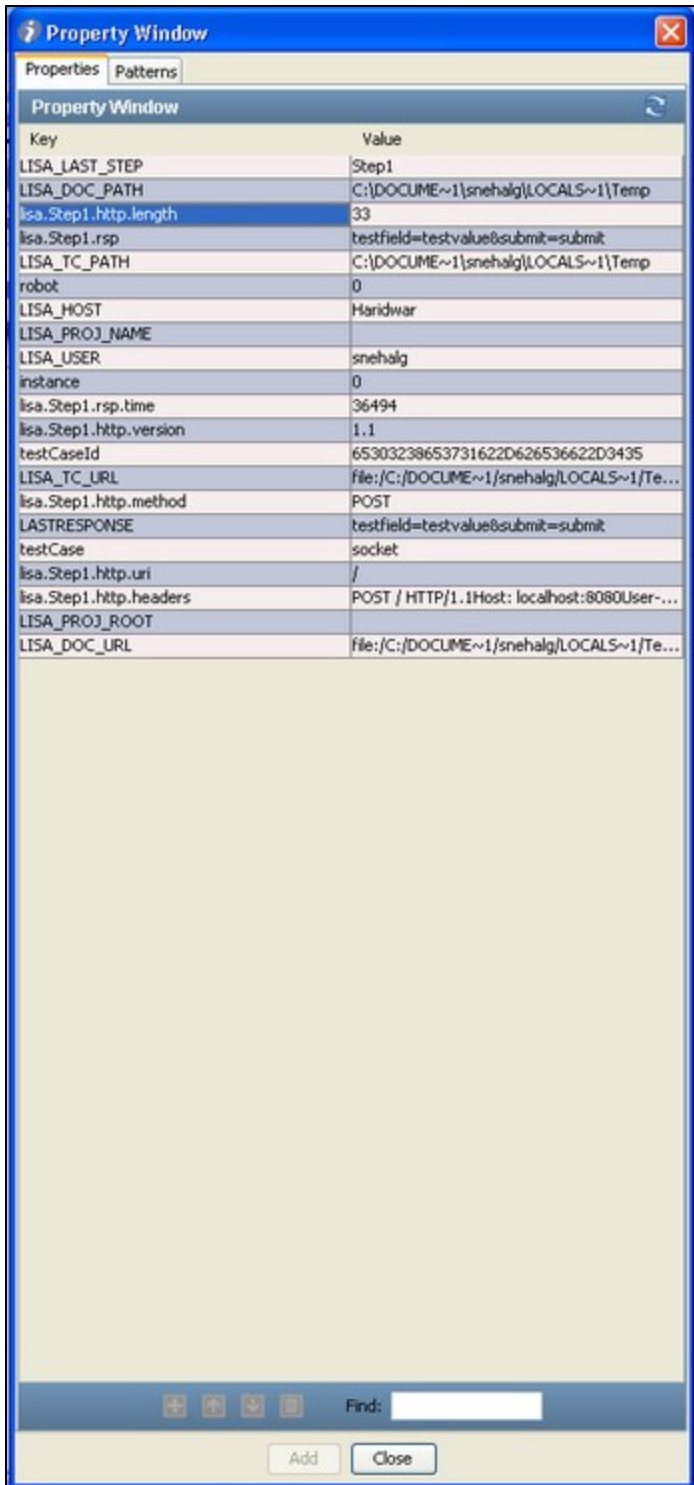
```

Request Response

Enter the data for the fields as described:

Field	Description
Process mode	From the list, select the process mode. The valid options are: Full Process, Asynchronous setup, Listen Only & Respond Only
Listen port	Enter the port on which LISA listens for the HTTP or Socket traffic.
Bind address	Enter the local IP address on which connections can come in. By default with no bind address specified, the listen step accepts connections on the specified port regardless of the NIC (or IP address) it comes in on.
Close immediately	Check to use design-time testing of this step. This option instructs the step to do the configured work and then immediately cleanup its network resources.
Use SSL	Check this box if you want to simulate a secure HTTPS Web site. Then supply the SSL keystore information.

SSL keystore file	Click Select to browse to your SSL keystore file. The same keystore file must be available to the VSE server to which the VS model is deployed.
Keystore password	Enter the keystore password, and click Verify .
Base Path	Identify the HTTP requested resource URIs that the listen step is to process. When the request comes in, the list of queue names is scanned for a name (base path) that starts the URI on the request. The queue name that matches is the one into which the request is placed and the listen step that is associated with the queue (by base path) processes the request.
If environment error	Select the step to go to if an exception occurs. The default is blank.
Record terminator	If the socket emulator is to simulate a record-based service, enter the character that marks the end of a record. If you leave this field blank, either line-oriented records or the HTTP protocol are simulated.
Ensure proper HTTP response format	By default, this option is checked. When in a process mode that sends a response and the response is to be a valid HTTP response, this option guarantees that the HTTP headers in the response text are correctly formatted and, if needed, that the Content-Length : HTTP response header is present and correct.
Listener status	Indicates whether the listener is running or not.
Test	Click to test the listener setup.
Clear Listener	Click to stop the test of the step.
Response tab	The Response to Send includes the text for the response.
Read Response From File	Click to browse the file system for a response.
Request tab	Last/Original Request is used only during design time. It displays the last request the step received.



lisa.step1.http.length property is seen getting added.

14.10 Messaging Virtualization Marker

14.10 Messaging Virtualization Marker

This step is used to designate that a message-based test case is designed for use in the Virtual Service Environment. If the test case is listening or responding through JMS, this step should be added to the VS model to ensure it can be deployed to the VSE.

14.11 Socket Server Emulator

14.11 Socket Server Emulator

The Socket Server Emulator allows to emulate a socket to act as a listener, a responder, both a listener and a responder, or a direct echo. Socket server emulator steps are the basis of the virtualized web service produced by the Virtual Web Service HTTP Recorder.

The Socket Server Emulator editor is shown below:

Virtual Socket Service - Step1

Virtual HTTP/Socket Setup Info

Process mode: Full Process

Listen port: 8080 Bind address: Close immediately

Use SSL SSL keystore file: Select...

Keystore password: Verify...

Base path: /

If environment error: Abort the Test Record terminator: Ensure proper HTTP response format

Listener status: Not running Test Clear Listener

Response to Send

Read Response From File...

Request Response

Base Tab

Enter the following information:

- Process Mode: Here one of the following options is chosen from drop down list.
 - 1) Asynchronous Setup
 - 2) Respond only
 - 3) Listen only
 - 4) Full Process.
- Listen Port: The port number the socket is listening on.
- Bind Address: The address to which it is bound.
- Close immediately
- Use SSL: Click here is SSL (Secured Socket Layer) is to be used. For SSL use, information of SSL keystore file (name with location which stores SSL key) & SSL keystore password need to be provided.

If environment error : From the list, select the step to go to if the test fails.

Click the **Clear Listener** button to clear the listener.

Click the **Test** button to test the emulator.

The Current Request window shows the last request.

Response Tab

Shows the response that will be sent if the Process Mode setting expects a response to be sent.

14.12 Compare Strings for Response Lookup

14.11 Compare Strings for Response Lookup

This step is used to look at an incoming request to a virtual service and determine the appropriate response in a completely stateless fashion, without referring to any service image. The stateful portions of VSE are not supported. You can match incoming requests using partial text match, regular expression, among others.

Text to match:

Range to match: Start: End:

If no match found:

If environment error:

☐ Store responses in a compressed form in the test case file.

Case Response Entries					
Enabled	Name	Delay Spec	Criteria	Compare Type	Response
<input checked="" type="checkbox"/>	str-compare	0	Simpson	Find in string	Simpson is found

Find:

Criteria

Response

The step components are:

Field	Description
Text to match	Enter the text against which criteria should be matched. This is typically a property reference, such as LASTRESPONSE .
Range to match	Enter the Start and End of the range.
If no match found	From the list, select the step to go to if no match is found.
If environment error	From the list, select the step to go to if the test fails.
Store responses in a compressed form...	Selected by default. This option compresses the responses in the test case file.
Case Response Entries	Add, move, and delete entries.
Enabled	Selected by default when you add an entry. Deselect to ignore an entry.
Name	Enter a unique name for the case response entry.
Delay Spec	Enter the delay specification range. The default is 1000-10000 , which indicates to use a randomly selected delay time between 1000 and 10000 milliseconds. (The syntax is the same format as Think Time specifications.)
Criteria	This area provides the string to compare against the Text to match field. To edit the criteria, in the Case Response Entries area select the appropriate row, and then select a different setting from the Criteria list.

Compare Type	Select an option from the list: <ul style="list-style-type: none"> Find in string (default) Regular expression Starts with Ends with Exactly equals
Response	This area provides the response of this step if the entry matches the Text to match field. To edit the response, in the Case Response Entries area select the appropriate row, and then select a different setting from the Response list.
Criteria	Allows for the update of the criteria string for an entry.
Response	Allows for the update of the step response for an entry."

14.13 Compare Strings for Next Step Lookup

14.11 Compare Strings for Next Step Lookup

This step is used to look at an incoming request and determine the appropriate next step. You can match incoming requests using partial text match, regular expression, among others.

Each matching criterion specifies the name of the step to which to transfer if the match succeeds.

Text to match:

Range to match: Start:

End:

If no match found:

If environment error:

Enabled	Name	Delay Spec	Criteria	Compare Type	Next Step
<input checked="" type="checkbox"/>	str-compare	0	Simpson	Find in string	FTP Step

+

↑

↓

✕

Find:

Criteria

Simpson

The step components are:

Field	Description
Text to match	Enter the text against which criteria should be matched. This is typically a property reference, such as LASTRESPONSE .
Range to match	Enter the Start and End of the range.
If no match found	From the list, select the step to go to if no match is found.
If environment error	From the list, select the step to go to if the test fails.
Next Step Entries	Add, move, and delete entries.
Enabled	Selected by default when you add an entry. Deselect to ignore an entry.

Name	Enter a unique name for the next step entry.
Delay Spec	Enter the delay specification range. The default is 1000-10000 , which indicates to use a randomly selected delay time between 1000 and 10000 milliseconds. (The syntax is the same format as Think Time specifications.)
Criteria	This area provides the string to compare against the Text to match field. To edit the criteria, in the Next Step Entries area select the appropriate row, and then select a different setting from the Criteria list.
Compare Type	Select an option from the list: <ul style="list-style-type: none"> • Find in string (default) • Regular expression • Starts with • Ends with • Exactly equals
Next Step	From the list, select the step to go to if the match is found.
Criteria	Allows for the update of the criteria string for an entry.

15. Custom Extension Steps

15. Custom Extension Steps

The following Steps are available in this chapter.

[15.1 Custom Test Step Execution](#)
[15.2 Java Script Test](#)
[15.3 Swing Test Step](#)
[15.4 Java Protocol Request Listener](#)
[15.5 JavaProtocol.Responder](#)
[15.6 Java Pass Through](#)
[15.7 JMS Pass through](#)
[15.8 MQ Pass through](#)

15.1 Custom Test Step Execution

15.1 Custom Test Step Execution

The Custom Test Step executes a test step custom written by your team using the LISA Extension Kit. This type of step is documented in the [LISA Developer's Guide \(LEK\)](#).

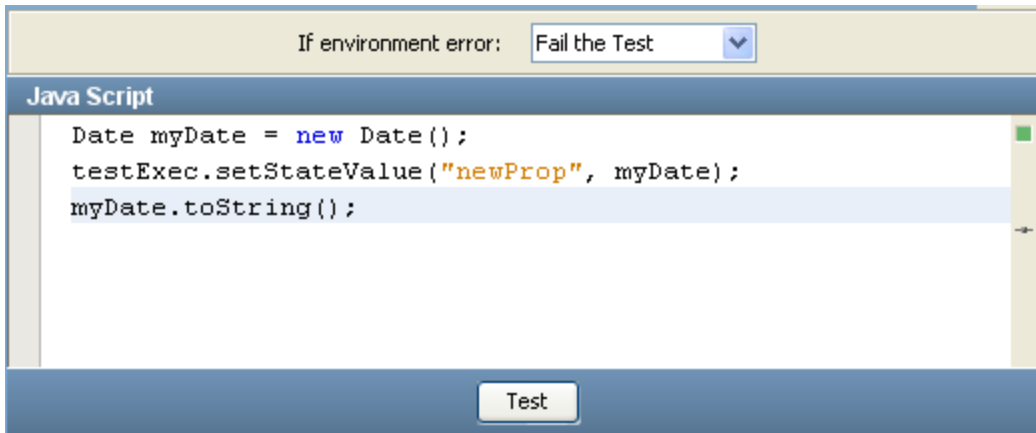
15.2 Java Script Test

15.2 Java Script Test

The Java Script step affords you the flexibility of writing and executing a Java script to perform some function or procedure. Your script is executed using the "BeanShell" interpreter. You have access to all the LISA properties in the test case, including built in objects.

Prerequisites - Some knowledge of BeanShell. For more information on BeanShell, see LISA for Java Developers or www.beanshell.org.

The script editor is shown below:



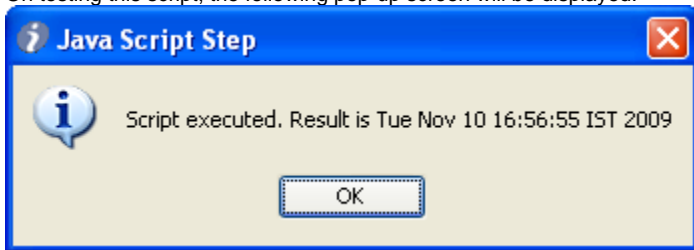
Here is the script editor where you write your scripts. Double clicking on an item in the list will paste that variable name into the editor. The last value exposed in the script will be saved as the response of this step.

Click the **Test** button to test your script. You will see the result from executing the script, or an error message describing the error that occurred.

The short example in the figure above shows that:

- A new Date object was created, initialized to the current date and time.
- This object was stored in a new LISA property, myprop, using one of LISA's exposed objects, testExec (see the [LISA LEK_Developer's Guide](#) for more details).
- The toString() value of the Date object was set as the response of the step.

On testing this script, the following pop-up screen will be displayed:



LISA property name syntax is very flexible and can include spaces. Property names that are not valid java identifiers are converted for use in this step. Invalid characters are automatically replaced by an underscore (_).

15.3 Swing Test Step

15.3 Swing Test Step

The Swing step allows LISA to **listen to mouse** and **keystrokes** on a Java Swing application, record them and play them back.

To do this LISA attaches to the AWT listener of the recorded application, the main class and .jar files for the application are needed. The Swing recorder will remotely launch the application and listen to the AWT events and then send the events back to LISA for recording. After recording is completed the **AWT events** are played back to the application just as if the keyboard and mouse were being used.

A **component browser** is provided to look at the Java objects in the application and find information on the running object. From LISA every property and method provided by the object can be run, Filtered and then asserted on. Assertions are created by attaching to the Swing Java object and running methods against it, for example, IF a radio button **isEnabled** or **isChecked** can be validated.

A Swing Test step can be added by clicking **Add Steps > Custom Extensions > Swing Test Step**.

The **Swing Step Editor** opens as follows:

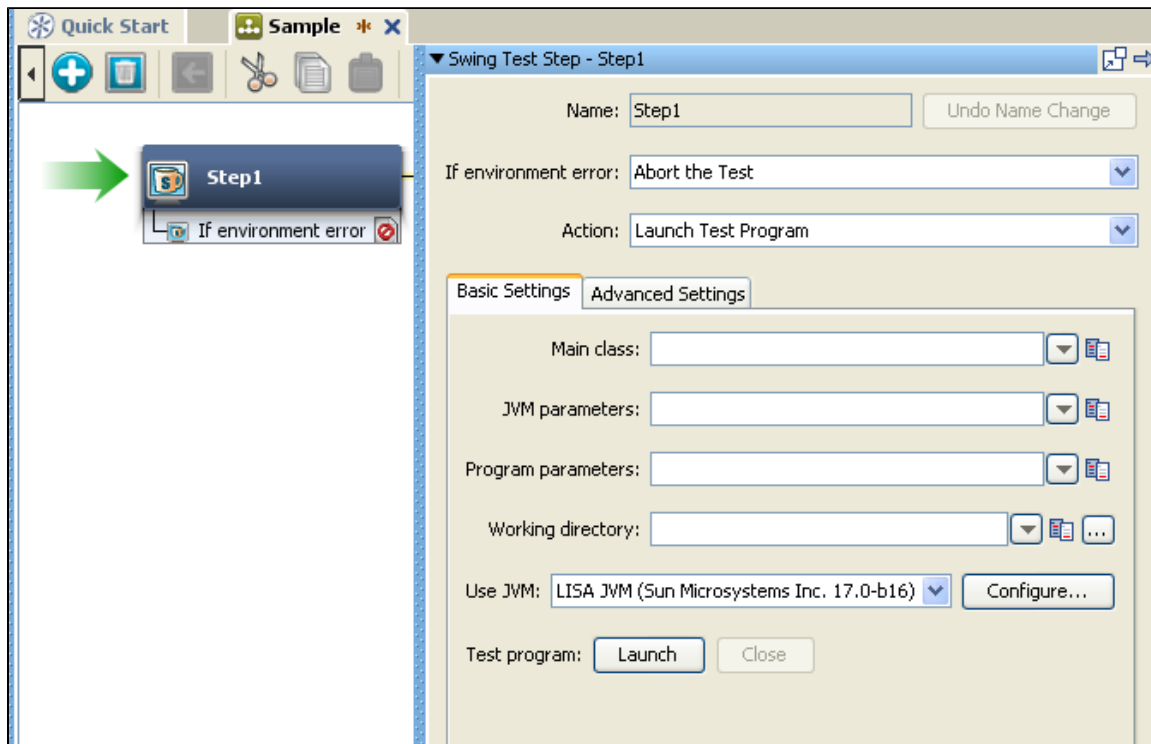
The top panel:

- **Name** - Enter the Name of the test step.
- **If Environment Error** - Select the necessary step in case of an error, from drop down option.
- **Action** - Select the necessary step to be taken as Action, from drop down option.

It has two tabs: **Basic Settings** and **Advanced Settings**.

Basic Settings Tab

By default it opens in the Basic settings tab.



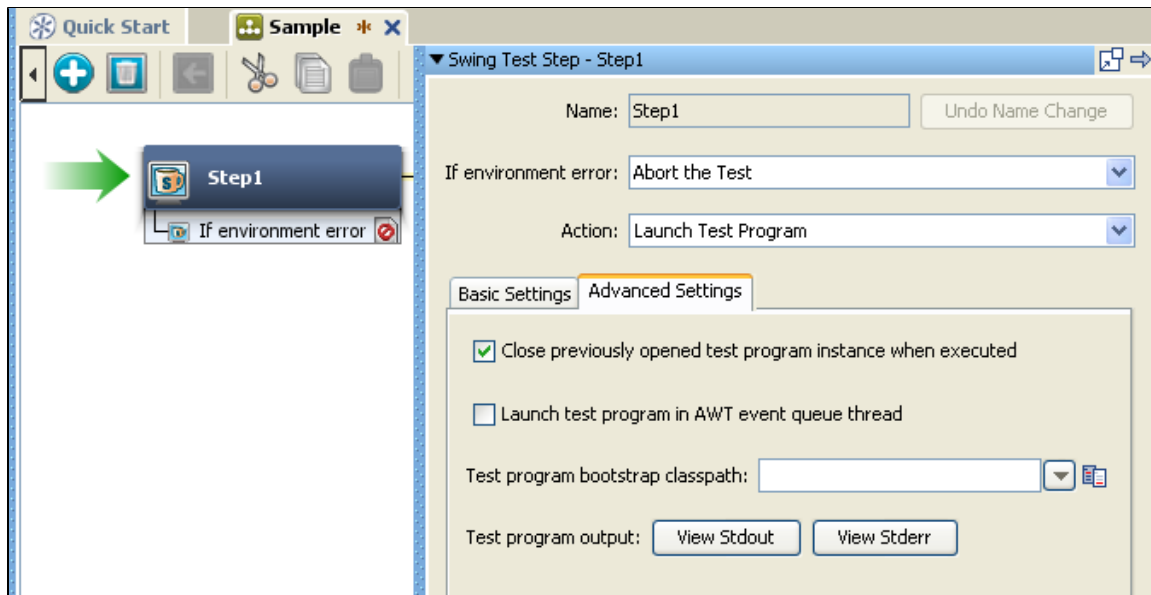
Mouse Input is created by LISA when recording mouse movements from the AWT listener.

Key Input is created by LISA when keys are pressed and recorded from the AWT listener.

Invoke Component Methods are used to attach to components (buttons, text, frames, etc.) and run methods on them.

- **Main Class:** Contains the class name with the Java main method.
- **JVM parameters:** Any parameter that should be sent to the JVM is entered here. The --classpath parameter contains all of the .jar files to run the applications. If the application jars are in the hot deploy directory this does not need to be set.
- **Program parameters:** Set any arguments that the application is expecting.
- **Working directory:** Points to the directory in which the application to be tested is installed. When the application is started it is started from this directory.
- **Use JVM:** This will default to the JVM LISA is running, an alternate JVM can be specified if the application requires a different JVM.
- **Test program:** Click the **Launch** button to open the application; the **Close** button will close the application.

Advanced Settings Tab



- **Close previously opened test program instances when executed** : This tells LISA to close any open applications and open a new one when playing back the steps recorded from the application. This makes sure the application always starts at the beginning of where the test case was recorded and prevents duplicate applications from being run.
- **Launch test program in AWT event queue thread**: This tells LISA where the AWT listener is to it can be attached to. If the Java application does not record in LISA, uncheck this box and LISA will try attaching to the AWT listener in the main of the program.
- **Test program bootstrap classpath**: This can be set for the application being tested. The bootstrap should only be used if the application is getting errors when loading classes. The classes will need to be moved from the classpath (-classpath "myjar.jar" or LISA hot deploy) to the bootstrap.
- **Test program output**: This allows for viewing of the Java application messages that are sent to Stdout (standard out) and Stderr (standard error) from the launched application.

Mouse Input

A Mouse Input step is created when recording a Swing application.

The information about which button, any extra keys complementing the click and the location is recorded. The component that was clicked on is listed in the Component window.

Pressing the **Select...** button will navigate the component browser to the object that was clicked. When a mouse click is played back, the component is found and the mouse is clicked in the middle of the component. This allows LISA to be tolerant to changes in a Swing User Interface.

Key Input

Key Input is recorded by LISA when a keystroke is done on the keyboard.

The recording can be a action like **ALT + TAB** to change between windows of the application or the typing of letters into a field. When a single letter is typed, the Key pressed, Key typed, and Key released steps are recorded to correspond to all actions. The Component that the key action was taken on is listed and can be viewed in the component browser by pressing the Select... button.

Invoke Component Methods

The Invoke Component Methods action is added by the tester when a component is available in the running application and methods of the component are to be run. For example, if there is a frame in the application and the title of the frame is wanted. To do this, Invoke Component Methods action is inserted into the workflow, the component is selected and the method of **getTitle** is available.

Component browser shows the active components of a Java application. The application must be launched from the Launch program step so that the component browser can attach to it. If the Component Browser does not navigate to the component, verify there is not a "*" asterisk next to the title Component Tree indicating a refresh of the tree is required. The other reason a component will not show up could be that the application has not been launch and the component is not available in the application, for example, it could be on a folder tab that is not visible.

When the select button is pressed, the component is sent back to the LISA step with information about the object. The Data Sheet contains information about the object, The call sheet will allow you to run methods on the object like **getTitle()**.

Assertions can now be done on the results of the method calls just like any other object in LISA.

15.4 Java Protocol Request Listener

15.4 Java Protocol Request Listener

This step is responsible for setting up communication with the LISA Agent (Pathfinder) to support Java virtualization. When this step is initialized, it creates a companion. The companion takes a list of agent names and classes from the step, and tells the named agents to begin virtualizing the named classes. The companion is also responsible for monitoring the agents, and re-initializing virtualization on target agents when they go offline and come back online.

The companion sets up a blocking queue to handle intercepted method calls from the agent. It registers a callback with LISA Agent (Pathfinder). The callback creates a blocking response queue, and puts that queue along with the VSEFrame into the request queue. It then waits for a maximum of 30 seconds for a response to appear in the response queue. If no response appears in 30 seconds, then it tells the agent to let the SUT process the method.

The Java listen step monitors the incoming request queue with a 10 second timeout; if no request is found within 10 seconds, the step will set it's next node to the current step and complete, causing it to be re-executed. Multiple Java listen steps can monitor this queue simultaneously, and the first step that gets the request will process it.

15.5 JavaProtocol.Responder

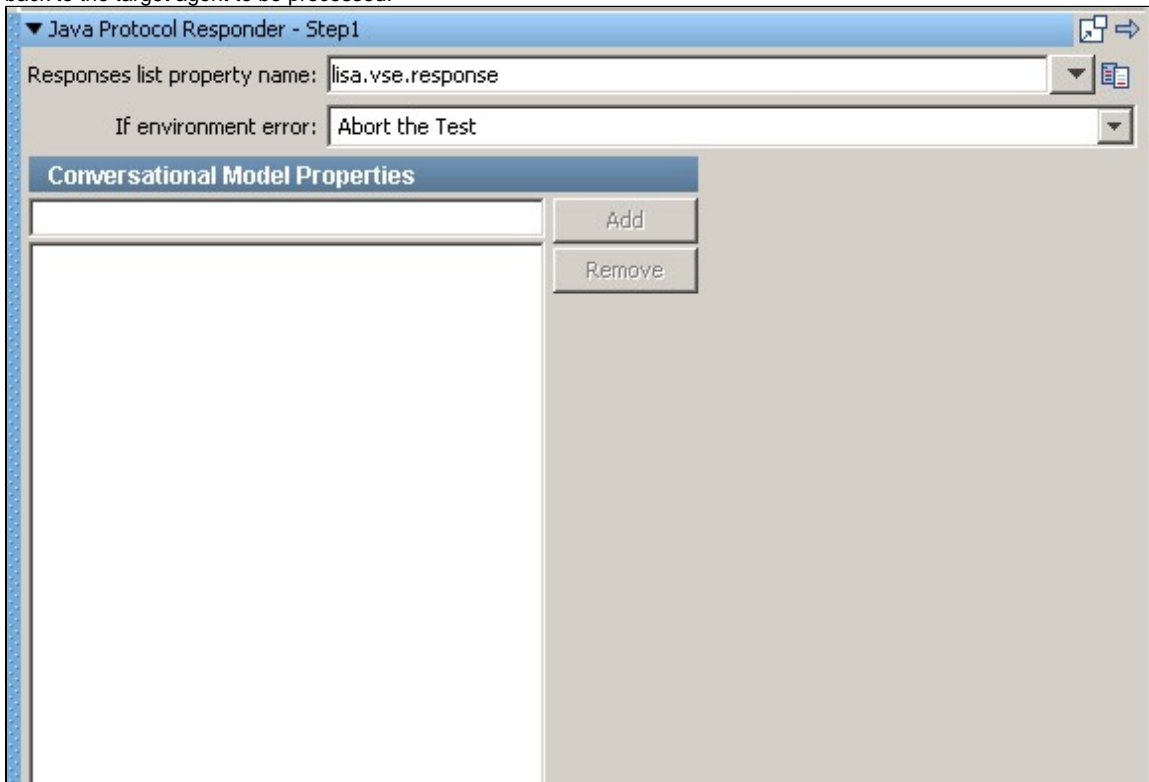
15.5 JavaProtocol.Responder

The respond step retrieves the VSE Response object from the test exec and runs it through BeanShell evaluation.

If the body of the response starts with a "<" symbol, then the respond step considers this to be XML. It retrieves the original VSE Frame from the test exec object, and calls setResultXML() passing in the body of the response.

- If the body of the response starts with any character other than a "<" symbol, the respond step considers it to be code. It calls setCode() on the frame, passing in the body of the response, and sets useCode() to true. Note that "return com.itko.lisa.remote.instrument.Rules.NO_HIJACK;" is the default unknown response for the Java protocol.
- If the body of the response is empty, then the frame is not updated.

The response body is logged in vse_matches.log. Then the frame is put into the response queue, where the callback retrieves it and sends it back to the target agent to be processed.



15.6 Java Pass Through

15.6 Java Pass Through

This step sets "return com.itko.lisa.remote.instrument.Rules.NO_HIJACK;" on the frame and puts the frame in the response queue, telling the target agent to allow the SUT to attempt to process the call.

15.7 JMS Pass through

15.7 JMS Pass through

A custom step described as "JMS Pass Through" of class `com.ITKO.lisa.vse.stateful.protocol.messaging.jms.JmsPassThroughStep`.

The screenshot shows a configuration window titled "JMS Pass Through - Step1". It contains the following fields and controls:

- Timeout:** An empty text input field.
- Request property name:** A dropdown menu with "lisa.vse.request" selected.
- Response property name:** A dropdown menu with "lisa.vse.response" selected.
- Format step response as XML:** A checked checkbox.
- If environment error:** A dropdown menu with "Abort the Test" selected.

15.8 MQ Pass through

15.8 MQ Pass through

A custom step described as "MQ Pass Through" of class `com.ITKO.lisa.vse.stateful.protocol.messaging.mq.MQPassThroughStep`.

This screenshot is identical to the one above, showing the "JMS Pass Through - Step1" configuration window with the same settings: Timeout (empty), Request property name (lisa.vse.request), Response property name (lisa.vse.response), Format step response as XML (checked), and If environment error (Abort the Test).

16. Standard Test Steps

16. Standard Test Steps

The following are some standard test steps:

- 16.1 Abort the Test
- 16.2 End the Test
- 16.3 Fail the Test

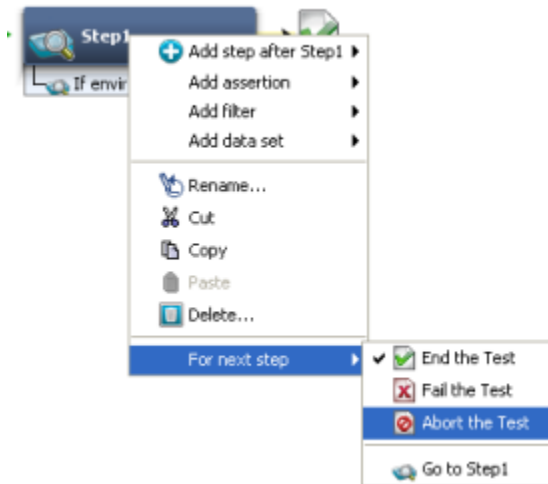
16.1 Abort the Test


16.1 Abort the Test

To Abort the test step,

- Right click the Test Step after which you want to **Abort** the test case.

- Click **For Next Step > Abort the Test**, as shown below:



Abort the Test,  step will quit the test case and mark the step as having aborted as shown below:

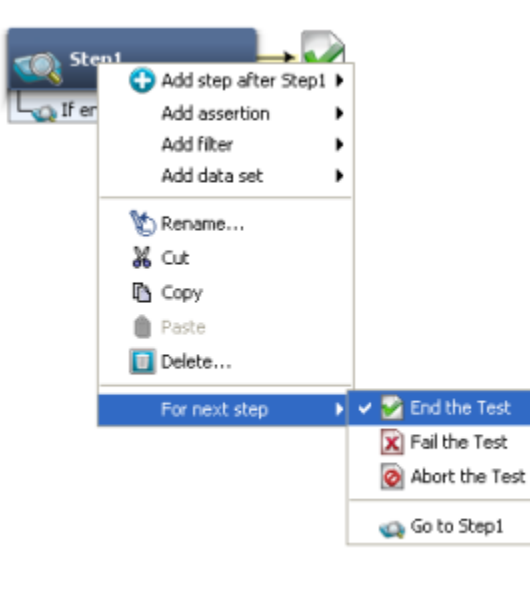



16.2 End the Test

16.2 End the Test

To End the test step,

- Right click the Test Step after which you want to **End** the test case.
- Click **For Next Step > End the Test**, as shown below:



End the Test,  step will complete the test and mark the test as having ended successfully as shown below:

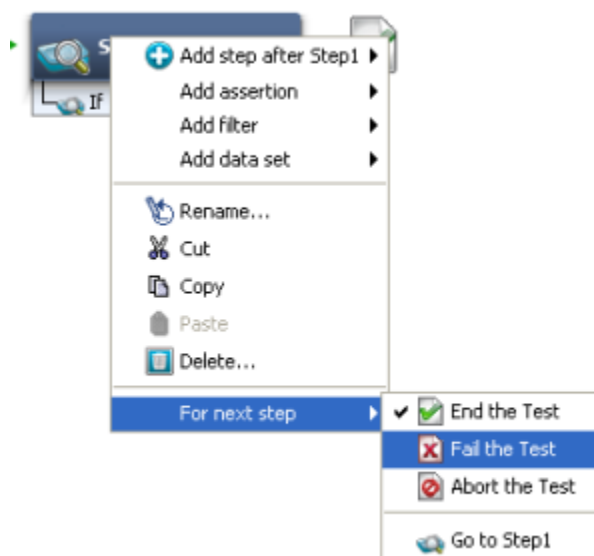



16.3 Fail the Test

16.3 Fail the Test

To Fail the test step,

- Right click the Test Step after which you want to **Fail** the test case.
- Click **For Next Step > Fail the Test**, as shown below:



Fail the Test,  step will fail the test case and mark the test as having failed as shown below:



PART 3 - Filters

PART 3 - Filters

This section describes each of the Filters that are available in LISA. In this Reference Guide we assume a general understanding of the LISA concepts and uses of filters in the test cases. For more information on Filters, see the [LISA User Guide - Adding Filters](#).

Regular expressions are used for comparison purposes in several filters. For more information on regular expressions, visit <http://java.sun.com/docs/books/tutorial/essential/regex/>.

The following Filters are available -

17. Utility Filters

- 17.1 Create Property Based on Surrounding Values
- 17.2 Store Step Response
- 17.3 Override "Last Response" Property
- 17.4 Save Property Value to File
- 17.5 Parse Property Value as Argument String
- 17.6 Save Property from one key to another
- 17.7 Time Stamp Filter

18. Database Filters

- 18.1 Extract Value from JDBC Result Set
- 18.2 Simple Result Set Filter
- 18.3 Size of JDBC Result Set
- 18.4 Set Size of a Result Set to a Property
- 18.5 Get Value For Another Value in a ResultSet Row

19. Messaging_ESBFilters

- 19.1 Extract payload and Properties from Messages
- 19.2 Convert a MQ Message to a VSE Request
- 19.3 Convert a JMS Message to a VSE Request

20. HTTP_HTML Filters

- 20.1 Create Resultset from HTML Table Rows
- 20.2 Parse Web Page for Properties
- 20.3 Parse HTML_XML Result for Specific Tag_Attributes Values
- 20.4 Parse HTML Result for Specific Tag_Attribute's Value and Parse It
- 20.5 Parse HTML Result for Tag's Child Text
- 20.6 Parse HTML Result for HTTP Header Value
- 20.7 Parse HTML Result for Attribute's Value
- 20.8 Parse HTML Result for LISA Tags
- 20.9 Parse HTML Result and Select Random Attribute Value
- 20.10 Parse HTML into List of Attributes
- 20.11 Parse HTTP Header Cookies
- 20.12 Dynamic Form Filter
- 20.13 Parse HTML Result by Searching Tag_Attribute Values

21. XML Filters

- 21.1 Parse text from XML
- 21.2 Read Attribute from XML Tag
- 21.3 Parse XML Result for LISA Tag
- 21.4 Choose Random XML Attribute
- 21.5 XML XPath Filter

22. Web 2.0 Filters

- 22.1 Web 2.0 Element Filter
- 22.2 Web 2.0 Text Filter
- 22.3 Web 2.0 Attribute Filter
- 22.4 Web 2.0 Java Script Filter
- 22.5 Web 2.0 Function Filter
- 22.6 Web 2.0 Composite Filter

23. Java Filters

- 23.1 Override "Last Response" Property
- 23.2 Store Step Response
- 23.3 Save Property Value to File

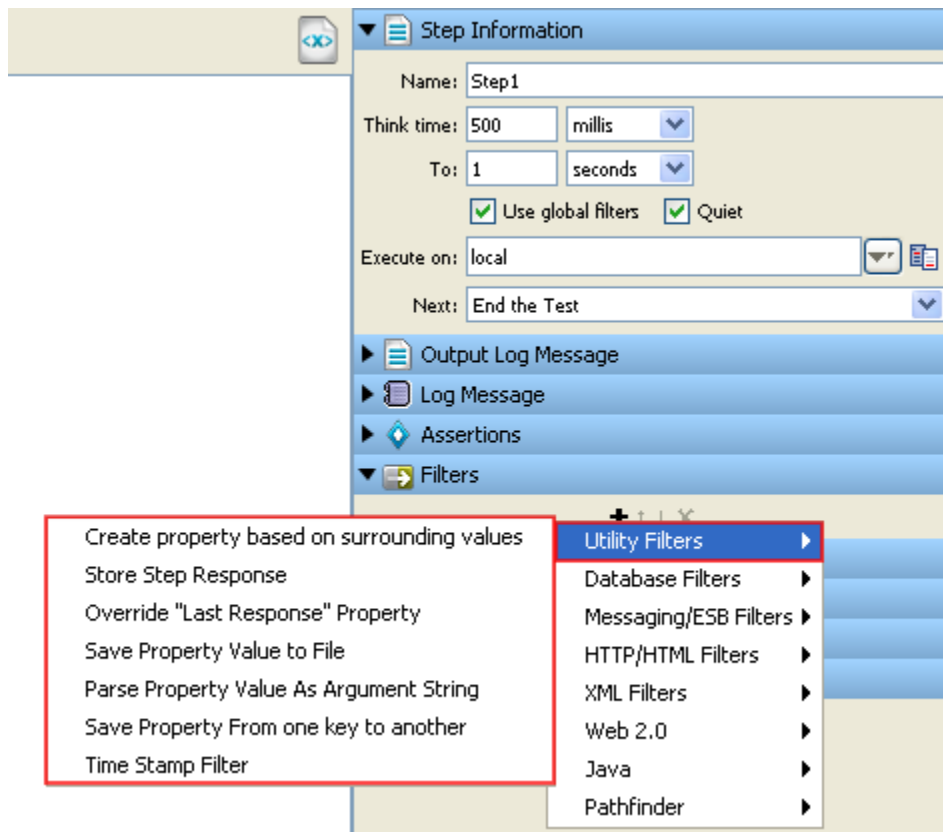
25. Pathfinder Filters

- 24.1 Global Filters
- 24.1 LISA Integration Support for Pathfinder
- 24.2 LISA Integration Support for webMethods Integration Server

17. Utility Filters

17. Utility Filters

These are the filters available in the Utility Filters list for any test step:



This list explains these Filters in the order they appear on screen...

- 17.1 Create Property Based on Surrounding Values
- 17.2 Store Step Response
- 17.3 Override "Last Response" Property
- 17.4 Save Property Value to File
- 17.5 Parse Property Value as Argument String
- 17.6 Save Property from one key to another
- 17.7 Time Stamp Filter

17.1 Create Property Based on Surrounding Values

17.1 Create Property Based on Surrounding Values

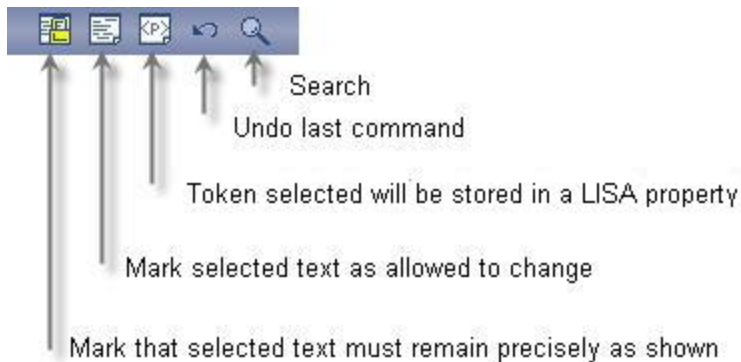
(Parse Text for Values)

The Parse Text for Values filter allows you to read textual content and filter it for information to be stored in LISA properties. It can be used on text, and XML and HTML treated as text. It uses a "Paint the Screen" technique.

Paint the Screen allows you great flexibility to define what in the text buffer you want to parse out as properties. There are three ways to mark the text:

1. Text that must appear in the response precisely as shown. A "Must" block.
2. Text that does not have to appear in the response, or can change. An "Any" block.
3. Text that will be stored in a LISA property. A "Property" block. Property blocks must always be bounded by Must blocks.

The text is marked using the icons at the bottom of the editor:



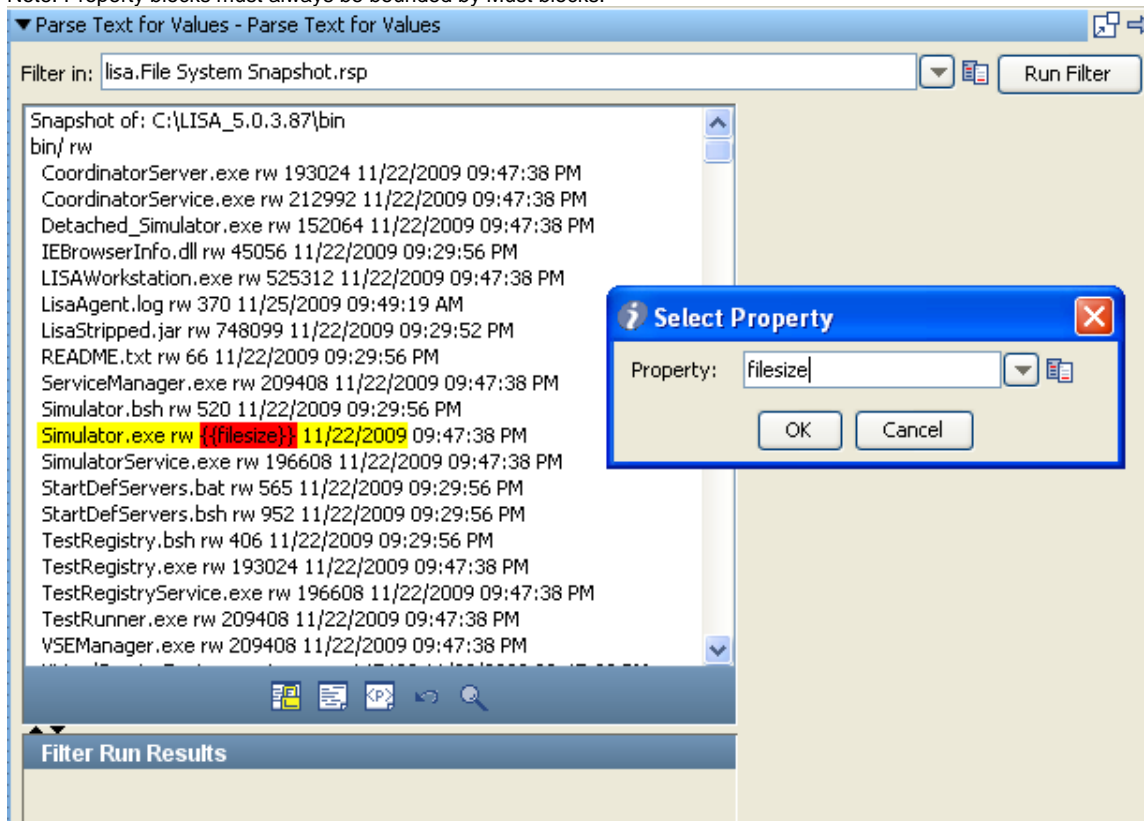
This technique is best explained by example.

In the following example, see the figure below, we want to store the size of a particular file in a property.

We have marked the text using the icons shown above, by selecting text and then clicking on the appropriate icon.

- Yellow background indicates text that must appear as shown (indicated by the arrows from "Text uses"). This is a "Must" block and is marked using the icon.
- Red background identifies the text that will be stored in the property entered into the dialog box (indicated by a single arrow). This is a "Property" block and is marked using the icon.

Note: Property blocks must always be bounded by Must blocks.

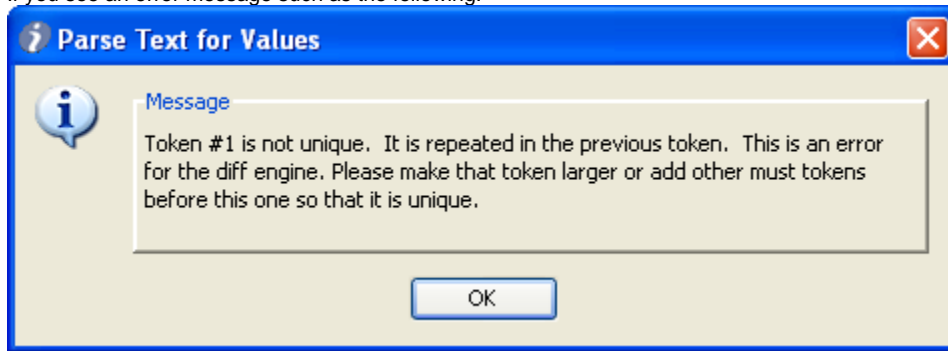


This screen shows the contents of the text buffer. We want to parse out the file size of the "Simulator.exe" file. The file size is the number that appears after 'Simulator.exe'. We have set the boundaries around the file size, and clicked the "Must" icon, . Then we selected the actual file size text inside the highlighted content, and clicked the "Property" icon, . We entered the property name into the dialog box. Note that the actual value of the file size has been replaced with the name of the LISA property.

When this filter is run, the property "filesize" will be assigned the size of the **Simulator.exe** file. You can repeat this process on this text buffer to have as many properties defined as you like.

Handling Non-unique Tokens

If you see an error message such as the following:



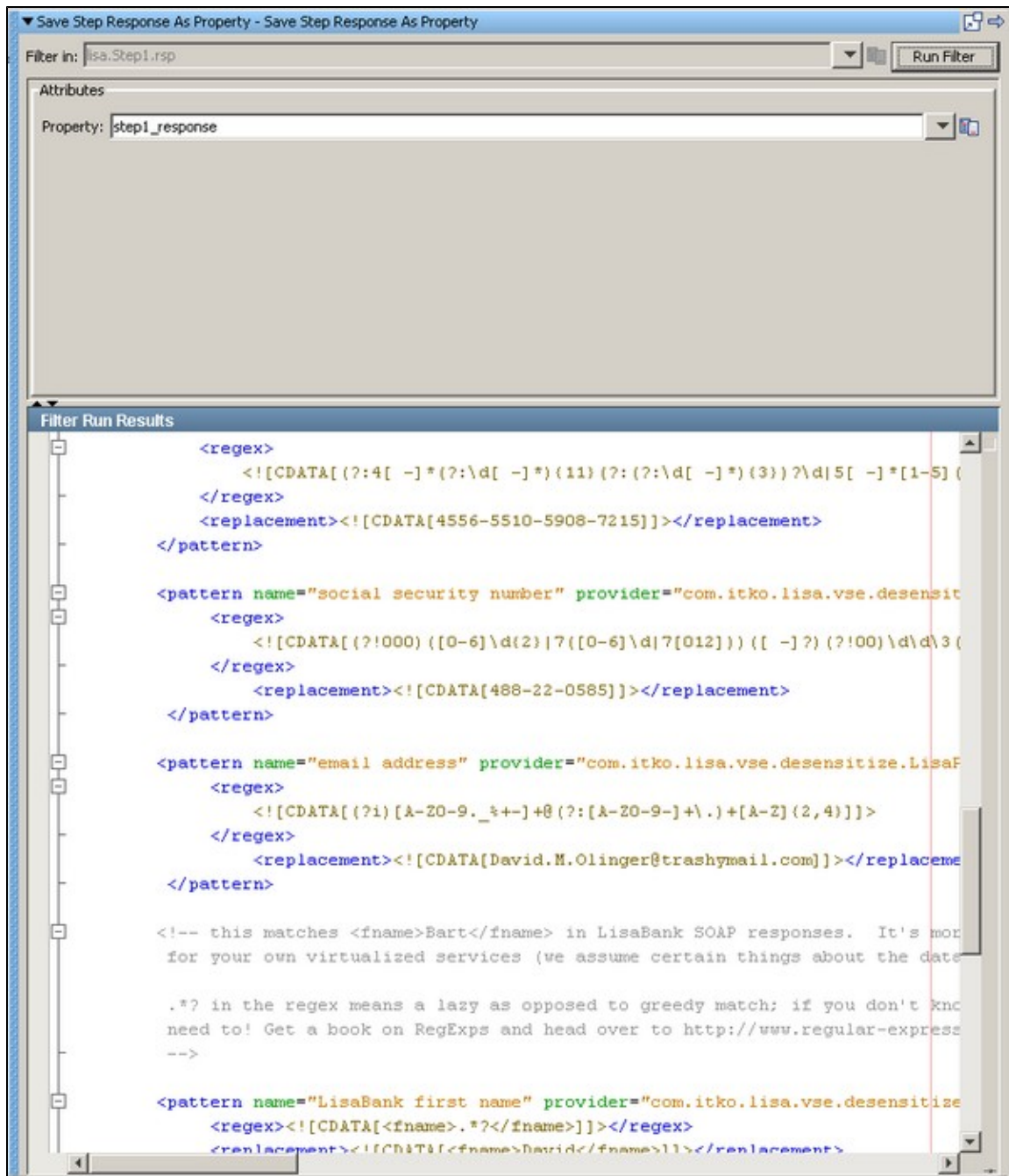
LISA is telling you that your selected token is not unique, the selection you just made is repeated in the token before it. To solve this in most cases, simply create another token to make the prior token a "Must" token as well. In other cases, when this doesn't work, a judicious placement of another "Must" block in between the two duplicate tokens should do the trick. This will work because LISA can distinguish between the two duplicate tokens based on their relative location.

17.2 Store Step Response

17.2 Store Step Response

(Save Step Response as Property)

The Save Step Response as Property filter allows you to save the last response as a property, for future use.



Enter the following parameter:

- **Filter in:** Where to apply the filter. The figure above shows list.Step1.rsp which means that the filter will be applied to the response of Step1. You cannot change this value for this filter.
- **Property:** The name of the property to store the last response.
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

17.3 Override "Last Response" Property

17.3 Override "Last Response" Property

(Convert Property Value into Last Response)

The Convert Property Value into Last Response filter allows you to replace the current value of the last response with the value of an existing LISA property. For example, assume you execute an EJB, but you eventually get a value back after making some method calls on the EJB. Instead of leaving the EJB object to be the last response, it may make more sense, in your test case, to make the result from one of your method

calls the last response. You can first save that call's return value as a property using a filter, and then you can use that property in this filter.

The screenshot shows a configuration window titled "Convert Property Value into Last Response - Convert Property Value into Last Response". It features a "Filter in:" text box containing "EJB2.rsp", a dropdown arrow, a document icon, and a "Run Filter" button. Below this is an "Attributes" section with a checkbox labeled "Convert to XML". At the bottom is a "Filter Run Results" section with a blue header bar.

Enter the following parameters

- **Filter in:** The name of the property you want considered as the step's last response. The property should be in the pull down menu, if not you can enter it. It must be an existing property.
- **Convert to XML:** Check this if you want the response to be converted to valid XML.
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

17.4 Save Property Value to File

17.4 Save Property Value to File

The Save Property Value to File filter allows you to save the value of an existing property to a file in your file system.

The screenshot shows a configuration window titled "Save Property Value to File - Save Property Value to File". It features a "Filter in:" text box containing "step1_response", a dropdown arrow, a document icon, and a "Run Filter" button. Below this is an "Attributes" section with a "Location:" text box containing "{LISA_HOME}examples/outProp.txt", a browse button "...", and a checkbox labeled "Append Mode". At the bottom is a "Filter Run Results" section with a blue header bar.

Enter the following parameters:

- **Filter in:** The name of the property whose value you want to write to the file.
- **Location:** The path name of the file to write the value to. You can browse to the file. You can use properties in the location.
- **Append Mode:** Check this checkbox if you want to append the information to an existing file.
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

17.5 Parse Property Value as Argument String

17.5 Parse Property Value as Argument String

The Parse Property Value As Argument String filter allows you to store the text of a specific attribute in a property. This is very useful as a second filter, where you parse a filtered value for information.

▼ Parse Property Value As Argument String - Parse Property Value As Argument String

Filter in:

Attributes

☐ IsURL

Attribute:

Property:

Default (if not found):

Filter Run Results

Enter the following properties:

- **Filter in:** The name of the existing property to parse. For example, if you want to parse the "lisa.delete user.cookies.rsp" property to return the value of the SESSIONID attribute, enter lisa.delete user.cookies.rsp.
- **IsURL:** Check this checkbox if the property value is a URL.
- **Attribute:** The attribute to retrieve. SESSIONID attribute, enter SESSIONID.
- **Property:** The name of the property to store the text of the attribute, here we used sessionID.
- **Default (if not found):** The default value to use if the attribute is not found.
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

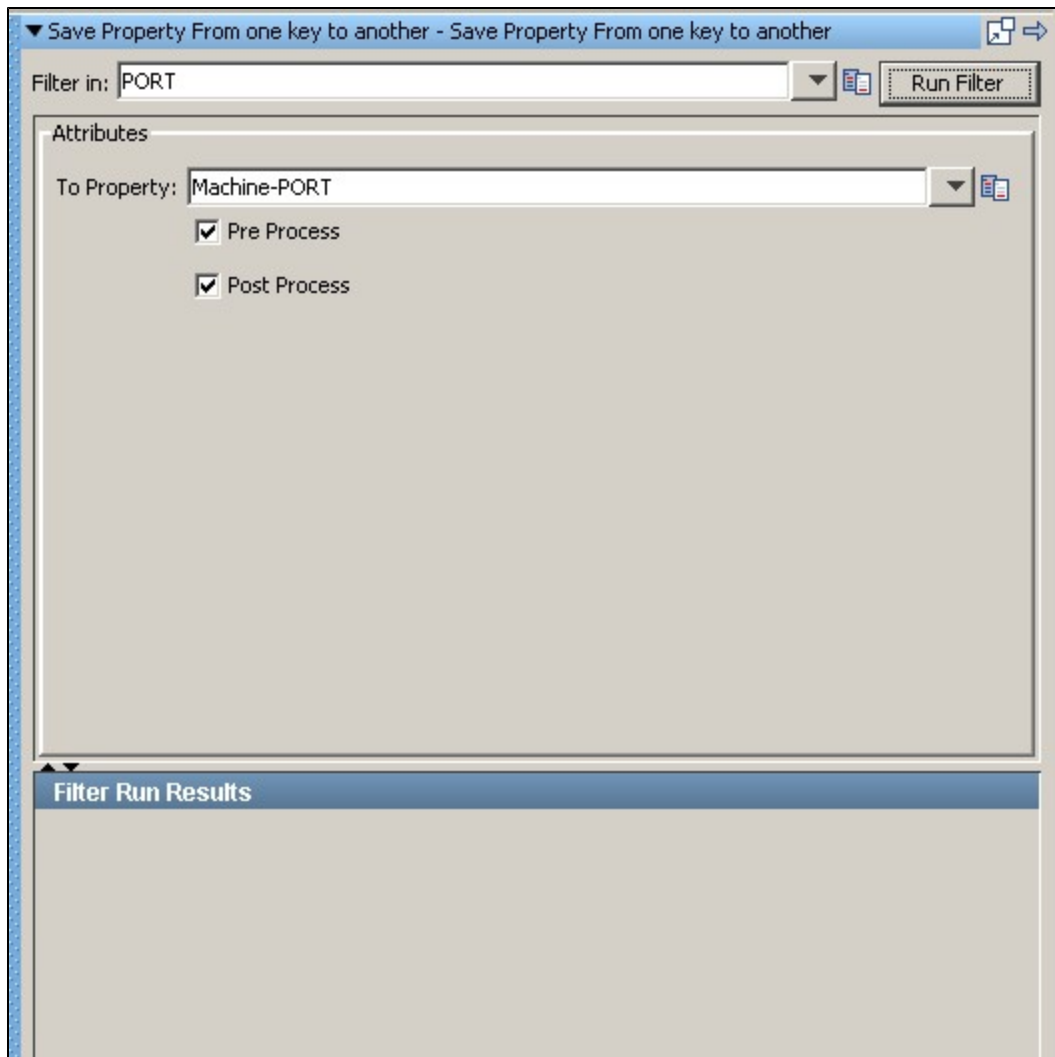
Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

17.6 Save Property from one key to another

17.6 Save Property from one key to another

This filter copies values from one key to another by reference where normal java rules apply.

This filter simply optimizes beanshell overhead for simple property copy.



Filter in: This field accepts the property content of which will be copied in some other property. So this is the input source property.

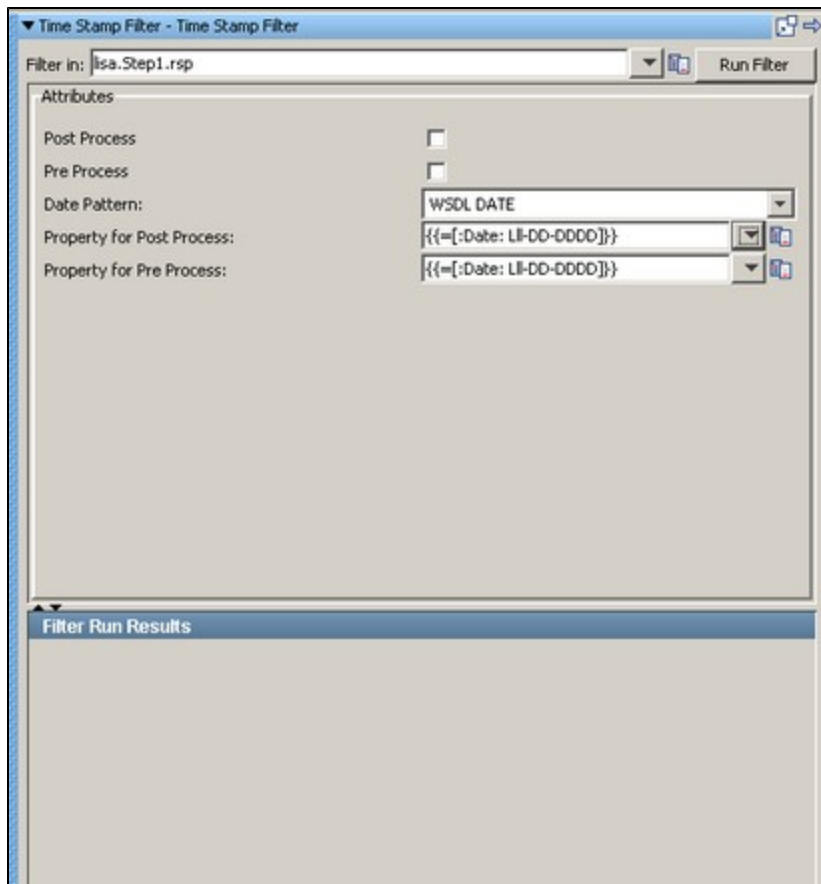
To Property: This is the property name where input property content will be copied.

Run Filter button allows to test filter immediately during developing test steps rather than waiting for writing whole test case and then checking it.

17.7 Time Stamp Filter

17.7 Time Stamp Filter

Timestamp filter is used to assign the current time and date to the property so that you may use this property in the following test steps.



Enter the following parameters:

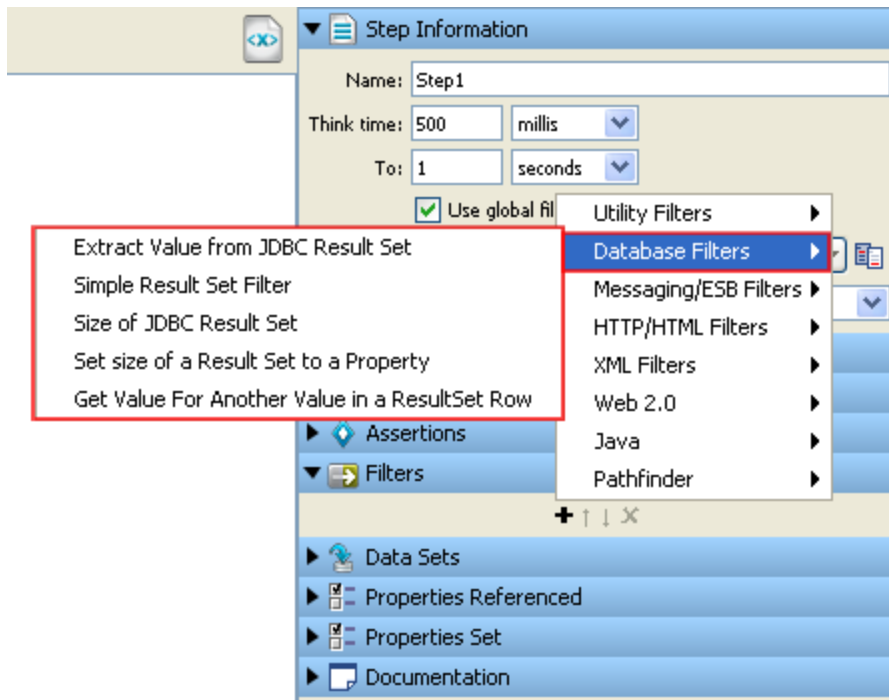
- **Filter in:** The name of the existing step. For example, if you want to parse the "lisa.delete user.cookies.rsp" property to return the value of the SESSIONID attribute, enter lisa.delete user.cookies.rsp.
- **Date Pattern:** Select the date pattern you want to display.
- **Property for Post Process:** Enter the name of the property to which you want the date to be stored.
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

18. Database Filters

18. Database Filters

These are the filters available in the Database Filters list for any test step:



This list explains these Filters in the order they appear on screen:

- 18.1 Extract Value from JDBC Result Set
- 18.2 Simple Result Set Filter
- 18.3 Size of JDBC Result Set
- 18.4 Set Size of a Result Set to a Property
- 18.5 Get Value For Another Value in a ResultSet Row

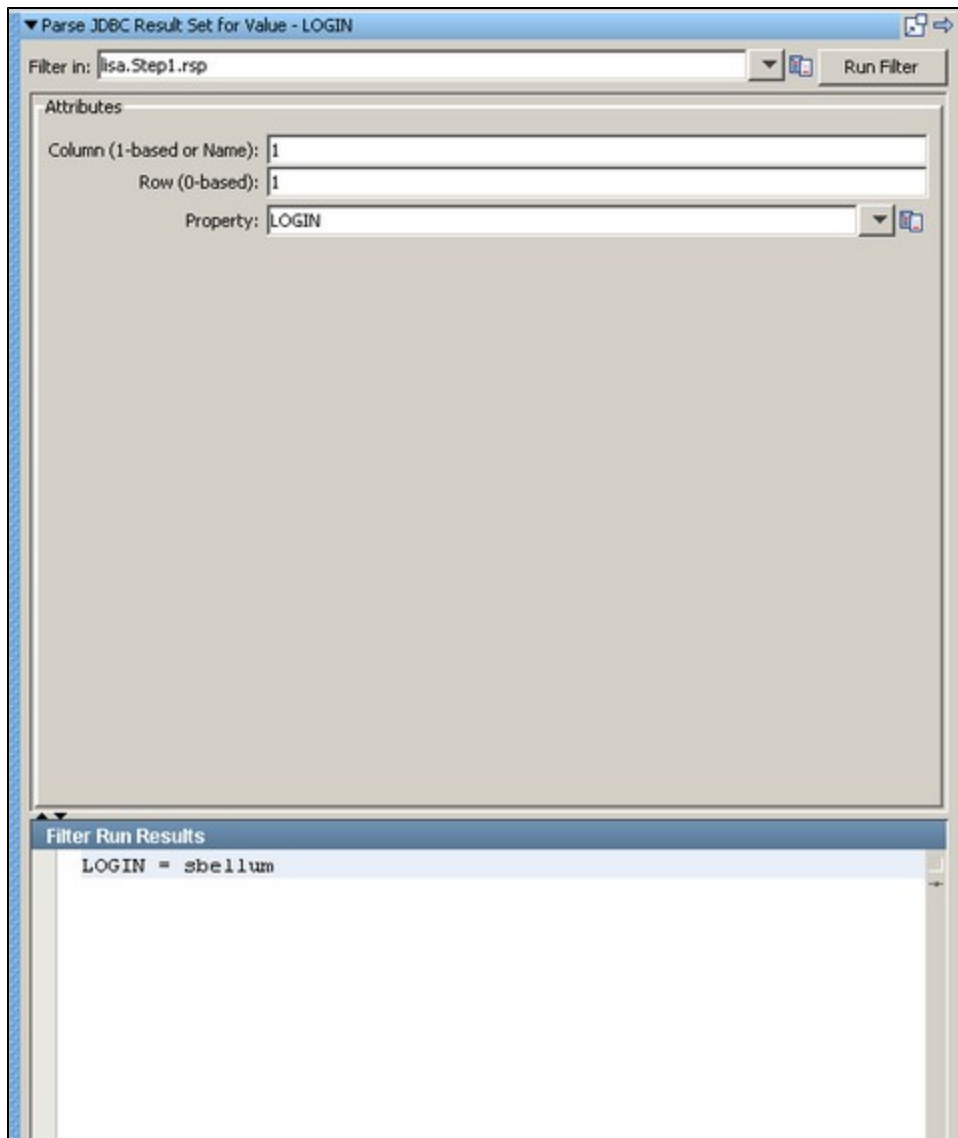
18.1 Extract Value from JDBC Result Set

18.1 Extract Value from JDBC Result Set

The Parse JDBC ResultSet for Value filter allows you to store the text of a specific JDBC result set value in a property.

This filter can be created in two ways, either as a manual filter from the filter list, or using the embedded filter commands on a result set response. We will look at both methods.

a) Creating the filter manually



Enter the following parameters:


- **Filter in:** The name of the property you want considered as the step's last response. The property should be in the pull down menu, if not you can enter it. It must be an existing property.
- **Column (1-based or name):** The index or name of the column (field).
- **Row (0-based):** The row to retrieve the value. This is a 0-based index.
- **Property:** The name of the property where the value in the cell at the row/column intersection is stored.
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

b) Creating the filter from a result set response


Display the step response that contains the result set. From within the result set select the cell of the value you want to store in the filter.

Result Set							
LOGIN	PWD	FNAME	LNAME	EMAIL	PHONE	ROLEKEY	SSN
admin	0DPIKuNIrrVm...	ITKO	Admin	lisabank-admin...	123-4567	2	434-47-5409
sbellum	26yJsXNpIb5A...	Sara	Bellum	sbellum@myco...	232-4345	1	614-40-1053
wpiece	/UuJ0MeQFuR...	Warren	Piece	wpiece@myco...	455-3232	1	546-71-4973
areck	AHDORRJD4AdI...	Amanda	Reckonwith	areck@mycom...	555-2244	1	350-02-1232
boaty	RQIiI0LdpTRd...	Boaty	Rabbit	boaty@rabbit...	333-4521	1	616-51-0344
itko	qUqP5cyxm6Y...	itko	test	itko.test@itko...	650-234-1212	1	140-72-2944
lisa_simpson	60fAFoq+W0...	lisa	simpson	lisa.simpson@i...	123-456-7890	1	295-20-0146

Select Generate Filter for Current Col/Row Value using the icon,  , shown by the arrow in the figure above.

In the dialog box that will open, check, or, reassign the columns for the search and the value, then enter the Property Key:

Please enter the property key for t...



OK

Cancel

Click the **OK** button.

LISA will create exactly the same filter as the one we created manually above.

In the example, the value in the cell in the 3rd column, and 3rd row, **sbellum**, will be stored in the property **theLogin**.

18.2 Simple Result Set Filter

18.2 Simple Result Set Filter

The Simple Result Set Filter is used to count the number rows in a Result Set Response.

Simple Result Set Filter

Attributes

☒ Result set has warnings

Row count >=:

Row count <=:

On error:

For more information refer to "Size of JDBC Result Set.

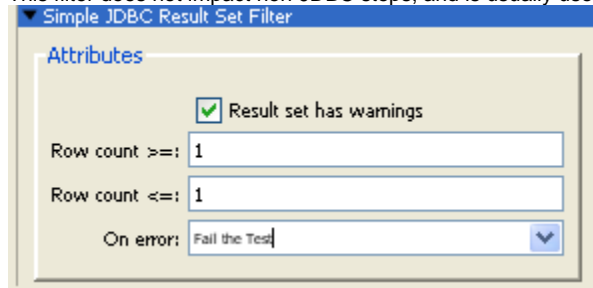
18.3 Size of JDBC Result Set

18.3 Size of JDBC Result Set

(Simple JDBC ResultSet Filter)

The Simple JDBC ResultSet filter allows you to check that the result set returned in each JDBC-based step matches the criteria specified. It is a simple filter to handle most common database errors automatically.

This filter does not impact non-JDBC steps, and is usually used as a global filter in a test case.



Enter the following parameters:

- **Result Set Has Warnings:** Some databases return warnings in the result set. If your database supports this feature and you wish to make a warning fire this filter's on error step, make sure the Result Set Has Warnings box is checked.
- **Row Count At Least (>=):** The minimum number of rows in the result set. If the result set contains less than this value, the filter sets the next step to the value specified in On Error Step.
- **Row Count No More Than (<=):** The maximum number of rows in the result set. If the result set contains more than this value, the filter sets the next step to the value specified in On Error Step.
- **On Error:** the step to execute if the filter's conditions are not met.

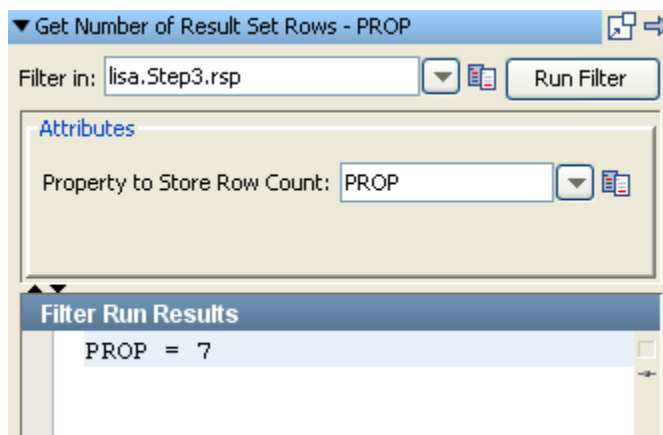
Note: This filter can serve the purpose of a general global assertion since you can choose a next step based on the presence of an error.

18.4 Set Size of a Result Set to a Property

18.4 Set Size of a Result Set to a Property

(Get Number of Result Set Rows)

The Set size of a Result Set to a Property filter allows you to store the count of a result set to a property provided.



Enter the following parameters:

- **Filter in:** The name of the property you want considered as the step's last response. The property should be in the pull down menu, if not you can enter it. It must be an existing property.
- **Property to Store Row Count:** User provided property name to store the row count. Default property name is **PROP**.
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

18.5 Get Value For Another Value in a ResultSet Row

18.5 Get Value For Another Value in a ResultSet Row

The Get Value for Another Value in a ResultSet row filter allows you to search a column (field) in a result set for a particular value. If the value is found the value in another column (field) and the same row is placed in a property.

This filter can be created in two ways, either as a manual filter from the filter list, or using the embedded filter commands on a result set response. We will look at both methods.

1. Creating the filter manually:

▼ Get Value For Another Value in a ResultSet Row - Get Value For Another Value in a ResultSet Row

Filter in: Run Filter

Attributes

Search Text (Regular Expression):

Search Column (1-based or Name):

Value Column (1-based or Name):

Property:

Filter Run Results

Enter the following parameters:

- **Filter in:** The name of the property you want considered as the step's last response. The property should be in the pull down menu, if not you can enter it. It must be an existing property.
- **Search Text (Regular Expression):** The search string.
- **Search Column (1-based or Name):** The index or name of the column to search.
- **Value Column (1-based or Name):** The index or name of the column to extract the property value.
- **Property:** The name of the property to store the value.
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

2. Creating the filter from a result set response:

Display the step response that contains the result set. From within the result set select the two values in different columns, using the control key.

Result Set							
LOGIN	PWD	FNAME	LNAME	EMAIL	PHONE	ROLEKEY	SSN
admin	ODPIKuNIrrVm...	ITKO	Admin	lisabank-admin@itko.com	123-4567	2	434-47-5409
sbellum	26yJsXNpIb5...	Sara	Bellum	sbellum@mycompany.com	232-4345	1	614-40-1053
wpiece	/UuJ0MeQFu...	Warren	Piece	wpiece@mycompany.com	455-3232	1	546-71-4973
areck	AHRRRjD4Ad...	Amanda	Reckonwith	areck@mycompany.com	555-2244	1	350-02-1232
boaty	RQil0LdpTRd...	Boaty	Rabbit	boaty@rabbit.org	333-4521	1	616-51-0344
itko	qUqP5cyxm6...	itko	test	itko.test@itko.com	650-234-1212	1	140-72-2944
lisa_simpson	60FAFoq+W0...	lisa	simpson	lisa.simpson@itko.com	123-456-7890	1	295-20-0146

Base Result Set Filter for a value and then get another column value

Select **Filter for a value** and then get another column value filter using the icon, , shown by the arrow in the figure above.

In the dialog box that opens, check, or, reassign the columns for the search and the value, then enter the Property Key:

Generate Value for Value Filter

Search Column (1-based or Name):

Value Column (1-based or Name):

Property Key to save value into:

OK Cancel

Click the **OK** button.

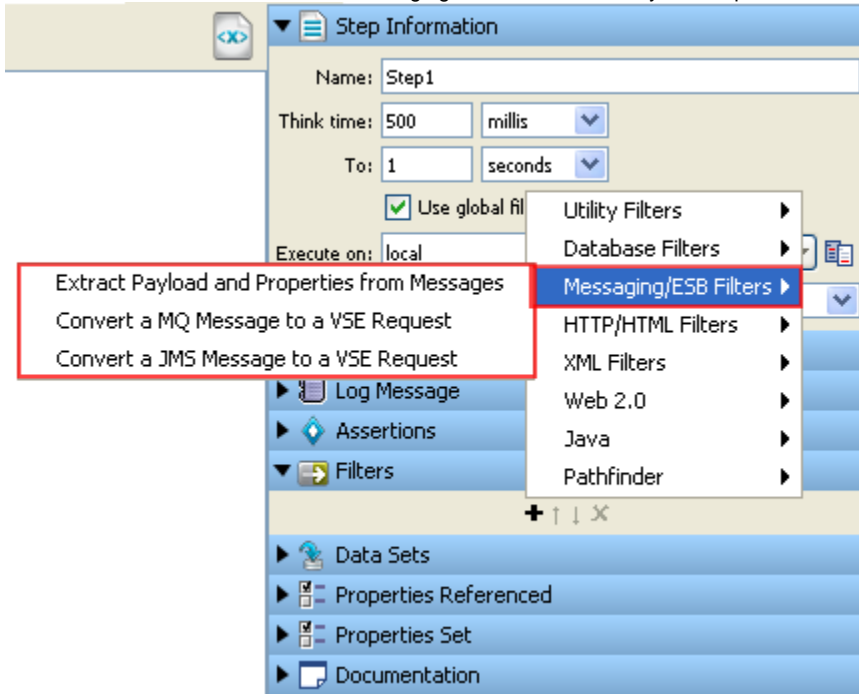
LISA will create exactly the same filter as the one we created manually above.

In the example, **sbellum** will be searched for, and if found, the value in the **EMAIL** column of that row will be placed in the property **theEmail**.

19. Messaging_ESBFilters

19. Messaging_ESBFilters

These are the filters available in the Messaging/ESBFilters list for any test step:



This list explains these Filters in the order they appear on screen:

- 19.1 Extract payload and Properties from Messages
- 19.2 Convert a MQ Message to a VSE Request
- 19.3 Convert a JMS Message to a VSE Request

19.1 Extract payload and Properties from Messages

19.1 Extract payload and Properties from Messages

There are several internal properties of messages that LISA will auto extract into properties in the test step using this filter. You can also select to auto extract the payload into a property. This is fast way to get data from message.

Different messaging platforms impose various restrictions & can be seen as warnings at execution.

The property names can default to lisa.stepName.message or you can specify the prefix. You can specify exact name for the payload.

▼ Auto Extract Message Data - Auto Extract Message Data

Filter in:

Attributes

☒ Get payload

Property key to store the payload:

Prefix for extracted details.:

☒ Get Message ID

☒ Get correlation ID where appropriate

☒ Additional extended properties

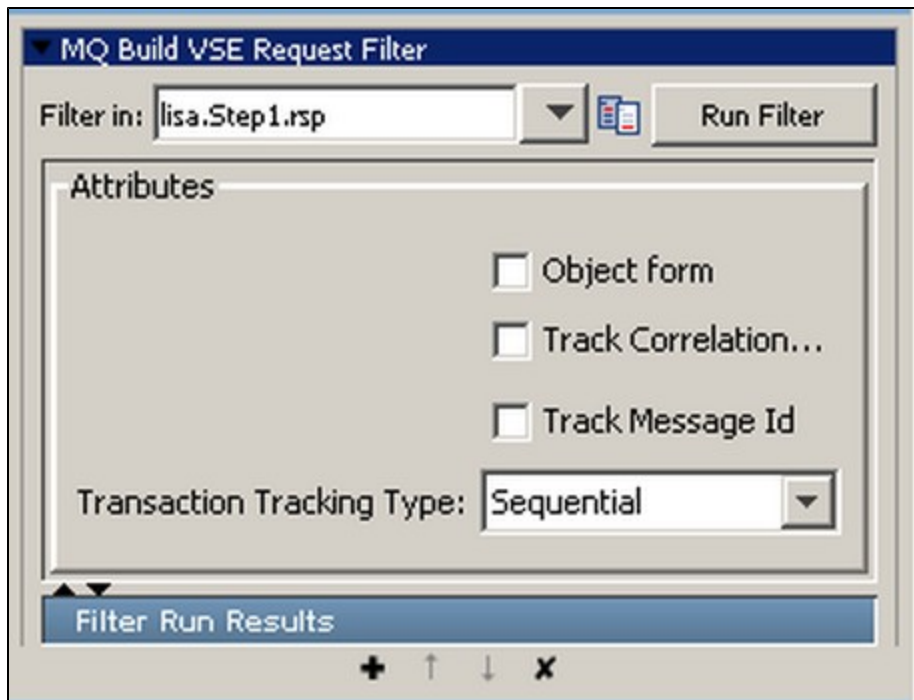
Filter Run Results

Property Name	Property Value
PROP	7
testCase	Test Case
testCaseId	31373561323230352D656635642D3430

19.2 Convert a MQ Message to a VSE Request

19.2 Convert a MQ Message to a VSE Request

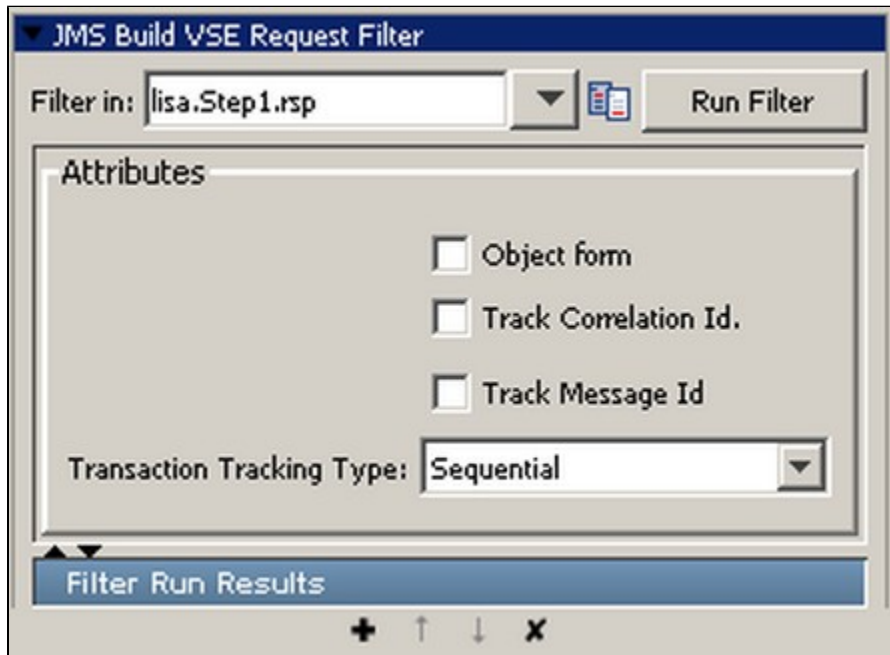
This filter is automatically added from the VSE recorder. It serves the specific purpose that enables proper functioning with recordings. It should not be used carefully & if it added to a step in VSE model, it should not typically be removed or edited.



19.3 Convert a JMS Message to a VSE Request

19.3 Convert a JMS Message to a VSE Request

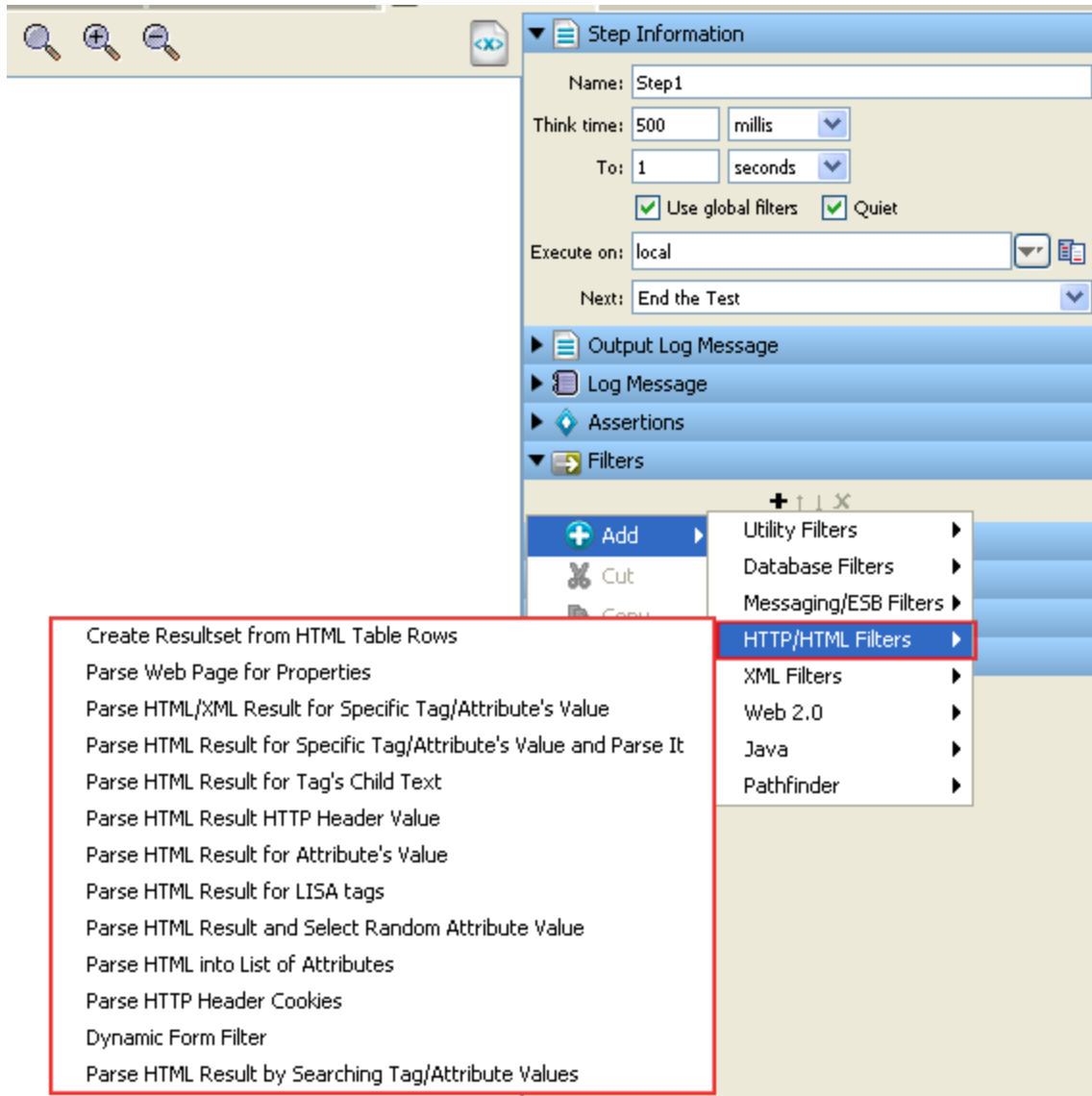
This filter is automatically added from the VSE recorder. It serves the specific purpose that enables proper functioning with recordings. It should not be used carefully & if it added to a step in VSE model, it should not typically be removed or edited.



20. HTTP_HTML Filters

20. HTTP_HTML Filters

These are the filters available in the HTTP/HTML Filters list for any test step:



This list explains these Filters in the order they appear on screen:

- 20.1 Create Resultset from HTML Table Rows
- 20.2 Parse Web Page for Properties
- 20.3 Parse HTML_XML Result for Specific Tag_Attributes Values
- 20.4 Parse HTML Result for Specific Tag_Attribute's Value and Parse It
- 20.5 Parse HTML Result for Tag's Child Text
- 20.6 Parse HTML Result for HTTP Header Value
- 20.7 Parse HTML Result for Attribute's Value
- 20.8 Parse HTML Result for LISA Tags
- 20.9 Parse HTML Result and Select Random Attribute Value
- 20.10 Parse HTML into List of Attributes
- 20.11 Parse HTTP Header Cookies
- 20.12 Dynamic Form Filter
- 20.13 Parse HTML Result by Searching Tag_Attribute Values

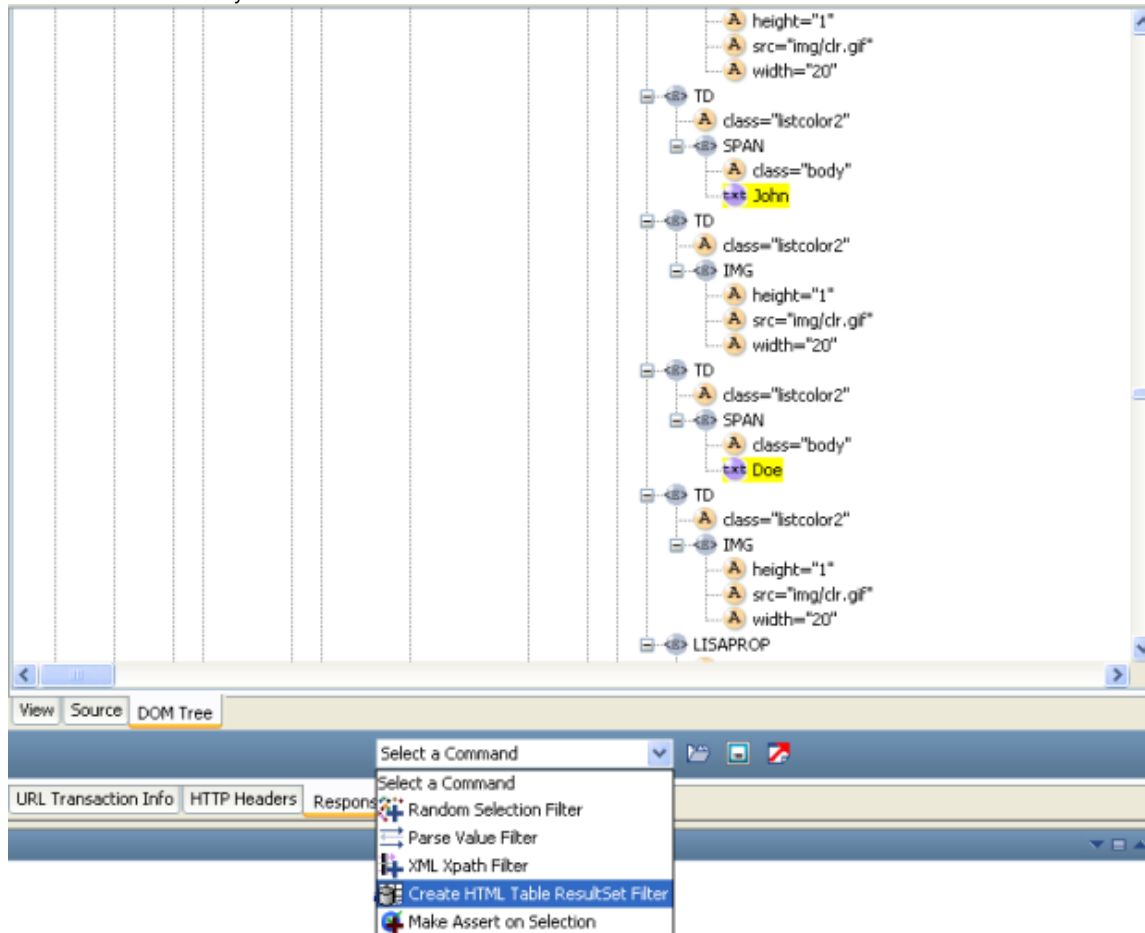
20.1 Create Resultset from HTML Table Rows

20.1 Create Resultset from HTML Table Rows

The Create Resultset from HTML Table Rows filter allows you to create a result set (i.e. a JDBC result set) from an HTML table returned in the HTML response. The columns and rows of an HTML table can be selected, and LISA will create a result set from them. The result set can then be used to generate assertions in the same way as it would in a database step.

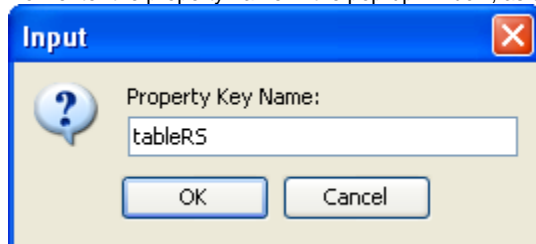
Although you can create this filter by choosing it from the filter list and filling in the parameters, it is far easier to create it directly from the HTTP/HTML Request step response using one of the filter commands available to that step. This is the approach we take here. The parameters produced here, i.e. the ones you would have had to calculate to manually create this filter, are shown later in this section.

To create a filter on a table, record a web page that contains the table, go to the appropriate HTML step, and view it from the DOM tree. Select the values that are to be placed in the table, using the control key, to select multiple fields. You only have to select one example value from each column in the table that you want to use in the result set:



Once highlighted, select Create HTML Table Results Filter, shown by the arrow in the figure above.

Now enter the property name in the pop-up window, as shown below:



The property will now be available in the test case.

LISA added this property to the current step. Shown below are the parameters that LISA calculated for this step. These are the parameters you would have had to supply to manually create this filter.

Filter in: Run Filter

Attributes

Save Into Property Key:

HTML Table for Search:

Defined Element Paths:

Filter Run Results

To show the results of this filter we added a step of type Save Property as Last Response and put in the property created by the filter. The result set panel is displayed below:

Property Key to Save as Step Result:

Result Set	
SPAN1-TXT-VALUE	SPAN2-TXT-VALUE
login	password
admin	0DPiKuNIrrVmD8IUCuw1hQxNqZc=
sbellum	26yJsXNpIb5AKFGPDryfAlCpmho=
wpiece	/UuJ0MeQFuRNHog0H0hR+Y82B5w=
areck	AHDRRjD4AdIUOCwt4gG/g75GR0c=
boaty	RQii0LdpTRdNaHCft+tsalrPGg=
itko	qUqP5cyxm6YcTAhz05Hph5gvu9M=
lisa_simpson	60fAFoq+W0R4HrLgsfPodkWRw9I=
webapp-38951908	AL2slstNsa15az09CwLQ/ThKVnU=

If you are editing an existing test case you may need to replay the test case to generate the property from the filter using the Replay test case to a specific point command. The Replay test case to a specific point command is activated using the Replay,



, icon on the toolbar, or from the Command menu. You can now use the embedded filters and assertions that are available at the bottom of the result set window of this step.

20.2 Parse Web Page for Properties

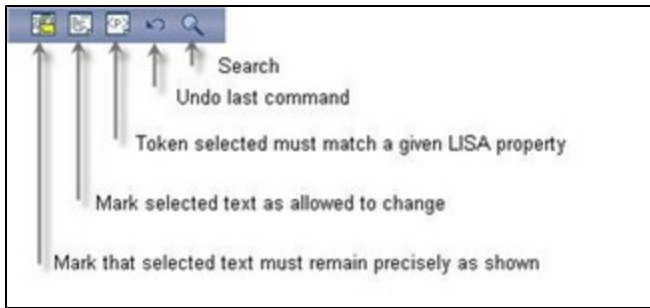
20.2 Parse Web Page for Properties

The Parse Web Page for Properties filter allows you to view a rendered web page to create properties from the HTML content. It uses the "Paint the Screen" technique.

Paint the Screen allows you great flexibility to define what in the HTML you want to parse out as properties. There are three ways to mark the text:

1. Text that must appear in the response precisely as shown. A "Must" block.
2. Text that does not have to appear in the response, or can change. An "Any" block.
3. Text that will be stored in a LISA property. A "Property" block.



The text is marked using the icons at the bottom of the editor:



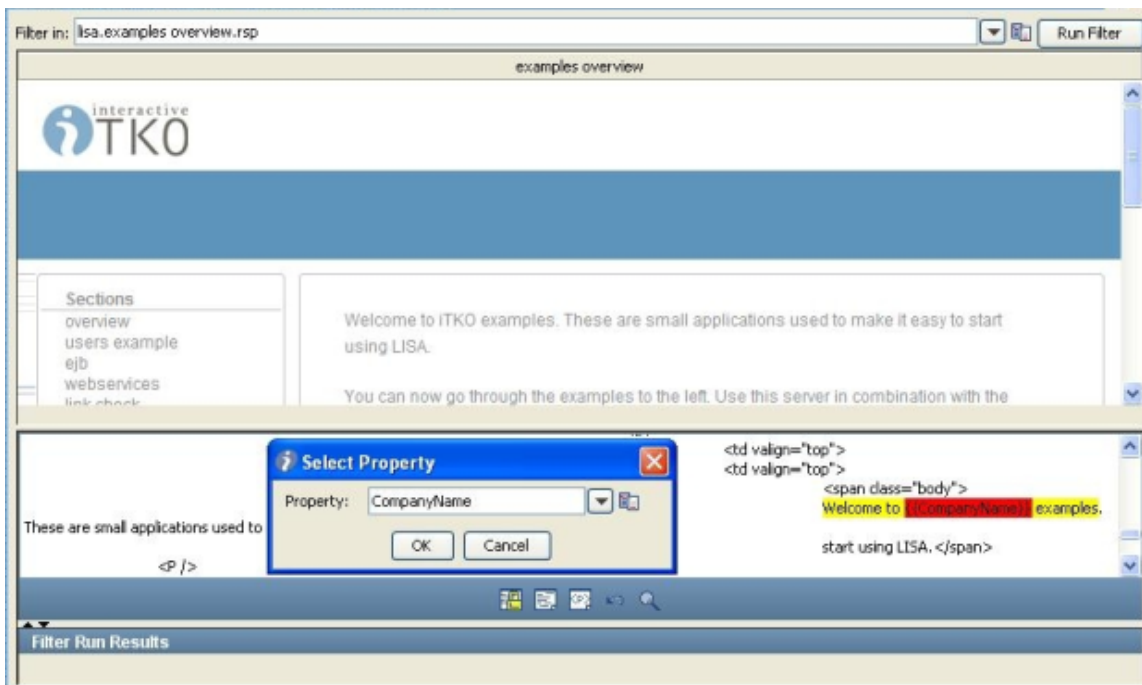
This technique is best explained by example.


In the following example, see the figure below, let's assume that we expect that the company name "iTKO" will change from user to user, and therefore needs to be stored as a LISA property.


We have marked the text using the icons shown above, by selecting text and then clicking on the appropriate icon.

- Yellow background indicates text that must appear as shown (indicated by the arrows from "Text uses"). This is a "Must" block and is marked using the  icon.
- Red background identifies the text that will be stored in the property entered into the dialog box (indicated by a single arrow). This is a "Property" block and is marked using the  icon.

Note: Property blocks must always be bounded by Must blocks.

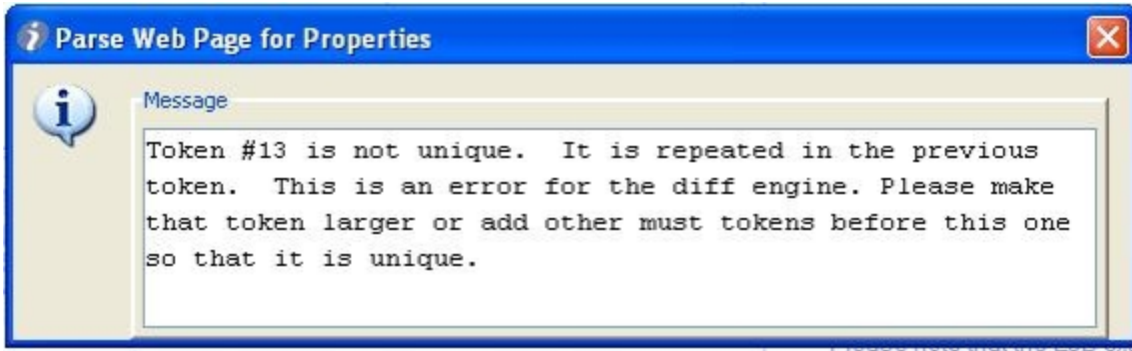


This screen shows the HTML rendered in a browser in the top panel, and the actual HTML text in the bottom panel. We want to parse out the company name in the phrase "Welcome to iTKO examples". We have set the boundaries around that, and clicked the "Must" icon, .

. Then we selected the company name text, "iTKO", inside the highlighted content, and clicked the "Property" icon, . We entered the property name into the dialog box. Note that the company name text has been replaced with the name of the LISA property. Frequently you can do this purely from the web page view by selecting the content in the web browser. At times, it will be easier just to click on the web browser in the area that you want to select, then make your actual selection in the HTML panel. Now when this filter is run, the property "CompanyName" will be assigned the current value that appears on the HTML page. Note that the phrase "Welcome to iTKO examples" can change location in the text buffer and it will still be located and parsed for the property. You can repeat this process on this text buffer to have as many properties defined as you like.

Handling Non-unique Tokens

If you see an error message such as the following:



LISA is telling you that your selected token is not unique, the selection you just made is repeated in the token before it. To solve this in most cases, simply create another token to make the prior token a "Must" token as well. In other cases, when this doesn't work, a judicious placement of another "Must" block in between the two duplicate tokens should do the trick. This will work because LISA can distinguish between the two duplicate tokens based on their relative location.

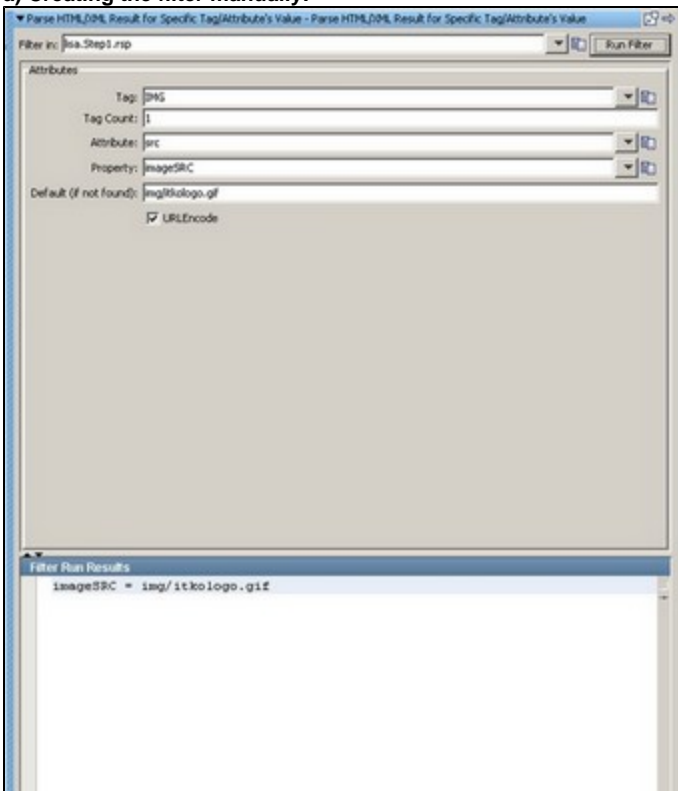
20.3 Parse HTML/XML Result for Specific Tag_Attributes Values

20.3 Parse HTML/XML Result for Specific Tag/Attributes Values

The Parse HTML for Specific Tag/Attribute's Values filter allows you to parse the HTML response for a given attribute of a given tag.

This filter can be created in two ways, either as a manual filter from the filter list, or using the embedded filter commands on a result set response. We will look at both methods.

a) Creating the filter manually:



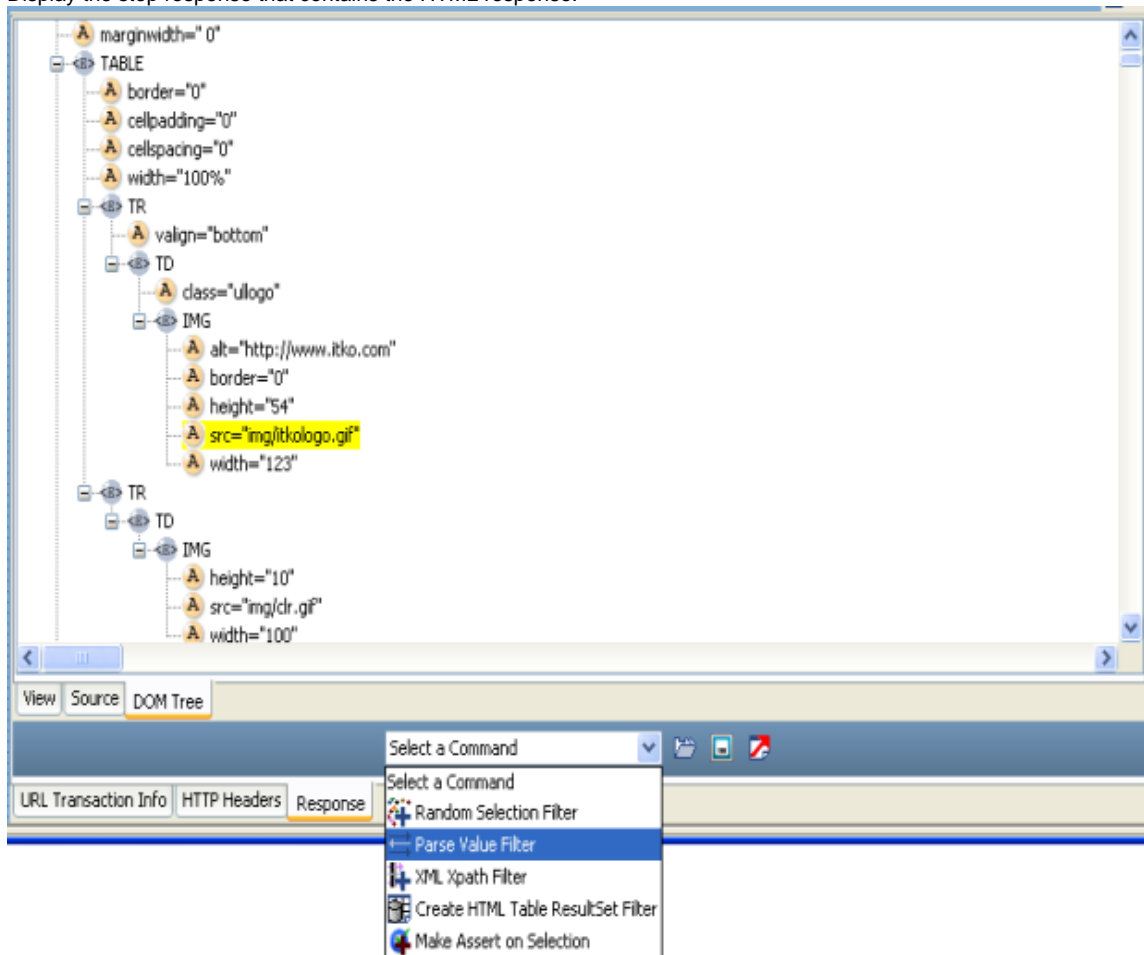
Enter the following parameters:

- **Filter in:** The name of the property you want considered as the step's last response. The property should be in the pull down menu, if not you can enter it. It must be an existing property.
- **Tag:** The name of the HTML tag, for an image tag enter IMG.
- **Tag Count:** The occurrence of the tag from the top of response, for the first image tag enter 1.
- **Attribute:** The name of the attribute to filter, for the source attribute enter src.
- **Property:** The property in which to store the value.
- **Default (if not found):** The value to use if the attribute value is not found.
- **URLEncode:** When checked, property value is URLEncoded.
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

b) Creating the filter from the HTTP/HTML request step response page:

Display the step response that contains the HTML response.



From Dom Tree view select the attribute whose value you want to store in a property. Here we choose the same attribute as above in a). Once highlighted, select Parse Value Filter shown by the arrow in the figure above.

Now enter the property name in the pop-up window, as shown below:

Note: You can also add an assertion at this point if you wish. This is a Property Value Expression Assertion. LISA will add this to the assertions for this step.

20.4 Parse HTML Result for Specific Tag_Attribute's Value and Parse It

20.4 Parse HTML Result for Specific Tag/Attribute's Value and Parse It

The Parse HTML Result for Specific Tag/Attribute's Value and Parse It filter is really a combination of two other filters: "Parse HTML Result for

Attribute's Value", and "Parse Property Value as Argument String".

This filter is designed to find a certain attribute in a web page, and then further parse that attribute. If the attribute is a URL, and not just a name-value pair, then there is a function for handling that information.

In this example, we see the filter finds the seventh anchor tag's "href" attribute, which is a URL. The filter takes the "command" parameter and stores that value in the "**cmdlist users_KEY**".

Filter in: ▼ 📄 Run Filter

Attributes

Tag: ▼ 📄

Tag Count:

Attribute: ▼ 📄

☒ IsURL

Argument To Parse:

Property: ▼ 📄

☐ URLEncode

Default (if not found):

Filter Run Results

Enter the following parameters:

- **Filter in:** The name of the property you want considered as the step's last response. The property should be in the pull down menu, if not you can enter it. It must be an existing property.
- **Tag:** The name of the HTML tag, for an anchor tag enter "a".
- **Tag Count:** The occurrence of the tag from the top of response, for the seventh anchor tag enter "7".
- **Attribute:** The name of the attribute to filter, for the href attribute enter "href".
- **IsURL:** Check the checkbox if the attribute value is a URL.
- **Argument to Parse:** The name of the argument to parse for it's value, in our case "cmd".
- **Property:** The property in which to store the value, in our case "cmdlist users_KEY".
- **URLEncode:** When checked, property value is URLEncoded.
- **Default (if not found):** The value to use if the attribute value is not found.
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

20.5 Parse HTML Result for Tag's Child Text

20.5 Parse HTML Result for Tag's Child Text

The Parse HTML Result for Tag's Child Text filter allows you to store the text of a tag's child text in a property.

Enter the following information:

- **Filter in:** The name of the property you want considered as the step's last response. The property should be in the pull down menu, if not you can enter it. It must be an existing property.
- **Tag:** The type of the tag. For example, for an h1 tag enter h1.
- **Tag Count:** The occurrence of the tag. For the child text of the third h1 tag, enter 3.
- **Property:** The name of the property to store the text.
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

20.6 Parse HTML Result for HTTP Header Value

20.6 Parse HTML Result for HTTP Header Value

The Parse HTML Result for HTTP Header Value filter allows you to store the value of a returned HTTP header key in a property.

A common use of this filter is saving the HTTP header Server in a property called SERVER_NAME.

Enter the following parameters:

- **Filter in:** The name of the property you want considered as the step's last response. The property should be in the pull down menu, if not you can enter it. It must be an existing property.
- **HTTP Header Key:** The name of the HTTP header, for example "Server"
- **Property:** the name of the property to store the header value
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

20.7 Parse HTML Result for Attribute's Value

20.7 Parse HTML Result for Attribute's Value

The Parse HTML Result for Attribute's Value filter allows you to store the text of a specific attribute in a property. The attribute can occur anywhere in the result, including scripting code.

The parameters set are:

- **Filter in:** The name of the property you want considered as the step's last response. The property should be in the pull down menu, if not you can enter it. It must be an existing property.
- **Attribute:** The type of attribute to retrieve. For example, if you want the URL of an anchor tag enter "href".
- **Count:** The occurrence of the tag. For example, if you want the URL of the third anchor tag on the page, enter "3".
- **Property:** the name of the property to store the text of the attribute, enter "anchor3".
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

20.8 Parse HTML Result for LISA Tags

20.8 Parse HTML Result for LISA Tags

The Parse HTML Result for LISA Tags filter provides a way for developers to test-enable their web applications. For an in-depth study on test-enabling, please see the [LISA Developer's Guide \(LEK\)](#).

LISA provides the ability to insert 'LISAPROP' tags into your web page. The LISAPROP tag has two attributes – name and value. The LISAPROP tags do not show up in your web pages. They function only to discretely provide valuable information about your web page to a tester. An example of a LISAPROP might be:

`<LISAPROP name="FIRST_USER" value="sbellum">.`

If a tester has installed this type of filter, LISA will automatically assign the property 'FIRST_USER' the value 'sbellum'. This removes any need on behalf the tester to parse for this value. This type of filter helps a developer make the testing easier.

Frequently a web page will not contain the information needed to perform proper validation, or that information is very difficult to parse. Even when it is there, the parsing can become incorrect due to subtle changes in the HTML that is generated. This LISAPROP filter can resolve many tedious parsing issues for web testing.



There are no parameters required.

20.9 Parse HTML Result and Select Random Attribute Value



20.9 Parse HTML Result and Select Random Attribute Value

The Parse HTML Result and Select Random Attribute Value filter allows you to store the text of a random selection from a set in a property. The attribute can occur anywhere in the result, including scripting code.



▼ Parse HTML Result and Select Random Attribute Value - Parse HTML Result and Select Random Attribute Value



Filter in:  

Attributes



Outer Tag:  



Tag Count:

Inner Tag:  

Filter Attribute:  

Filter Value:

Attribute:  

Property Key:  

Filter Run Results

Enter the following parameters:

- **Filter in:** The name of the property you want considered as the step's last response. The property should be in the pull down menu, if not you can enter it. It must be an existing property.
- **Outer Tag:** The outer element that contains the list from which to pick. For example, to select a drop-down menu, you would enter the text "select".
- **Tag Count:** The occurrence of the outer tag. For example, to select the second drop-down menu, you would enter the text "2".
- **Inner Tag:** The tag to randomly pick the attribute from. To pick a random item in the drop-down menu, you would enter the text "option".
- **Filter Attribute:** Optional field to specify attribute names that should not appear in the pick list.
- **Filter Value:** Optional field to specify attribute values that should not appear in the pick list.
- **Attribute:** The attribute from which to retrieve text. If this is blank, the child text of the inner tag is returned.
- **Property Key:** The name of the property to store the text of the attribute.
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

20.10 Parse HTML into List of Attributes

20.10 Parse HTML into List of Attributes

The Parse HTML into List of Attributes filter allows you to store the text of a set of attributes, as a list, in a property.

- **Filter in:** The name of the property you want considered as the step's last response. The property should be in the pull down menu, if not you can enter it. It must be an existing property.
- **Outer Tag:** The outer element that contains the list of tags to parse. For example, to store all of the links from all the anchor tags in a table, enter table.
- **Outer Tag Count:** The occurrence of the outer tag. For the second table enter 2.
- **Inner Tag:** The tag to retrieve the values from. For example, for all of the anchor tags in the table enter a.
- **Filter Attribute:** Optional field to specify attribute names that should not appear in the pick list.
- **Filter Value:** Optional field to specify attribute values that should not appear in the pick list.
- **Attribute:** The attribute of the Inner Tag to retrieve the text from. If this is blank, the child text of the Inner Tag is returned. To store all of the links from all of the anchor tags in a table enter href.
- **Property Key:** the name of the property to store the text of the attribute.
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

20.11 Parse HTTP Header Cookies

20.11 Parse HTTP Header Cookies

The Parse HTTP Header Cookies filter allows you to parse the HTTP header for cookie values, and store them in a property starting with a specific prefix:

Enter the following parameter:

- **Filter in:** The name of the property you want considered as the step's last response. The property should be in the pull down menu, if not you can enter it. It must be an existing property.
- **Property Prefix:** A text string that will be prefixed to the cookie name to provide the property name to use. The full names of these properties are therefore dependent on the names of the cookies that have been returned. The cookie names can be identified in the property tab of the Interactive Test Run (ITR).
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

20.12 Dynamic Form Filter

20.12 Dynamic Form Filter

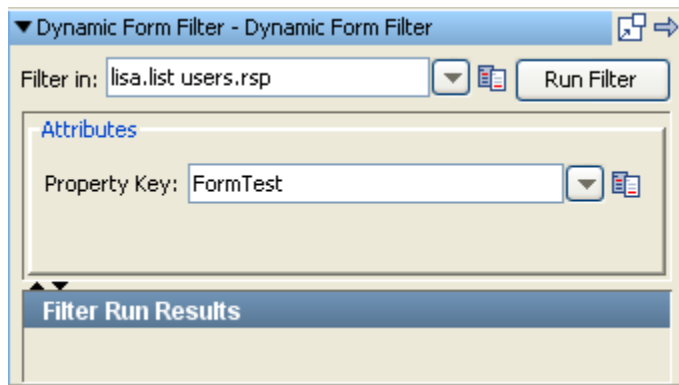
The Dynamic Form filter identifies dynamically generated forms in HTML responses and parses them into a set of properties. The Property Key that you enter becomes part of the property name for each form element in each form. This is easier to understand by examining the example that follows.

You might test an HTML page with two dynamically generated forms:

```
<form name="F001" action="index.jsp"> <input type="text" name="0001A" value="default" /> <input type="text" name="0001B" value="" /></form>
```

```
<form name="F002" action="orders.jsp"> <input type="text" name="0002A" value=Key"" /> <input type="text" name="0002B" value="" /></form>
```

Using a Property Key of FormTest in the filter panel:



would create the following key-value pairs:




Key	Value
FormTest.Form1.text1.name	0001A
FormTest.Form1.text1.value	default
FormTest.Form1.text2.name	0001B
FormTest.Form1.text2.value	
FormTest.Form2.text1.name	0002A
FormTest.Form2.text1.value	
FormTest.Form2.text2.name	0002B
FormTest.Form2.text2.value	

20.13 Parse HTML Result by Searching Tag_Attribute Values



20.13 Parse HTML Result by Searching Tag/Attribute Values



The Parse HTML Result by Searching Tag/Attribute Values filter allows you to filter the value of a tag attribute by searching for the name and value of another attribute in that tag. If more than one tag fits the criteria you can specify which one you want.

▼ Parse HTML Result by Searching Tag/Attribute Values - Parse HTML Result by Searching Tag/Attribute Values

Filter in:    Run Filter



Attributes



Tag:  

Search Criteria Attribute:  

Search Criteria Value Expression:

Tag Count:

Attribute:  

Property:  

Default (if not found):

☒ URLEncode

Filter Run Results

The parameters set are:

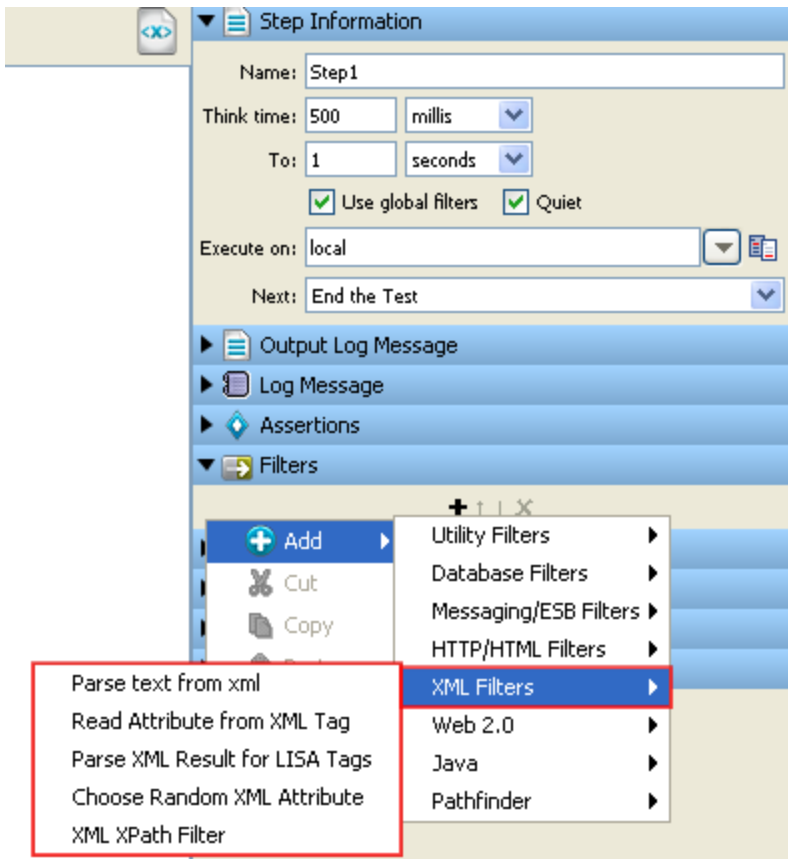
- **Filter in:** The name of the property you want considered as the step's last response. The property should be in the pull down menu, if not you can enter it. It must be an existing property.
- **Tag:** The name of the tag to search.
- **Search Criteria Attribute:** The attribute to search for.
- **Search Criteria Value Expression:** The attribute expression to search for.
- **Tag Count:** The specific tag to use from those that satisfy the search criteria.
- **Attribute:** The attribute whose value you want.
- **Property:** The property in which to store the value.
- **Default (if not found):** The value to use if search is not successful.
- **URLEncode:** When checked, the value is URLEncoded.
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

21. XML Filters

21. XML Filters

These are the filters available in the XML Filters list for any test step:



This list explains these Filters in the order they appear on screen:

- 21.1 Parse text from XML
- 21.2 Read Attribute from XML Tag
- 21.3 Parse XML Result for LISA Tag
- 21.4 Choose Random XML Attribute
- 21.5 XML XPath Filter

21.1 Parse text from XML

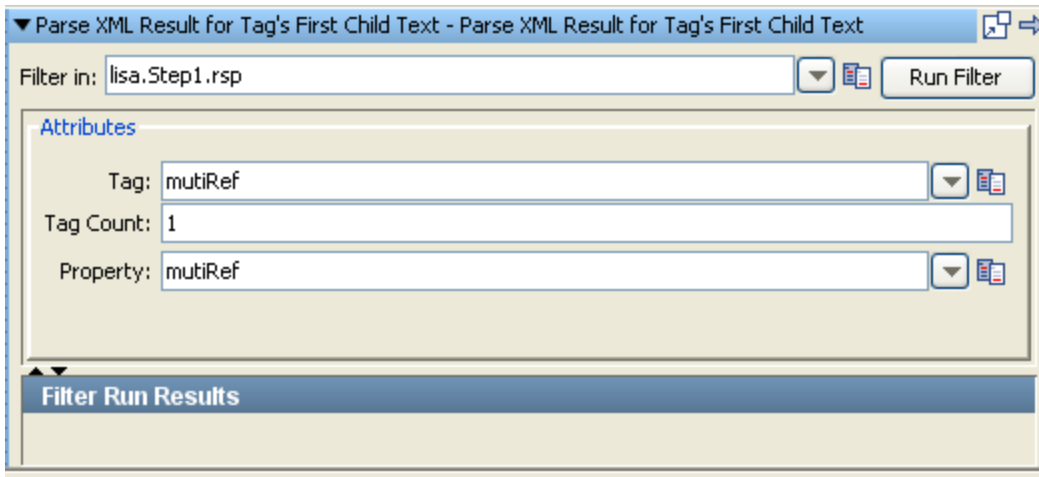
21.1 Parse text from XML

(Parse XML Result for Tag's First Child Text)

The Parse XML Result for Tag's First Child Text filter stores the text of a tag's child text in a property. To define a Parse XML Result for Tag's Child Text filter, set the Type of the filter and set the three attributes:

This filter can be created in two ways, either as a manual filter from the filter list, or using the embedded filter commands on an XML response. We will look at both methods.

Creating the filter manually



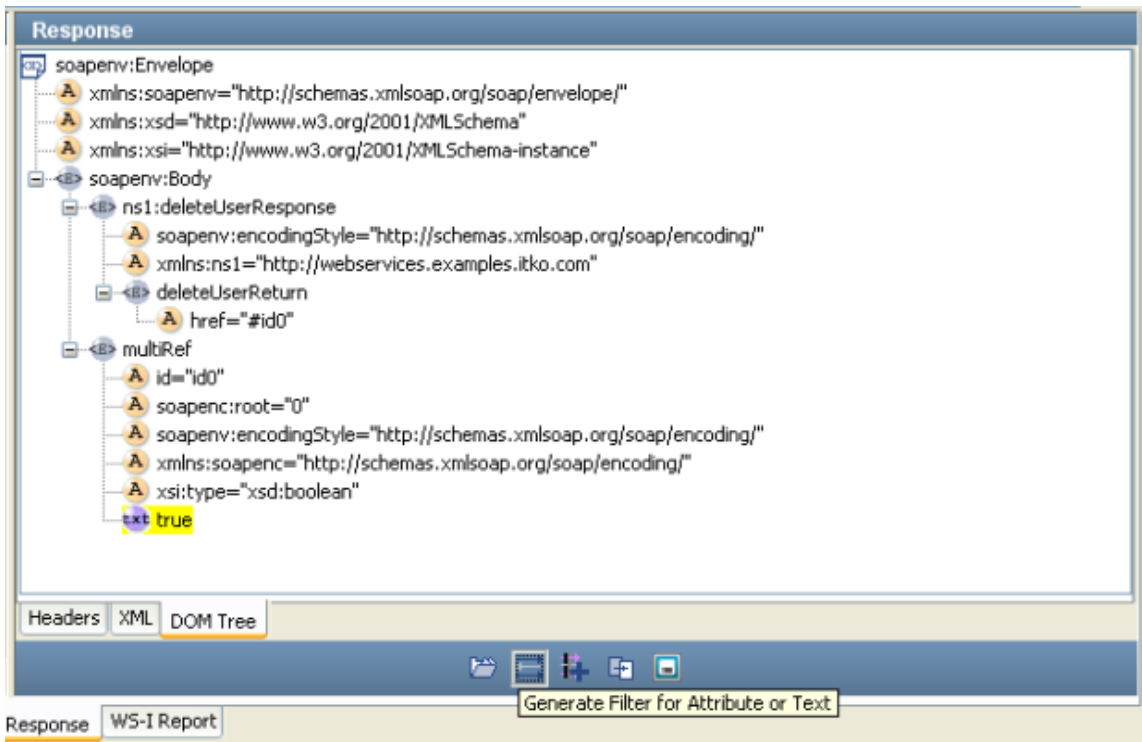
Enter the following parameters:

- **Filter in:** Where to apply the filter. The figure above shows list.Step1.rsp which means that the filter will be applied to the response of Step1. You can edit this value for this filter.
- **Tag:** the type of the tag. For example, if you want the child text of the multiRef tag, enter multiRef
- **Tag Count:** the occurrence of the tag. For example, if you want the child text of the first multiRef tag, set the Count to 1
- **Property:** the name of the property to store the text.
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Run Filter: Click the **Run Filter** button to execute the filter and see the results in the Filter Run Results.

Creating the filter directly from the response page

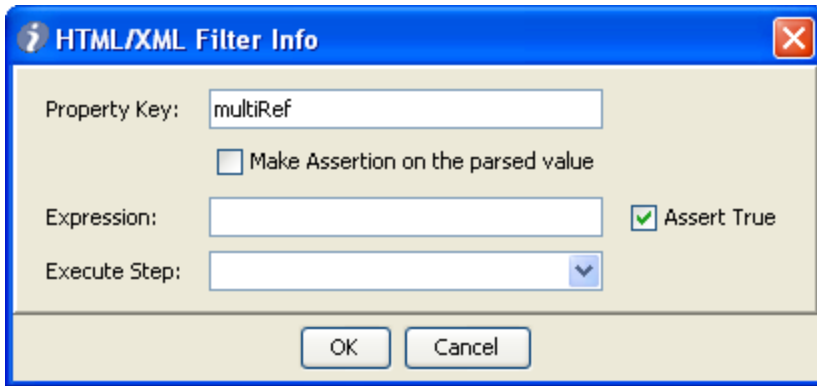
Displays the step response that contains the XML:



From Dom Tree view select the attribute whose value you want to store in a property. Here we choose the same attribute as above in A.

Once highlighted, select Generate Filter for Attribute or Text shown by the arrow in the figure above.

Now enter the property name in the pop-up window, as shown below:



HTML/XML Filter Info

Property Key:

☐ Make Assertion on the parsed value

Expression: ☒ Assert True

Execute Step: ▼

OK Cancel

Note: You can also add an assertion at this point if you wish. This is a Property Value Expression Assertion. LISA will add this to the assertions for this step.

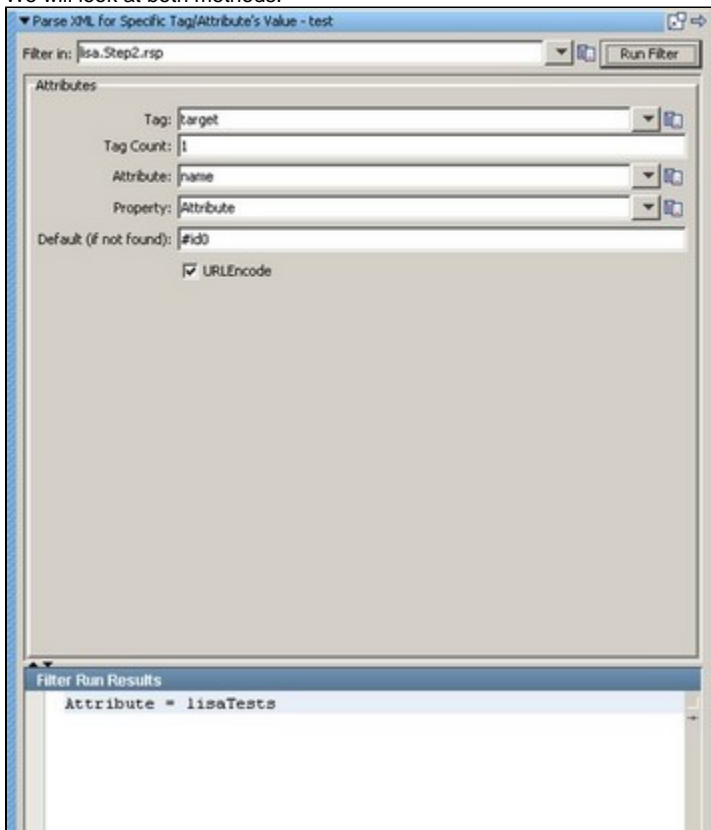
21.2 Read Attribute from XML Tag

21.2 Read Attribute from XML Tag

(Parse XML for specific Tag/Attribute's Value)

The Parse XML Result for Attribute's Specific Tag/Attribute's Value filter allows you to store the text of a specific attribute in a property. The attribute can occur anywhere in the result.

This filter can be created in two ways, either as a manual filter from the filter list, or using the embedded filter commands on an XML response. We will look at both methods.



▼ Parse XML for Specific Tag/Attribute's Value - test

Filter in: Run Filter

Attributes

Tag:

Tag Count:

Attribute:

Property:

Default (if not found):

☒ URLEncode

Filter Run Results

Attribute = lisaTests

Creating the filter manually

Enter the following parameters:

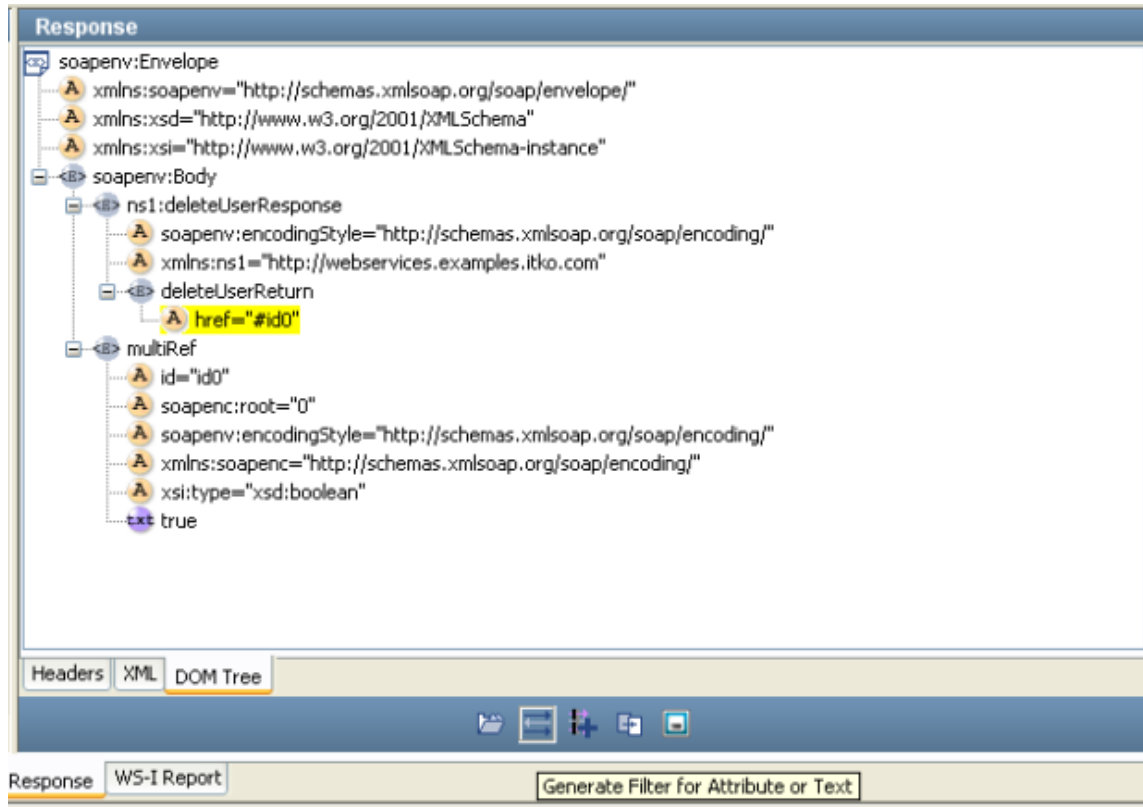
- Filter in: Where to apply the filter. The figure above shows list.Step1.rsp which means that the filter will be applied to the response of Step1.
- Tag: The name of the XML tag, for example deleteUserReturn.
- Tag Count: The occurrence of the tag from the top of response, for the first tag enter 1.
- Attribute: The name of the attribute to filter, for the href attribute enter href.

- **Property:** The property in which to store the value, say myAttribute.
- **Default:** The value to use if the attribute value is not found.
- **URLEncode:** When checked, property value is URLEncoded.
- **Filter Run Results:** Displays the property and values set as a result of running the filter.

Click the Run Filter button to execute the filter and see the results in the Filter Run Results.

Creating the filter from the directly from the response page

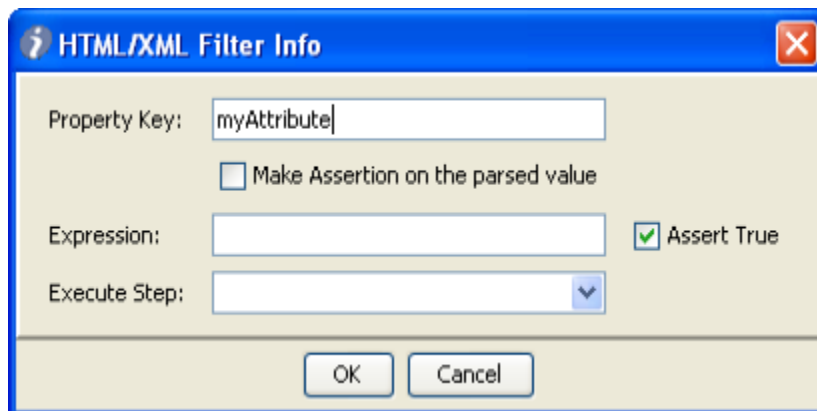
Display the step response that contains the XML:



From Dom Tree view select the attribute whose value you want to store in a property. Here we choose the same attribute as above in A.

Once highlighted, select Generate Filter for Attribute or Text shown by the arrow in the figure above.

Now enter the property name in the pop-up window, as shown below:



Note: You can also add an assertion at this point if you wish. This is a Property Value Expression Assertion. LISA will add this to the assertions for this step.

21.3 Parse XML Result for LISA Tag

21.3 Parse XML Result for LISA Tag

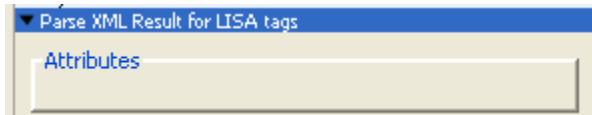
The Parse XML Result for LISA Tags filter provides a way for developers to test-enable their XML applications. For an in-depth study on test-enabling, please see the LISA Developer Guide.

LISA provides the ability to insert 'LISAPROP' tags into your XML page. The LISAPROP tag has two attributes – name and value. The LISAPROP tags function only to discretely provide valuable information about your XML to a tester. An example of a LISAPROP might be:

```
<LISAPROP name="FIRST_USER" value="sbellum">.
```

If a tester has installed this type of filter, LISA will automatically assign the property 'FIRST_USER' the value 'sbellum'. This removes any need on behalf the tester to parse for this value. This type of filter helps a developer make the testing easier.

The XML may not contain the information needed to perform proper validation, or that information is very difficult to parse. Even when it is there, the parsing can become incorrect due to subtle changes in the XML that is generated. This LISAPROP filter can resolve many tedious parsing issues.



There are no parameters required.

21.4 Choose Random XML Attribute

21.4 Choose Random XML Attribute

(Parse HTML Result and Select Random Attribute Value)

The Parse XML Result and Select Random Attribute Value filter allows you to store the text of a random selection from a set in a property. The attribute can occur anywhere in the result.

For more information refer to [Parse HTML Result and Select Random Attribute Value](#).

21.5 XML XPath Filter

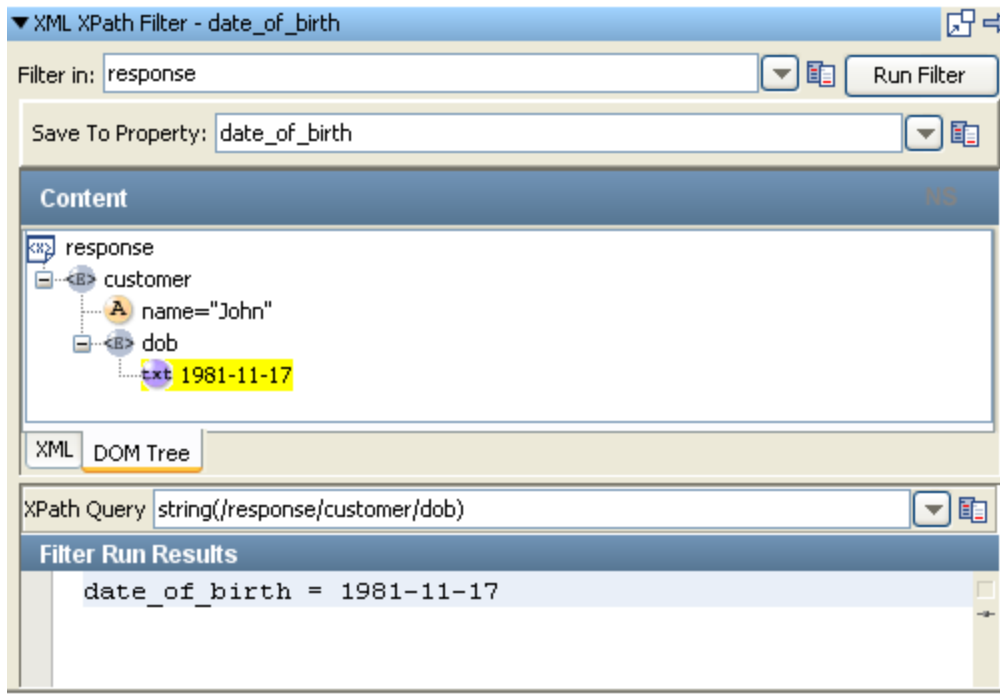
21.5 XML XPath Filter

The XML XPath filter allows you to use an XPath query that will be run on a property, or the last response and store it in a property. When this filter is selected, the last response is loaded into the content panel.

The response can be viewed as an XML document or as a DOM Tree. However, the XPath selection can only be made from the DOM Tree view.

There are three ways to construct the XPath Query:

- Manually enter the XPath expression in the XPath Query text box.
- Select an element from the DOM tree and let LISA construct the XPath expression.
- Select an element from the DOM tree, and then edit the XPath that LISA constructs. For example you may want to modify it to use a LISA property, or a counter data set.

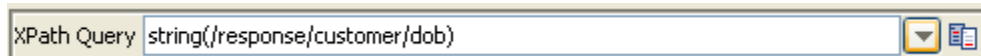


Enter the following parameters:

- Filter In: Enter either the last response or a named property.
- Save To Property: The property in which to store the result of the XPath query.

Now construct the XPath query using one of the methods described above.

After an XPath query has been constructed, test it by clicking the **Run XPath Query** button (shown by the arrow in the figure below) on the right side of the XPath Query panel:



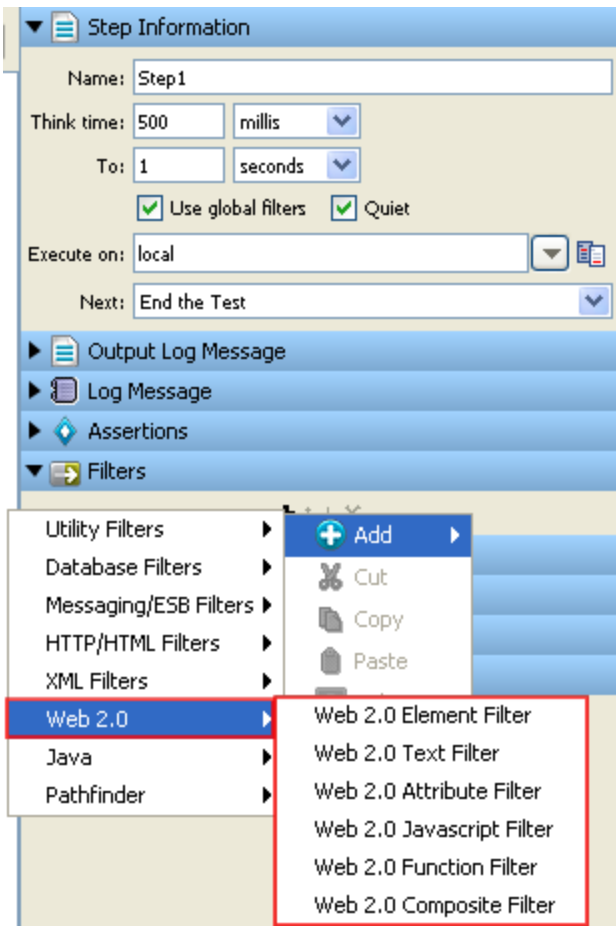
The results of the query are displayed in the **Filter Run Results** pane.

Click the **Run Filter** button to run the filter now.

22. Web 2.0 Filters

22. Web 2.0 Filters

These are the filters available in the Web 2.0 Filters list for any test step:



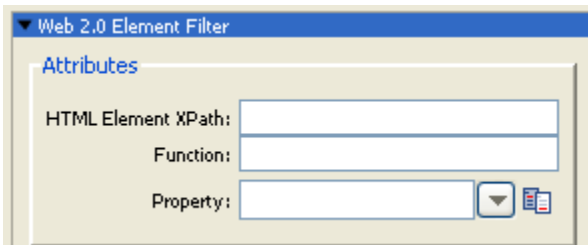
This list explains these Filters in the order they appear on screen:

- 22.1 Web 2.0 Element Filter
- 22.2 Web 2.0 Text Filter
- 22.3 Web 2.0 Attribute Filter
- 22.4 Web 2.0 Java Script Filter
- 22.5 Web 2.0 Function Filter
- 22.6 Web 2.0 Composite Filter

22.1 Web 2.0 Element Filter

22.1 Web 2.0 Element Filter

The Web 2.0 Element Filter allows you to Retrieve an HTML element from the response and store it in a property:



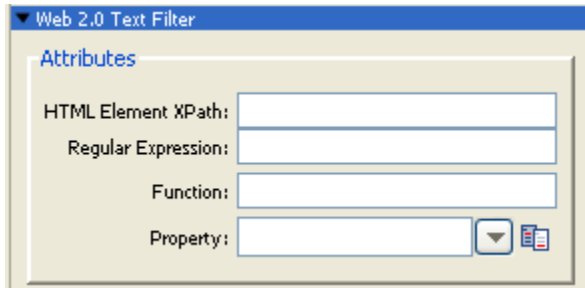
Enter the following parameters:

- HTML Element XPath: the XPath expression that uniquely identifies the DOM element that was the target of the event.
- Function: Name of predefined web2.0 function.
- Property: The name of the property to store the result.

22.2 Web 2.0 Text Filter

22.2 Web 2.0 Text Filter

The Web 2.0 Text Filter allows you to retrieve text from the response using an XPath expression and then a regular expression, and then store the text in a property:



The screenshot shows a configuration window titled "Web 2.0 Text Filter". Inside, there is a section labeled "Attributes" with four input fields: "HTML Element XPath:", "Regular Expression:", "Function:", and "Property:". The "Property:" field has a dropdown arrow and a document icon to its right.

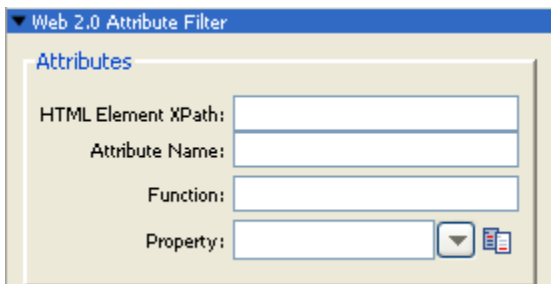
Enter the following parameters:

- HTML Element XPath: the XPath expression that uniquely identifies the DOM element that was the target of the event.
- Regular Expression: a regular expression that is applied to the result of the filter to further control what gets returned.
- Function: Name of predefined web 2.0 function.
- Property: The name of the property to store the result.

22.3 Web 2.0 Attribute Filter

22.3 Web 2.0 Attribute Filter

The Web 2.0 Attribute Filter allows you to retrieve an HTML attribute value from the response and store it in a property:



The screenshot shows a configuration window titled "Web 2.0 Attribute Filter". Inside, there is a section labeled "Attributes" with four input fields: "HTML Element XPath:", "Attribute Name:", "Function:", and "Property:". The "Property:" field has a dropdown arrow and a document icon to its right.

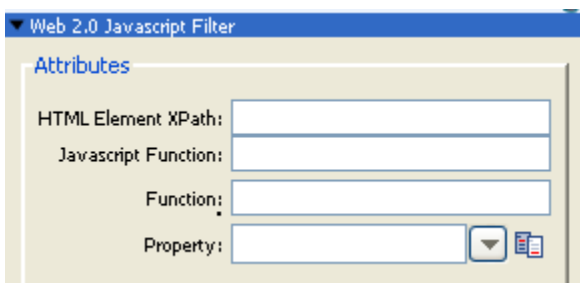
Enter the following parameters:

- HTML Element XPath: the XPath expression that uniquely identifies the DOM element that was the target of the event.
- Attribute Name: the optional DOM attribute name used to execute DOM attribute filters.
- Function: Name of predefined web2.0 function.
- Property: The name of the property to store the last response.

22.4 Web 2.0 Java Script Filter

22.4 Web 2.0 Java Script Filter

The Web 2.0 JavaScript Filter allows you to retrieve arbitrary information from the response using a JavaScript expression.



The screenshot shows a configuration window titled "Web 2.0 Javascript Filter". Inside, there is a section labeled "Attributes" with four input fields: "HTML Element XPath:", "Javascript Function:", "Function:", and "Property:". The "Property:" field has a dropdown arrow and a document icon to its right.

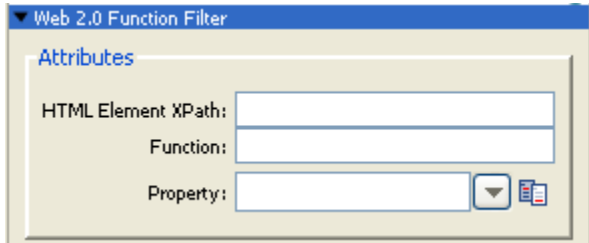
Enter the following parameters:

- HTML Element XPath: the XPath expression that uniquely identifies the DOM element that was the target of the event.
- Javascript Function: a snippet of valid JavaScript code that gets executed by the filter. It should return an object.
- Function: Name of predefined web2.0 function.
- Property: The name of the property to store the result.

22.5 Web 2.0 Function Filter

22.5 Web 2.0 Function Filter

The Web 2.0 Function Filter allows you to execute a predefined web20 function.

The screenshot shows a dialog box titled "Web 2.0 Function Filter". Inside, there is a section labeled "Attributes". It contains three input fields: "HTML Element XPath:" with a text box, "Function:" with a text box, and "Property:" with a text box and a dropdown arrow. To the right of the "Property:" field is a small icon of a document with a red 'X'.

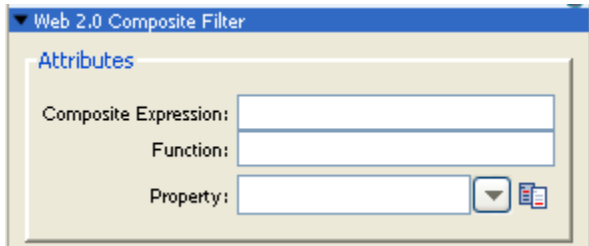
Enter the following parameters:

- HTML Element XPath: the XPath expression that uniquely identifies the DOM element that was the target of the event.
- Function: Name of predefined web2.0 function.
- Property: The name of the property to store the last response.

22.6 Web 2.0 Composite Filter

22.6 Web 2.0 Composite Filter

The Web 2.0 Composite Filter allows you to combine other web 2.0 filters using a string or an arithmetic expression.

The screenshot shows a dialog box titled "Web 2.0 Composite Filter". Inside, there is a section labeled "Attributes". It contains three input fields: "Composite Expression:" with a text box, "Function:" with a text box, and "Property:" with a text box and a dropdown arrow. To the right of the "Property:" field is a small icon of a document with a red 'X'.

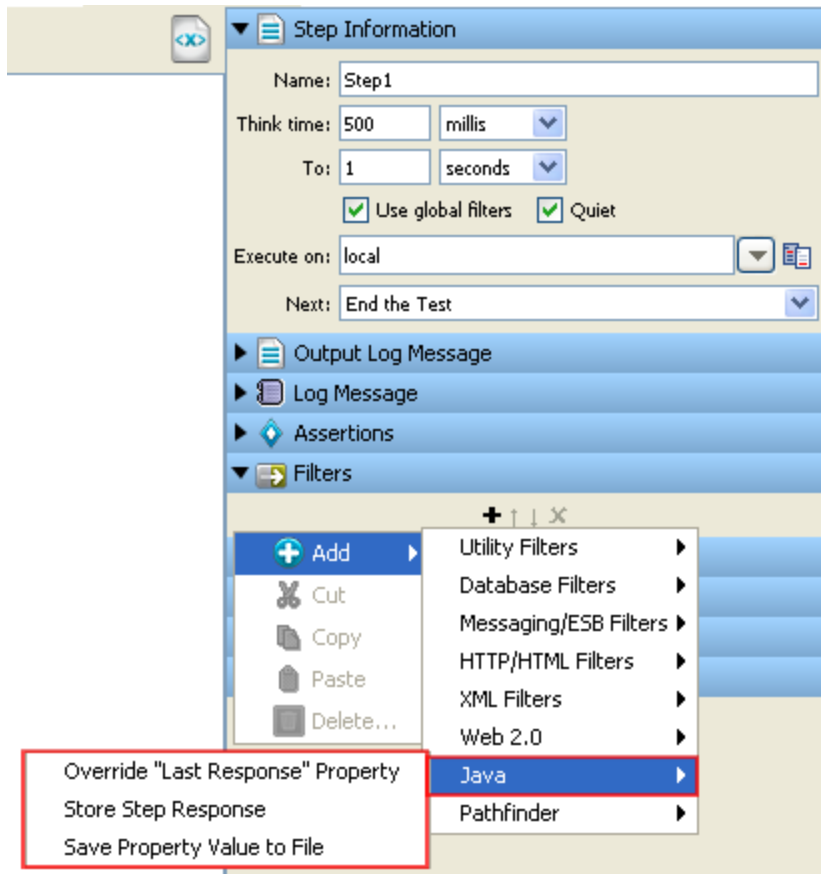
Enter the following parameters:

- Composite Expression: Expression with web 2.0 filters using a string or arithmetic expression.
- Function: Name of predefined web2.0 function.
- Property: The name of the property to store the result.

23. Java Filters

23. Java Filters

These are the filters available in the Java Filters list for any test step:



This list explains these Filters in the order they appear on screen:

- 23.1 Override "Last Response" Property
- 23.2 Store Step Response
- 23.3 Save Property Value to File

23.1 Override "Last Response" Property

23.1 Override "Last Response" Property

(Convert Property Value into Last Response)

This filter allows you to replace the current value of the last response with the value of an existing LISA property.

For the detailed information refer to Utility Filters: [Override Last Response Property](#).

23.2 Store Step Response

23.2 Store Step Response

(Save Step Response as Property)

This filter allows you to save the last response as a property, for future use.

For the detailed information refer to Utility Filters: [Store Step Response](#).

23.3 Save Property Value to File

23.3 Save Property Value to File

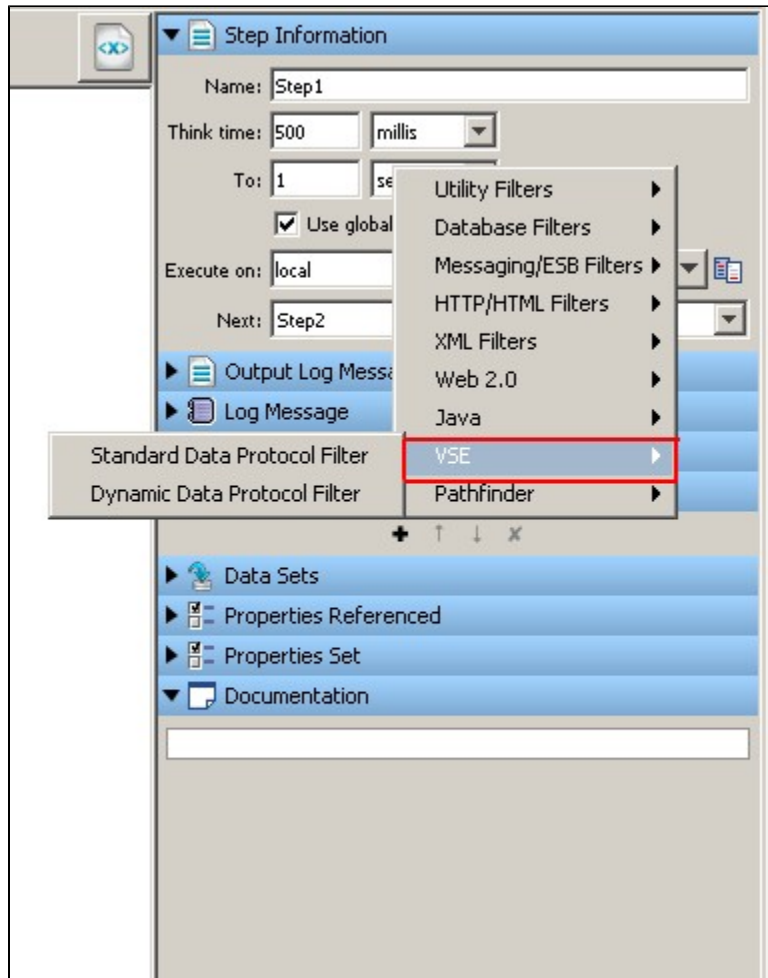
This filter allows you to save the value of an existing property to a file in your file system.

For the detailed information refer to Utility Filters: [Save Property Value to File](#).

24. VSE Filters

24. VSE Filters

These are the filters available in the VSE Filters list for any test step:



This list explains these Filters in the order they appear on screen:

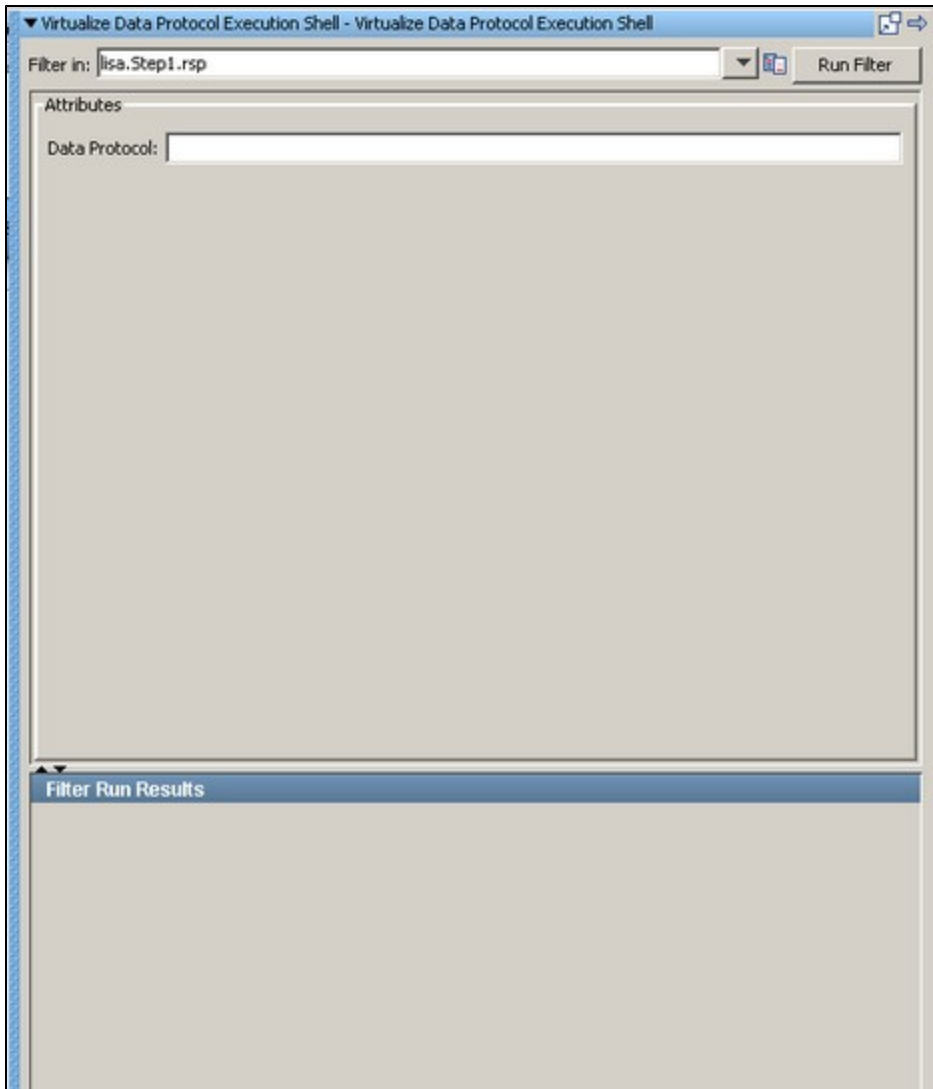
24.1 Standard Data Protocol Filter.

24.2 Dynamic Data Protocol Filter.

24.1 Standard Data Protocol Filter

24.1 Standard Data Protocol Filter

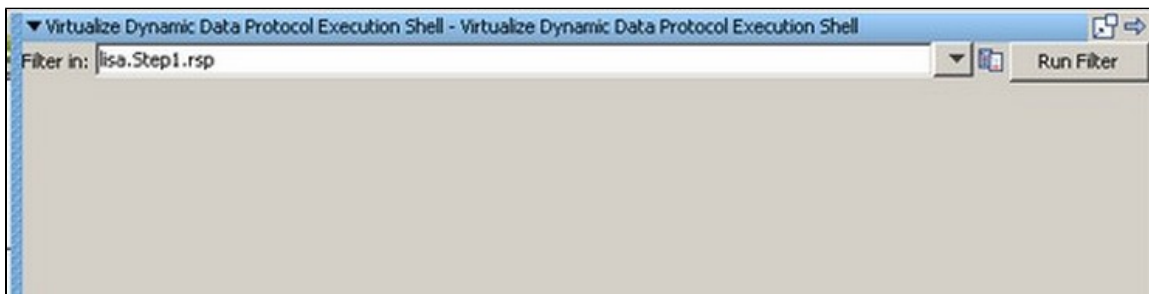
This filter is used on protocol-specific listen steps for virtual models. It provides the necessary wrapper for a data protocol to act as a filter, the appropriate way for things to work in the runtime side of VSE.



24.2 Dynamic Data Protocol Filter

24.2 Dynamic Data Protocol Filter

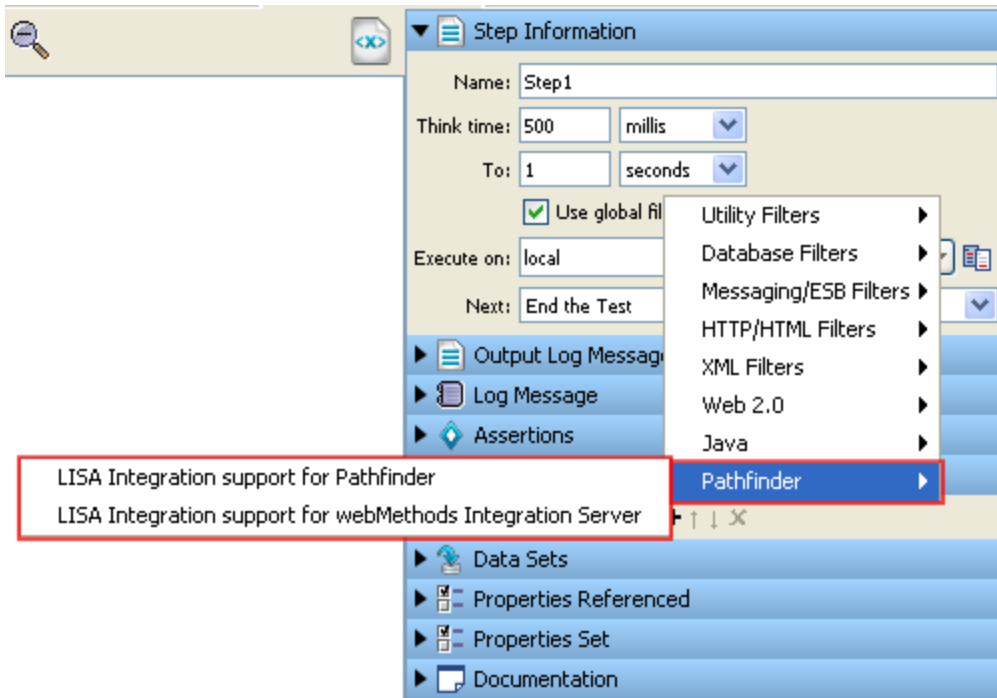
This filter is used on protocol-specific listen steps for virtual models. It provides the necessary wrapper for a data protocol to act as a filter, the appropriate way for things to work in the runtime side of VSE.



25. Pathfinder Filters

25. Pathfinder Filters

These are the filters available in the Pathfinder Filters list for any test step:



The following topics are available in this chapter.

- [24.1 LISA Integration Support for Pathfinder](#)
- [24.2 LISA Integration Support for webMethods Integration Server](#)

24.1 LISA Integration Support for Pathfinder

24.1 LISA Integration Support for Pathfinder

This is a common filter to enable pathfinder for all the technologies supported by LISA.

This filter collects additional information from a pathfinder application.

Presently LISA supports integration with web services, JMS, servlets, EJB & Java objects.

▼ LISA Integration support - LISA Integration support

☒ Send a LISA integration header (enable integration)

Error if Max Build Time (millis) Exceeds: 0

On Transaction Error Step: Fail the Test

On Pathfinder Warnings Step: Generate Error

☒ Report Component Content

☐ Force a Garbage Collection on the server at the start & end of...

☒ Fail test if server-side exception is logged

Log4J Level to capture in test events: INFO

Log4J Logger to temporarily change (blank is Root Logger):

Enter the following parameters:

Error if Max Build Time(millis) Exceeds : Enter build time in milliseconds. If it exceeds the time specified error will be generated.

On Transaction Error Step : Select the step to redirect to on Transaction error after filter is set to run .

On Pathfinder Warning Step : Select the step to redirect to on Pathfinder Warning Step after filter is set to run .

Report Component Content checkbox : Generate a report of the component content.

Force a Garbage Collection on the server at the start & end of the request checkbox: forces a Garbage collection on the server at the start & end of the request.

Fail test if server-side exception is logged checkbox: Fail the test case if exception is thrown at the server side.

Log4J level to capture in the test events : Select the log4J level that is to be captured in the test events.

Log4J Logger to temporarily change(blank is Root Logger) : Enter the name of the Logger.

24.2 LISA Integration Support for webMethods Integration Server

24.2 LISA Integration Support for webMethods Integration Server

This filter collects additional information from pathfinder-enabled web methods integration server

Attributes

Error if Max Build Time (millis) Exceeds: 0

On Transaction Error Step: Fail the Test

On Pathfinder Warnings Step: Generate Error

Enter the following parameters:

Error if Max Build Time(millis) Exceeds : Enter build time in milliseconds. If it exceeds the time specified error will be generated.

On Transaction Error Step : Select the step to redirect to on Transaction error after filter is set to run .

On Pathfinder Warning Step : Select the step to redirect to on Pathfinder Warning Step after filter is set to run .

PART 4 - Assertions

PART 4 - Assertions

This chapter describes each of the Assertions that are available in LISA. In this Reference Guide we assume a general understanding of the concepts and uses of assertions in test cases.

Regular expressions are used for comparison purposes in many assertions. For more information on regular expressions, visit <http://java.sun.com/docs/books/tutorial/essential/regex/>.

25. HTTP Assertions

- 25.1 Highlight HTML Content for Comparison
- 25.2 Check HTML for Properties in page
- 25.3 Ensure HTTP Header Contains Expression
- 25.4 Check HTTP Response Code
- 25.5 Simple Web Assertion
- 25.6 Check Links on Web Responses

26. Database Assertions

- 26.1 Ensure Result Set Size
- 26.2 Ensure Result Set Contains Expression

27. Web 2.0 Assertions

- 27.1 Web 2.0 Basic Assertion
- 27.2 Web 2.0 Validation Assertion
- 27.3 Web 2.0 Branching Assertion

28. XML Assertions

- 28.1 Highlight Text Content for Comparison
- 28.2 Ensure Result Contains String
- 28.3 Ensure Step Response Time
- 28.4 Graphical XML Side-by-Side Comparison
- 28.5 XML Side-by-Side Comparison
- 28.6 XML XPath Assertion
- 28.7 Ensure XML Validation

29. Virtual Service Environment Assertions

- 29.1 Assert on Execution Mode

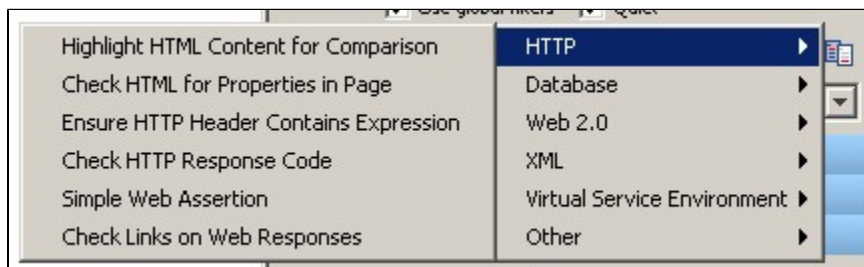
30. Other Assertions

- 30.1 Highlight Text Content for Comparison
- 30.2 Ensure Non-Empty Result
- 30.3 Ensure Result Contains String
- 30.4 Ensure Result Contains Expression
- 30.5 Ensure Property Matches Expression
- 30.6 Ensure Step Response Time
- 30.7 Scripted Assertion
- 30.8 Ensure Properties Equal
- 30.9 Assert on Invocation Exception
- 30.10 File Watcher Assertion
- 30.11 Check Content of Collection Object
- 30.12 WS-I Basic Profile 1.1 Assertion
- 30.13 Messaging VSE Workflow Assertion

25. HTTP Assertions

25. HTTP Assertions

These are the assertions available in the HTTP assertions list for any test step:



This list explains these assertions in the order they appear on screen:

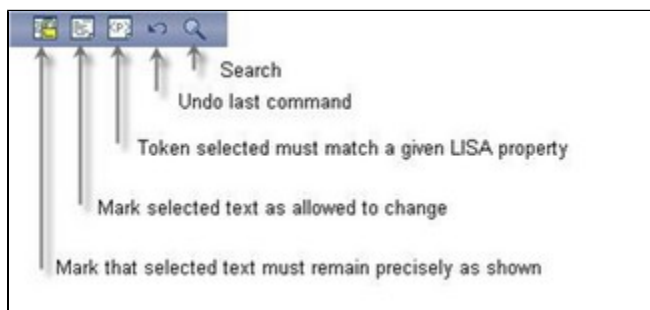
25.1 Highlight HTML Content for Comparison
25.2 Check HTML for Properties in page
25.3 Ensure HTTP Header Contains Expression
25.4 Check HTTP Response Code
25.5 Simple Web Assertion
25.6 Check Links on Web Responses

25.1 Highlight HTML Content for Comparison

25.1 Highlight HTML Content for Comparison

The Web HTML Text assertion allows you to make a comparison based on the contents on an HTML page. This assertion use "paint the screen" technique specifically designed to work with HTML pages. For example, if there is a large HTML document, then the user identifies the data before and after the "content of interest". Then after user simply identify what the "content of interest" will be compared against (usually this would be an expected value supplied in a dataset).

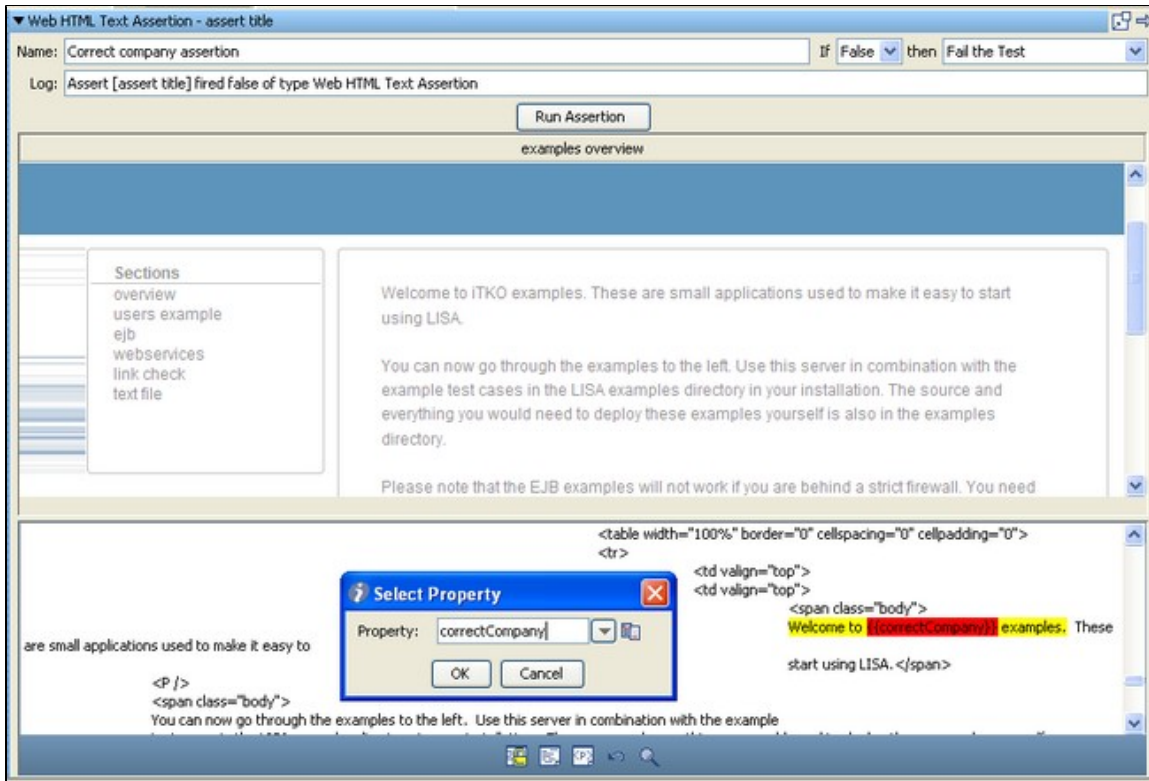
The text is marked using the icons at the bottom of the editor:



This technique is best explained by example.

In the below example, see the figure below, we want to make sure that the company name, currently iTKO, that appears in the phrase "Welcome to iTKO examples" matches the value in a specified LISA property. We have marked the text, using the three icons shown above, by selecting text and then clicking on the appropriate icon.

- Yellow background indicates text that must appear as shown.
- White background indicates text that need not be present, or can change.
- Red background identifies the text that must match the property entered into the dialog box.



This screen shows the HTML rendered in a browser in the top panel, and the actual HTML text in the bottom panel. We want to make the phrases "Welcome to" and "examples" required. We have set the boundaries around that, and clicked the "Must" icon, . Then we selected the company name text, "iTKO", inside the highlighted content, and clicked the "Property" icon, . We entered the property name, correctCompany, into the dialog box. This property will be compared to the text that appears between the two bounding phrases. Note that the company name text has been replaced with the name of the LISA property.

Click the Run Assertion button to execute an assertion.

Now when this assertion is run, the value of the property "correctCompany" will be inserted between the phases "Welcome to" and "examples" and the resulting phase will be compared to the corresponding phase in the HTML response. Note that the phase "Welcome to correctCompany examples" can change location in the HTML and it will still be located.

This was a simple example.

25.2 Check HTML for Properties in page

25.2 Check HTML for Properties in page

(Assert on Properties from HTML Result)

The Assert on Properties from HTML Result assertion is useful for Web testing when there is property data in the Web page that might be used for the assertion. LISA makes the property data available for assertion by parsing the Web page for meta tags, title tags, hidden form fields, and other tags that LISA can automatically parse, including <lisaprop> tags and the LISA Integration API.

Here is a sample of the available properties table:

▼ Assert on Properties from HTML Result - Assert718

Name: If then

Log:

Properties Referenced on Page

Property	Observed Value	Expression
list_user_22	ramesh/ramesh	
itko_examples_page	user manage	
title	list users	
list_prop_command	/list	
list_user_11	multi/test	
list_user_10	{{unique_user}}/pwd	
list_size	29	
list_user_15	John/Doe	
list_user_14	fred/flintstone	
list_user_13	multitier-1093520423/example-pwd	
list_user_12	WSExampleUser-1784496840/userPass	
list_user_19	doc3/writer3	
list_user_18	tracy/password	
list_user_17	md/password	
list_user_16	integrator-953936408/example-integrator	
list_user_7	kris/testpass	

Find:

Enter the following parameters:

- **Name:** Enter the name of the assertion.
- **If:** Select the behavior of the assertion using the dropdown box.
- **then:** Select the step to redirect to if the assertion fires.
- **Log:** The text that will be printed out as event text if the assertion fired.

Click the **Run Assertion** button to execute an assertion.

Note: LISA may prompt you to install the **LISA tag filter**.

25.3 Ensure HTTP Header Contains Expression

25.3 Ensure HTTP Header Contains Expression

(HTTP Result Header Field)

The HTTP Result Header Field assertion allows you to check that a specific HTTP result header contains a field that matches a specified regular expression.

Name:

If then

Log:

Header Field:

RegExpression:

Enter the following parameters:

- **Name:** The name of this assertion. This will help you identify events for this assertion.
- **If:** Select the behavior of the assertion using the dropdown box.
- **then:** Select the step to redirect to if the assertion fires.
- **Log:** The text that will be printed out as event text if the assertion fired.

- **Header Field:** The name of the header field.
- **RegularExpression:** The regular expression that must appear in the header field.

Click the **Run Assertion** button to execute an assertion.

25.4 Check HTTP Response Code

25.4 Check HTTP Response Code

The HTTP Response Code assertion allows you to check that the HTTP response code matches a specified regular expression.

Name: FourHundredRange

If: False

then: Fail the Test

Log:

Run Assertion

RegularExpression: 4\d\d

Enter the following parameters:

- **Name:** The name of this assertion. This will help you identify events for this assertion.
- **If:** Select the behavior of the assertion using the dropdown box.
- **then:** Select the step to redirect to if the assertion fires.
- **Log:** The text that will be printed out as event text if the assertion fired.
- **RegularExpression:** the regular expression that must appear in the response code. For example, to check that the HTTP response code is in the 400-499 range, set the RegularExpression to 4\d\d.

Click the **Run Assertion** button to execute an assertion.

25.5 Simple Web Assertion

25.5 Simple Web Assertion

This assertion reads the return code from the web application.

If the application returns code 404 (page not found), 500 (server error) or any other error then this assertion returns true.

Assert203

Name: Assert203

If: True

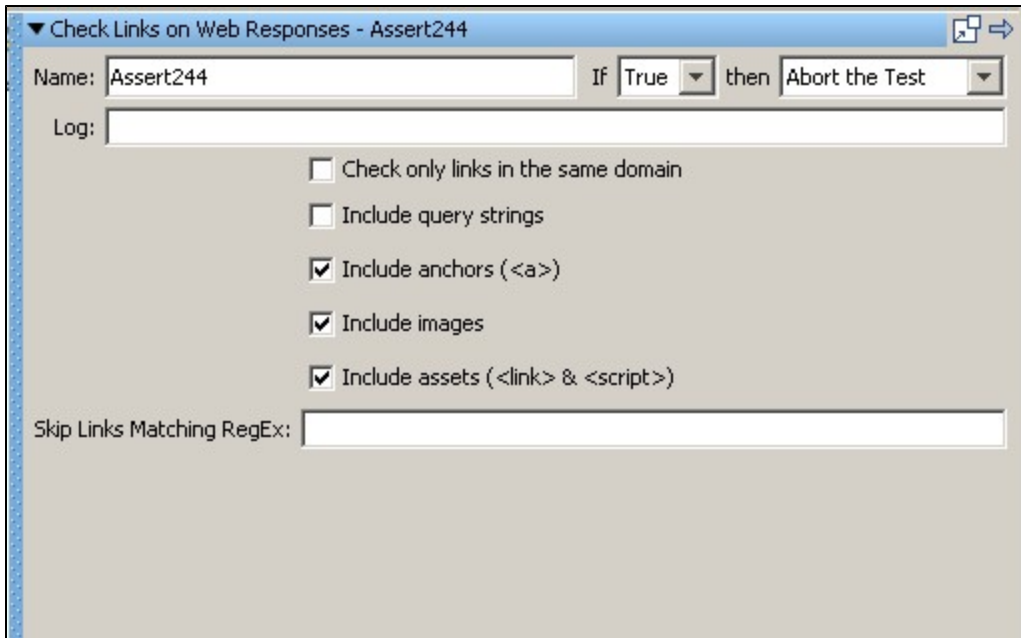
then: Abort the Test

Log:

25.6 Check Links on Web Responses

25.6 Check Links on Web Responses

This assertion checks every link on the returned web page to make sure that it contains a valid page & doesn't return a HTTP error like 404 error etc. This is commonly used to make sure that the links are working properly across the application & there are no inactive links on the page.

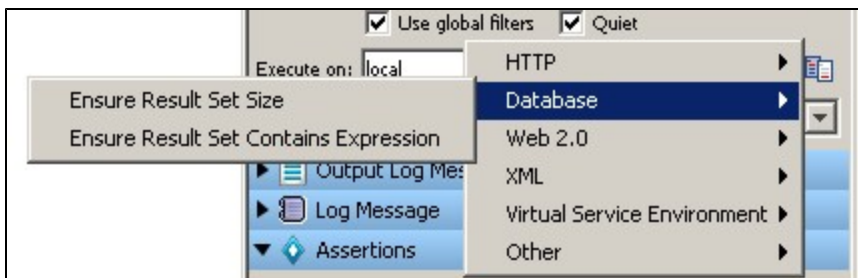


- **Check only links in the same domain** : Checks only links in the current domain of the returned web page.
- **Include query strings** : If any query strings are present on the returned web page then those are checked.
- **Include anchors(<a>)** : Any anchor links in the current web page are checked.
- **Include images** : All the images on the returned web page are checked.
- **Include assets(<link> & <script>)** : Current webpage is checked for script & links.

26. Database Assertions

26. Database Assertions

These are the assertions available in the Database assertions list for any test step:



This list explains these assertions in the order they appear on screen:

- 26.1 Ensure Result Set Size
- 26.2 Ensure Result Set Contains Expression

26.1 Ensure Result Set Size

26.1 Ensure Result Set Size

This assertion will count the number of rows in a Result Set and ensure that the size falls between an upper and lower value supplied by the user.

An example of this assertion could be checking to make sure number of rows in an HTML table matches a supplied expected value from a data set.

Enter the following parameters:

- **Name:** The name of this assertion. This will help you identify events for this assertion.
- **If:** Select the behavior of the assertion using the dropdown box.
- **then:** Select the step to redirect to if the assertion fires.
- **Log:** The text that will be printed out as event text if the assertion fired.
- **Result set has warnings:** If checked, the database may return warnings in the result set. Check with your system administrator to determine whether your database supports warnings in the result set.
- **Row Count >=:** The minimum number of rows in the result set. -1 indicates no minimum.
- **Row Count <=:** The maximum number of rows in the result set. -1 indicates no maximum.

Click the **Run Assertion** button to execute an assertion.

For example, to make sure that a Database Assertion step returns one and only one row, set the **Row Count At Least** field to 1 and the **Row Count No More Than** field to 1.

26.2 Ensure Result Set Contains Expression

26.2 Ensure Result Set Contains Expression

This assertion will check a particular column in a Result Set and ensure that the supplied expression matches at least one value.

Enter the following parameters:

- **Name:** The name of this assertion. This will help you identify events for this assertion.
- **If:** Select the behavior of the assertion using the drop down box.
- **then:** Select the step to redirect to if the assertion fires.
- **Log:** The text that will be printed out as event text if the assertion fired.
- **Column:** The column that contains the text to check. This can be a column name or an index.
- **Regular Expression:** The regular expression to match in the column.

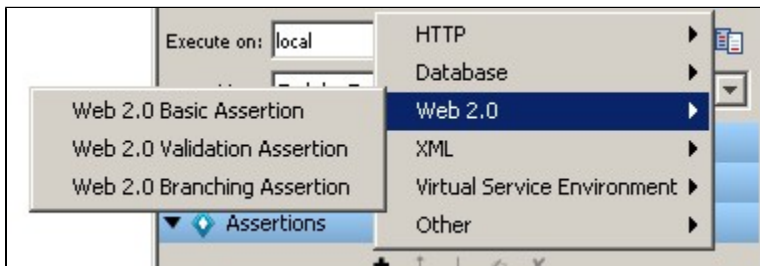
Click the **Run Assertion** button to execute an assertion.

For example, to check that at least one of the rows returned from a query has a login value that starts with **wp**, set the **Column** field to **login** and the **RegExpression** field to **wp.***.

27. Web 2.0 Assertions

27. Web 2.0 Assertions

These are the assertions available in the web2.0 assertions list for any test step:



This list explains these assertions in the order they appear on screen:

[27.1 Web 2.0 Basic Assertion](#)
[27.2 Web 2.0 Validation Assertion](#)
[27.3 Web 2.0 Branching Assertion](#)

27.1 Web 2.0 Basic Assertion

27.1 Web 2.0 Basic Assertion

(Web 2.0 Assertions)

Web 2.0 assertions are intended to be created from the DOM web browser. The basic assertion provides the ability to compare properties, for example comparing a bank balance before and after making a deposit. For instance the basic assertion could be used to make sure the `{{after}}EQUALS{{before + deposit}}`.

The Web 2.0 Basic Assertion evaluates a unary or binary expression using predefined operators and properties. Typically, you will use a variable created by a filter on the left side and then equal it or match it to a constant value or another variable on the right side. In the example of the previous section with the table rows count, you would use something like "£3 Equals 36".

A screenshot of a configuration form for a 'Web 2.0 Basic Assertion'. The form has several fields: 'Name' (set to 'Assert285'), 'If' (set to 'False'), 'then' (set to 'Fail the Test'), 'Log' (empty), 'And Assertion' (checkbox), 'Left Operand' (empty), 'Operator' (set to 'Equals'), and 'Right Operand' (empty). Each field has a dropdown arrow on the right side.

Enter the following parameters:

- **If:** Select the behavior of the assertion using the drop down box.
- **then:** Select the step to redirect to if the assertion fires.
- **Log:** The text that will be printed out as event text if the assertion fired.
- **And Assertion :** (TODO)
- **Left Operand:** Left side of the expression for comparison.
- **Operator:** One of the following operators:
 - **Equals:** those compare the 2 sides of the expression for equality.
 - **Not Equals:** those compare the 2 sides of the expression for non-equality.
 - **Matches:** those attempt to match (as regular expression) the left or the right side of the expression against the other side.
 - **Not Matches:** those attempt not to match (as regular expression) the left or the right side of the expression against the other side.
 - **Less Than:** those compare the 2 sides of the expression as numeric values for order. Returns false on non-numeric values.
 - **More Than:** those compare the 2 sides of the expression as numeric values for order. Returns false on non-numeric values.
 - **Exists:** those verify if the entity represented by the left side exists in the current page context.
 - **Evaluate:** those evaluate arbitrary JavaScript code in the context of the current page. Must return a boolean.

- **Right Operand:** Right side of the expression for comparison, or blank if a unary comparison.

27.2 Web 2.0 Validation Assertion

27.2 Web 2.0 Validation Assertion

Web 2.0 assertions are intended to be created from the DOM web browser. This assertion will validate the entire HTML result to ensure it complies with W3C standards. This is important in web 2.0 applications because there can be many tools to generate the page, so it is good to ensure it is standards compliant.

The Validate the HTML page assertion validates the HTML page for errors, warnings or broken links.

The screenshot shows a configuration window titled "Web 2.0 Validation Assertion - Assert326". It has a "Name" field with the value "Assert326". To the right of the name field is a conditional logic section: "If False" (with a dropdown arrow) "then Fail the Test" (with a dropdown arrow). Below the name field is a "Log:" label followed by an empty text box. Underneath the log field are five checkboxes for validation options:

- ☐ And Assertion
- ☐ Validate HTML against W3C Errors
- ☐ Validate HTML against W3C Warnings
- ☐ Validate HTML scripts, css, inputs and images
- ☒ Validate HTML outgoing references (links)

Select one or more of the following validations:

- **And Assertion:** (TODO).
- **Validate HTML against W3C Errors:** HTML page is evaluated for W3C errors.
- **Validate HTML against W3C Warnings:** HTML page is evaluated for W3C warnings.
- **Validate HTML scripts, css, input and images:** Page is evaluated for images, css scripts and inputs.
- **Validate HTML outgoing references (links):** HTML Page is evaluated for valid links.

27.3 Web 2.0 Branching Assertion

27.3 Web 2.0 Branching Assertion

Web 2.0 Branching assertion is created from the DOM web browser.

The screenshot shows a configuration window titled "Web 2.0 Validation Assertion - Assert206". It has a "Name" field with the value "Assert206". To the right of the name field is a conditional logic section: "If True" (with a dropdown arrow) "then Fail the Test" (with a dropdown arrow). Below the name field is a "Log:" label followed by an empty text box. Underneath the log field are five checkboxes for validation options:

- ☐ And Assertion
- ☐ Validate HTML against W3C Errors
- ☐ Validate HTML against W3C Warnings
- ☐ Validate HTML scripts, css, inputs and images
- ☐ Validate HTML outgoing references (links)

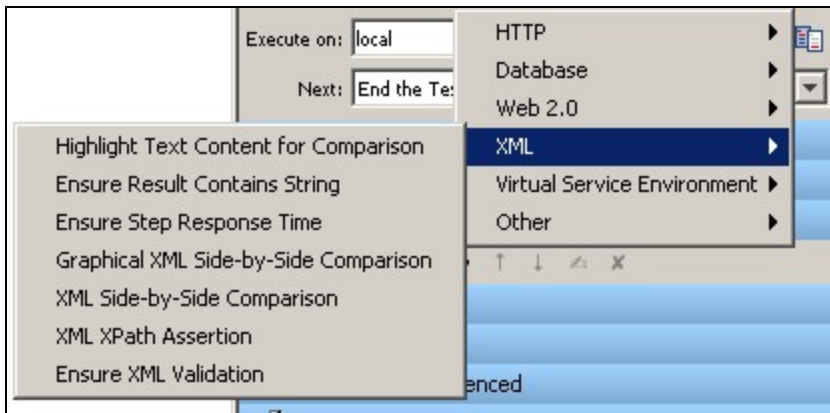
Select one or more of the following validations:

- **And Assertion:** (TODO).
- **Validate HTML against W3C Errors:** HTML page is evaluated for W3C errors.
- **Validate HTML against W3C Warnings:** HTML page is evaluated for W3C warnings.
- **Validate HTML scripts, css, input and images:** Page is evaluated for images, css scripts and inputs.
- **Validate HTML outgoing references (links):** HTML Page is evaluated for valid links.

28. XML Assertions

28. XML Assertions

These are the assertions available in the XML assertions list for any test step:



This list explains these assertions in the order they appear on screen:

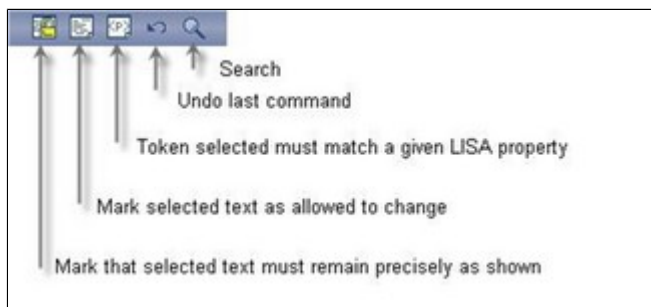
- 28.1 Highlight Text Content for Comparison
- 28.2 Ensure Result Contains String
- 28.3 Ensure Step Response Time
- 29.4 Graphical XML Side-by-Side Comparison
- 29.5 XML Side-by-Side Comparison
- 28.6 XML XPath Assertion
- 28.7 Ensure XML Validation

28.1 Highlight Text Content for Comparison

28.1 Highlight Text Content for Comparison

This assertion uses the "paint the screen" technique specifically designed to work with HTML pages. For example, if there is a large HTML document, then the user identifies the data before and after the "content of interest". Then after user simply identify what the "content of interest" will be compared against (usually this would be an expected value supplied in a dataset).

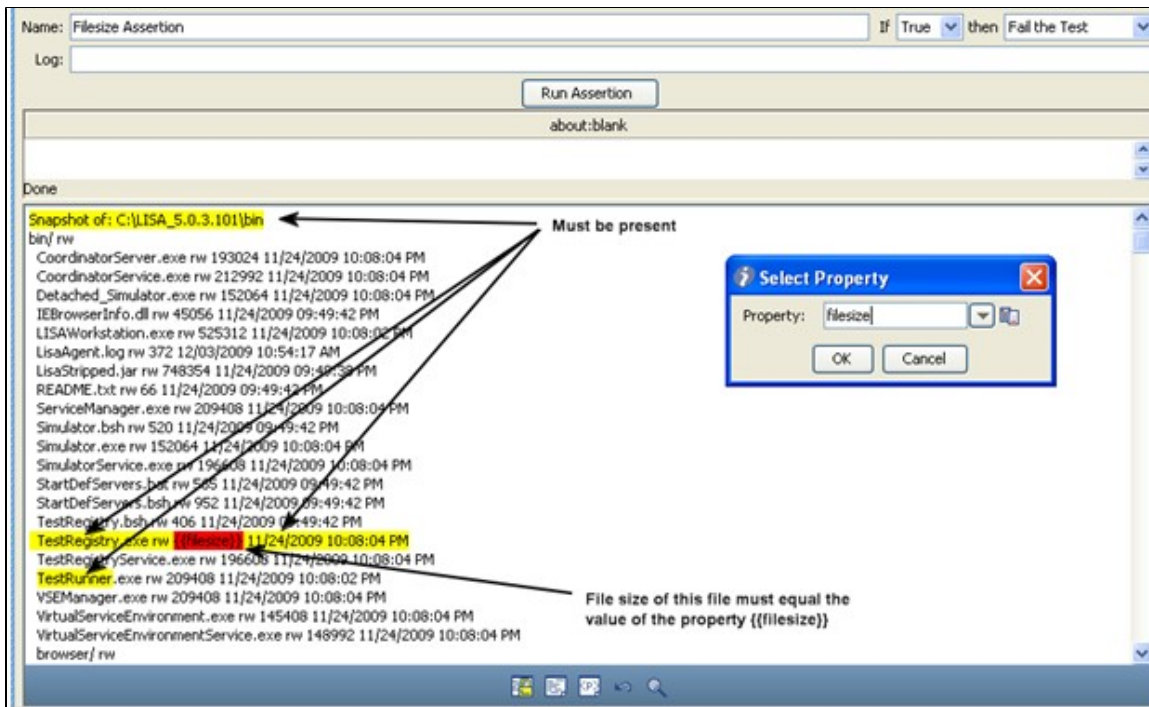
The text is marked using the icons at the bottom of the editor:



This technique is best explained by example.

In the following example, see the figure below, we want to make sure that certain files appear in the buffer, and one of the file sizes needs to be compared to the value of a property. We have marked the text using the 3 icons shown above, by selecting text and then clicking on the appropriate icon.

- Yellow background indicates text that must appear as shown.
- White background indicates text that need not be present, or can change.
- Red background identifies the text that must match the property entered into the dialog box.



The set of tokens shown above can be read this way:

- The buffer must start with the phrase in yellow: "Snapshot of: C:\LISA_5.0.3.101\bin".
- There are a number of files that may or may not be in the buffer in the next token, but since it is an "Any" token the variance is immaterial.
- The file "TestRegistry.exe" and "rw" attributes must appear.
- The red `filesize` means that the value associated with the property key "filesize" will be swapped into the expression, then the comparison made.
- The text "11/24/2009" must appear.
- The file "TestRunner.exe" must appear.
- The buffer can have any amount of content afterward.

After you have finished the markup, enter the following parameters:

- Name: The name of this assertion. This will help you identify events for this assertion.
- If: Select the behavior of the assertion using the drop down box.
- then: Select the step to redirect to if the assertion fires.
- Log: The text that will be printed out as event text if the assertion fired.

Click the Run Assertion button to execute an assertion.

Note: Property blocks must always be bounded by Must blocks.

28.2 Ensure Result Contains String

28.2 Ensure Result Contains String

(Result as String Contains Given String)

The Result as String Contains Given String assertion allows you to search the response (as text) for a string.

Enter the following parameters:

- **Name:** The name of this assertion. This will help you identify events for this assertion.
- **If:** Select the behavior of the assertion using the drop down box.
- **then:** Select the step to redirect to if the assertion fires.
- **Log:** The text that will be printed out as event text if the assertion fired.
- **Contains String:** The string to search for in the step result – this can contain a property.

28.3 Ensure Step Response Time

28.3 Ensure Step Response Time

(Assert Step Response Time Thresholds)

The Assert Step Response Time Thresholds assertion allows you to define upper and lower bounds on the response time and assert that the response time is within those bounds.

Enter the following parameters:

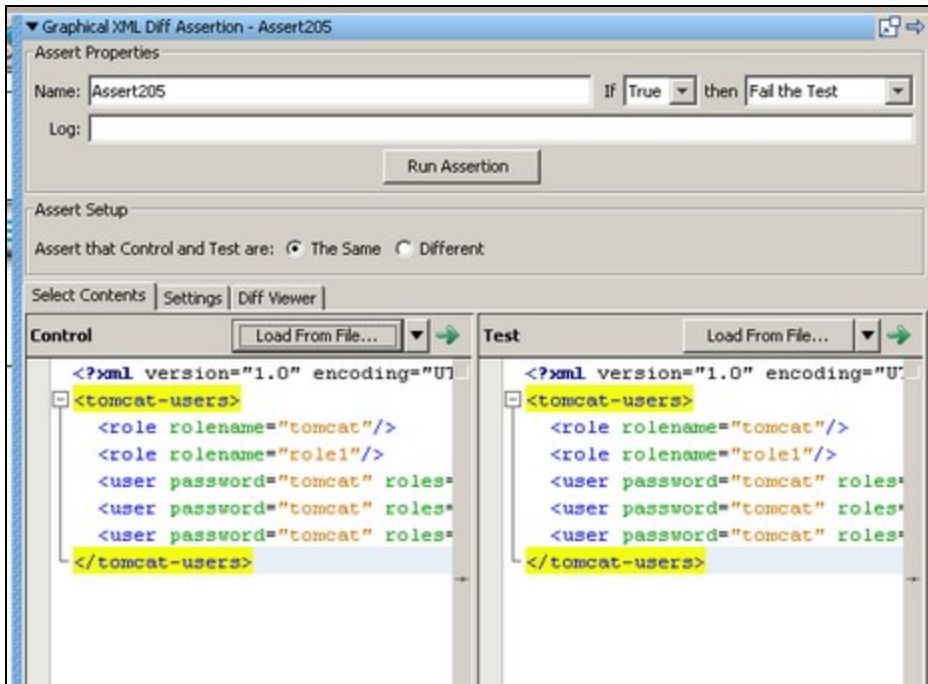
- **Name:** The name of this assertion. This will help you identify events for this assertion.
- **If:** Select the behavior of the assertion using the dropdown box.
- **then:** Select the step to redirect to if the assertion fires.
- **Log:** The text printed out as event text if the assertion fired.
- **Time must be at least (millis):** Enter the lower bound in milliseconds.
- **Time must not be more than (millis):** Enter the upper bound in milliseconds. This is ignored if set to -1.

Note: Parameters can contain properties.

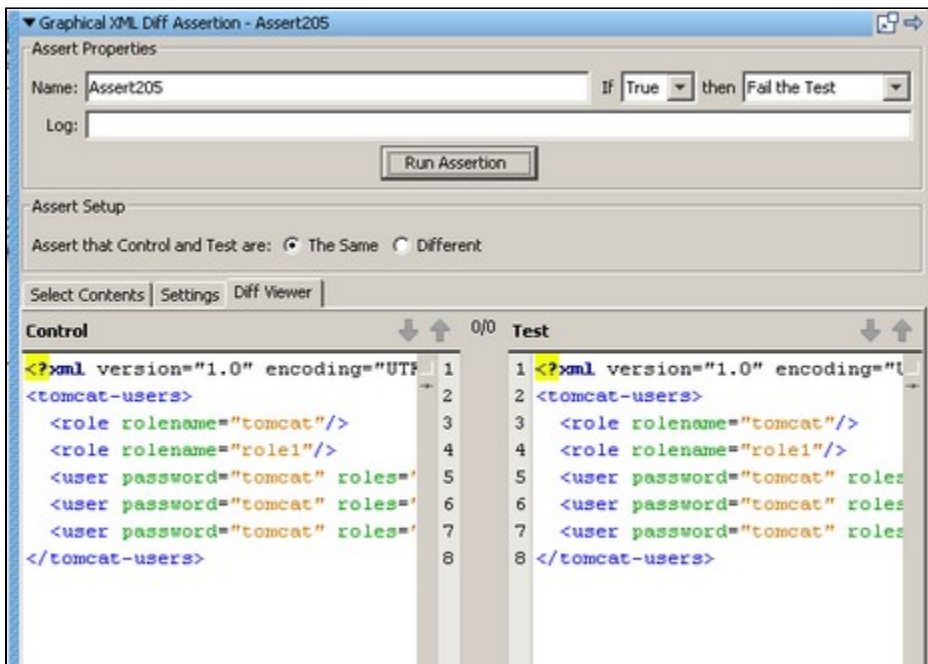
28.4 Graphical XML Side-by-Side Comparison

28.4 Graphical XML Side-by-Side Comparison

This assertion allows to compare a test XML value received from a test with a control XML value. The assertion can return true if the responses are the same or different. This provides a flexible ability to compare XML documents at various steps in a business process to make sure they match expected criteria. This is known as "exclusive" testing where an entire response is compared except for a few values known to change.



In the example above, the comparison was completed and the XML files were found similar.



28.5 XML Side-by-Side Comparison

28.5 XML Side-by-Side Comparison

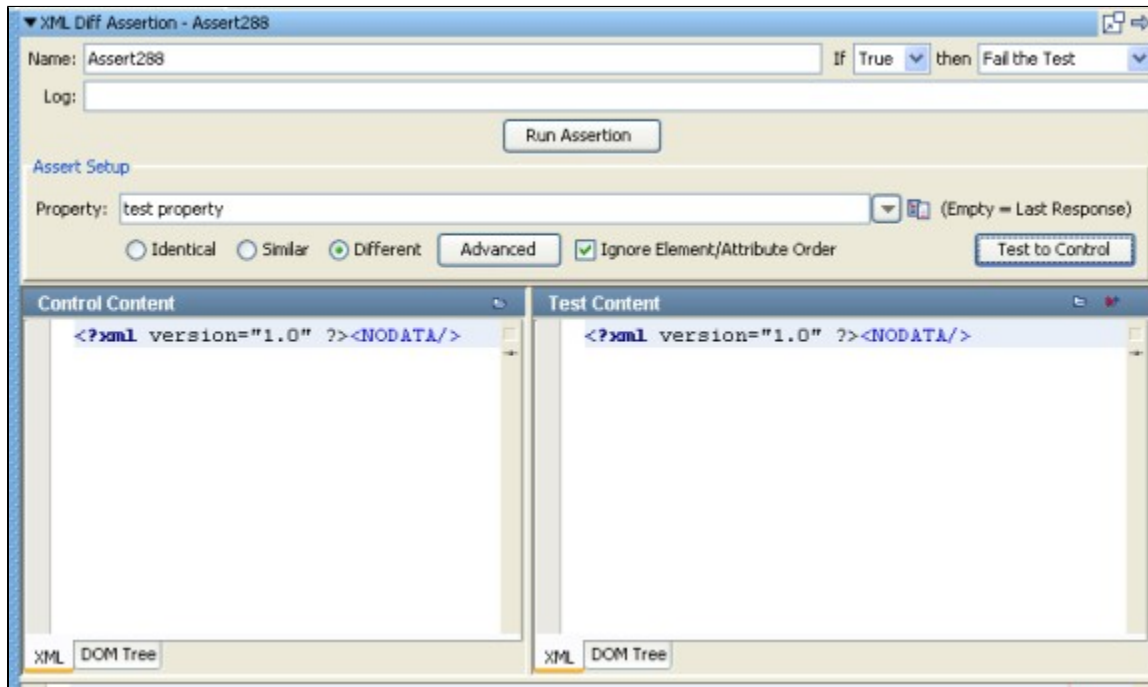
The XML Diff assertion allows you to compare the XML in a response, or the XML saved in a property to an XML control document. Since this is an XML comparison, you can choose the strictness of the comparison: identical, similar or different:

Identical: The content and sequence of the nodes in the documents are exactly the same.

Similar: The content of the nodes in the documents are the same, but minor differences exist, for example, sequencing of sibling elements, values of namespace prefixes, use of implied attribute values.

Different: the contents of the documents are fundamentally different.

You can also choose to exclude XML elements from the comparison by providing an **Ignore List**.



If comparison is made for Different then after comparison is made the XML files were found to be different and the result is displayed in the bottom panel.

Enter the following parameters:

Name: The name of this assertion. This will help you identify events for this assertion.

Log: The text that will be printed out as event text if the assertion fired.

Assert: Select the behavior of the assertion using the radio buttons.

Execute: Select the step to redirect to if the assertion fires.

Property: The property that contains the XML. If this is left blank, the last response is used.

Identical, Similar and Different radio buttons: Choose the strictness level of the comparison. See the definitions above.

Advanced button: Displays the Ignore List window that allows you to add the XML elements that you want excluded from the comparison.

Ignore Element/Attribute Order: check if you want to ignore element and attribute order in the comparison.

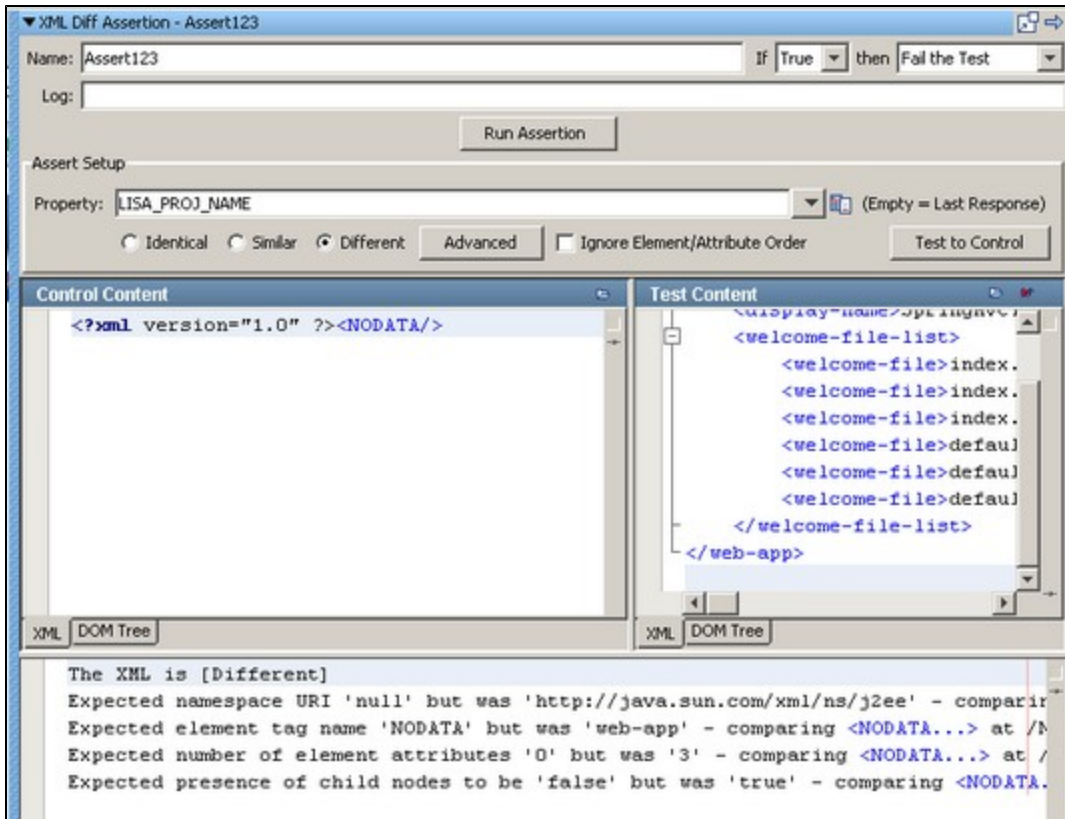
The Control Content panel contains the XML that will be compared to our xml response. The XML content can be typed in, copied from a file, or it can be constructed using copy/paste from the Test content panel.

The Test Content panel displays the last response XML or the XML from the chosen property.

The DOM tree view shows the DOM representations of the XML. (These are read only displays).

Note: Properties can be used in both the Test and Control Content areas.

After the Control Content has been finalized, a test comparison can be made using the **Compare** button (shown by the cursor in the top right corner in the figure below).



In the example above, the comparison was completed and the XML files were found different.

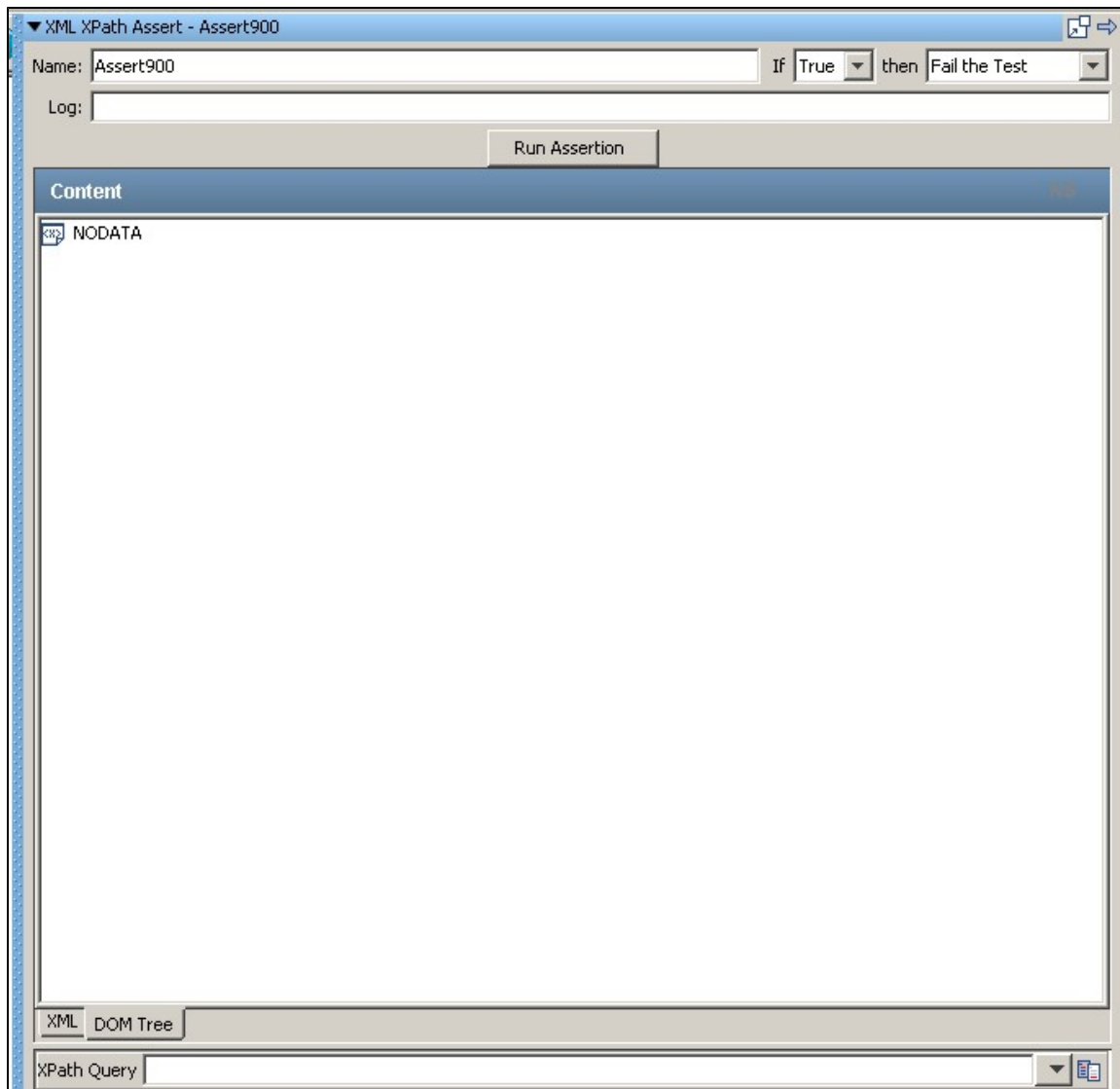
28.6 XML XPath Assertion

29.6 XML XPath Assertion

The XML XPath assertion allows you to use an XPath query that will be run on the response. When this assertion is selected, the last response is loaded into the content panel.

XPath can be thought of as the 'SQL for XML'. It is a very powerful query language that makes parsing XML simple. XPath assertions are very useful when you need to validate a web service response in a more sophisticated way than simply parsing the entire result for a given string. For example you might want to make sure the second and third order item contains 'TKO' and the value of those line items is > 10.

The response can be viewed as an XML document or as a DOM Tree. However, the XPath selection can only be made from the DOM Tree view.



There are three ways to construct the XPath Query:

- Manually enter the XPath expression in the XPath Query text box.
- Select an element from the DOM tree and let LISA construct the XPath expression.
- Select an element from the DOM tree, and then edit the XPath that LISA constructs. For example, you may want to modify it to use a LISA property, or a counter data set.

Enter the following parameters:

- Name: The name of this assertion. This will help you identify events for this assertion.
- Log: The text that will be printed out as event text if the assertion fired.
- Assert: Select the behavior of the assertion using the radio buttons.
- Execute: Select the step to redirect to if the assertion fires.

Now construct the XPath query using one of the methods described above.

After an XPath query has been constructed, test it by clicking the **Run XPath Query** button (shown by the arrow in the figure below) on the right side of the XPath Query panel:

The results of the query are displayed in the Query Results panel.

The example above uses the fourth occurrence of the **<wsdl:part>** tag.

It is common to select an XPath node in a web service result and compare the node to a text value to quickly assert that a response contains the desired value. For this common use case, you can add an equality operator to the end of the initial expression that LISA provides. For example, If we select the new password returned in the response (**BobPass**):

LISA builds the following XPath expression:

`string(/env:Envelope/env:Body/ns2:updatePasswordResponse/[name()='return']/[name()='pwd'])=`

If we add `=NewPassword`, we are comparing the string of the result to the value of the LISA property we used to set the new password. If the equality test does not match, the assertion fails.

So the entire XPath expression becomes:

`string(/env:Envelope/env:Body/ns2:updatePasswordResponse/[name()='return']/[name()='pwd'])=NewPassword`

Here we are checking the web service response, looking for the new password, and making sure it matches what we think it should match.

28.7 Ensure XML Validation

28.7 Ensure XML Validation

(XML Validation)

The XML Validation assertion allows you to validate an XML document. You can check to see if the XML document is well-formed, you can validate against a DTD, or against one or more schemas. If you have an XML fragment you can choose to have LISA add the XML declaration tag. You can also specify that warnings should be treated as errors. You enter the XML to validate as a property.

▼ XML Validation - Assert121

Name: WSDL validation multi-tier If False then Fail the Test

Log:

Run Assertion

Source (empty for default):

Validate:

- ☒ Well Formed XML
- ☒ DTD Conformance
- ☒ Schema(s)
- ☒ XML Fragment
- ☐ Treat Warnings As Errors

Validation Schemas

Validation Type: ☐ No Errors Allowed ☒ Error Message Expression Run Validation

Validation Error List (Select to Ignore)

- ☐ line 1, column 23 : Content is not allowed in prolog.
- ☐ line 1, column 23 : Error reading XML or XSD: Content is not allowed in prolog.

Enter the following parameters:

- **Name:** The name of this assertion. This will help you identify events for this assertion.
- **If:** Select the behavior of the assertion using the drop down menu.
- **then:** Select the step to redirect to if the assertion fires.
- **Log:** The text that will be printed out as event text if the assertion fired.
- **Source:** The property that contains the XML. If this is left blank, the last response is used.
- **Validate:** Multiple validation option can be selected:
- **Well Formed XML:** check that XML is well-formed.
- **DTD Conformance:** check conformance with a DTD.
- **Schema(s):** check conformance with one or more schemas.
- **XML Fragment:** If XML is a fragment, LISA will add XML declaration to top of XML fragment.
- **Treat Warnings as Errors:** Warning will be reported as errors.

Click the **Run Assertion** button to execute an assertion.

Validation Tab

You can run the validation by clicking the **Run Validation** button. Any resulting validation errors are displayed in the **Validation Error List**. Now you can use the **Validation Type** radio buttons to choose how to handle the errors:

- **No errors allowed:** The validation fails on any error
- **Error Message expressions:** Errors can be marked to be ignored in the validation. The errors to ignore can be check in the Validation

Error List if you choose this option. Two errors are shown in the figure above that can be ignored if desired.

Schema Tab

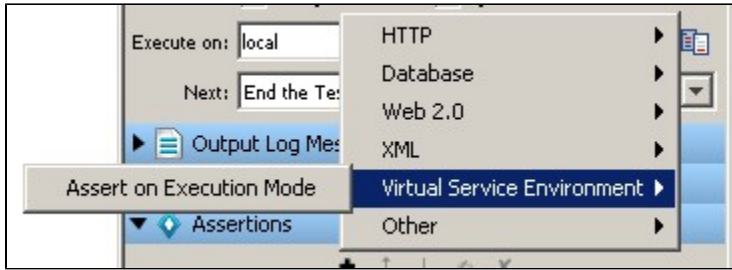
Enter the information for each schema you want to use in the validation. You can also specify the default schema:

- Default Schema: Optionally specify the default schema.
- Other Schemas: Specify the namespace and the URL of the schemas to be used.

29. Virtual Service Environment Assertions

29. Virtual Service Environment Assertions

These are the assertions available in the Virtual Service Environment assertions list for any test step:



This VSE Assertion is explained in the following topic:

29.1 Assert on Execution Mode

29.1 Assert on Execution Mode

29.1 Assert on Execution Mode

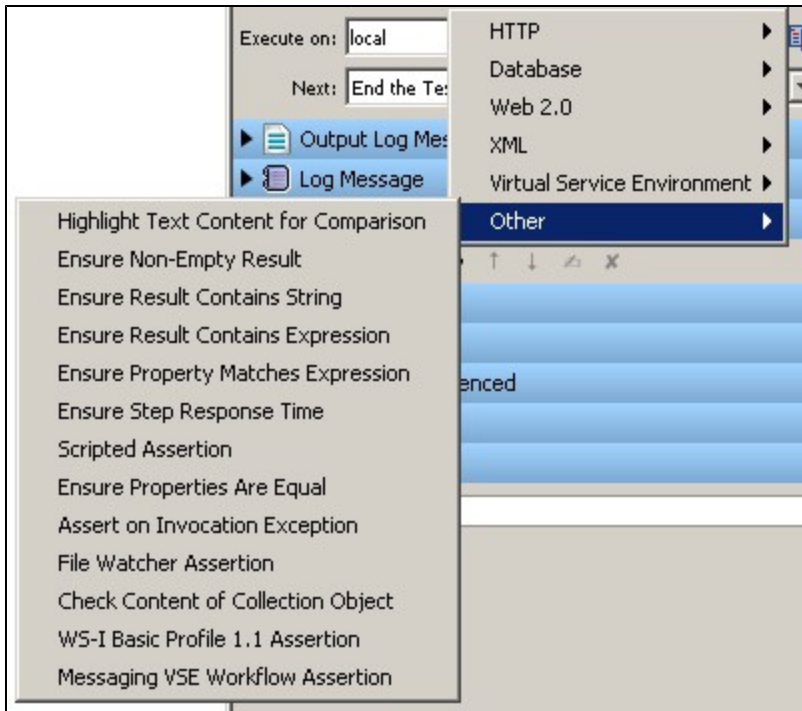
A screenshot of a configuration dialog for the 'Assert on Execution Mode' assertion. The dialog has a 'Name' field with the value 'Assert370'. Below it is an 'If' dropdown set to 'True', followed by a 'then' dropdown set to 'Fail the Test'. There is a 'Log' field. A 'Run Assertion' button is located below the 'Log' field. At the bottom, there is an 'Execution mode' dropdown set to 'Image Validation'. The dialog also features standard navigation icons (plus, up, down, left, right, and close) at the bottom.

This assertion will check the current execution mode to its reference and fire if they match. This is primarily used to control step flow for virtual service models.

30. Other Assertions

30. Other Assertions

These are the assertions available in the "Other" assertions list for any test step:



This list explains these assertions in the order they appear on screen:

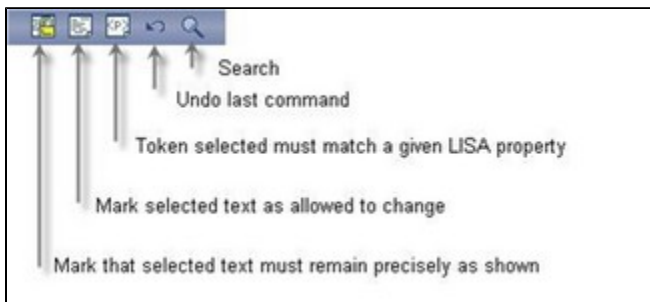
- 30.1 Highlight Text Content for Comparison
- 31.2 Ensure Non-Empty Result
- 30.3 Ensure Result Contains String
- 30.4 Ensure Result Contains Expression
- 30.5 Ensure Property Matches Expression
- 30.6 Ensure Step Response Time
- 30.7 Scripted Assertion
- 30.8 Ensure Properties Equal
- 30.9 Assert on Invocation Exception
- 30.10 File Watcher Assertion
- 30.11 Check Content of Collection Object
- 31.12 WS-I Basic Profile 1.1 Assertion
- 30.13 Messaging VSE Workflow Assertion

30.1 Highlight Text Content for Comparison

30.1 Highlight Text Content for Comparison

This assertion use "paint the screen" technique specifically designed to work with HTML pages. For example, if there is a large HTML document, then the user identifies the data before and after the "content of interest". Then after user simply identify what the "content of interest" will be compared against (usually this would be an expected value supplied in a dataset).

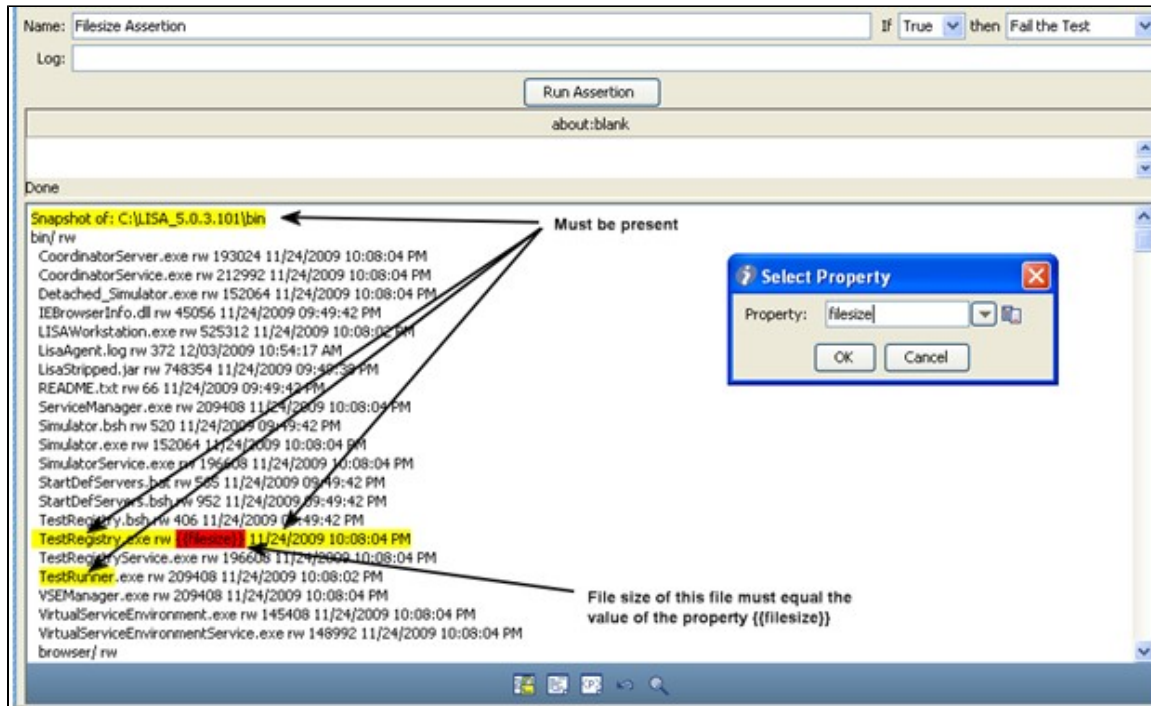
The text is marked using the icons at the bottom of the editor:



This technique is best explained by example.

In the following example, see the figure below, we want to make sure that certain files appear in the buffer, and one of the file sizes needs to be compared to the value of a property. We have marked the text using the 3 icons shown above, by selecting text and then clicking on the appropriate icon.

- Yellow background indicates text that must appear as shown.
- White background indicates text that need not be present, or can change.
- Red background identifies the text that must match the property entered into the dialog box.



The set of tokens shown above can be read this way:

- The buffer must start with the phrase in yellow: "Snapshot of: C:\LISA_5.0.3.101\bin".
- There are a number of files that may or may not be in the buffer in the next token, but since it is an "Any" token the variance is immaterial.
- The file "TestRegistry.exe" and "rw" attributes must appear.
- The red `filesize` means that the value associated with the property key "filesize" will be swapped into the expression, then the comparison made.
- The text "11/24/2009" must appear.
- The file "TestRunner.exe" must appear.
- The buffer can have any amount of content afterward.

After you have finished the markup, enter the following parameters:

- Name: The name of this assertion. This will help you identify events for this assertion.
- If: Select the behavior of the assertion using the drop down box.
- then: Select the step to redirect to if the assertion fires.
- Log: The text that will be printed out as event text if the assertion fired.

Click the Run Assertion button to execute an assertion.

Note: Property blocks must always be bounded by Must blocks.

30.2 Ensure Non-Empty Result

30.2 Ensure Non-Empty Result

(Any Non-Empty Result)

Check the return from the step to ensure that some value has been returned. If there is no response (timeout) or a null value returned, this assertion will return true.

Name:

If:

then:

Log:

Enter the following parameters:

- **Name:** The name of this assertion. This will help you identify events for this assertion.
- **If:** Select the behavior of the assertion using the drop down menu.
- **then:** Select the step to redirect to if the assertion fires.
- **Log:** The text printed out as event text if the assertion fired.

Click the **Run Assertion** button to execute an assertion.

No other attributes are required.

Note: This Assertion should be used with caution, as it implements no content validation.

30.3 Ensure Result Contains String

30.3 Ensure Result Contains String

If the value being searched is found anywhere inside the response, this assertion will return true. This is typically used to make sure the response contains a required value such as a unique id that was supplied during the request.

For more information refer to [Ensure Result Contains String](#).

30.4 Ensure Result Contains Expression

30.4 Ensure Result Contains Expression

(Result as String Contains Expression)

The Result as String Contains Expression assertion allows you to check that a specified regular expression occurs somewhere in the result, as text.

Name:

If:

then:

Log:

Regular Expression:

Enter the following parameters:

- **Name:** The name of this assertion. This will help you identify events for this assertion.
- **If:** Select the behavior of the assertion using the drop down menu.
- **then:** Select the step to redirect to if the assertion fires.
- **Log:** The text that will be printed out as event text if the assertion fired.
- **RegularExpression:** The regular expression to search for in the step result. For example, to check that some number in the 400s appears somewhere in the result, set the RegExpression to **4/d/d**.

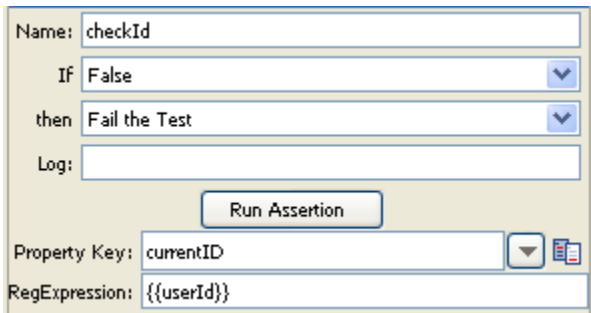
Click the **Run Assertion** button to execute an assertion.

30.5 Ensure Property Matches Expression

30.5 Ensure Property Matches Expression

(Property Value Expression)

The Property Value Expression assertion allows you to check that the current value of a property matches a specified regular expression.



The screenshot shows a configuration window for a 'Property Value Expression' assertion. It contains the following fields and controls:

- Name:** A text box containing 'checkId'.
- If:** A dropdown menu set to 'False'.
- then:** A dropdown menu set to 'Fail the Test'.
- Log:** An empty text box.
- Run Assertion:** A button located below the 'Log' field.
- Property Key:** A text box containing 'currentID' with a small icon to its right.
- RegularExpression:** A text box containing '{{userId}}'.

Enter the following parameters:

- **Name:** The name of this assertion. This will help you identify events for this assertion.
- **If:** Select the behavior of the assertion using the drop down box.
- **then:** Select the step to redirect to if the assertion fires.
- **Log:** The text that will be printed out as event text if the assertion fired.
- **Property Key:** The name of the property to be checked.
- **RegularExpression:** The regular expression that must appear in the current value of the property.

Click the **Run Assertion** button to execute an assertion.

30.6 Ensure Step Response Time

30.6 Ensure Step Response Time

This assertion lets a user define an upper and lower threshold for application response time. If the performance is either too fast or too slow, the test case can be failed by using this assertion. Sometimes an application that returns a response very quickly can be a sign that the transaction was not properly processed.

For more information refer to [Ensure Step Response Time](#).

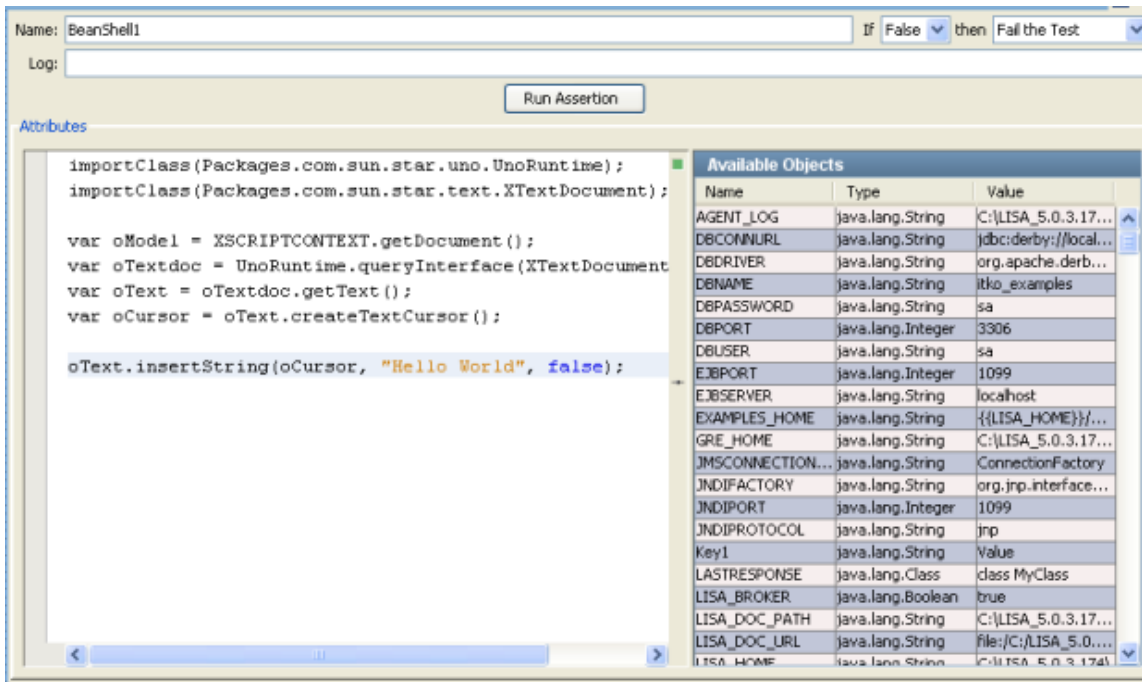
30.7 Scripted Assertion

30.7 Scripted Assertion

(Assert by Script Execution)

The Assert by Script Execution assertion allows you to write and execute Java script in the Bean Shell interpreter. The result must be a Boolean, or **false** is returned.

The Assert by Script Execution editor is shown below:



Enter the following parameters:

- **Name:** The name of this assertion. This will help you identify events for this assertion.
- **If:** Select the behavior of the assertion using the drop down box.
- **then:** Select the step to redirect to if the assertion fires.
- **Log:** The text printed out as event text if the assertion fired.

Click the **Run Assertion** button to execute an assertion.

Enter your script into the Script Editor on the left.

LISA lists all the objects available for use in the Script Editor in the Available Objects Panel on the right. This will include primitive types of data like strings and numbers, but will also include objects like any EJB response objects that have been executed in the test case.

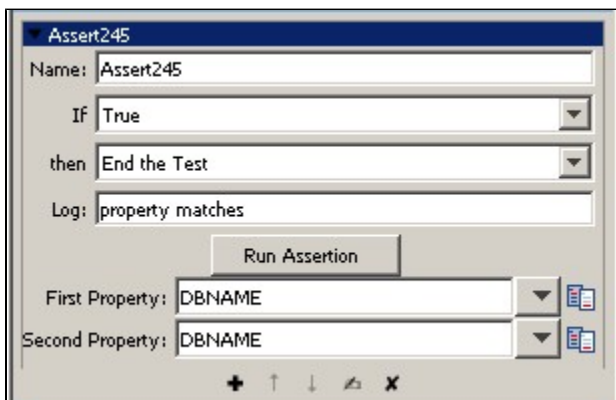
Double-click an entry in the Available Objects table to paste that variable name into the editor area.

Click **Test** to open a window with the result of the script execution or a description of the errors that occurred.

30.8 Ensure Properties Equal

30.8 Ensure Properties Equal

The Assert Properties Equal assertion allows you to compare the values of two properties to make sure that they are same. Typically this is used with a data set and supplied "expected value" to make sure that the application functionality is correct.



Enter the following parameters:

- **Name:** The name of this assertion. This will help you identify events for this assertion.
- **Log:** The text printed out as event text if the assertion fired.
- **Assert:** Select the behavior of the assertion using the radio buttons.
- **Execute:** Select the step to redirect to if the assertion fires.
- **First Property:** The first property in the comparison: type the name or pick from the list.
- **Second Property:** The second property in the comparison: type the name or pick from the list.

30.9 Assert on Invocation Exception

30.9 Assert on Invocation Exception

The Assert on Invocation Exception assertion allows you to alter the test flow based on the occurrence of a Java exception. The assertion will assert true if a particular java exception is returned in the response.

Enter the following parameters:

- **Name:** The name of this assertion. This will help you identify events for this assertion.
- **Log:** The text printed out as event text if the assertion fired.
- **Assert:** Select the behavior of the assertion using the radio buttons.
- **Execute:** Select the step to redirect to if the assertion fires.
- **Expression:** The expression to search for in the invocation exception. It can be a regular expression. It is common to use the expression, '.*'

30.10 File Watcher Assertion

30.10 File Watcher Assertion

The File Watcher assertion allows you to monitor a file for given content, and react to the presence (or absence) of a given expression. This assertion is running in the background while your test case is executing.

Enter the following parameters:

- **Name:** The name of this assertion. This will help you identify events for this assertion.
- **Log:** The text that will be printed out as event text if the assertion fired.
- **The amount of time (in seconds) to delay before checking the file contents:** The number of seconds to wait before checking the file at the beginning of the step that contains this assertion.
- **The amount of time (in seconds) to wait between checks on the file contents:** The number of seconds to wait between each check.
- **The time (in seconds) the FileWatcher will give up watching for the expression:** The total number of seconds this assertion will check for the expression.

- The url of the file to watch: The URL or path to the file being watched.
- The expression to watch for in the file: The regular expression being watched for in the response.

Note: The times are in seconds and must be integers. They default to 0.

30.11 Check Content of Collection Object

30.11 Check Content of Collection Object

The Assert on Contents of Collection Object assertion allows you to make simple assertions on the contents of a collection. This is a useful way to find out whether certain tokens are in the collection, with the option of adding some simple constraints. For example, if one of the pieces of the data returned from a bank web service is a list of account(checking,saving,loan etc.), this assertion can check to make sure all the account ids match expected values.

Enter the following parameters:

- **Name:** The name of this assertion. This will help you identify events for this assertion.
- **Log:** The text printed out as event text if the assertion fired.
- **Assert:** Select the behavior of the assertion using the radio buttons.
- **Property to Check (blank for whole response):** The name of the property holding the object you want to use in the assertion. Leave this blank to use the last response. The object must be of type Java Collection or Array.
- **Field to Check (blank for "toString"):** Enter the name of a field and LISA will call its get method.
- **Tokens To Find (value1, value2):** Comma-delimited string of tokens to check for.
- **Exact Match Only:** The token names much match exactly.
- **Must be in this order:** Check this box if the tokens must be found in same order as the order in the Tokens To Find string.
- **Must only contain these tokens (no extra objects):** The tokens in the Tokens To Find string must be the only tokens found.

30.12 WS-I Basic Profile 1.1 Assertion

30.12 WS-I Basic Profile 1.1 Assertion

The WS-I Basic Profile 1.1 assertion allows you to get a WS-I Basic Profile compliance report for a specific web service. This report is delivered in the standard format specified by the WS-I Basic Profile specification.

WS-I Basic Profile 1.1 Assertion - Assert1146

Name: If then

Log:

Report Type:

Auto-Select Port:

Service Name:

Service Namespace:

Port Name:

Enter the following parameters:

- Name: The name of this assertion. This will help you identify events for this assertion.
- Log: The text that will be printed out as event text if the assertion fired.
- Assert: Select the behavior of the assertion using the radio buttons.
- Execute: Select the step to redirect to if the assertion fires. It is recommended that you redirect to 'Continue' here instead of 'Fail' since it is common for a web service to be found non-compliant.
- Report Type: Select the level of (WS-I) assertions to include in the report. There are 4 levels:
 - Display All Assertions.
 - Display All But Info Assertions.
 - Display Only Failed Assertions.
 - Display Only Not Passed Assertions.
- Auto-Select Port: Determine how the port will be selected – a specific port or 'Don't Auto-select'.
- Service Name: Select a Service Name from the list. This may auto populate from the step.
- Service Namespace: is automatically populated based on Service Name.
- Port Name is automatically populated based on Service Name.

Click **Test** to run the assertion.

Shown below is the header of the WS-I formatted Conformance Report. This is normally a long report.

30.13 Messaging VSE Workflow Assertion

30.13 Messaging VSE Workflow Assertion

This assertion is automatically added from the VSE recorder. It serves a specific purpose that enables proper function with VSE recordings. It should not be used unless you know what you are doing and if it is added to a step in a VSE model it should not be typically removed or edited.

WS-I Basic Profile 1.1 Assertion - Assert1228

Name: If then

Log:

Report Type:

Auto-Select Port:

Service Name:

Service Namespace:

Port Name:

Enter the following parameters:

- Name: The name of this assertion. This will help you identify events for this assertion.
- Log: The text that will be printed out as event text if the assertion fired.
- Assert: Select the behavior of the assertion using the radio buttons.
- Execute: Select the step to redirect to if the assertion fires. It is recommended that you redirect to 'Continue' here instead of 'Fail' since it is common for a web service to be found non-compliant.
- Report Type: Select the level of (WS-I) assertions to include in the report. There are 4 levels:
- Display All Assertions.
- Display All But Info Assertions.
- Display Only Failed Assertions.
- Display Only Not Passed Assertions.
- Auto-Select Port: Determine how the port will be selected – a specific port or 'Don't Auto-select'.
- Service Name: Select a Service Name from the list. This may auto populate from the step.
- Service Namespace: is automatically populated based on Service Name.
- Port Name is automatically populated based on Service Name.

Click **Test** to run the assertion.

PART 5 - Data Sets

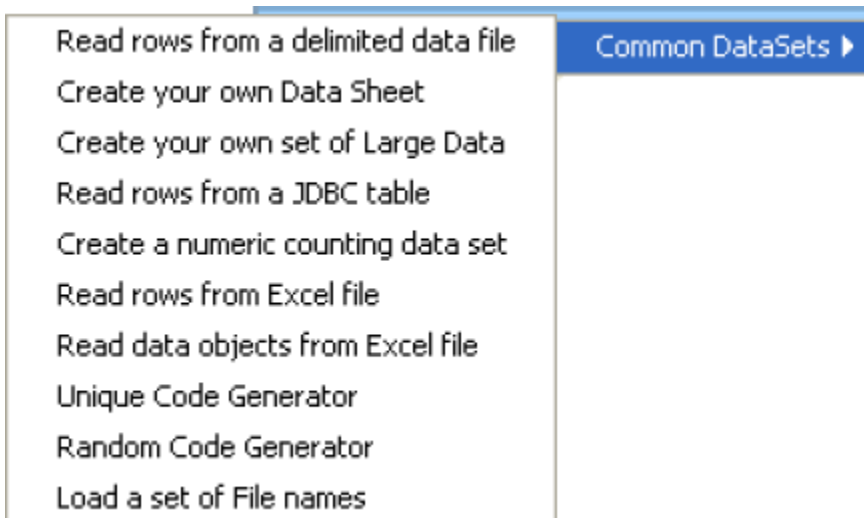
PART 5 - Data Sets

The following chapter is available..

32. Data Sets

32. Data Sets

32. Data Sets



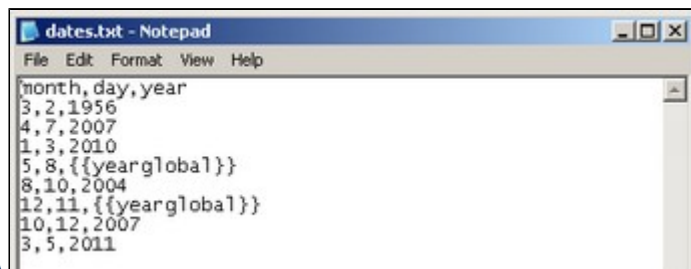
The Data Sets are explained as they appear on the screen:

32.1 Read Rows from a Delimited Data File
 32.2 Create your own Data Sheet
 32.3 Create your own Set of Large Data
 32.4 Convert Data Sheet to Delimited File Data Set
 32.5 Read Rows from JDBC Table
 32.6 Create a Numeric Counting Data Set
 32.7 Read Rows from Excel File
 32.8 Read DTO's from Excel File
 32.9 Read Rows from a JDBC ResultSet
 32.10 Unique Code Generator
 32.11 Random Code Generator
 32.12 Load a Set of File Names

32.1 Read Rows from a Delimited Data File

32.1 Read Rows from a Delimited Data File

The Read Rows from a Delimited Data File data set assigns values to properties based on the contents of a text file. This is the most commonly used type of data set in LISA. The first line of the text file specifies the names of the properties into which the data values will be stored. Subsequent lines list the data values to be used for these properties. The text file is created using a simple text editor.

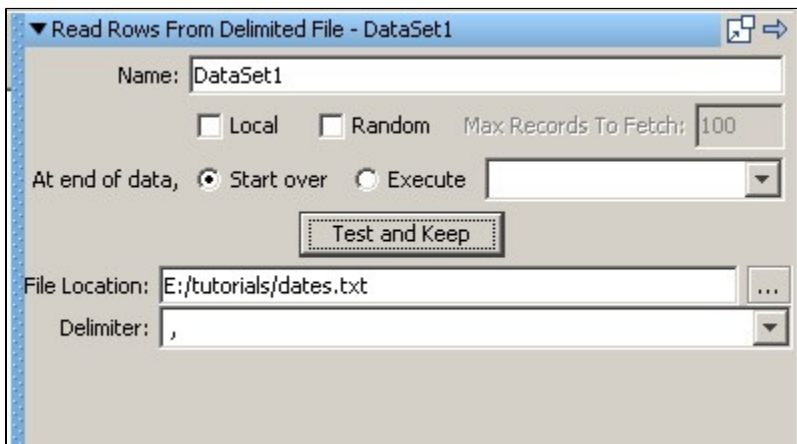


```

month, day, year
3, 2, 1956
4, 7, 2007
1, 3, 2010
5, 8, {{yearglobal}}
8, 10, 2004
12, 11, {{yearglobal}}
10, 12, 2007
3, 5, 2011
  
```

Below is an example of a comma-delimited data file:
 The first row, which is highlighted, shows the property names in this data set.

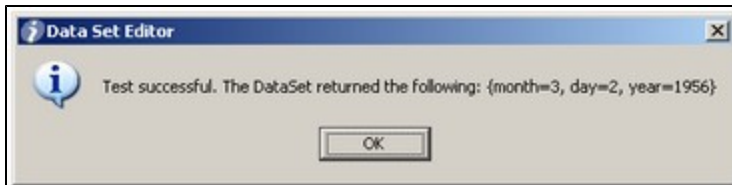
The Data Set Editor is shown below:



In the Data Set Editor enter the following:

- Name: The name of the data set
- At End Of Data: Choose whether you want to Start Over and read values from the start of the data set, or Execute the step you select in the pull-down menu
- Local: Check the textbox if you want a local data set. The default is global (unchecked)
- File Location: The full path name of the text file, or browse to file with the browse button
- Delimiter: The delimiter being used. Any character is allowed as a delimiter. LISA provides common delimiters in the drop-down menu

Click the **Test & Keep** button to test and load the data. You will get a popup that confirms that the data set can be read, and shows the first set of data:



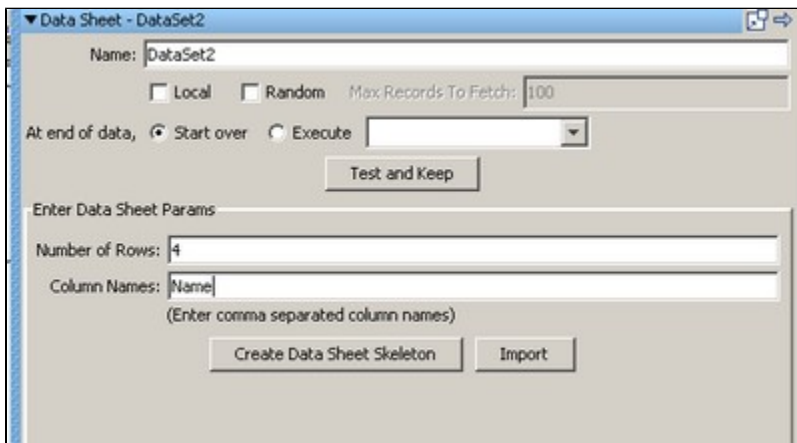
Choose the Steps that will use the dataset.

32.2 Create your own Data Sheet

32.2 Create your own Data Sheet

The Data Sheet data set allows you to generate your data set data within LISA, without requiring any reference to an external file.

The Data Sheet consists of a data table; the column headings specify the property names and the table rows specify the data values for those properties. The table skeleton is built by specifying the number of rows and the column names (properties).

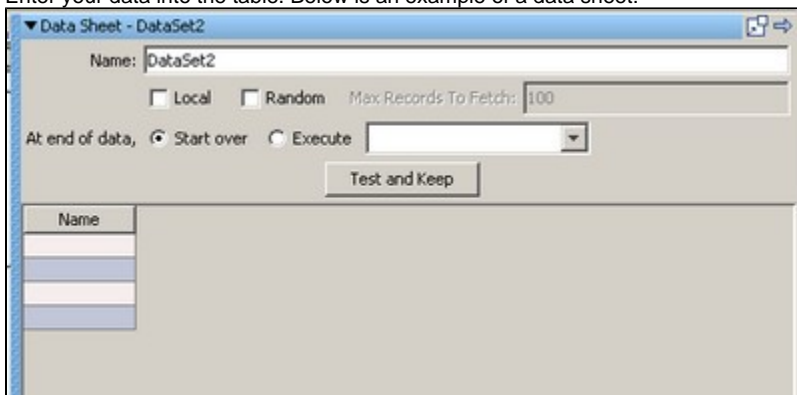


In the Data Set Editor enter the following:

- **Name:** The name of the data set
- **At End Of Data:** Choose whether you want to Start Over and read values from the start of the data set, or Execute the step you select in the pull-down menu
- **Local:** Check the textbox if you want a local data set. The default is global (unchecked)
- **Number of Rows:** Initial estimate of number of rows (it can be modified later).
- **Column Names:** Comma delimited list of column names (these are also the property names).

Click the **Create Data Sheet Skeleton** button.

Enter your data into the table. Below is an example of a data sheet:

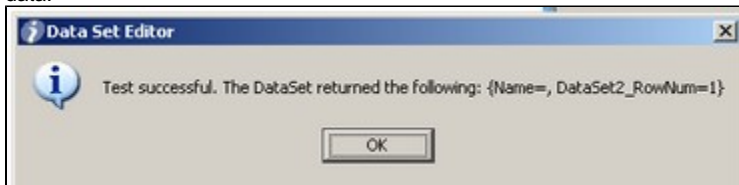


Use the functions in the bottom toolbar to create and modify the table:

- **Add:** Add rows to the table
- **Delete:** Delete the highlighted row
- **Up and Down:** Highlight a row and move it up or down in the table
- **Add Column:** Add a column to your table
- **Delete Column:** Delete a column. Select a cell in the column you want deleted, making sure that the cell is highlighted for editing. Then

click this button.

Click the **Test & Keep** button to test and load the data. You will get a popup that confirms that the data set can be read, and shows the first set of data:



You can also:

- Double-click the column header to sort the rows
- Right click on the column name to change the column name
- Select a cell, and then right-click on a cell to select the above functions and to Launch Extended View to edit the cell

Note: Moving rows up or down in the table may affect the outcome of a test. The order of columns will not affect the outcome of the test.

Choose the Steps that will use the dataset.

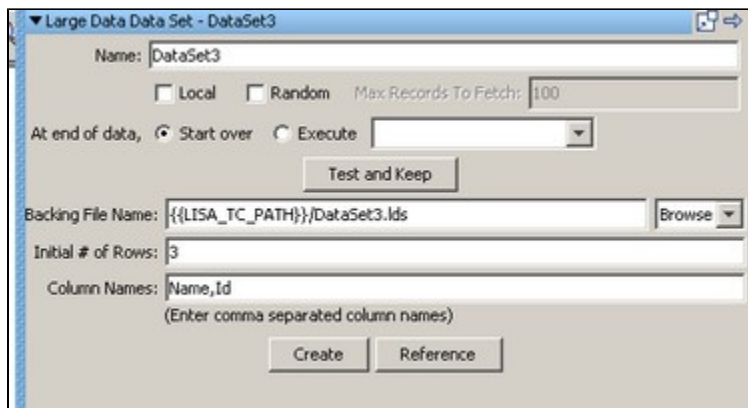
The example above shows a data set called DataSheet3, that will be used as many times as required. To refer to the values in your test steps, use the column names in property notation, for example, `firstName`, `password`.

32.3 Create your own Set of Large Data

32.3 Create your own Set of Large Data

This data set allows user to define custom data table that can be arbitrarily large.

The data can have any number of rows & columns. A backing file name is the file in which all the data is stored.

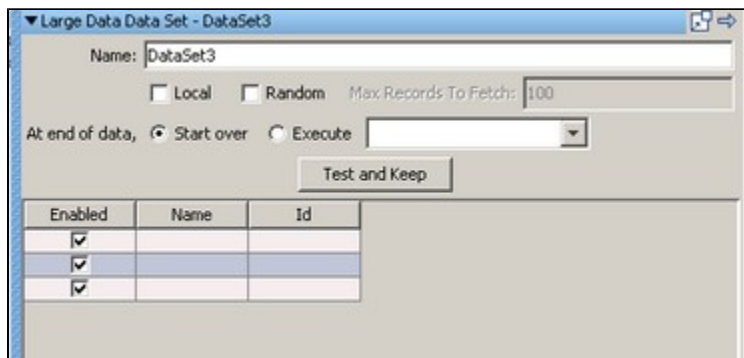


Backing file name: This is the name of the file in which data is stored. This file is created automatically & data which we supply, is inserted.

Initial # of Rows: This field gives initial number of rows which are created.

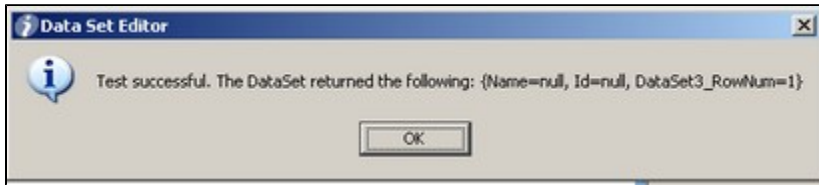
Column Names: This field expects column names separated by comma that will go into data set.

Create button will create the data set file.



Test and Keep button

Pressing **Test and Keep** button after entering data in the columns which we created before, data will be copied in the backing file created.



32.4 Convert Data Sheet to Delimited File Data Set

32.4 Convert Data Sheet to Delimited File Data Set

The Convert Data Sheet to Delimited File data set is a utility that can convert an existing internal data sheet into a comma delimited set, and saves it as a external text file. When selected, the utility constructs and displays a list of all the currently available data sheets in the test case. You select which data sheet you wish to convert, and supply the name of the file where the data will be written.

The Data Set Editor is shown below:

In the Data Set Editor enter the following:

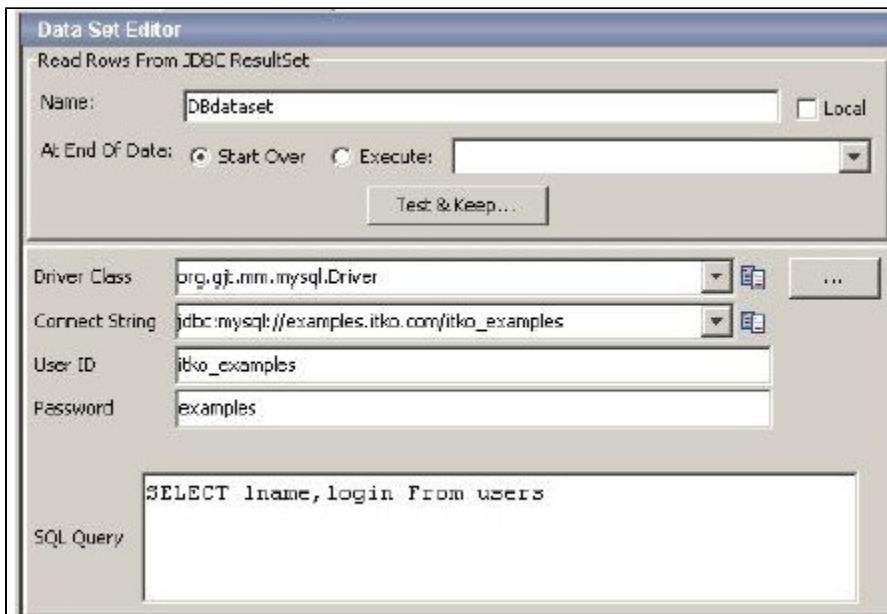
- **Data Sheet To Convert:** Select a Data Sheet from pull down menu.
- **External File Name:** The file name (full path name) of the external file where the data will be saved.

If the file already exists, you will be prompted to check if you want to overwrite it.

32.5 Read Rows from JDBC Table

32.5 Read Rows from JDBC Table

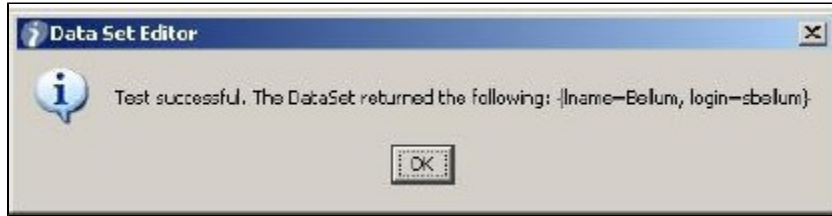
This DataSet is used to read source test case data from a database. The data is read using a JDBC driver (which must be supplied by the user). Each column in the table of data will be represented as a lisa property. The DataSet will then loop across the rows returned from the SQL query.



In the Data Set Editor enter the following:

- **Name:** The name of the data set
- **At End Of Data:** Choose whether you want to Start Over and read values from the start of the data set, or Execute the step you select in the pull-down menu
- **Local:** Check the textbox if you want a local data set. The default is global (unchecked)

Click the **Test & Keep** button to test and load the data. You will get a popup that confirms that the data set can be read, and shows the first set of data:



32.6 Create a Numeric Counting Data Set

32.6 Create a Numeric Counting Data Set

The Create a Numeric Counting data set assigns a number to a property. The number assigned starts at a given value and changes by a fixed step every time the data set is used until it exceeds a known limit. This data set is used to simulate a "for" loop, or, to set the number of times something will occur. For example, you might want to make one hundred calls to the same step. The example below illustrates how to do this using the Create a Numeric Counting Data Set.

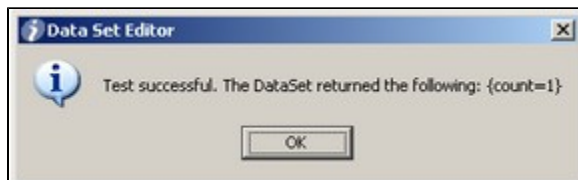
The Data Set Editor is shown below:



In the Data Set Editor enter the following:

- **Name:** The name of the data set
- **At End Of Data:** Choose whether you want to Start Over and read values from the start of the data set, or Execute the step you select in the pull-down menu
- **Local:** Check the textbox if you want a local data set. The default is global (unchecked)
- **Property Key:** The name of the property into which the counter value will be stored.
- **From:** The initial counter value
- **To:** The final counter value
- **Increment:** The step increment for the counter. The counter data set can be used to count backwards by assigning a negative increment.

Click the **Test & Keep** button to test and load the data. You will get a popup that confirms that the data set can be read, and shows the first set of data:



Choose the Steps that will use the dataset.

In our example, the data set is configured to start at 1, increment by 1 until it exceeds 100.

Note: To iterate on a single step, change that step's "next step" to be itself. Do this by going to the step's Base Step Info and selecting Next for the Step element.

32.7 Read Rows from Excel File

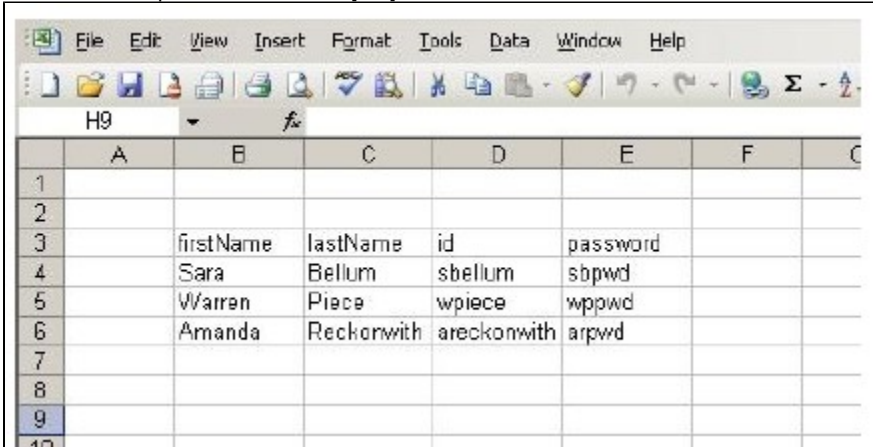
32.7 Read Rows from Excel File

The Read Rows from Excel File data set assigns values to properties based on the contents of an Excel spreadsheet.

The first non-blank row of the Excel spreadsheet specifies the names of the properties to which the data values will be assigned. Subsequent rows list the data values to be used for these properties. The first full row of empty cells is treated as the end of data.

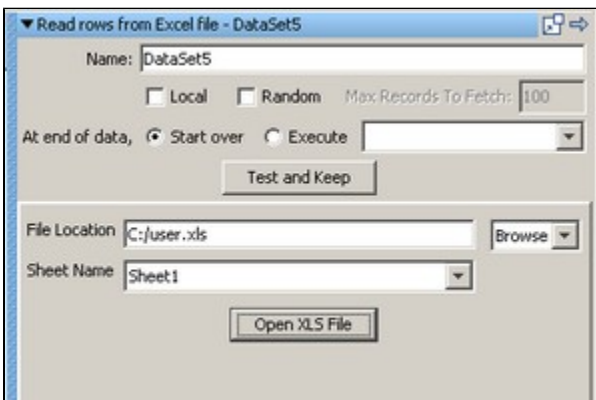
For example, you might want to test a set of first name, last name, id and password combinations.

Here is an example of an Excel data [file:]



	A	B	C	D	E	F	G
1							
2							
3		firstName	lastName	id	password		
4		Sara	Bellum	sbellum	sbpwd		
5		Warren	Piece	wpiece	wppwd		
6		Amanda	Reckonwith	areckonwith	arpwd		
7							
8							
9							
10							

The Data Set Editor is shown below:



▼ Read rows from Excel file - DataSet5

Name: DataSet5

☐ Local ☐ Random Max Records To Fetch: 100

At end of data, ☒ Start over ☐ Execute

Test and Keep

File Location: C:\user.xls Browse

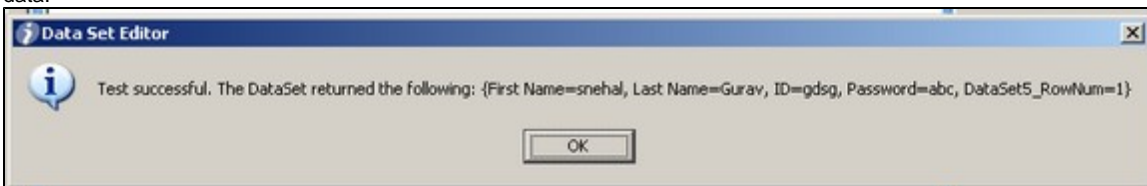
Sheet Name: Sheet1

Open XLS File

In the Data Set Editor enter the following:

- Name: The name of the data set
- At End Of Data: Choose whether you want to Start Over and read values from the start of the data set, or Execute the step you select in the pull-down menu
- Local: Check the textbox if you want a local data set. The default is global (unchecked)
- File Location: The full path name of the Excel file, or browse to file with the browse button. You can use a property in the path name (for instance LISA_HOME)
- Sheet Name: The name of the sheet in the Excel spreadsheet

Click the **Test & Keep** button to test and load the data. You will get a popup that confirms that the data set can be read, and shows the first set of data:



Choose the Steps that will use the dataset.

The example above shows a data set called **DataSet2** that will be used as many times as required. Data is loaded from Excel file.

Note: You can click the Open XLS File button to open and edit the Excel spreadsheet

32.8 Read DTO's from Excel File

32.8 Read DTO's from Excel File

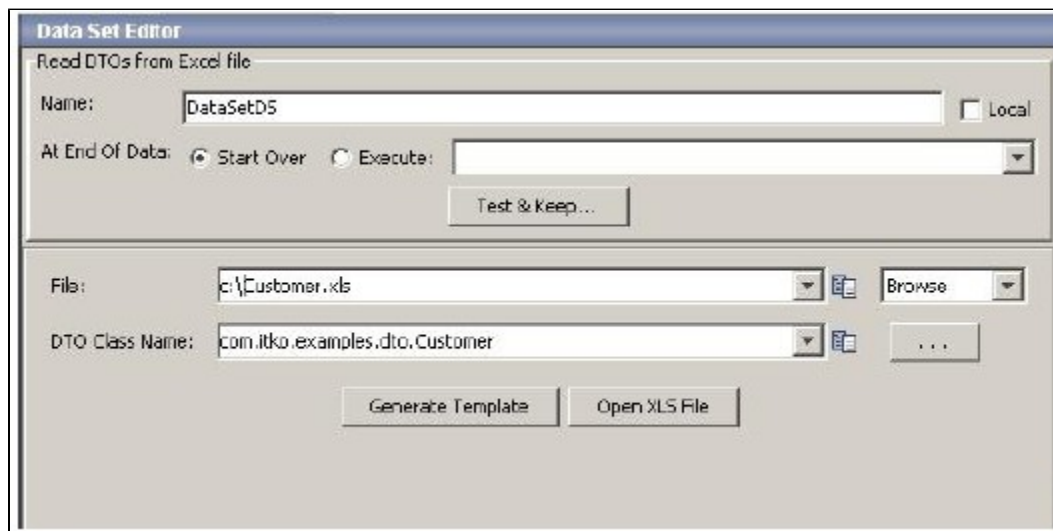
The Read DTO's from Excel file data set allows you to parameterize Java data transfer objects (DTOs) in your test steps and gives you an easy way, through Excel spreadsheets, to provide data values for those parameters.

The Read DTO's from Excel file data set assigns values to the properties of a DTO and stores the object in a LISA property. This property can then be used whenever the DTO is required as a parameter. The data in the data set can be simple data types like numbers or strings, or complex data types such as DTOs, arrays and collections. LISA will convert the data represented in Excel into the proper data types automatically when needed. The only complex part of using this dataset is the initial creation of the Excel spreadsheet. Fortunately, LISA does this for you! Given the package name of the DTO, LISA will create a template, using one or more Excel sheets, that represent the object. Data types such as primitives, strings, arrays of primitives and simple individual DTOs can be represented on a single sheet. More complex data types, such as arrays of objects, will require additional Excel sheets in order to represent the full DTO.

It is often the case that a web service endpoint expects complex DTOs. The Excel data set makes it very simple to create objects to use as parameters to the web service. When the web service is first referenced in LISA, it is given a name and a URL for the WSDL. LISA then automatically generates java DTO classes in the form **com.lisa.wsgen.SERVICENAME.OBJECTNAME** and makes them available on the classpath. You can browse to that generated class in the DTO class browser (see below), generate an Excel file and simply fill in the template.

Building the data set is a two step process. First, we let LISA build the template in Excel, second we open the Excel spreadsheet and fill in the data fields in all the sheets LISA produced.

The Data Set Editor is shown below:



To build the template, enter the following:

- Name: The name of the dataset. This becomes the property that LISA uses to store the current DTO object.
- At End Of Data: Choose whether you want to Start Over and read values from the start of the data set, or Execute the step you select in the pull-down menu.
- Local: Check the textbox if you want a local data set. The default is global (unchecked).
- File: The fully qualified path name, or browse to the Excel file using the browse pull-down menu.
- DTO Class Name: The full package name, or browse to, the DTO object. The class file must be on LISA's Test Manager. Your class can be copied to the hotDeploy directory to put it on the Test Manager.

Click the **Generate Template** button.

LISA will build the template for you. In the system messages you will see a message that the file has been built.

Click the **Open XLS File** button.

The spreadsheet contains everything needed to construct the object. We will show how to add data in the next section.
Close the XLS file.

Click the **Test & Keep** button to test and load the data. You will get a popup window that confirms that the data set can be read, and shows the first set of data.

Building the Excel spreadsheet

To facilitate this explanation, we will use an actual DTO object: **com.itko.example.dto.Customer**. This class is included with the LISA examples and can be found by browsing the Test Manager using the **Browse** button.

The Customer DTO has the following properties:

Property Name	Type
balance	Double
id	int
name	String
poAddr	Address
since	Date
types	int[]
locations	Address[]

The Address DTO has the following properties:

Property Name	Type
city	String
line1	String
line2	String
state	String
zip	String

The first six Customer DTO properties can appear on one Excel spreadsheet. However, the locations property, an array of Address objects, requires a second Excel spreadsheet.

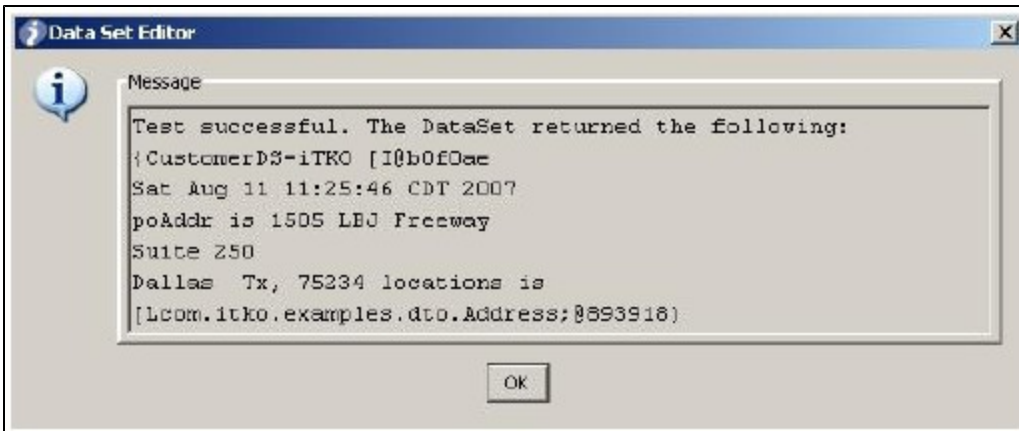
Looking at the first spreadsheet, at the top, LISA lists the DTO spec (Customer) and the current DTO object (Customer). It would also list the Java doc location, if available. The actual data sheet appears below, with a row specifying property names, followed by a row specifying the data types. The first field (column) is not a DTO property, but a special field (Primary key), that holds a unique value for each row.

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2		Spec:	com.itko.examples.dto.Customer									
3		DTO:	com.itko.examples.dto.Customer									
4		Docs:	No docs available									
5		Static Constructor Meth:	No methods available									
6		Static Constructor Field:	No fields available									
7												
8		Primary Key	balance	id	name	poAddr.city	poAddr.line1	poAddr.line2	poAddr.state	poAddr.zip	since	types
9		(unique per row)	double	int	String	String	String	String	String	String	Date	int[]
10			1	50	101 itko	Dallas	1505 LBJ F Suite 250	Tx		75234	8/11/2007	1,5,10
11			2	275	102 Ron	Plano	4509 Postbridgeave	Tx		75024	6/9/2007	2,4,9
12			3	1000	103 John	Dallas	1505 LBJ F Suite 250	Tx		75234	6/7/2007	3,7,9
13												
14												
15												

Looking at the data sheet we can see the following:

- Each row contains the data for a single Customer DTO
- The poAddr property, of type Address, has been "flattened" and its properties are listed on this sheet. These properties are prefixed with poAddr and then the property name in the address object (i.e. poAddr.city)
- The since property, of type date, is prefilled with today's date. This is to show you the required format for the date mm/dd/yyyy. All dates must have this format in the Excel template
- The types property, of type int[], is a single cell which can contain the array elements as a comma separated list. This is only possible for arrays of primitives or strings

The location property, of type Address[], does not appear on this sheet. It is in fact on the second sheet in the Excel file. This is because it is an array of objects. This sheet contains the data for an Address object in each row. There are two special fields in this sheet, "Primary Key", and the "reference the containing DTO" field that is used to link the rows in this sheet to the rows in the primary sheet.



Choose the Steps that will use the dataset.

Depending on the complexity of your DTO, you could have several more Excel sheets in the Excel workbook. However, the process is the same as above.

To see how you might use this Customer DTO as a property, see the sections on testing Java objects in [PART 2 - Test Steps](#).

32.9 Read Rows from a JDBC ResultSet

32.9 Read Rows from a JDBC ResultSet

The Read Rows from a JDBC ResultSet data set assigns values to properties based on the result set obtained from executing an SQL query. For example, you might want to use a set of login ids and last names in your test. You can create that data by querying the database where they are stored using the following SQL query:

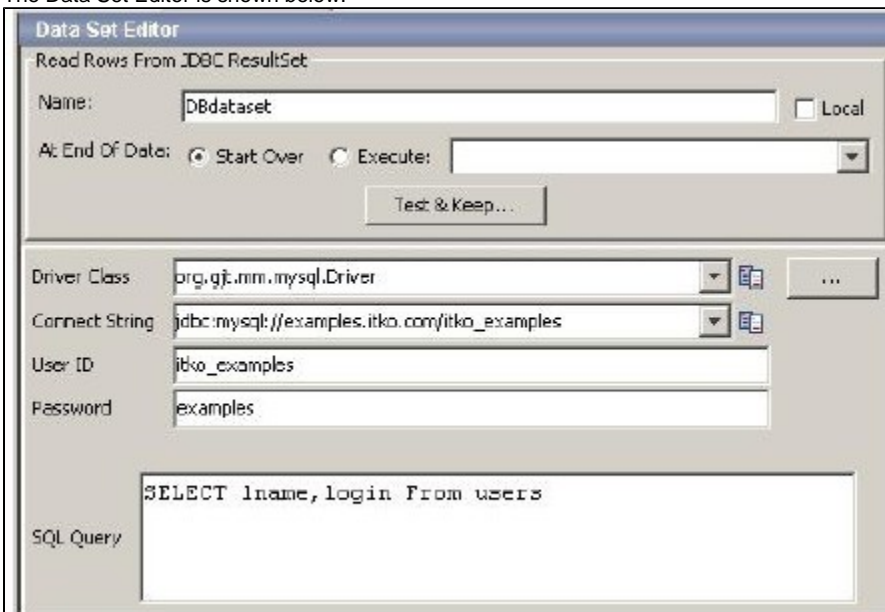
lname	login
Bellum	sbellum
Piece	wpiece
Redonwith	areck
	itko

SELECT lname, login, FROM users;

The result set of the query contains two fields, lname and login, and four records, as seen below:

The field (column) names define the properties set by the data set. Each result set record (row) specifies the values to be used for the properties.

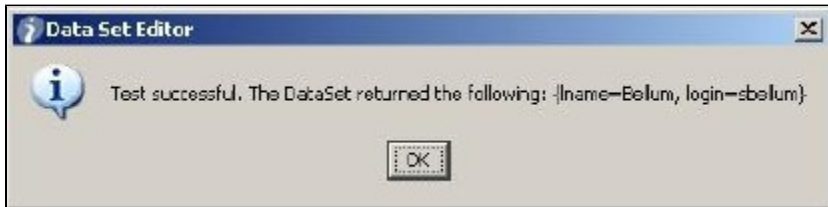
The Data Set Editor is shown below:



In the Data Set Editor enter the following:

- Name: The name of the data set
- At End Of Data: Choose whether you want to Start Over and read values from the start of the data set, or Execute the step you select in the pull-down menu
- Local: Check the textbox if you want a local data set. The default is global (unchecked)
- Driver Class: The fully qualified name of the appropriate database driver class
- Connect String: The connection string (JDBC URL) used to locate the database
- User ID: The ID to use when connecting to the database (if required)
- Password: The password to use when connecting to the database (if required)
- SQL Query: The SQL query used to create the data set

Click the **Test & Keep** button to test and load the data. You will get a popup that confirms that the data set can be read, and shows the first set of data:



Choose the Steps that will use the dataset.

The example above shows a data set called **DBdataSet** that will be used as many times as required. Data is loaded from the result set obtained from querying the users table from the **itko_examples** database.

32.10 Unique Code Generator

32.10 Unique Code Generator

The Unique Code Generator data set provides a unique token (or code) each time it is called. The token can be numeric or alphanumeric, and can have a user defined prefix pre-pended to it. This is commonly used in testing to create new users, accounts, etc. to ensure they do not use a value which is already inside a system.

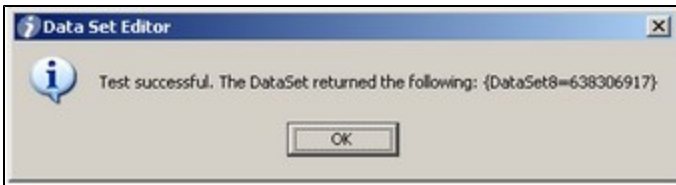
The Data Set Editor is shown below



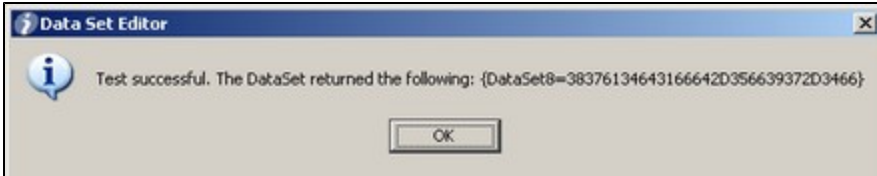
In the Data Set Editor enter the following:

- Name: The property that is assigned the value. This is also the name of the data set
- At End Of Data: Choose whether you want to Start Over and read values from the start of the data set, or Execute the step you select in the pull-down menu
- Local: Check the textbox if you want a local data set. The default is global (unchecked)
- Type: Type of token to return. Choices are Number or Alphanumeric
- Prefix (opt): Prefix to be pre-pended (optional)

Click the **Test & Keep** button to test and load the data. You will get a popup that confirms that the Data set can be read, and shows the first set of data:



The next **Test & Keep** result shows the result of choosing an alphanumeric token:



Choose the steps that will use the dataset.

Note: Since this data set will always return a token, the At End of Data section of the Data Set Editor has no meaning for this data set type.

32.11 Random Code Generator

32.11 Random Code Generator

This data set generates numeric or alphanumeric data randomly for use in a test case. This is similar to the unique code generator, but it allows the user the ability to set a length for the result. This can be used to create a particular type of unique value such as a phone number, social security number, etc.



In the Data Set Editor enter the following:

- Name: The name of the data set
- At End Of Data: Choose whether you want to Start Over and read values from the start of the data set, or Execute the step you select in the pull-down menu
- Local: Check the checkbox if you want a local data set. The default is global (unchecked)

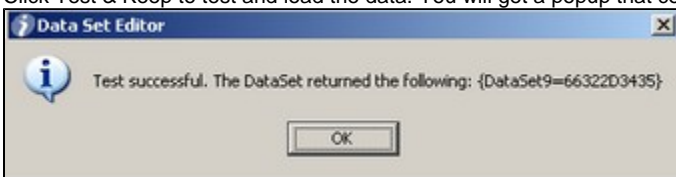
Random Max Records: This field takes number which is nothing but the records generated randomly by this data set at run time.

PRexif(opt) adds prefix to the generated data. This field is optional.

Type: This field allows either **Alphanumeric** or **Number** type of data set to be generated.

Length: This field restricts the length of the random data generated to the value set here.

Click Test & Keep to test and load the data. You will get a popup that confirms that the data set can be read, and shows the first set of data:



32.12 Load a Set of File Names

32.12 Load a Set of File Names

The Load a Set of File Names data set assigns a value to a property based on a filtered set of filenames from the file system. The set of filenames can include all the files in a given directory, or a set filtered by a "file pattern". There is also an option to recursively include sub-directories in the set. The files are returned in a case-sensitive, alphabetical order, top directories first, followed by sub-directories (using depth first ordering). Each time the data set is used, it returns the next filename (full path name) in the dataset. This filename is stored in a property whose name is the same as that of the data set.

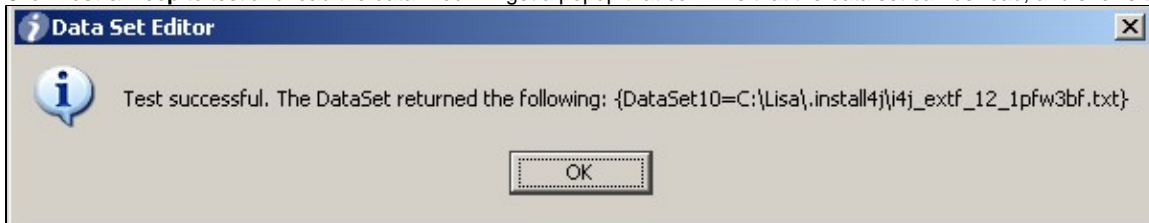
The Data Set Editor is shown below:



In the Data Set Editor enter the following:

- Name: The name of the data set
- At End Of Data: Choose whether you want to Start Over and read values from the start of the data set, or Execute the step you select in the pull-down menu
- Local: Check the textbox if you want a local data set. The default is global (unchecked)
- Directory Location: The full pathname of the directory to scan, or you can browse using the browse button
- File Pattern: A filter pattern string, using an * as a wild card is desired (optional)
- Include Files from Sub-directories: Check if files in sub-directories are to be listed

Click **Test & Keep** to test and load the data. You will get a popup that confirms that the data set can be read, and shows the first set of data:]



Choose the Steps that will use the dataset.

32.13 XML Data Set

32.13 XML Data Set

Like other data sets in LISA, the XML Data Set allows user-specified content to be added to distinct records. However, the XML Data Set specializes in handling XML content.

XML Data Set - DataSet1

Basic Settings | Advanced Settings

Name: DataSet1

☐ Local ☐ Random Max Records To Fetch: 0

At end of data, ☒ Start over ☐ Execute

Test and Keep

First Previous Next Last New Copy Delete Save Revert

Record 1 of 2

Visual XML Raw XML

Node	Occurs	Nil	Nilable	Value
pattern		<input type="checkbox"/>		
name		<input type="checkbox"/>		LisaBank first name
provider		<input type="checkbox"/>		com.itko.lisa.vse.desensitize.LisaPropReplacer
regex		<input type="checkbox"/>		<fname>.*?</fname>
replacement		<input type="checkbox"/>		<fname>Timothy</fname>
pattern		<input type="checkbox"/>		
name		<input type="checkbox"/>		LisaBank last name
provider		<input type="checkbox"/>		com.itko.lisa.vse.desensitize.LisaPropReplacer
regex		<input type="checkbox"/>		<lname>.*?</lname>
replacement		<input type="checkbox"/>		<lname>Hoskins</lname>
pattern		<input type="checkbox"/>		
name		<input type="checkbox"/>		LisaBank phone
provider		<input type="checkbox"/>		com.itko.lisa.vse.desensitize.LisaPropReplacer
regex		<input type="checkbox"/>		<phone>.*?</phone>
replacement		<input type="checkbox"/>		<phone>316-284-8483</phone>

Validation Results | View XML Schema Source | Error Log

Introduction

At design-time, the first record of an XML Data Set populates the value for a property in LISA test state via clicking the **Test and Keep** button. The LISA property name is the same as the name of the data set, e.g. if the data set is named "DataSet1", then the property populated in test case is "{DataSet1}". At run-time, all of the records can be accessed sequentially to create a data-driven test case.

Basic Settings Tab

XML Data Set - DataSet1

Basic Settings | Advanced Settings

Name: DataSet1

☐ Local ☐ Random Max Records To Fetch: 0

At end of data, ☒ Start over ☐ Execute

Test and Keep

Name: The name of this data set. This value will be used as the name of the property in test step. For example, an XML Data Set named **DataSet1** will populate the property `{{DataSet1}}`.

Local: Whether this should act as a global or local data set. Global is the default. Local data sets are created one-per-simulator; global data sets are created once and shared by all simulators.

Random: Whether the record after the current record (sequential access) will be read, or a random record will be read. Sequential reading is the default.

Max Records to Fetch: The upper bound on the number of records to fetch for random access. This text field is disabled if the **Random** check box is not selected.

At end of data, Start over: After the last record is read, loop back to the very first record. This is the default behavior.

At end of data, Execute: After the last record is read, branch to the specified test step.

Test and Keep: At design-time, populate the LISA property associated with the data set in test state with the value of the first record.

Advanced Settings Tab

XML Data Set - DataSet1

Basic Settings | Advanced Settings

Directory path: {{LISA_PROJ_ROOT}}/Data/datasets/xml/30393134663437332D343861332D3431

Directory path: This read-only value represents the directory on the file system where records are saved. There is a one-to-one mapping between a record and a file in the directory path. If an XML Data Set is created within a project, the directory path will start with `"{{LISA_PROJ_ROOT}}/Data/datasets/xml/"`. If no project is used, the directory path will start with `"{{LISA_HOME}}/datasets/xml/"`.

Record Editing Panel

First

Previous

Next

Last

New

Copy

Delete

Save

Revert

Record 1 of 2

Visual XML

Raw XML

Node		Occurs	Nil	Nilable	Value
[-]	pattern		<input type="checkbox"/>		
	name		<input type="checkbox"/>		LisaBank first name
	provider		<input type="checkbox"/>		com.itko.lisa.vse.desensitize.LisaPropReplacer
	regex		<input type="checkbox"/>		<fname>.*?</fname>
	replacement		<input type="checkbox"/>		<fname>Blanche</fname>
[-]	pattern		<input type="checkbox"/>		
	name		<input type="checkbox"/>		LisaBank last name
	provider		<input type="checkbox"/>		com.itko.lisa.vse.desensitize.LisaPropReplacer
	regex		<input type="checkbox"/>		<lname>.*?</lname>
	replacement		<input type="checkbox"/>		<lname>Bogle</lname>
[-]	pattern		<input type="checkbox"/>		
	name		<input type="checkbox"/>		LisaBank phone
	provider		<input type="checkbox"/>		com.itko.lisa.vse.desensitize.LisaPropReplacer
	regex		<input type="checkbox"/>		<phone>.*?</phone>
	replacement		<input type="checkbox"/>		<phone>864-634-6257</phone>

Action Buttons

- First:** Move to the first record in the XML Data Set.
- Previous:** Move to the previous record.
- Next:** Move to the next record.
- Last:** Move to the last record.
- New:** Create a new record.
- Copy:** Create a copy of the current record at the end of the data set and move to it.
- Delete:** Delete the current record.
- Save:** Save all pending changes.
- Revert:** Revert all pending changes

Record Number Selector

Record X of X: Enter the record number to jump to a specific record.

Node	Occurs	Nil	Nillable	Value
pattern				
name				LisaBank first name
provider				com.itko.lisa.vse.desensitize.LisaPropReplacer
regex				<fname>.*?</fname>
replacement				<fname>Timothy</fname>
pattern				
name				LisaBank last name
provider				com.itko.lisa.vse.desensitize.LisaPropReplacer
regex				<lname>.*?</lname>
replacement				<lname>Hoskins</lname>
pattern				
name				LisaBank phone
provider				com.itko.lisa.vse.desensitize.LisaPropReplacer
regex				<phone>.*?</phone>
replacement				<phone>316-284-8483</phone>

Visual XML Tab
Visual XML Editor

Moving the mouse over the Node and Value columns will display a tool tip. This can help if the value in the table is too long to be fully displayed.

```

<pattern name="LisaBank first name" provider="com.itko.lisa.vse.desensitize.LisaPropR
  <regex><![CDATA[<fname>.*?</fname>]]></regex>
  <replacement><![CDATA[<fname>{={:First Name:}}</fname>]]></replacement>
</pattern>
<pattern name="LisaBank last name" provider="com.itko.lisa.vse.desensitize.Li
  <regex><![CDATA[<lname>.*?</lname>]]></regex>
  <replacement><![CDATA[<lname>{={:Last Name:}}</lname>]]></replacement>
</pattern>
<pattern name="LisaBank phone" provider="com.itko.lisa.vse.desensitize.LisaPr
  <regex><![CDATA[<phone>.*?</phone>]]></regex>
  <replacement><![CDATA[<phone>{={:Telephone:}}</phone>]]></replacement>
</pattern>

```

Raw XML Tab

This tab contains a raw text view of the XML contained in a record.

Tip: Moving the mouse over a LISA property displayed in the editor will show the current value of that property in design-time test state in a

tooltip. For example, in the image above, a tooltip is displayed for the property `{{[:First Name:William]}}`.

Tip: Double-clicking the left border (where the line numbers exist in the image above) will toggle the visibility of a top toolbar, line numbers bar, and bottom editing info bar in the editor.

32.14 CorrelationID data set

The correlation ID data set is a specialised unique code generator useful for messaging. It generates a 24 byte unique code - designed specifically for IBM MQ Series correlation ID, but can also be used for any JMS provider. This data set sets creates or updates two special lisa properties 'lisa.jms.correlation.id' and 'lisa.mq.correlation.id'. The messaging steps recognise these properties and set the message correlation ID appropriately. There are no user settings for this data set type.

PART 6 - Companions

PART 6 - Companions

The following chapter is available..

33. Companions

33. Companions

33. Companions

LISA has following types of Companions –

Web Browser Simulation
Browser Bandwidth Simulation
Configure LISA to use a Web Proxy
Set up a Synchronization Point
Set up an Aggregate Step

Common Companions ►
Other Companions ►

- **Common Companions**
 1. Web Browser Simulation
 2. Browser Bandwidth Simulation
 3. Configure LISA to use a Web Proxy
 4. Set up a Synchronization Point
 5. Set up an Aggregate Step
- **Other Companions**

Create a Sandbox Class Loader for Each Test
Set Final Step to Execute
Negative Testing Companion
Fail Test Case Companion
XML Diff Ignored Nodes Companion

Common Companions ►
Other Companions ►

1.
 - a. Create a Sandbox Class Loader for each test.
 - b. Set Final Step to Execute
 - c. Swing Test Settings Companion
 - d. Resource Manager Companion
 - e. Negative Testing Companion.
 - f. Fail Test Case Companion

g. XML Diff Ignored Nodes Companion

This section describes each of the Companions that are available in LISA. You should have a general understanding of the concepts and uses of companions in test cases.

33.1 Common Companions

33.1 Common Companions

Common Companions has following options under it:

[33.2 Web Browser Simulation.](#)

[33.3 Browser Bandwidth Simulation.](#)

[33.4 Configure LISA to use a Web Proxy.](#)

[33.5 Set Up a Synchronization Point.](#)

[33.6 Set up an Aggregate Step.](#)

33.2 Web Browser Simulation

33.2 Web Browser Simulation

The Web Browser Simulation companion allows you to simulate a variety of web browsers. Web browsers identify themselves to a web server via the User-Agent HTTP header. You configure LISA to simulate several User-Agents when you are running several virtual users in a staged test. Each User-Agent string is assigned a relative weight, allowing one browser to appear more often than others.

The default Browser Selection Companion Editor is shown below:

▼ Web Browser Selection - Web Browser Selection

Browser Selection Attributes

User-Agent	Weight
Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)	5
Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)	5
Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)	1
Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)	1
Mozilla/4.0 (compatible; MSIE 4.5; Windows NT 5.0)	1
Mozilla/4.0 (compatible; MSIE 5.0; Windows 95)	1
Mozilla/4.76 (Macintosh; U; PPC)	2
Mozilla/4.0 (compatible; MSIE 5.0; Mac_PowerPC)	2
Mozilla/3.01 (X11; I; Linux 2.4.18-6mdk i686)	2
Mozilla/4.76 (X11; I; Linux 2.4.18-6mdk i686)	2
w3m/0.1.9	0
Wget/1.5.3	0

Find:

To configure the Web Browser Selection companion, enter or edit the list of browser agents and the weights:

- User-Agent: The browser to simulate
- Weight: The weight for this browser. For example, to assign weights of 25%, 25% and 50% for three browsers, enter weights of 1, 1, and 2 for the three rows, and zero for the others (or delete the extra rows)

To add an additional User-Agent, use the **Add** button.

To delete a line, use the **Delete** button.

33.3 Browser Bandwidth Simulation

33.3 Browser Bandwidth Simulation

The Browser Bandwidth Simulation companion allows you to simulate varied bandwidths for the virtual users. Some testing scenarios call for the simulations of different types of internet connections.

The Browser Bandwidth Simulation Attributes Editor is shown below:



To configure the Browser Bandwidth Simulation Companion, enter the following parameters:

- **BytesPerSec:** The connection speed. For example, to simulate a connection speed of 56K, enter a BytesPerSec of 7000 (56000 bits / 8 bits per byte = 7000 bytes per second)
- **ConnectPause:** The number of milliseconds to pause before starting the test case, to simulate the time to establish a connection
- **MaxReadSize:** The maximum size of a single read allowed by the simulated browser. For example, if the simulated browser only supports 32K per read, enter a MaxReadSize of 32000

- **Weight:** The weight given to this row. For example, to assign weights of 25%, 25% and 50% to three rows, enter values of 1, 1 and 2 on the weight column of the three rows

To add an additional line, use the **Add** button.

To delete a line, use the **Delete** button.

33.4 Configure LISA to use a Web Proxy

33.4 Configure LISA to use a Web Proxy

The Web Proxy Setup companion allows you to set up a proxy for all Web testing steps. If there are times when your environment dictates the use of a proxy, use this Companion. Proxy information is specific to your organization. We suggest you consult your operations team for your company's proxy settings.

The Web Proxy companion Editor is shown below:

To configure the Web Proxy Setup companion enter the following parameters:

- Web Proxy Server (Host & Port): The name or IP address of the proxy server in the first field, and the port number in the second field
- Secure Web Proxy Server (SSL Proxy Host & Port): The name or IP address of the SSL proxy server in the first field, and the port number in the second field
- Bypass secure web proxy for these Hosts & Domains: Here enter names of the domains/hosts you wish secure proxy to be bypassed.
- Proxy Server Authentication: Here domain name along with username & password is required for authenticating to proxy server.

LISA can also use its **local.properties** file to assign a web proxy for all test cases. This file is in the LISA home directory. Update the **lisa.http.webProxy.host** and **lisa.http.webProxy.port** properties as appropriate and restart LISA. If you do not already have a **local.properties** file in the LISA home directory, rename the existing **_local.properties** to **local.properties**, and use it

33.5 Set Up a Synchronization Point

33.5 Set Up a Synchronization Point

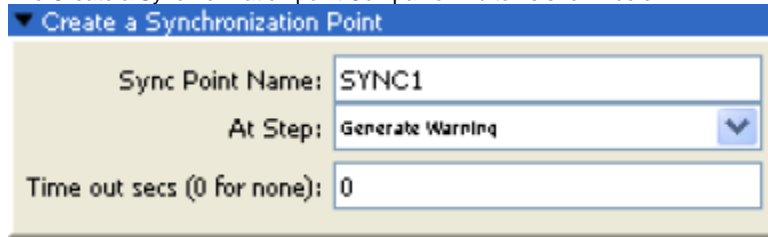
The Create a Synchronization Point companion allows you to select a test step that will be used as a synchronization point in a test case or a test suite. At a synchronization point, virtual users will pause and wait until each one has reached this step. Then all virtual users will be released to execute the step at the same time. This is very useful when setting up load tests for concurrent testing or peak resource utilization.

For example you could set up a 100 user test to have all users log in to your app and then all order the same seat in a theater session at the same time.

Synchronization Points apply to a single test or you can apply them to all tests within a test suite. In test suites, the Synchronization Point name must be the same across the suite, but the At Step can be different. Multiple test scenarios (and test suites) must be set to run in parallel, since

serial tests will not be able to hit the Synchronization Point at the same time by definition. Multiple Synchronization Points can be defined in a test case or test suite.

The Create a Synchronization point Companion Editor is shown below:



To configure a Create a Synchronization Point Companion, enter the following parameters:

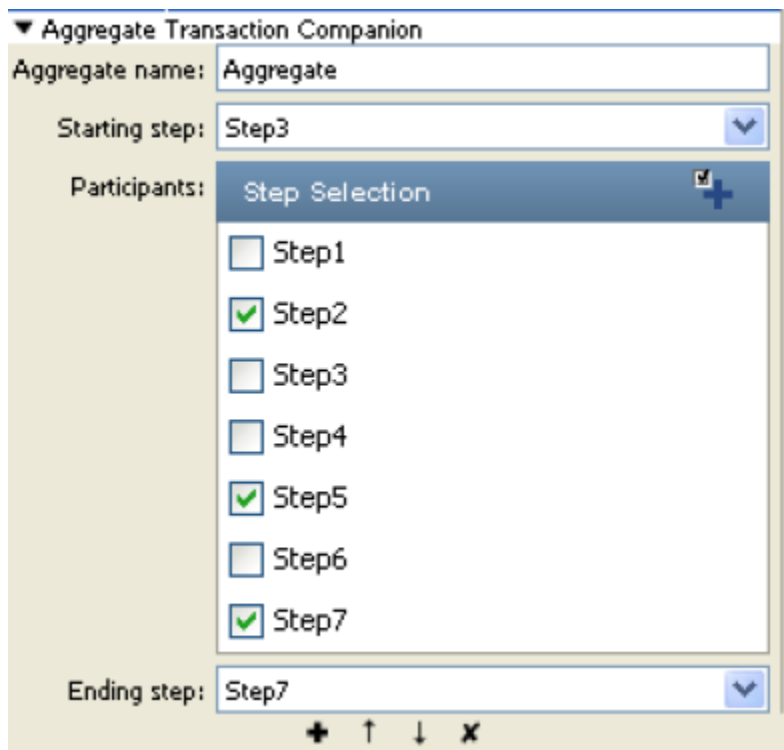
- **Sync Point Name:** The name you specify for the Synchronization Point
- **At Step:** Select the step for the Synchronization Point from the drop down list. Virtual users will pause before executing this step.
- **Time out secs (0 for none):** The number of seconds to wait for synchronization to occur. All virtual users must reach the **At Step** before the time out period elapses

33.6 Set up an Aggregate Step

33.6 Set up an Aggregate Step

The Aggregate Transaction companion allows you to aggregate and report several physical test steps as one logical step for metrics collections and reporting purposes. Unless you select the **Quiet** option on a step, LISA also collects metrics and report events for the individual steps in an aggregate. Multiple aggregation points can be set.

The Aggregate Transaction Companion Editor is shown below:



To configure the Aggregate Transaction companion, enter the following parameters:

- **Aggregate Step Name:** The name of the aggregation step (spaces are allowed)
- **Starting Step:** Select the start (first) step of the aggregation from the pull down menu
- **Participants:** Check the steps to include in the aggregate (exclude the Starting and Ending steps)
- **Ending Step:** Select the end (last) step of the aggregation from the pull down menu

The **Add** icon can be used to select or deselect all test steps.

All reports shows the aggregate step and any individual steps that are not set to quiet.

This can be seen in the example that follows where the aggregate step is named **WebAggregate**. It combines the response time of the "new user" step and the "add user" step into a response time for the new WebAggregate step. This can be viewed in the Test Events tab of the Interactive Test Run (ITR) by selecting the final step in the aggregation and searching for the Aggregate step name in the short info.

Test Events tab in ITR
Performance Summary
Performance Summary Graph

33.7 Other Companions

33.7 Other Companions

Other Companions has following options under it:

[33.8 Create a Sandbox Class Loader for Each Step.](#)

[33.9 Set Final Step to Execute.](#)

[33.10 Negative Testing Companion.](#)

[33.11 Fail Test Case Companion.](#)

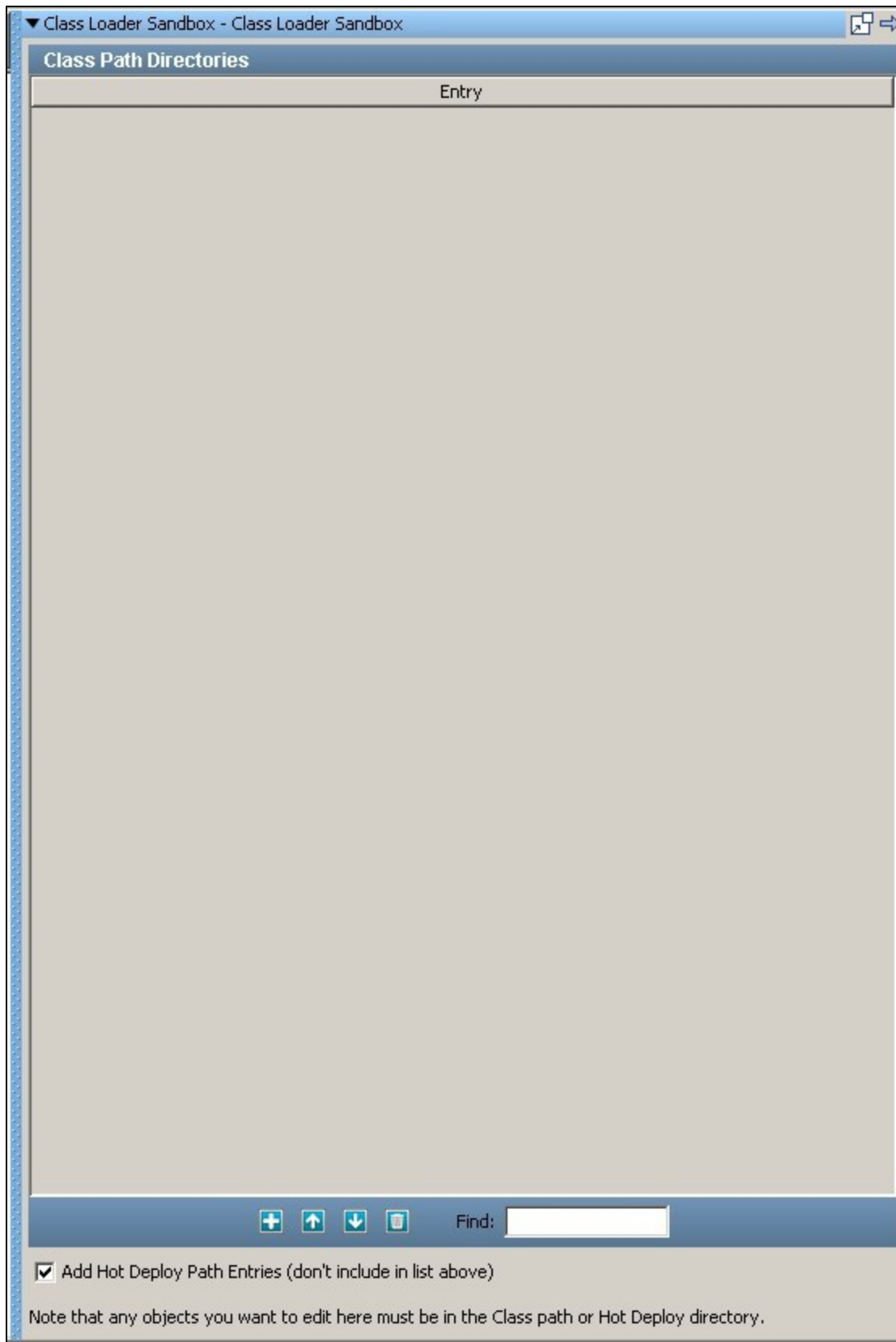
[33.12 XML Diff Ignored Nodes Companion.](#)

33.8 Create a Sandbox Class Loader for Each Step

33.8 Create a Sandbox Class Loader for Each Step

The Class Loader Sandbox companion allows you to ensure that every test run executes in its own Java Class loader (JVM). This is valuable when testing local Java objects that are not designed for multi-threaded or multi-user access. Most testing will not require this companion; it is usually only necessary when testing local Java objects with the Dynamic Java Execution step.

The Class Path Sandbox companion editor is shown below:



To configure the Class Loader Sandbox companion do the following:

- If the class you want to run is already in the **hotDeploy** directory, check the **Add Hot Deploy Path Entries** checkbox
- If the class you want to run is not in the **hotDeploy** directory, then use the **Add** buttons to add a line in the Class Path Directories list and add the appropriate class path for the class

Any Java objects you want to edit or run...

- Must be in the class path or the **hotDeploy** directory.
- Must not be in the LISA **lib** or **bin** directories. The Class Loader Sandbox will not work due to the way the Java VM loads classes

33.9 Set Final Step to Execute

33.9 Set Final Step to Execute

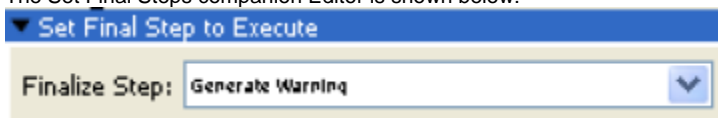
The Set Final Steps to Execute companion allows you to ensure that the system under test is left in a consistent state regardless of the outcome of the test. You specify which step are always run last when the normal test flow is circumvented.

A common use is to make sure that resources are released at the end of the test. Specifying the first step is not commonly needed, but there are many scenarios where specifying the final step is important.

LISA will execute the Final step even if the test case reaches an End step or the test case is instructed to end abruptly.

The Final step is not shown in the **Execution History** list in the ITR, but the results can be seen in the **Events** tab for the last step executed.

The Set Final Steps companion Editor is shown below:



To configure the Set Final Steps companion enter the following parameters:

- Finalize Step: Select the final step to execute from the pull down menu.

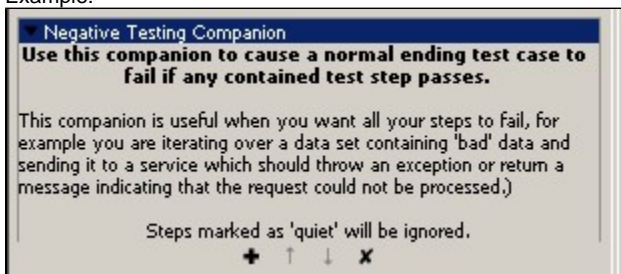
33.10 Negative Testing Companion

33.10 Negative Testing Companion

When you want all your steps to fail, this companion is useful.

Use this companion to cause a normal ending test case to fail if any contained test step passes.

Example:



When iterating over a data set having bad or corrupted data & sending it to a service which should throw an exception or return a message indicating request could not be processed.

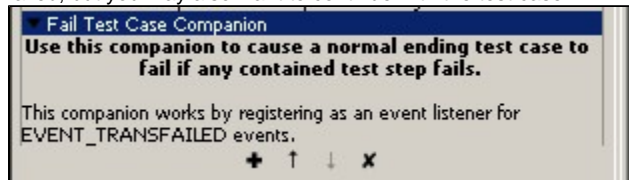
There are no configuration parameters for this companion.

33.11 Fail Test Case Companion

33.11 Fail Test Case Companion

The Fail Test Case companion will mark a test case that ended normally as failed if any of the test steps failed. This would typically happen when a failed test redirected to a step other than the fail step. An event listener for the EVENT_TRANSFAILED event is registered in this companion.

An example of this would be when you have a WSDL validation assertion on a web service step. You want to be informed that the validation failed, but you may also want to continue with the test case.

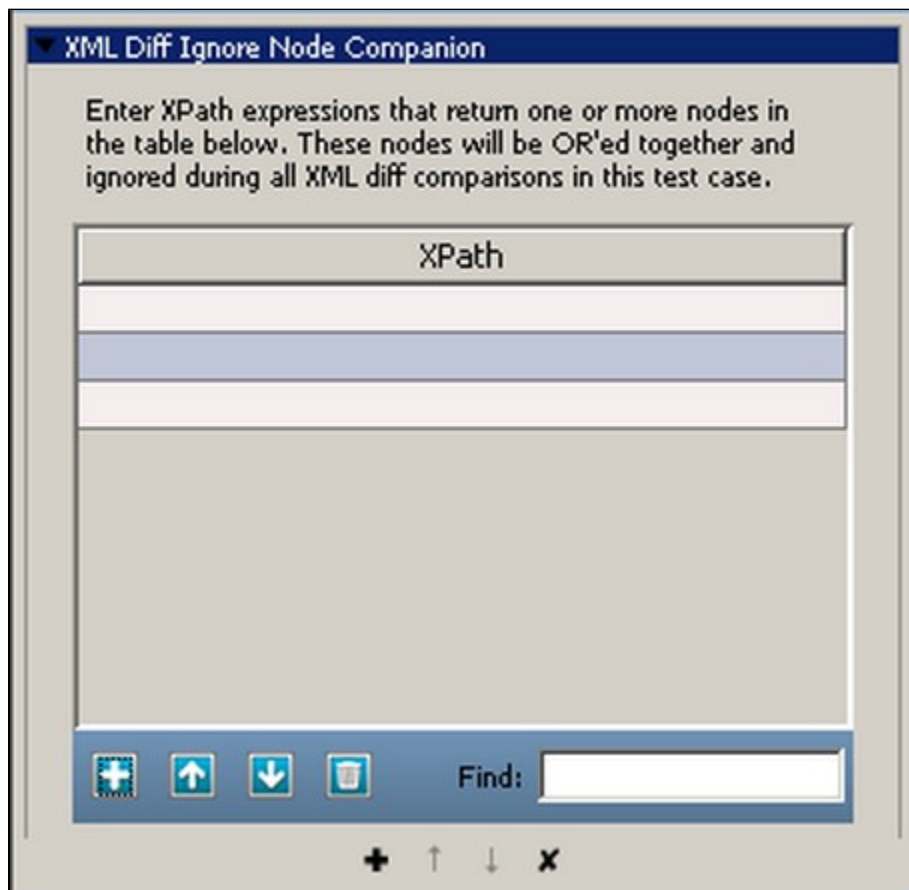


There are no configuration parameters for this companion.

33.12 XML Diff Ignored Nodes Companion

33.12 XML Diff Ignored Nodes Companion

Enter XPath expressions that return one or more nodes in the table below. These nodes will be 'OR'ed together and ignored during all xml diff companions during this test case.



PART 7 - Events

PART 7 - Events

The following chapter is available..

[34. Events](#)

34. Events

34. Events

The following two sub topics are available.

[34.1 Test Events](#)
[34.2 LISA Test Events Table](#)

34.1 Test Events

34.1 Test Events

An Event is a message broadcast from LISA informing any interested parties, including you, that an action occurred. LISA creates events for every major action that occurs in a test case. An event is created every time a step is executed, a property is set, a response time is reported, a result is returned, a test succeeds or fails and more. LISA provides you the ability to see every event or to filter out the events that are not of interest.

Events are important to you when you are monitoring tests, or analyzing test results. You can observe events during an Interactive Test Run (ITR), by clicking the Test Events Tab in the ITR, as the following figure shows:

▼ Interactive Test Run - 1

Execution History

- Step1
- End the Test

Response Properties Test Events

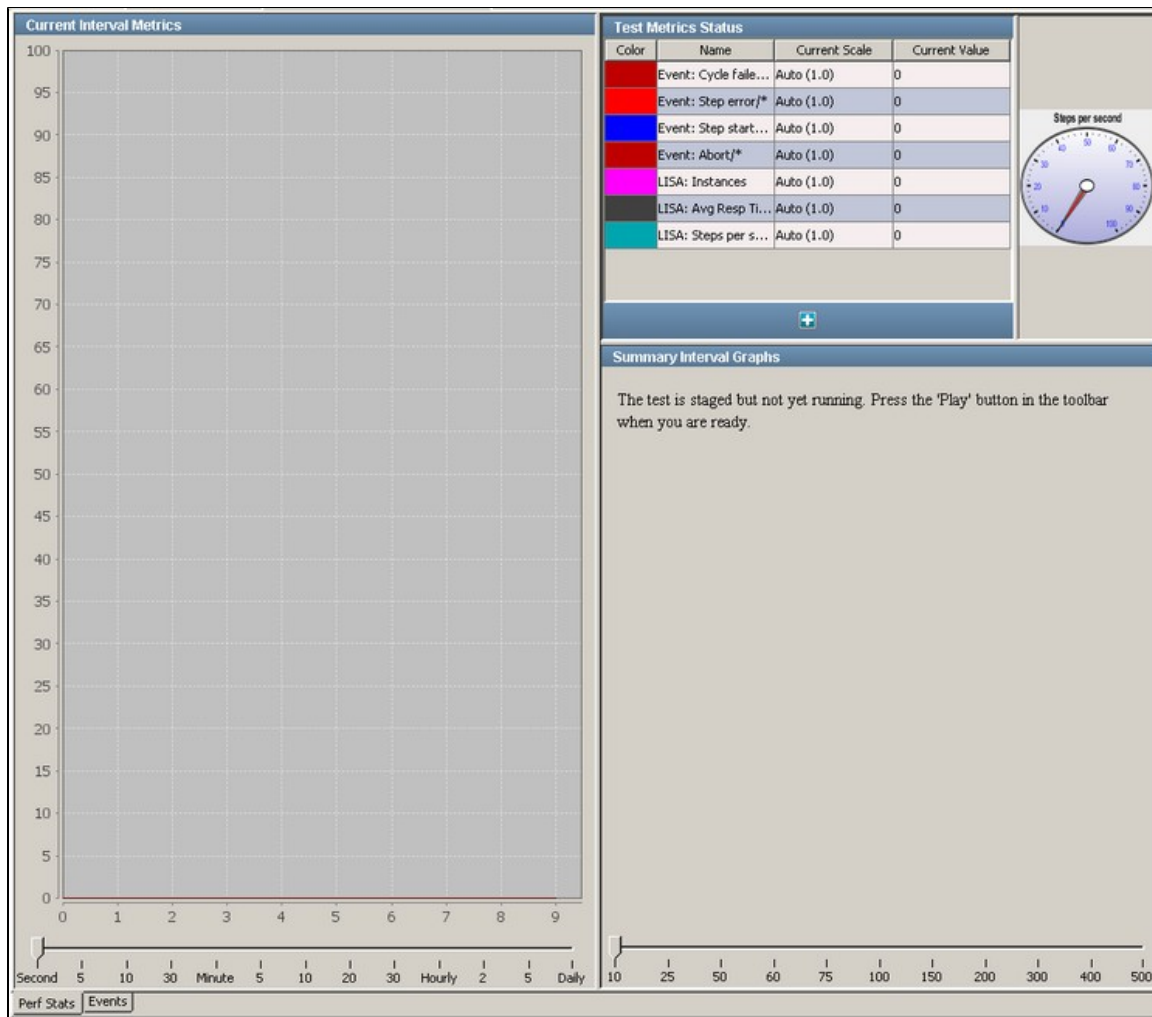
EventID	Timestamp	Short	Long
Property set	Fri Apr 09 16:36:...	Step response	true
Step started	Fri Apr 09 16:36:...	Step1	
Property set	Fri Apr 09 16:36:...	isa.http.persistentH...	
Property set	Fri Apr 09 16:36:...	isa.Step1.http.resp...	302
Property set	Fri Apr 09 16:36:...	isa.Step1.http.head...	Locatio...
Property set	Fri Apr 09 16:36:...	isa.Step1.http.resp...	200
Property set	Fri Apr 09 16:36:...	isa.Step1.http.requ...	http://...
Property set	Fri Apr 09 16:36:...	isa.Step1.http.head...	Date=F...
Step request	Fri Apr 09 16:36:...	Step1	GET / H...
Step target	Fri Apr 09 16:36:...	Step1	http://...
Info message	Fri Apr 09 16:36:...	Step1	Reques...

Long Info Field

Location=http://www.google.co.in/&Cache-C...

Run Settings

You can observe and filter events in a Quick Test, as shown below:



The same can be done while monitoring a staged test or a test suite.

You can select events to be included in reports, and, select events to be used as metrics that can be monitored and included in reports.

For more information on LISA reports, see Chapter 31 Reports in the User Guide. For more information on metrics, see Chapter 32 Metrics in the [LISA User Guide](#).

Note: Internal to LISA, a step may also be referred to as a node, explaining why some events have node in the EventID.

An event is characterized by an EventID, a Short value, and a Long value. EventIDs are keywords that indicate the type of event. The Short values and Long values contain information about the event; their contents will vary with the type of event.

The table that follows, displays the EventID, a description of the event, and an explanation of the Short and Long values - for each test event type. The events are listed in the same order as they appear in the event lists in LISA. An example of the event list can be seen in one of the figures above.

34.2 LISA Test Events Table

34.2 LISA Test Events Table

Event ID	Description	Short	Long
EVENT_COORDSERVERCREATED	The Coordinator Server was created	The name of the Coordinator Server	
EVENT_REMOVECOORDSERVER	The Coordinator Server named in the short description ended	The name of the Coordinator Server	
EVENT_NEWCOORDINATOR	A new Coordinator named in the short description was created	The name of the Coordinator Server	

EVENT_REMOVECOORDINATOR	The Coordinator name in the short description was removed	The name of the Coordinator Server	
EVENT_TESTSTARTED	The Coordinator started the test	The name of the test case	
EVENT_TESTENDED	The Coordinator stopped the test	The name of the test case	
EVENT_VUSERSSTART	The Simulator created an instance	The name of the Simulator	
EVENT_VUSERSEND	An instance in the Simulator finished	The name of the Simulator	
EVENT_SIMSTARTED	A Simulator started	The name of the Simulator	
EVENT_SIMEND	A Simulator ended	The name of the Simulator	
EVENT_INIT	The test was initialized (staged)	The name of the test case	
EVENT_START	The test instance and cycle started	The name of the test case	
EVENT_NORMALEND	The test execution completed in a successful state	The name of the test case	
EVENT_TESTFAILED	The test execution failed (usually means that it reached the 'failed' step)	The name of the test case	
EVENT_SETPROP	A LISA property was set	Name of the property	Value of the property as a string
EVENT_TRANSACTION	A step is being executed (transaction is a synonym for step)	The name of the step	
EVENT_ASSERT	An assertion on a step 'fired'	The name of the assertion	The log message of the assertion, or a LISA-generated message if there is no log message set
EVENT_NODERESPONSE	The step executed and a response was generated. It is displayed in Long field.	The name of the step	The response data as a string
EVENT_RESPTIME	The time to execute the step. This does not include think time or LISA overhead.	The name of the step	The number of milliseconds to execute the step
EVENT_BANDWIDTH	Approximate amount of data sent and received from the system under test for the step execution	The name of the step	Actual number of bytes read/received
EVENT_TRANSFAILED	An error has occurred in the system under test. For example, no response from a web server. This is on a per step basis. The EVENT_ TESTFAILED refers to the complete test case.	The name of the step	If available, a message to help determine the cause of the failure
EVENT_LOGMSG	Basic logging data that LISA considers valuable to record, but can be turned off, when filtering events, to minimize overhead	The message sent to the log	
EVENT_NODEMSG	Basic logging data that LISA considers valuable to record. For example, the HTTP/HTML request step will send this message with the step name in the short field and the URL being sent to the server in the long field	Usually the name of the step during which this message was generated	Usually a message that LISA generated
EVENT_TESTDEFERROR	A test case error was discovered during execution of the test. For example, the name is constructed from a property that does not exist	Varies but it is usually the name of the test element (i.e. step, data set, or filter) that has the error	Usually a message that explains the error

EVENT_STOPTESTSIGNAL	The instances have been instructed to stop testing		
EVENT_TESTRUNERROR	A LISA error occurred. For example, the Coordinator lost its connection to a Simulator while running a test	Varies but it is usually the name of the test element (i.e. step, data set, or filter) that has the error	Usually a message that explains the error
EVENT_REQUEST	Steps that support this event use it to report the actual request made to the system under test	The name of the step	The request data as a string
EVENT_CALL	Steps that perform object calls like Web Services or EJBs use this event to report each call that is made on the object	The name of the step	The call as a string, like void addUser([string] Joe, [string] mypass)
EVENT_CALLRESULT	Steps that perform object calls like Web Services or EJBs use this event to report the response they get from calls	The name of the step	The call response as a string. If the response is an object, an XML view of the object is presented
EVENT_SUITE_STARTING	A suite is starting to run	The name of the suite	
EVENT_SUITE_TESTSTAGED	When a test is staged to run as part of a suite	The name of the suite	The name of the test, path to the test, and other information
EVENT_SUITE_TESTPASSED	When a test running as part of a suite ends successfully	The name of the suite	
EVENT_SUITE_TESTFAILED	When a test running as part of a suite ends in failure	The name of the suite	
EVENT_SUITE_FINISHED	When all the tests running as part of a suite have finished, the suite is completed	The name of the suite	
EVENT_SUITE_SKIPPED	When a setup test (defined in a suite document) has failed, then the suite will not run the tests defined in the suite. This event informs you that this has happened	The name of the suite	
EVENT_SUITE_SETUPTEARDOWN	The suite has a setup or teardown test defined and that test has just started to run	The name of the suite	The name of the test, path to the test, and other information
EVENT_METRIC ALERT	A metric has been collected and is reporting its value	The short name of the metric, like LISA: Avg Response Time	The value of the metric collected
EVENT_INTEGRATION	The system being tested has LISA integration enabled. This event contains the LISA integration XML data that was captured	The name of the step	The XML representation of the LISA Integration data captured

PART 8 - Appendix

PART 8 - Appendix

Several LISA elements have more than one name. There is the Wizard Name that appears on the Wizards panels for each element, and the Short Name that appears in pull-down menus, and in the Test Case Tree.

The Reference Guide uses the Short Name.

To help you find the element you are looking for in the Reference Guide, the following table shows the elements that have more than one name, and lists the Wizard Name and the Short Name.

The following chapter is available.

35. Appendix_ Alternative Wizard Names

35. Appendix_ Alternative Wizard Names

The following topic is available.

35.1 Alternative Names Table

35.1 Alternative Names Table

35.1 Alternative Names Table

Element	Wizard name	Short Name
Assertions	Check for any non empty response	Any Non-Empty result
Assertions	Check for a given string in response	Result as String Contains Given String
Assertions	Check the response for a given regular expression	Result as String Contains Expressions
Assertions	Make diff-like comparison on the text result	Highlight Content for Comparison
Assertions	Make diff-like comparison on the HTML result	Web HTML Text Assertion
Assertions	Check HTML for properties in Page	Assert on Properties from HTML Result
Assertions	Assert HTTP header field with expression	HTTP Result Header field
Assertions	Check HTML response code with expression	HTTP Response Code
Assertions	Check size of JDBC result set	Result Set General
Assertions	Look for value in JDBC result set	Result Set Contents
Assertions	Check size of JDBC result set	JDBC Result Set General
Assertions	Look for value in JDBC result set	JDBC Result Set Contents
Assertions	XML path expression	XML Xpath Assert
Assertions	XML Validity check	XML Validation
Companions	Configure LISA to use a Web Proxy	Web Proxy Setup
Companions	Web Browser Simulation	Web Browser Selection
Companions	Read Properties from a File	External Property Reader
Companions	Browser Bandwidth Simulation	Bandwidth Simulation
Companions	Create a Sandbox loader for each Test	Class Loader Sandbox
Companions	Scan a File for Content	Tail File for Expression
Companions	Set up a Synchronization Point	Create a Synchronization Point
Companions	Set up an Aggregation Step	Aggregate Transaction Companion
Datasets	Read rows from a delimited data file	Read Rows From delimited file
Datasets	Read Rows from JDBC table	Read Rows From JDBC result set
Datasets	Create numeric counting data set	Generate Numeric values (Counter)
Datasets	Create your own Data Sheet	Data Sheet

Datasets	Read Rows from Excel file	Read rows from Excel file
Datasets	Read data objects from Excel file	Read DTOs from Excel file
Filters	Parse text for properties	Pass Text for Values
Filters	Save this steps response/result in a property	Save Step Response As Property
Filters	Retrieve property and set as last response	Convert Property value into Last Response
Filters	Save the value of a property to a file	Save Property Value to File
Filters	Retrieve JDBC result set value and store in a property	Parse JDBC Result Set for Value
Filters	Install the simple JDBC filter	Simple JDBC Result Set Filter
Filters	Retrieve JDBC result set value and store in a property	Parse result Set for Value
Filters	Install the simple JDBC filter	Simple Result Set Filter
Filters	Filter for value and get a second columns value ?????	Get Value for Another Value in ResultSet Row
Filters	Install the web response link checker	Check links on Web Responses
Filters	Parse HTML for properties	Parse Web page for Properties
Filters	Parse HTML for certain values	Parse HTML Result for Specific Tag/Attributes Value
Filters	Parse HTML for a certain attribute and parse that attribute	Parse HTML Result for Specific Tag/Attributes Value and Parse it
Filters	Parse HTML for certain tags child text	Parse HTML Result for Tag's First Child text
Filters	Get HTTP header value	Parse HTML Result HTTP Header Value
Filters	Parse HTML for a certain attribute and parse that attribute	Parse HTML Result for Attributes Value
Filters	Install the LISA properties filter (web)	Parse HTML Result for LISA tags
Filters	Randomly pick a child tag from a list and parse it	Parse HTML Result and Select Random Attribute Value
Filters	Parse HTML for a list of tags	Parse HTML into List of Attributes
Filters	Get XML tags child text	Parse XML Result for Tag's First Child text
Filters	Get XML tags value	Parse XML for Specific Tag/Attribute Value
Filters	Scan XML for LISA tags	Parse XML result for LISA tags
Filters	Select a random child tag of a given tag and get its attribute	Parse XML Result and Select Random Attribute Value
Filters	Install the web testing filter	Simple Web Filter
Filters	Execute an XPath query to filter XML from this document	XML XPath Filter
Test Steps	Execute a Web Service via Soap	Web Service Execution
Test Steps	Launch embedded web server	Start or Stop Web Server
Test Steps	Web Service Simulation	Simulate a Web Service
Test Steps	Invoke calls on a local Java object	Dynamic Java Execution
Test Steps	RMI Server execution	RMI Server Execution
Test Steps	Execute an Enterprise Java Bean (EJB)	Enterprise Java Bean Execution
Test Steps	Execute a SQL call on a database	SQL Database Execution (JDBC)
Test Steps	Execute a Command Line Application	Execute External Command

LISA User Guide

LISA User Guide

The User Guide online documentation is divided into six parts:

PART 1 - Getting Started: This section provides a first look at LISA Workstation, the central hub of any LISA installation, whether you are running just a single workstation or a large distributed LISA Server environment. In addition there is information on the Anatomy of a Test Case and some tutorials to help get started in LISA.

PART 2 - Building Test Cases: This is where you find out how to build LISA Projects and Test Cases. It starts with a description of creating a LISA Project, LISA Test Case, and then goes on to show how to build individual test steps, and add LISA elements to the Test Case and the test steps.

PART 3 - Recorders & Test Generators: In addition to building Test Cases from the ground up, LISA provides many tools to automate or semi-automate the creation of a Test Case. These tools range from recorders that can follow your actions through a system and produce a Test Case for you, to test smart test generators that can build a Test Case for you based on some basic information that is made available to LISA. For example, if you have the Web Service Definition Language (WSDL) file from a web service, LISA can build a Test Case to test that web service.

PART 4 - Running Test Cases: LISA provides some utilities within the Workstation that facilitates to run Test Cases, not as production tests, but more to validate and tune your Test Case.

PART 5 - Starting, Staging, & Running Tests: This section has information about Staging and running individual tests and test suites, in the workstation environment and on LISA Server.

PART 6 - LISA Advanced Features: This section deals with advanced features and explains ways for Java developers to customize and extend LISA, and use it in ways that do require knowledge of Java.

Several components require parameters that must be obtained from people knowledgeable about the System Under Test (SUT). This information is identified throughout the guide. You will rarely be able to proceed with building and running the component without this information.

PART 1 - Getting Started

PART 1 - Getting Started

Welcome to TKO's LISA™ Users Guide!

Introduction to TKO's LISA™

iTKO's LISA™ Test is a complete and collaborative automated testing solution.

LISA provides complete test coverage, with the ability to invoke and verify the behavior of each component across the end-to-end application. LISA provides automated testability for all of the components in the technology stack.

LISA also builds portable, executable test cases that are easy to extend, easy to chain into workflows with other tests, and simple to integrate with existing test repositories. LISA test cases are designed to be shared across different teams and environments, with the ability to easily attach prior results and artifacts to extend them, and the ability to readily execute with different underlying data.

User Guide Structure

The User Guide is structured into many Parts. All the Parts of the Users Guide describe important sections of TKO's LISA™.

You should ideally start with Part 1 - Getting Started section. You will find introductory type information and tutorials to get you started in LISA.

Later on you can go to each Part, where the Part Title describes the kind of information included in it.

In this section, the following topics are covered:

1. LISA Basic Concepts
2. LISA Workstation Overview
3. Anatomy of a LISA Test Case
4. LISA Tutorials

1. LISA Basic Concepts

1. LISA Basic Concepts

This section describes basic concepts that you must understand in order to use LISA effectively. The following terminology is used extensively while working within LISA.

LISA Project

A LISA Project is the root of all Test cases. A LISA Project contains all the necessary testcases, suites, configuration files, Data Sets, Staging documents, Virtual service documents etc attached to it. It contains a single or a number of Test Cases, which in turn contain several Test Steps. In LISA Workstation only one Project can be opened at a time.

For more information regarding LISA Projects, see [Creating a LISA Project](#).

Test Step

A Test step is an element in the LISA Test Case workflow that represents a single test action that is to be performed. There are different types of Test Steps within LISA, Ex: Web Services, Java Beans, JDBC, JMS Messaging to name a few. A Test Step can have LISA elements - Filters, Assertions, Data Sets attached to it.

For more information regarding LISA Test Steps, see [Building Test Steps](#).

Test Case

A LISA Test Case contains single or many Test Steps. It is a complete specification of how to test a business component in the 'system under test', or in some cases the complete 'system under test'. A Test Case consists of a collection of one or different Test Steps, Data Sets, Assertion and Filters, applied to test steps within it.

For more information regarding LISA Test Cases, see [Building Test Cases](#)

Test Suite

A Test Suite consists of a group of Test Cases and/or other Test Suites that are scheduled to execute one after other. A Test Suite Document specifies the contents of the suite, the reports to generate, and the metrics to collect.

For more information regarding Test Suites, see [Building Test Suites](#) .

LISA Registry

All the major services within LISA Workstation, like the Reporting database, VSE database, Pathfinder Agent and other services, are provided through the LISA Registry.

A LISA Registry is mandatory for LISA Workstation.

You can choose from either the **LISA Registry** or the **Local Registry**.

For more information, refer to [LISA Registry](#) section in the Install Guide.

LISA Properties

LISA uses Properties to store data. These are **Name (key) – value pairs**, where the values are obtained by specifying the key. Properties can store many different kinds of data. At runtime, test steps can retrieve the values of these properties given the names, and set them.

For more information about Properties, see [Creating Properties](#).

Configuration

LISA Configuration is a collection of properties, available at the beginning of a LISA Project. One may look at a configuration as a place to set the various location, port and similar configuration parameters required for Test Case execution, so as to avoid hard coding of these parameters at any other place. There can be several configurations defined for a LISA Project, identified by unique names. Only one of the Configurations can be used at the time for Test Case execution. Thus they provide alternative contexts for the execution of a Test Case.

For more information, see [LISA Configurations](#).

Staging Document

Staging Document is a document in LISA, that contains information about "how" to run the test. The Staging documents has details like Run name, number of Instances, Test Cycles, Reports and Metrics to be generated etc.

For more information see, [Building Staging documents](#)

Filters

A Filter is a LISA code element that runs before and after a Test Step. This gives you the opportunity to process the data in the result, or store values in properties. Global Filters defined in a Test Case apply to each test step in the Test Case.

For more information regarding LISA Filters, see [Adding Filters](#).

Assertions

An Assertion is a LISA code element that runs after a test step and all its Filters have run. They are used to verify that the results from running the step match expectations. An Assertion is typically used to alter the flow of the Test Case execution. Global Assertions defined in a Test Case apply to each test step in the Test Case.

For more information regarding LISA Assertions, see [Adding Assertions](#).

Data Sets

A Data Set is a collection of values that can be used to set properties in a Test Case while a test is running. This provides a mechanism to introduce external test data into a Test Case.

For more information regarding Data Sets, see [Using Data Sets](#).

Companions

A Companion is a LISA element that runs before and after every Test Case execution. They can be understood as Filters applicable to the entire Test Case as opposed to individual test steps. Companions are used to configure global (to the Test Case) behavior in the Test Case.

For more information regarding Companions, see [Using Companions](#).

Events

An Event is a message broadcast from LISA informing about an action has occurred. LISA creates Events for every major action that occurs in a Test Case. These Events can be configured at the Test Case level.

For more information on Events, See [LISA Events](#).

Metrics

Metrics allow you to apply **quantitative methods** and **measurements** to the performance and functional aspects of your tests, and the "system under test".

For more information on Metrics, see [LISA Metrics](#).

Interactive Test Run

Interactive Test Run, is a utility in LISA that allows you to execute a Test Case step by step. With ITR, you can modify the Test Case at run time and re-run to check the results.

For more information on ITR, see [Running Interactive Test Run](#).

Quick Stage Run

Staging a Quick Test, is a utility in LISA that allows you to run a Test Case with minimal set up. To stage a Quick test, you need to have a Test case and a Staging document.

For more information on Quick Tests, see [Staging a Quick Test](#).

LISA Utilities / Portals

LISA Reports - LISA Report viewer is a tool to view the generated LISA 5.0 reports. For older reports, Legacy report viewer can be used.

For more details, see the [Reports](#) section.

Pathfinder - Pathfinder is a utility in LISA, which allows you to probe into the system under test, to examine the components behind the initial request or method call.

For more details, see the [Pathfinder](#) section.

CVS Dashboard - The Continuous Validation Service (CVS) utility within LISA, allows you to schedule Tests and Test suites to run on a regular basis, over an extended time period.

For more information, see the [CVS Dashboard](#) section.

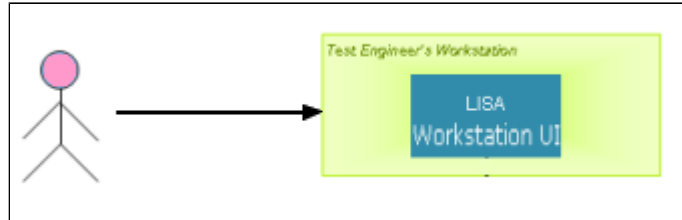
2. LISA Workstation Overview

2. LISA Workstation Introduction

LISA provides a single, easy to use, integrated environment - **LISA Workstation** for developing, staging and monitoring tests, with almost no coding required.

You can either work in a LISA Standalone version or work in a LISA Server Environment. For more information, refer the [LISA Installation Guide](#). Depending on the installation, the LISA components will be installed.

In **LISA Workstation** the tests are also managed and run within the Workstation environment. LISA Workstation is the only application you will



use to create, build and run Test Cases in LISA.

In **LISA Server** the tests are run in the Server environment. The Workstation then connects to the Server to deploy and monitor tests that were developed in LISA Workstation.

The Workstation is one of four applications that make up the LISA Server environment. In addition to the Workstation, there is at least an instance of each of the following:

- **Test Registry:** A registry that keeps track of all the Coordinator and Simulator Servers. LISA Workstation attaches to this test registry.
- **Coordinator Server:** Receives the test run information in the form of documents, and coordinates the tests that are run on one or more Simulator Servers. The Coordinator Server manages Metric collection and Reporting. A LISA Server environment can have only one Coordinator Server.
- **Simulator Server:** Runs the tests under the supervision of the Coordinator Server. The Simulator Servers instantiate virtual users to run test instances against the system under test (SUT). For large tests with many virtual users, virtual users can be distributed amongst several Simulator Servers.

Any number of Workstations can attach to the Test Registry and share the Server environment.

The descriptions and corresponding images in this chapter use several LISA concepts that are explained in detail throughout this online documentation.

For the purposes of understanding this chapter, below is a very brief description of these concepts.

- [2.1 Starting LISA Workstation](#)
- [2.2 LISA Workstation Main Menu](#)
- [2.3 LISA Workstation Toolbar](#)
- [2.4 LISA Quick Start](#)
- [2.5 LISA Workstation Views](#)

2.1 Starting LISA Workstation

2.1 Starting LISA Workstation

Once you install LISA and enter LISA License credentials, you are ready to work with LISA.

- Start the LISA Workstation from the **Start Menu > LISA 5.0 > LISA Workstation**.
- Or from command line by running **LISAWorkstation.exe**

Once LISA workstation is invoked, the first screen that pops up - is of setting LISA Registry.

Note - Post 5.0, LISA registry is mandatory for LISA workstation.



- You can choose from either a LISA Registry or local registry.

LISA Registry - Enter the name of the LISA Registry or select from drop down list.

Local Registry - Click to select the Local registry, which will start the local application server, which resides under <LISA_HOME>\webserver folder.

- Click **OK** to enter the LISA Workstation.

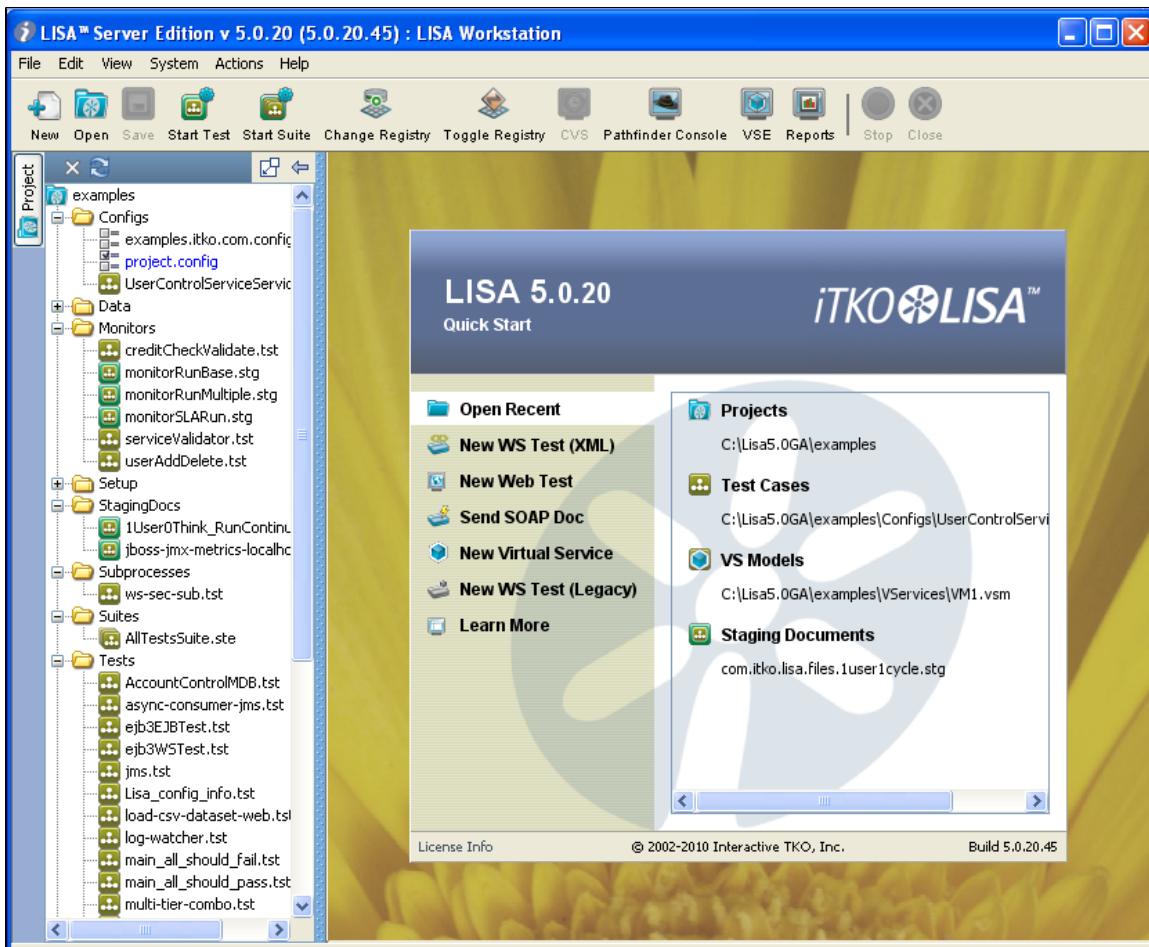
To Change the Registry,

- Open the Registry dialog from **Start Menu > LISA 5.0 > Registry**.
- Select the required registry from the drop down menu and click OK.

For more information on Registry, please refer to [LISA Registry](#) section.

LISA Workstation First Screen

This is the first screen when you open the LISA Workstation.



The screen is divided into two parts:

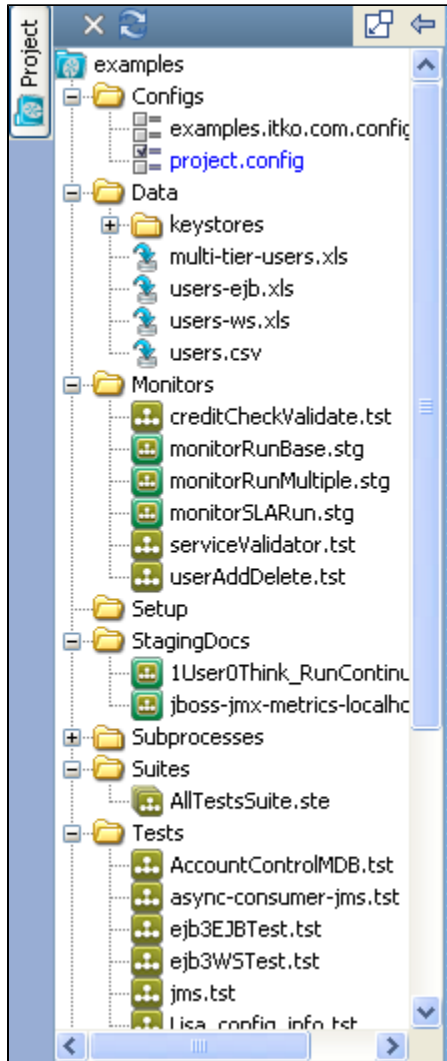
It has a **Project pane** on the left side and a LISA **Quick Start** menu on the right side.

Project Pane

For a new installation of LISA workstation, the "examples" project will open by default.

This project has various folders like Configs, Data, StagingDocs, Subprocesses, Suites, Tests and VServices which has default documents given by ITKO.

These folders contain all example samples that LISA supports. The samples contain examples of Test Steps, Test Suites, Configurations, Staging documents, Data Sets etc. You can see the same folder structure in the %LISA_HOME% directory where LISA is installed.



All the sample files and default files that come with the installation are installed in the individual folders as shown above.

Quick Start

The LISA Quick Start is described in detail in the [LISA Quick Start Menu](#) section.

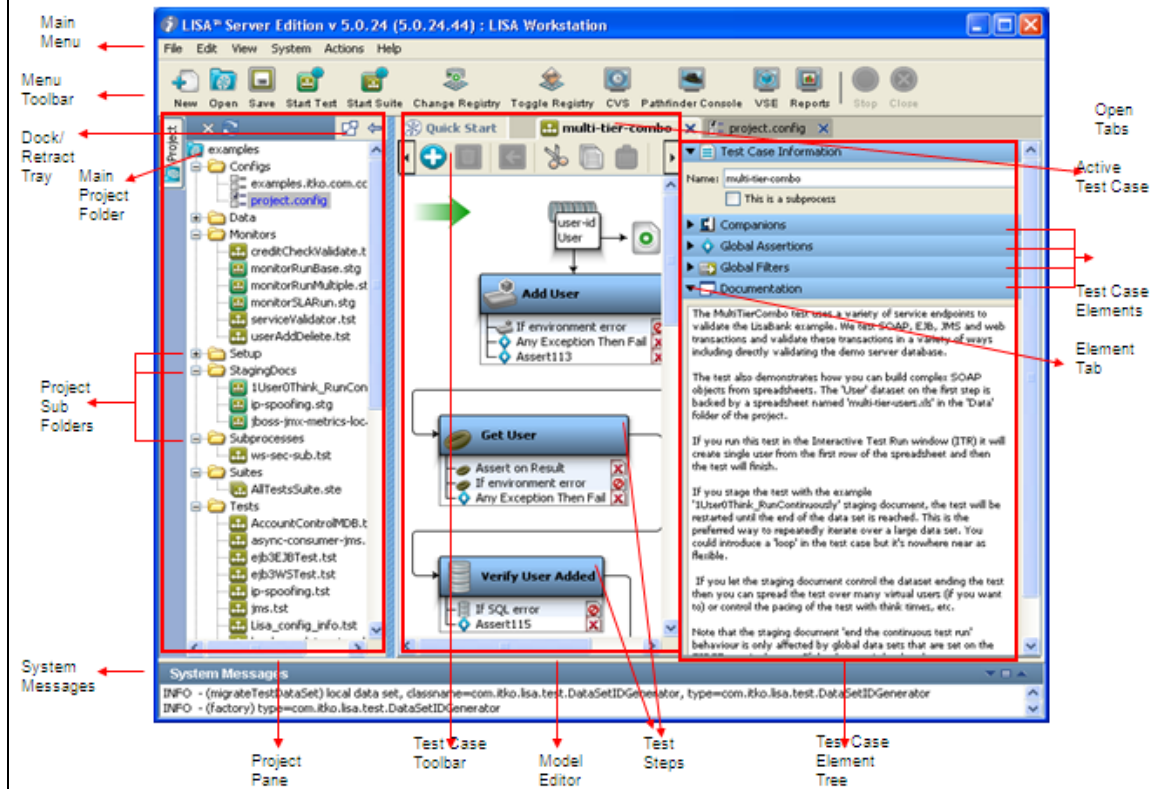
Getting Started

To get started with LISA Workstation for creating documents, you first need to create a LISA Project.

For more information about Creating a Project, see [12.1 Creating a LISA Project](#).

The figure below shows a detailed view of LISA Workstation with some tabs open:

LISA Workstation



The LISA Workstation User Interface is explained below:

Feature	Description
Main Menu	The LISA main menu appears once you login to LISA and remains there during all your interactions with LISA Workstation. Depending on the selected transactions, some menus maybe active or inactive.
Main Toolbar	The LISA main menu appears once you login to LISA and remains there during all your interactions with LISA Workstation. Depending on the selected transactions, some menus maybe active or inactive.
Project Tree/Pane	This is the Project pane. By default it is open when you start LISA. It is a dockable pane and can be clicked to open/hide the Project. At a time only one Project can be opened in LISA.
Project Main Folder	This is the main Project that is created. It is at the root of the project and has a project tree. It contains several subfolders for elements like – Configs, Data, Staging Docs, Suites, Tests, VServices - which are all part of this Project. You need to click on Project Pane to open or close the Project tree.
Project Subfolder	These are the subsets of the Project and appear once you open a Project. You can Right click on a folder to add that element into the Project. You can Add a Test Case, Staging Document, Test Suite, Configuration, VServices into a project. You can see the added items listed here under the respective folders.
Test Case Toolbar	The Test Case toolbar appears once you create or open a Test Case. Depending on the selected transactions, some menus maybe active or inactive.
Test Case Element Tree	This is on the right side of LISA Workstation and consists of all LISA elements/components which can be applied to a Test Case. All LISA elements like Filters, Assertions, Data Sets etc can be added, deleted or modified from here.
Model Editor	This is the LISA Workstation Test case Editor. This area will show a visual representation of your Test Case i.e. the steps added, Filters, and Assertions etc applied to a Test Case which forms a graphical workflow.
Element Toolbar	Each element in LISA has its own toolbar to add, edit, delete or reorder the elements within the Workflow. The toolbar is at the bottom of each element in the element tree. Once you open or expand an element, the toolbar appears at the bottom of the element.
System messages	System messages display relevant system messages and can be viewed at the bottom of the Workstation window. You can control the type of messages from the View main menu.

Each of these interfaces and navigational areas are discussed in more detail in the other chapters of this online documentation.

For the purpose of illustration, at many instances, we have shown the multi-tier-combo Test Case in the LISA examples directory (multi-tier-combo.tst).

The multi-tier-combo test case is explained in detail in the next section.

2.2 LISA Workstation Main Menu

2.2 LISA Workstation Main Menu

The LISA main menu has items for all the major functions available in the LISA Workstation.

These are the global functions. There are many more menus, usually drop down menus and toolbars, throughout the Workstation in the individual editors.

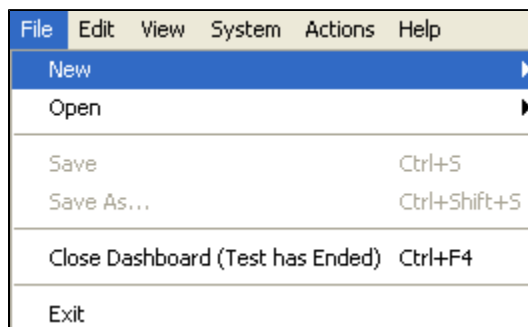
- 2.2.1 File menu
- 2.2.2 Edit menu
- 2.2.3 View menu
- 2.2.4 System menu
- 2.2.5 Actions menu
- 2.2.6 Help menu

2.2.1 File Menu

2.2.1 File Menu

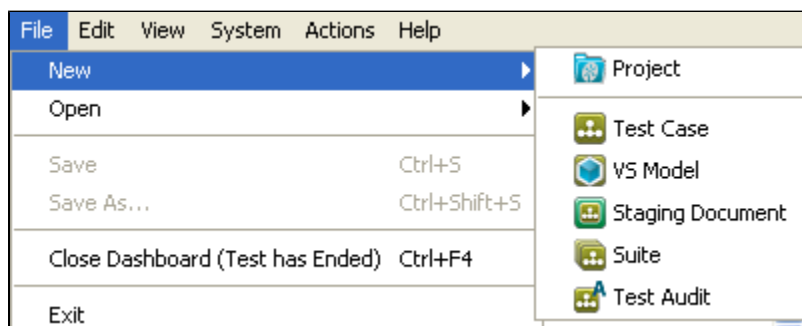
The File main menu has the standard items related to file manipulation.

File main menu



File -> New shows a secondary menu where you can create a LISA Project or create one of the five major test documents - Test Case, VS Model, Staging Document, Suite or Test Audit.

From this menu, we can create LISA documents within the project that's open in LISA Workstation or outside of the project. By default, the current directory selection is the **"Tests"** folder of the open project.



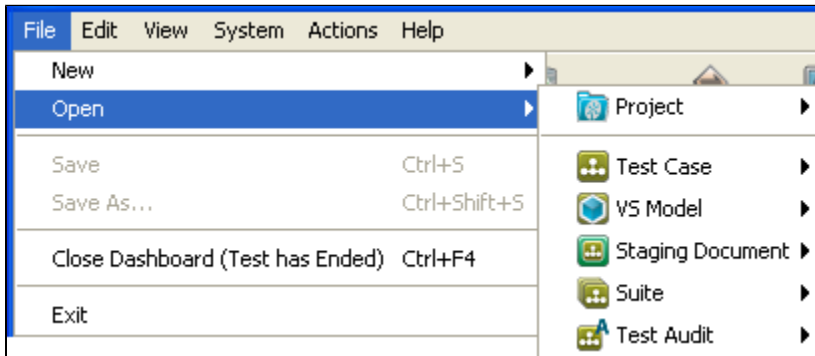
- **Project:** Creates a New Project. For more information refer to [See Creating Projects](#).
- **Test Case:** Creates a new Test Case.
- **VS Model:** Creates a new VS Model. A VS model is a specialized Test Case used in LISA virtualization.



Note: This is a licensed component and can be used by only those who have a license to this module.

- **Staging Document:** Creates a new Staging document.
- **Suite:** Creates a new Test Suite.
- **Test Audit:** Creates a new Audit Document.

File -> Open shows a secondary menu where you can open either an existing LISA Project or one of the five major LISA documents - Test Case, VS Model, Staging Document, Suite or Test Audit.



You can browse for the document to open, by looking in the file system, on the LISA classpath, or as a URL. LISA keeps track of the most recent documents you have opened, and lists them on the first page in the Quick Start menu.

- **Project:** Opens a Projects directory folder from where you can choose the project to open.
- **Test Case:** Opens a Test Case document
- **VS Model:** Opens a VS Model. A VS model is a specialized Test Case used in LISA virtualization.



Note: This is a licensed component and can be used by only those who have a license to this module.

- **Staging Document:** Opens a Staging document.
- **Suite:** Opens a test suite.
- **Test Audit:** Opens an Audit document.

You can open any LISA document like a Test Case/Staging document/VS Model etc either from a valid LISA Project or from outside a project, by using the main menu File > Open > menu option.

A non-project document (Ex: a test case) will open as it used to in earlier versions. If however, we choose a test case within a different project from this menu item, the current project will close, and open the project that the test case resides in, before opening the test case.

File -> Save: Saves the current document.

File -> Save As...: Saves the current document under a different name.

File -> View Reports: Displays a list of the reports that are available.

File -> Close: Closes the active tab.

File -> Exit: Exits from LISA Workstation.



Note: Be careful that you save your Test Case before exiting as there is no confirmation dialog box.

2.2.2 Edit Menu

2.2.2 Edit Menu

The Edit menu has the normal set of editing options.

Edit main menu

Edit	View	System	Actions
Cut		Ctrl+X	
Copy		Ctrl+C	
Paste		Ctrl+V	
Property Paste		Ctrl+Shift+V	

Edit -> Cut: Cuts the selected text

Edit -> Copy: Copies the selected text

Edit -> Paste: Pastes the selected text in the selected area

Edit -> Property Paste: Pastes the selected text in property format.








2.2.3 View Menu

2.2.3 View Menu

The View menu contains a list of menus which deal with the viewing of the Reporting portal, CVS Dashboard, VSE Dashboard, Pathfinder Console etc.

You can also set the look and feel of LISA Workstation here and visit the LISA portal for more information related to LISA.

View Main Menu

View	System	Actions	Help
	Reporting Console		
	CVS Dashboard		
	Pathfinder Console		
	VSE Dashboard		
	Virtual Service Images		
	LISA Portal		
	Toggle Zoom Panel	Ctrl+Shift+Z	
	Application Toolbar Settings		▶
	Model Editor Toolbar Settings		▶
	Class Path		

- **Reporting Console** : When selected invokes the LISA reporting console, wherein you can view various reports like Functional Report, Performance Report etc.
- **CVS Dashboard** : When selected, invokes the Continuous Validation Service utility, wherein you can schedule multiple tests to be run at scheduled intervals.
- **Pathfinder Console** : When selected invokes the Pathfinder console, wherein you can track the transactions and their paths.
- **VSE Dashboard** :When selected, invokes the Virtual Service Environment. It displays the currently registered virtual environments. For more information, see [LISA VSE Guide](#).

NOTE: VSE is a licensed component and will be able to use by only those who have a license to this module. For more information, contact ITKO support.

- **Virtual Service Images:** When selected, allows you to view the Service Images that are used in LISA Virtualization. For more information, see [LISA VSE Guide](#)
- **LISA Portal:**When selected, invokes the LISA portal.
- **Application Toolbar Settings:** Allows you to choose the interface of the Application Toolbar. You can select the toolbar settings to display Icons and Labels, Icons only, Small Icons or no toolbar at all.
- **Model Editor Toolbar Settings:** Allows you to choose between Large icons or Smaller icons to be displayed on the LISA Model Editor menu.
- **Class Path:** Allows you to view and set the class path.

From LISA 5.0, ITKO has added new Portal based application launchers to LISA Workstation.

Following applications can be launched through this portal:

- **LISA Pathfinder**
- **Reporting Dashboard**
- **CVS Dashboard**

To open the Portal, click <http://localhost:1505>.

Note - Registry needs to be running to start the Portal.

The following screen appears once the web page opens:



Click on the required icon to open it within the web browser.

2.2.4 System Menu

2.2.4 System Menu

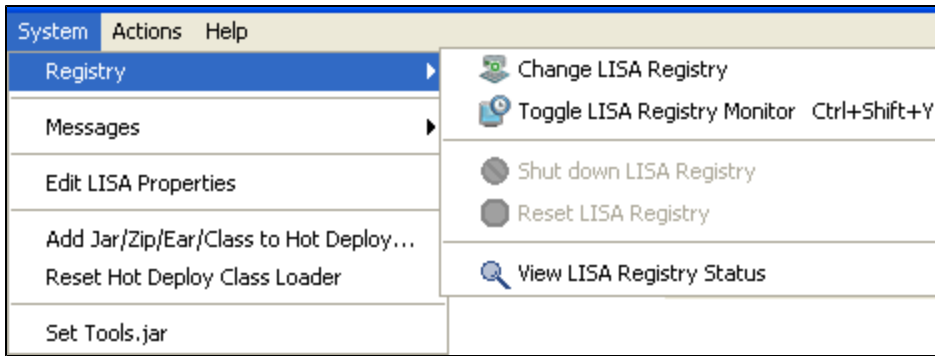
The System menu has a menu to manage the system registries and system messages.

You can also edit the LISA properties, add JAR or zip files to Hot Deploy or reset Hot Deploy Class Loader and set the Tools.jar file here.

Systems Main Menu

System	Actions	Help
Registry		►
Messages		►
Edit LISA Properties		
Add Jar/Zip/Ear/Class to Hot Deploy...		
Reset Hot Deploy Class Loader		
Set Tools.jar		

Registry: You can do all activities related to the LISA Registry in this menu.

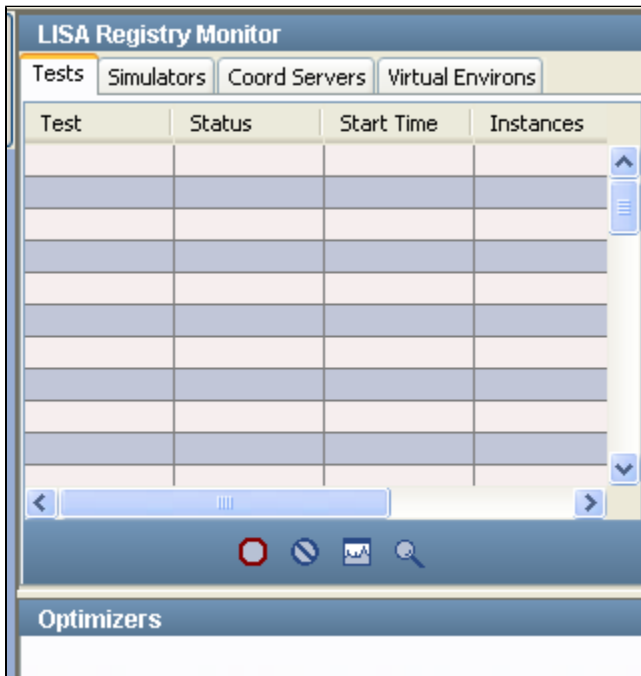


- **Change LISA Registry:** When selected, displays a dialog box listing all current test registries available in the LISA Server installation. This allows you to set the current test registry. It is uncommon, but possible to have more than one test registries running.



For more information, refer the [LISA Installation and Configuration Guide](#)

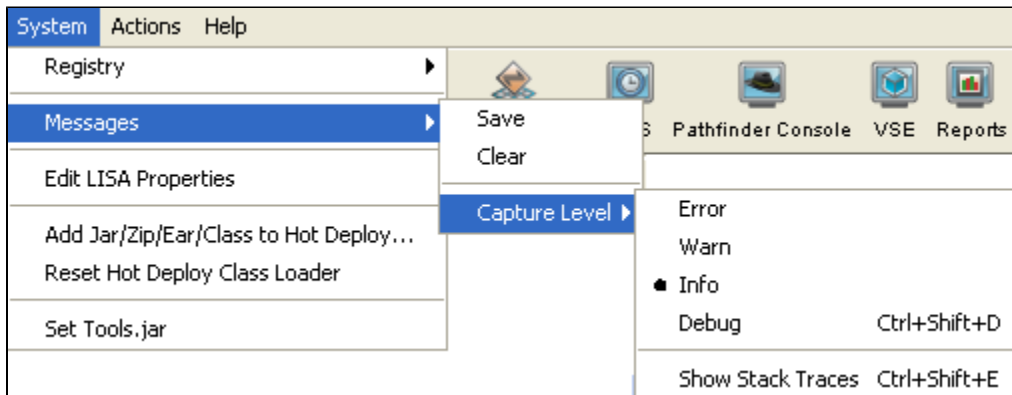
- **Toggle LISA Registry Monitor:** You can Toggle the LISA Registry Monitor, so that it can be viewed in the LISA Workstation.



- **Shut Down Test Registry:** When selected, shuts down the Test Registry.
- **Reset the Test Registry:** When selected, resets the test Registry which will reset the registry information in the system.
- **View LISA Registry Status:** When selected, displays a dialog box as shown below, listing the number of Coordinator Servers and Simulator Servers that are attached to the current LISA Registry.



- **Messages:** This menu is for the System Messages

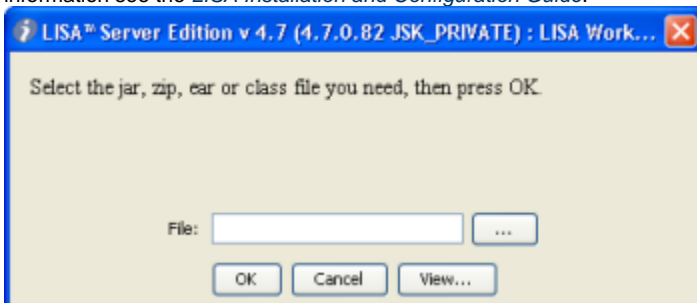


- **Save:** Click to Save the system messages.
- **Clear:** Click to clear the system messages
- **Capture Level:** Click and select the type of system messages to view: Errors, Warnings, Information or Debugging or show Stack Traces.
- **Edit LISA Properties:** Opens the **lisa.properties** file in an editable window.



Caution: Use extreme caution if you choose to edit this file.

- **Add Jar/Zip/Ear to Hot Deploy...:** When selected, displays a dialog box as shown below, to allow you to enter the name of a file that you want LISA to upload into the hot deploy directory. The file you upload contains items that you wish to add the Lisa's classpath. For more information see the [LISA Installation and Configuration Guide](#).



- **Reset Hot Deploy Class Loader:** When selected, displays a popup window where you can change the location of the current hot deploy directory.

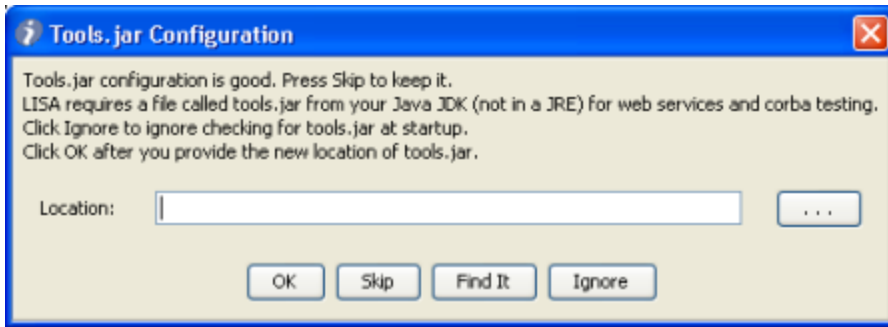


For more information see the [LISA Installation and Configuration Guide](#).

- **Set Tools.jar:** When selected, displays a dialog box as shown below, where you can enter, or browse to the location of a Tools.jar file. LISA requires this file if it needs to compile Java code. Several test steps and generators need to compile code.

Note: If LISA has found a Tools.jar file it will mention so in the dialog box below.

For more information see the [LISA Installation and Configuration Guide](#).

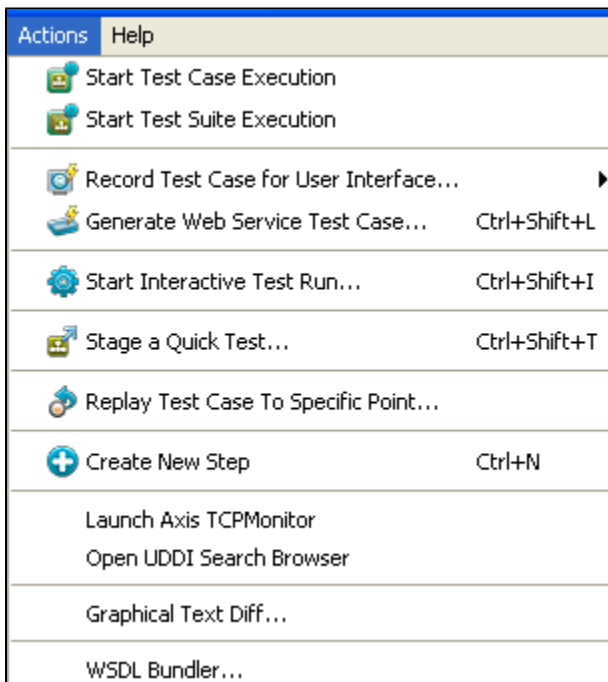


2.2.5 Actions Menu

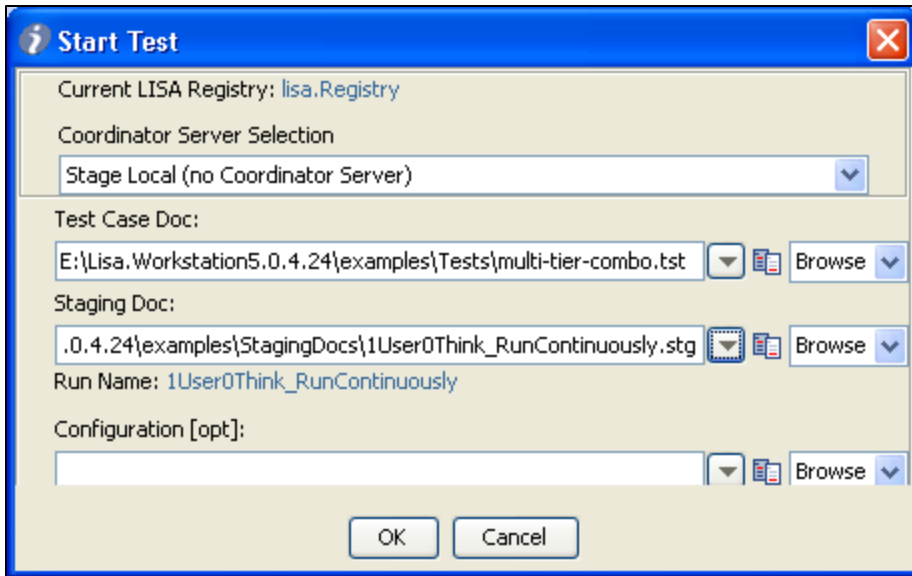
2.2.5 Actions Menu

The Actions menu has items related to Test Step or Test Suite Execution, Recorders for Web page, DOM Events etc, Generators and test execution tools in Workstation, and to some external tools that are integrated into the LISA environment.

Actions Main Menu

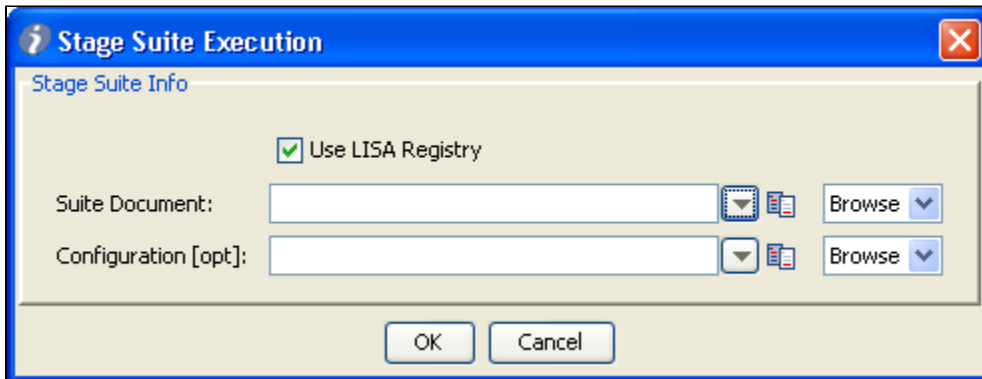


- **Start Test Case Execution** - When selected, opens a dialog box wherein we can specify the test case to be executed.



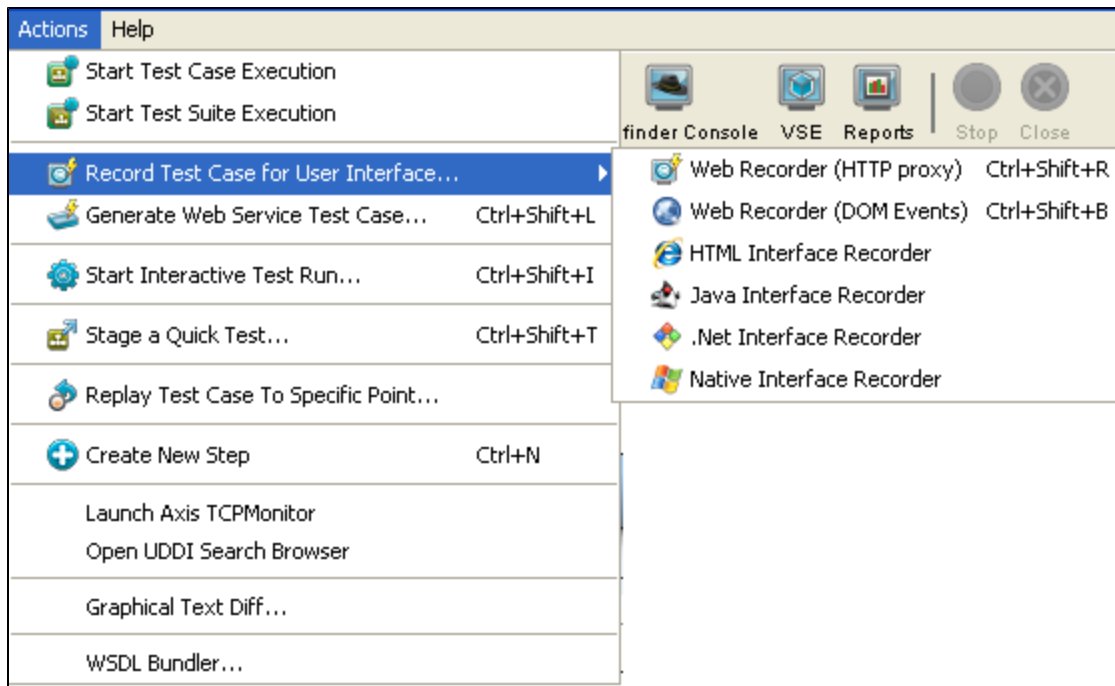
For more information, refer to 18. Running a Single Test.

- **Start Test Suite Execution** - When selected, opens a dialog box wherein we can specify the test suite to be executed.



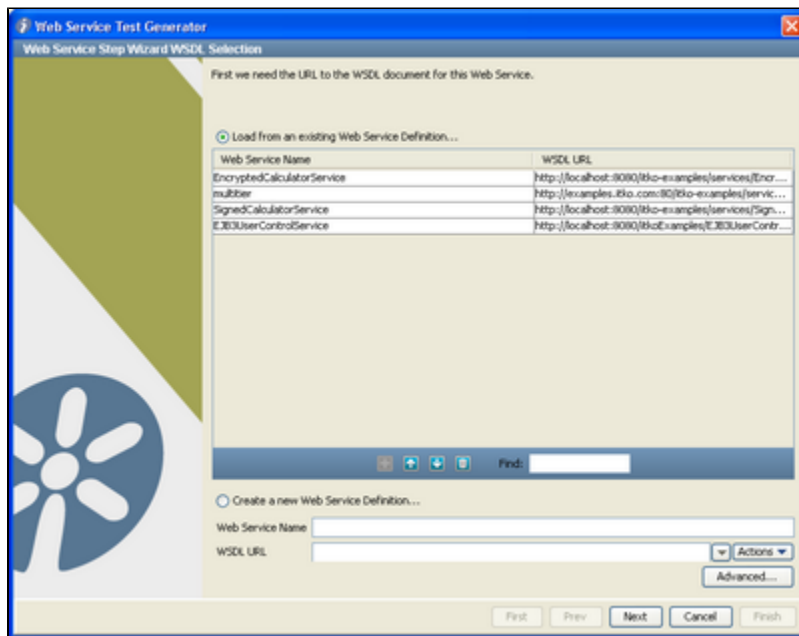
For more information, refer to 14. Building Test Suites

- **Record Test Case for User Interface** – When selected, opens a new sub menu which helps to Record Test Cases for HTTP Proxy, DOM Events, HTML, Java, .NET and Native Interfaces.



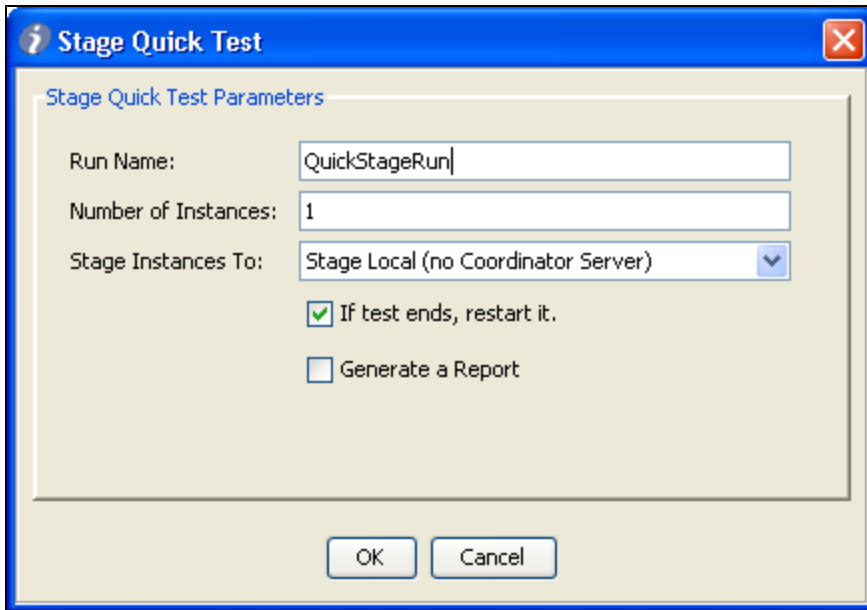
For more information, refer to Part 3 - Recorders and Test Generators.

- **Generate Web Service Test Case:** Click to generate a Web Service Test Case



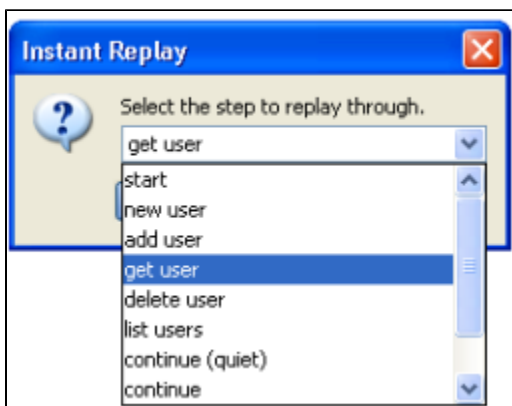
For more information, refer to 26. Generating a Web Service.

- **Start Interactive Test Run (ITR):** Click to start the Interactive Test Run (ITR) utility. For more information, refer to [PART 4 - Running Test Cases](#).
- **Stage a Quick Test:** Click to Stage a Quick Test.



For more information, refer to [PART 4 - Running Test Cases](#)

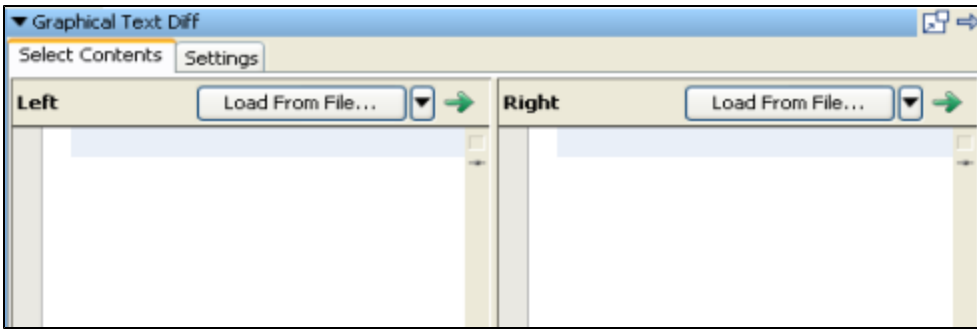
- **Replay Test to Specific Point:** Click to run a test to a specific point in a Test Case. You can replay the test in the Interactive Test Run facility to any given test step. A dialog box as shown below is displayed, to allow you to enter the last step of the replay. This will populate the known state up to that point. This is particularly useful if you want to examine properties like LASTRESPONSE which store the value of the last step response.



- **Create New Step:** When selected, opens a Steps panel with a list of all the available test steps in LISA. The selected step will be added to the Test Case workflow.

For more information, refer to [PART 2 - Building Test Cases](#)

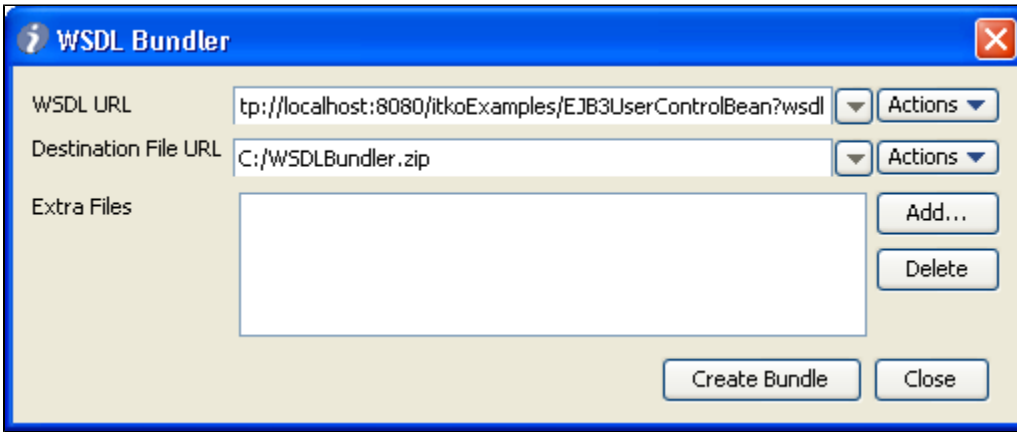
- **Launch Axis TCPMonitor:** When selected, launches an external application, TCPMonitor, which allows you to monitor client/Server communications. For more information see <http://ws.apache.org/axis/>.
- **Open UDDI Search Browser:** When selected, launches a UDDI search browser. For more information see the Web Service Test Step in Test Steps in the [LISA Reference Guide](#).
- **Launch Swing Component Browser:** When selected, launches a Swing Component browser. This is used in conjunction with the Swing recorders.
- **Graphical Text Diff...:** When selected, opens a new graphical XML diff engine and visualizer which has been implemented in LISA 5.0. This menu is used to start a graphical text differentiator for XML comparison.



For more detailed information, please refer to the [21.3 Graphical Diff Utility](#) Graphical Diff Facility.

- **WSDL Bundler:** When selected, allows you to bundle a WSDL and any supporting documents into a single zip file. The zip file can be sent to iTKO support if you are experiencing problems using the WSDL. The figure below shows an example.

Note: The Destination file is specified as a file URL.

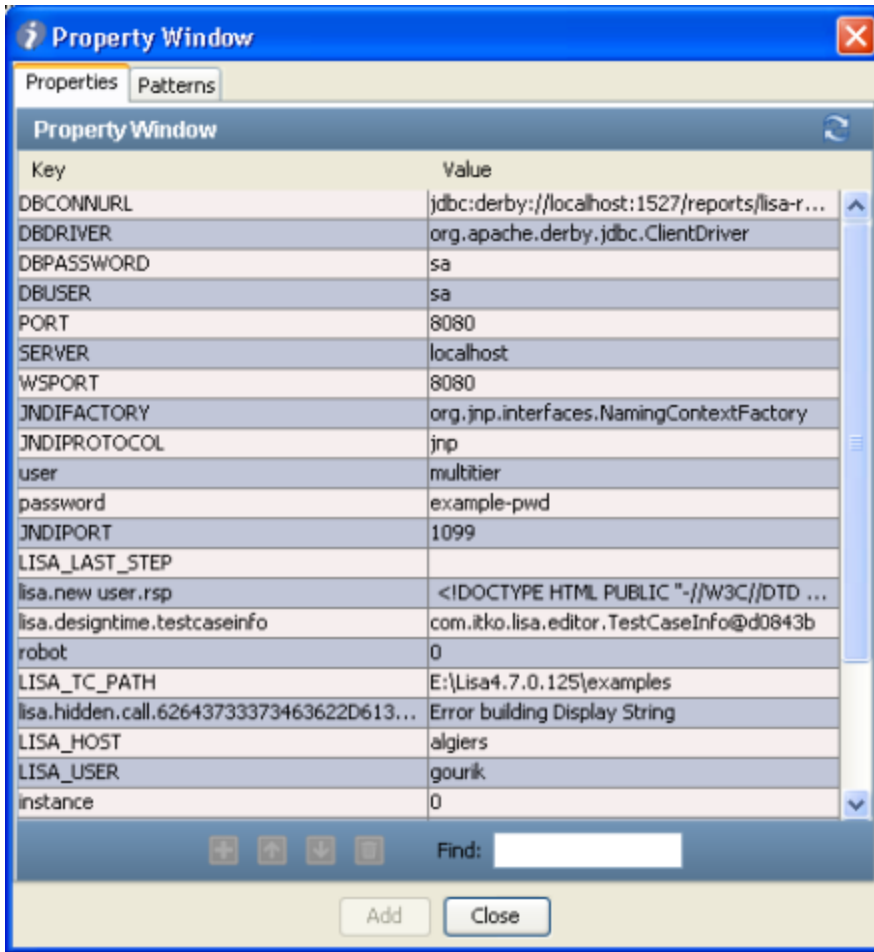


2.2.6 Help Menu

2.2.6 Help Menu

LISA Help menu has the Help menu functions that talk about LISA, its Documentation, Runtime Information, LISA License settings, Debugger etc.

- **Documentation:** When selected, will open the LISA documentation page.
- **About:** When selected, shows the LISA version and build numbers.
- **LISA Runtime Info:** When selected, shows current runtime information in two tabs, License Info and System Properties. For more details, please refer to [LISA Runtime information](#).
- **Property Window:** When selected, opens up the LISA property Window as shown below:



All the LISA properties are listed here. You can also generate string patterns from the **Patterns** tab.

- **LISA License Settings:** When selected, shows the current license settings.
For more information on LISA licensing and license settings see the [LISA Installation Guide](#).
- **Enable Debug:** When selected, turns on an extra level of debugging, and shows Java stack traces in the output log.

2.3 LISA Workstation Toolbar

2.3 LISA Workstation Toolbar

The LISA Workstation toolbar reproduces the common functions present in the menus.

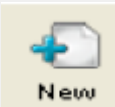
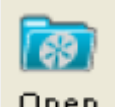



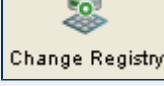

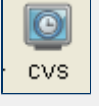

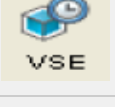



Below is the [Main Toolbar](#) followed by the [Test Case Toolbar](#).


Main Toolbar

The LISA main toolbar is shown below. The icons are described from left to right, with menu equivalents shown in parenthesis:



Icon	Description
------	-------------















 New	Creates a New Project (File > New > Project)
 Open	Opens an existing Project (File > Open > Project)
 Save	Saves the Current document (File -> Save, or Ctrl 5)
 Start Test	Starts Test Case Execution (Start -> Test Case Execution)
 Start Suite	Starts Stage Suite Execution
 Change Registry	Changes Test Registry Blue dot indicates that a Test Registry is attached, red dot indicates an error(Start > Change Test Registry)
 Toggle Registry	Toggle Test Registry Monitor Toggles the Test Registry(View -> Toggle Test Registry Monitor, or Ctrl+Shift+Y)
 CVS	CVS Dashboard Invokes the CVS Dashboard (System -> CVS Dashboard)
 Pathfinder Console	Pathfinder Console invokes the Pathfinder Console (View > Pathfinder Console)
 VSE	Invokes the VSE Dashboard (System -> VSE Dashboard)
 Reports	Invokes the Reports portal (File -> View Report)
 Play	Play Test Run. This icon is enabled only when you stage a test to run. Once clicked, this icon changes to a Stop icon, which is used to Stop a Test Run.
 Stop	Stop the test Run. This will turn into a Play icon and will be active when you stage a Test Case to run.

 Close	Close Test Run.
--	-----------------

Test Case Toolbar


This toolbar opens once you open a Test Case in the Model Editor. All the tasks here are specific to a Test Case.









Icon	Description
	Create New Step (Command -> Create New Step, or Ctrl+Shift+S)
	Delete a Test Step
	Set the currently selected step as the Starting step in the workflow
	Cut the selected text
	Copy the selected text
	Paste the selected text
	<div data-bbox="251 1224 1239 1549">  <div> Record Test Case for User Interface... <div>  Web Recorder (HTTP proxy) Ctrl+Shift+R  Web Recorder (DOM Events) Ctrl+Shift+B  HTML Interface Recorder  Java Interface Recorder  .Net Interface Recorder  Native Interface Recorder </div> </div> </div>

Click to open a menu and create Steps by *Recording a Test Case for User Interface You can:

- Record a Test case via **Proxy** (Commands -> Record Web Site Test Case Via Proxy or Ctrl+Shift+R)
- Record a Test case via **Dom Events** (Commands -> Record Web Site Test Case Via Dom Events or Ctrl+Shift+S),
- Use it as a HTML Interface Recorder, Java Interface Recorder, .NET Interface Recorder or Native Interface Recorder.
- Use it as a Java Interface Recorder,
- Use it as a .NET Interface Recorder or Native Interface Recorder. ||

	Start a New Interactive Test Run (Commands -> Start a New Interactive Test Run or Ctrl+Shift+I)
---	---

	Stage a Quick Test (Commands -> Stage a Quick Test or Ctrl+T)
	Replay Test To Specific Point (Commands -> Replay Test To Specific Point or Ctrl+Shift+R)
	Show Model Property (Help -> Property Window)
	Reset Zoom Scale to 1:1
	Zoom In
	Zoom Out
	View XML source

2.4 LISA Quick Start Menu

2.4 LISA Quick Start Menu

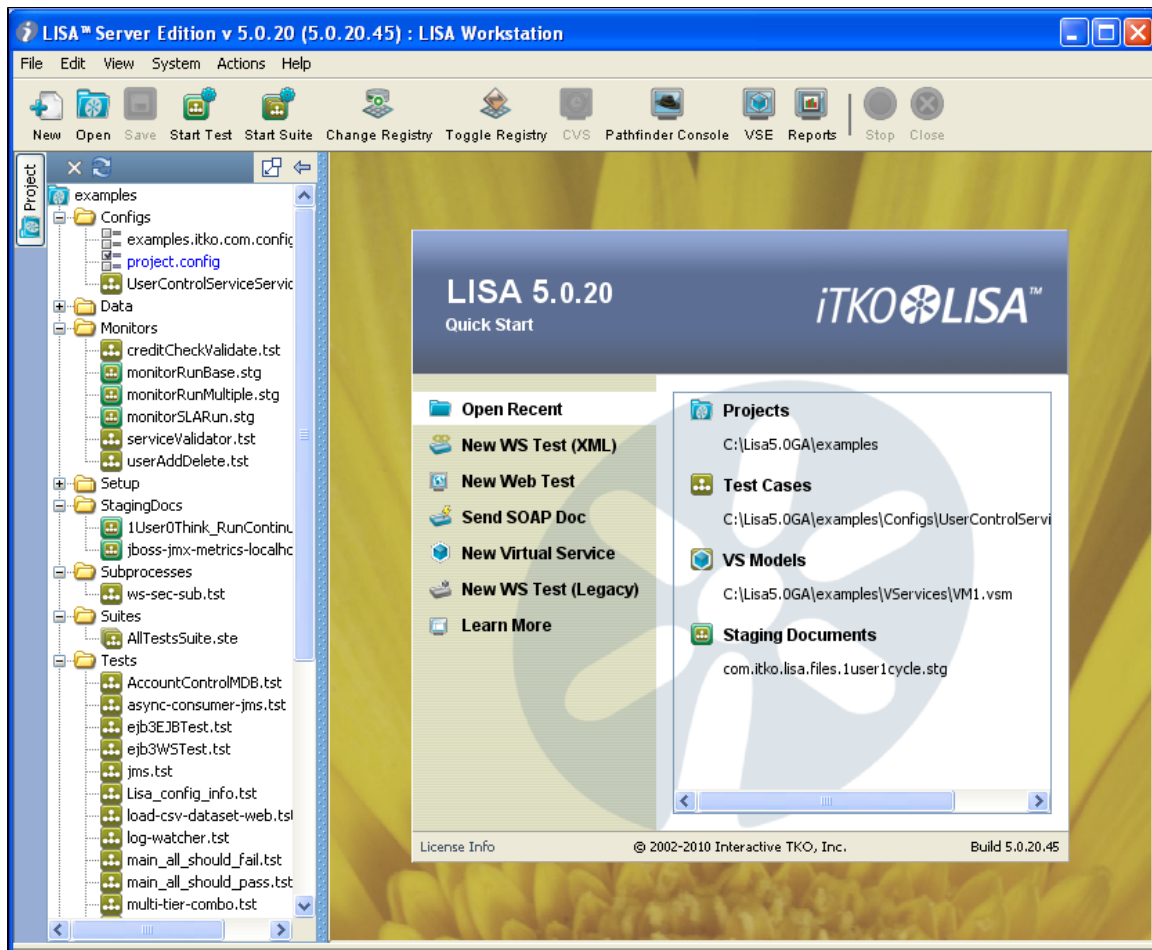
Once you install LISA and enter LISA License credentials, you are ready to work with LISA.

Note - Post 5.0, LISA registry is mandatory for LISA workstation.

So before starting to work in LISA Workstation, we need to set the Registry.

LISA Workstation First Screen

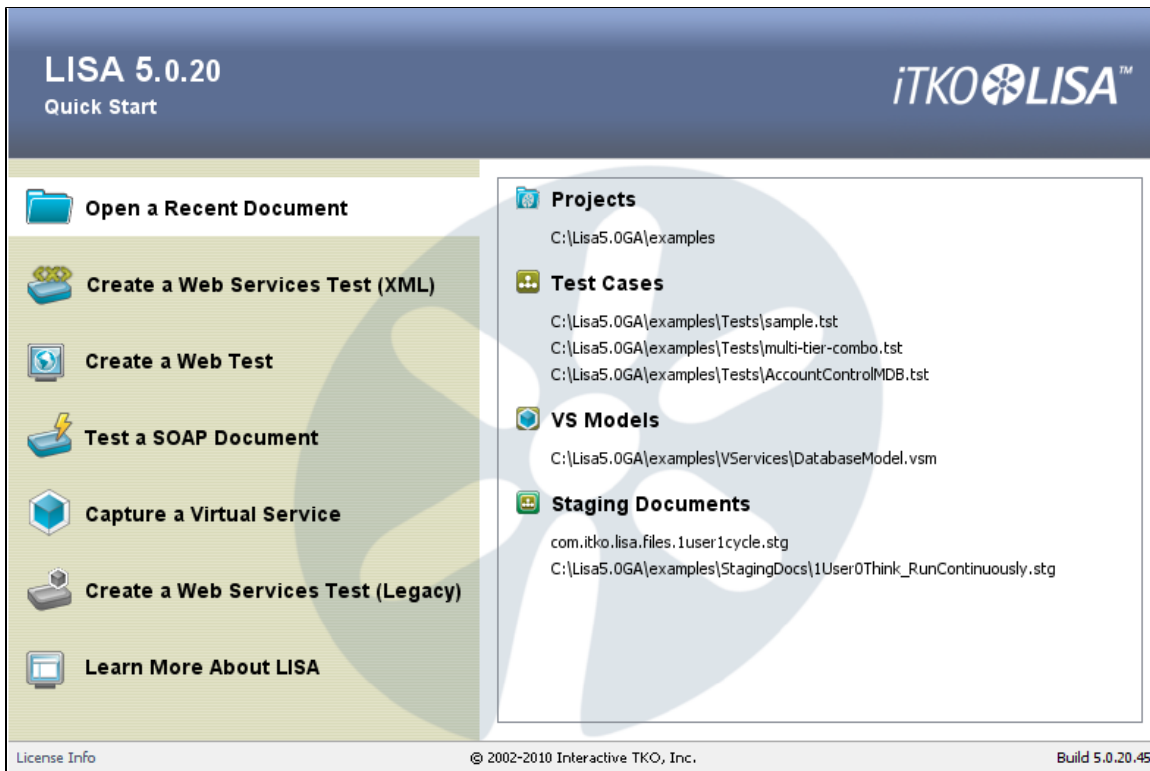
This is the first screen when you open the LISA Workstation.



Quick Start

The **Quick Start** menu is on the right of LISA workstation and has a list of quick menus and a list of recently opened items.

Note - If you are new to LISA workstation, the recently opened items list will be empty.



The Quick Start panel has some of the most useful options listed within LISA Workstation. So you can directly start creating a document from here, without opening the LISA workstation.

Once you click on a option, its properties are listed in the right pane.

Open a Recent Document

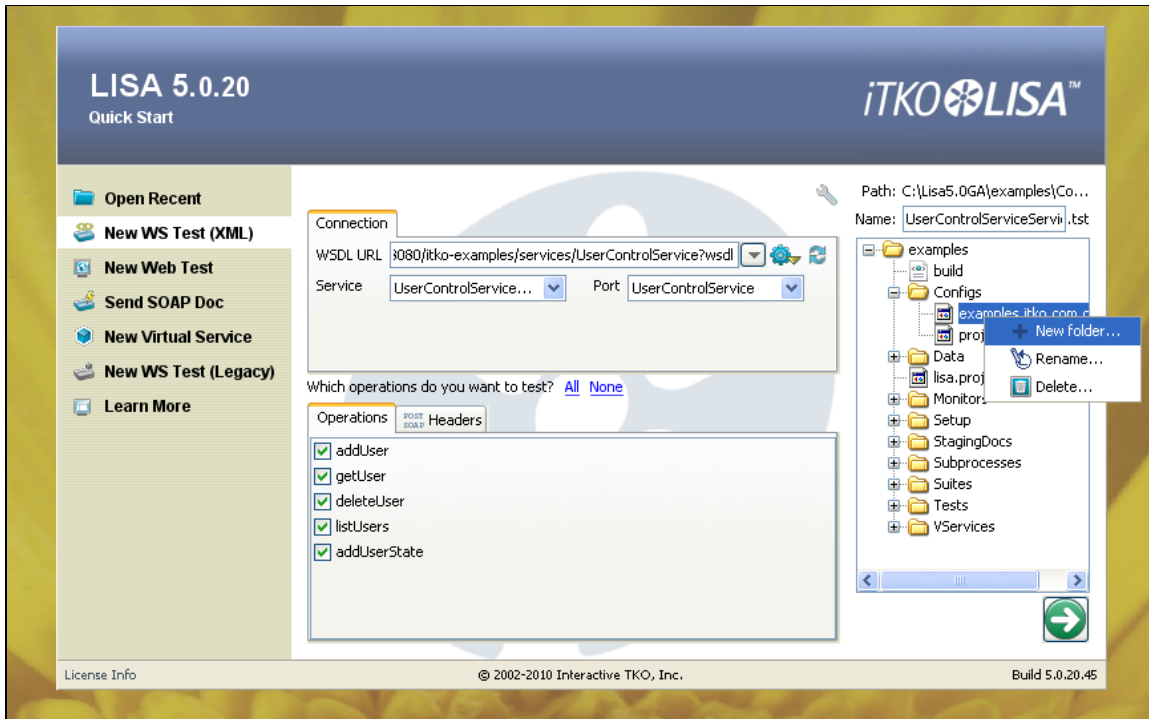
When LISA Workstation opens up, by default this option is selected.

The right pane, displays the recently opened Projects/Test Cases/Suites in the right window as shown in the screen above.

Create a Web Service Test (XML)

Click on this menu, to "Create a Web Service" test case.

The properties needed are displayed in the right pane.



- Enter the **WSDL URL**, **Service** and **Port** related details.
- Check on the **properties** that you want to test from "All" or "None" or select each item individually.
- In the right most pane, enter the **Name of the test** and select the path where you would like to **Save** it, within the project folder. When you select the path, it will be seen in the Path field. Within this pane, you can right click on any folder and create a new one or rename or delete the same as shown above.

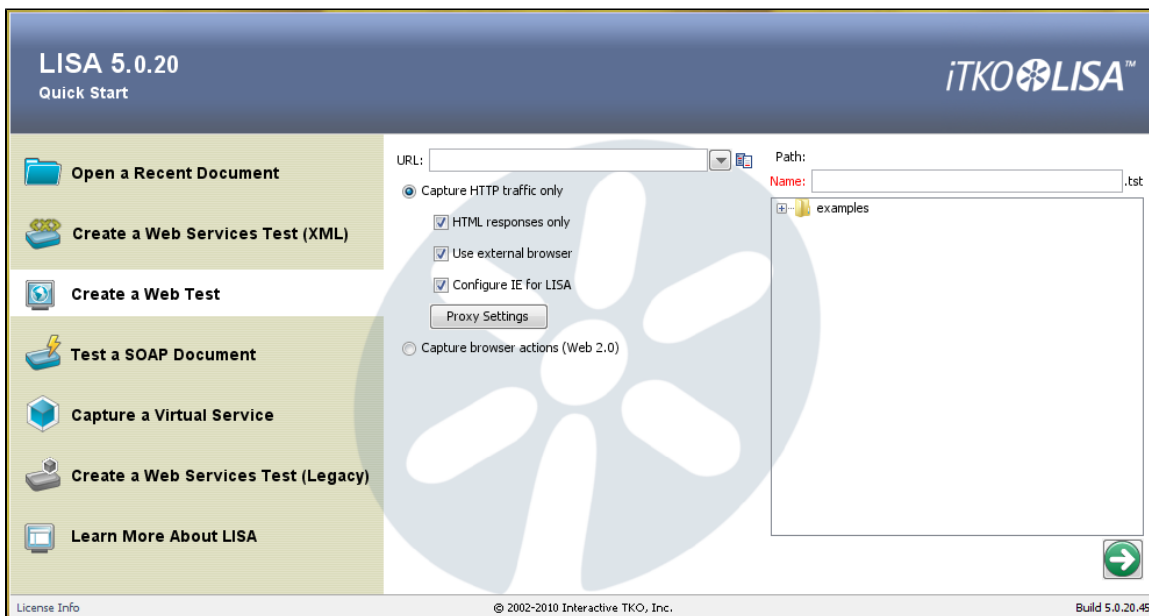


- Click on the Green arrow to get the document created.

For more information, refer to [Generating a XML Web Service](#) .

Create a Web Test

Click on this menu to "Create a web" test.



- Enter the **URL** for the web test.
- Select if you want to Capture **HTTP traffic** or Capture **Browser actions**.

- In the right most pane, enter the **Name of the test** and select the path where you would like to **Save** it, within the project folder. When you select the path, it will be seen in the Path field. Within this pane, you can right click on any folder and create a new one or rename or delete the same as shown above.

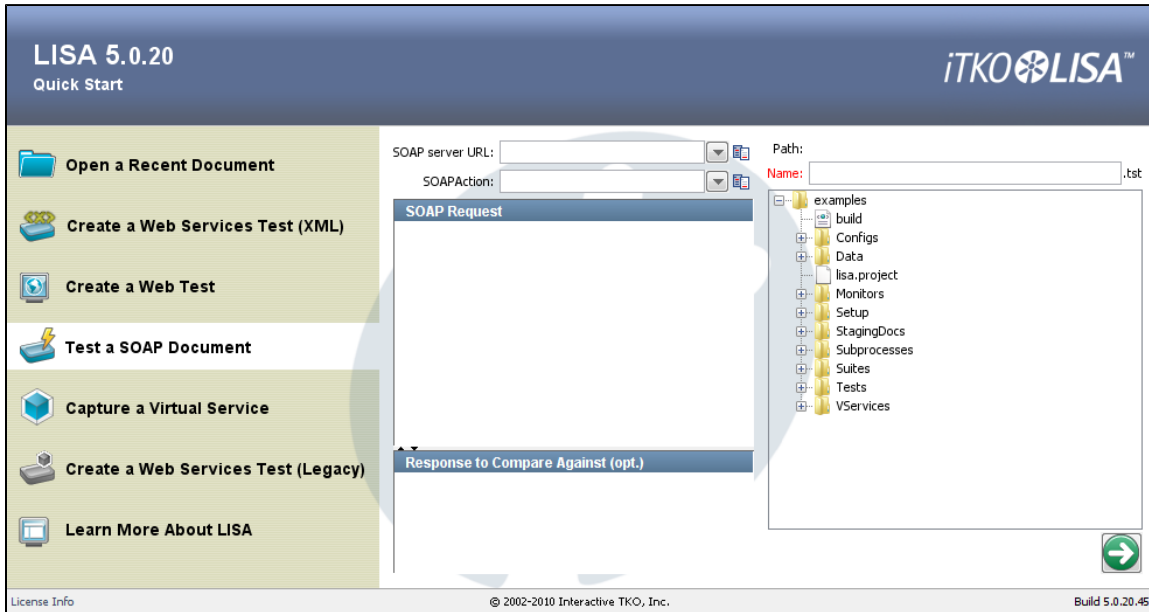


- Click on the Green arrow  to get the document created.

For more information, refer to [Recording a Web Site](#) .

Test a SOAP Document

Click this menu, to "Create a SOAP Request" document.



- Enter the **SOAP server URL** and **SOAP Action**.
- In the right most pane, enter the **Name of the test** and select the path where you would like to **Save** it, within the project folder. When you select the path, it will be seen in the Path field. Within this pane, you can right click on any folder and create a new one or rename or delete the same as shown above.



- Click on the Green arrow  to get the document created.

For more information, refer to the [LISA Reference Guide](#)

Capture a Virtual Service

Click this menu to "Create a Virtual Service" Test case.

LISA 5.0.20

Quick Start

- Open Recent
- New WS Test (XML)
- New Web Test
- Send SOAP Doc
- New Virtual Service
- New WS Test (Legacy)
- Learn More

Name: VM1

Listen/Record on port: 8001

Web service URL:

-- or --

Target host: 80

Target port: 80

Path:

Name: VM1 .vsm

- examples
 - build
 - Configs
 - Data
 - lisa.project
 - Monitors
 - Setup
 - StagingDocs
 - Subprocesses
 - Suites
 - Tests
 - VServices

Note: Be sure to update your endpoints:
From: http://80/...
To: http://GAURILAPTOP:8001/...

License Info

© 2002-2010 Interactive TKO, Inc.

Build 5.0.20.45

- Enter the **Name** of the Virtual Service
- Enter or select the **port number to Listen or Record** the service.
- Enter the **Web service URL** or enter the **Target host** and **port** number.
- Check the **SSL** for SSL connection.
- In the right most pane, enter the **Name of the test** and select the path where you would like to **Save** it, within the project folder. When you select the path, it will be seen in the Path field. Within this pane, you can right click on any folder and create a new one or rename or delete the same as shown above.



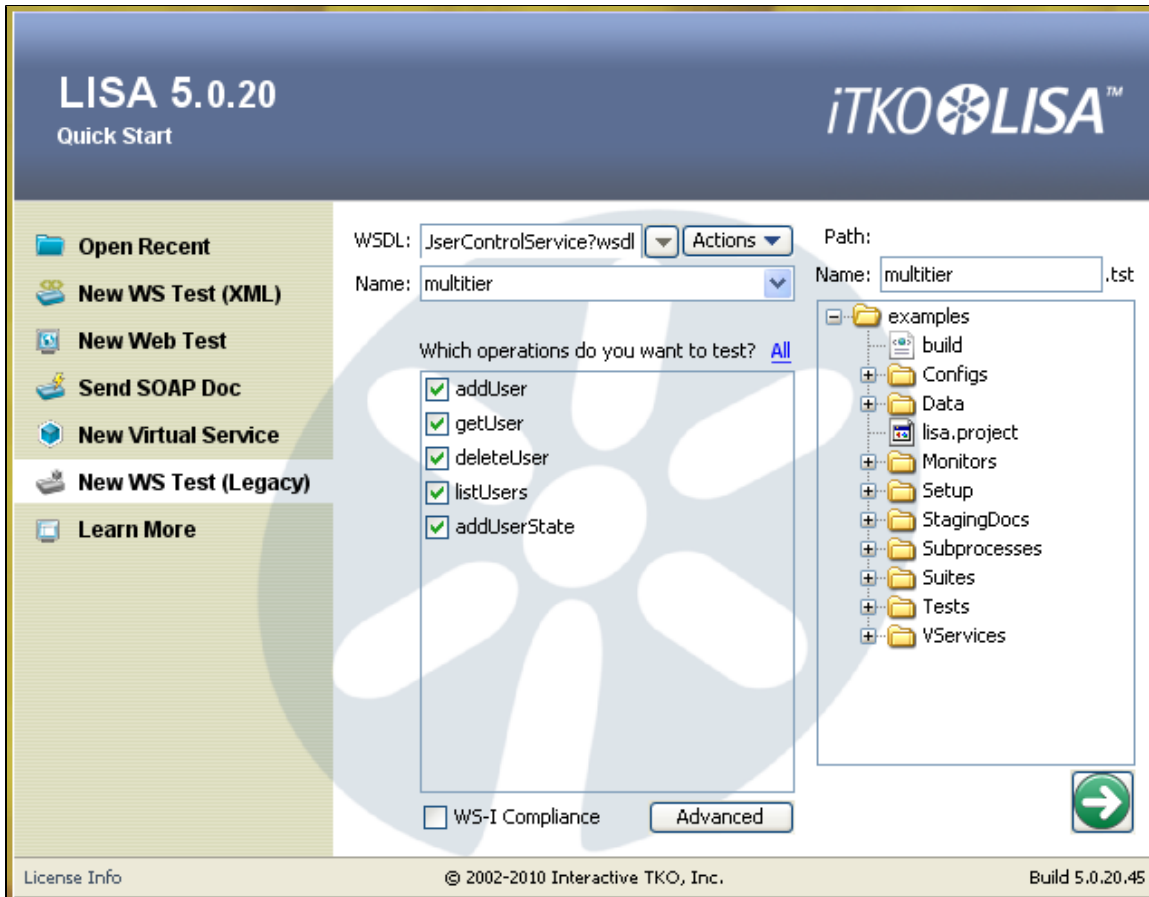
- Click on the Green arrow to get the document created.

For more information, refer to the [LISA VSE Guide](#) .

Create a Web Service Test (Legacy)

Click this menu to "Create a Web Service Legacy" test case.

- Enter the **WSDL URL** or select from the existing one from the drop down.
- Enter the **Name** of the Test case and select the operations that you want to **test** by selecting the given test steps.



- Check/Uncheck the **WS-I Compliance**.
- In the right most pane, enter the **Name of the test** and select the path where you would like to **Save it**, within the project folder. When you select the path, it will be seen in the Path field. Within this pane, you can right click on any folder and create a new one or rename or delete the same as shown above.



- Click on the Green arrow  to get the document created.

For more information, refer to [Generating a Legacy Web Service](#)

Note - From LISA 5.0, the Web Services Test panel will also allow WSDL Bundle URLs in the WSDL URL field.

For more types of advanced settings, click **Advanced** button to open the Advanced settings dialog:

- In the right most pane, enter the **Name of the test** and select the path where you would like to **Save** it, within the project folder. When you select the path, it will be seen in the Path field. Within this pane, you can right click on any folder and create a new one or rename or delete the same as shown above.



- Click on the Green arrow  to get the document created.

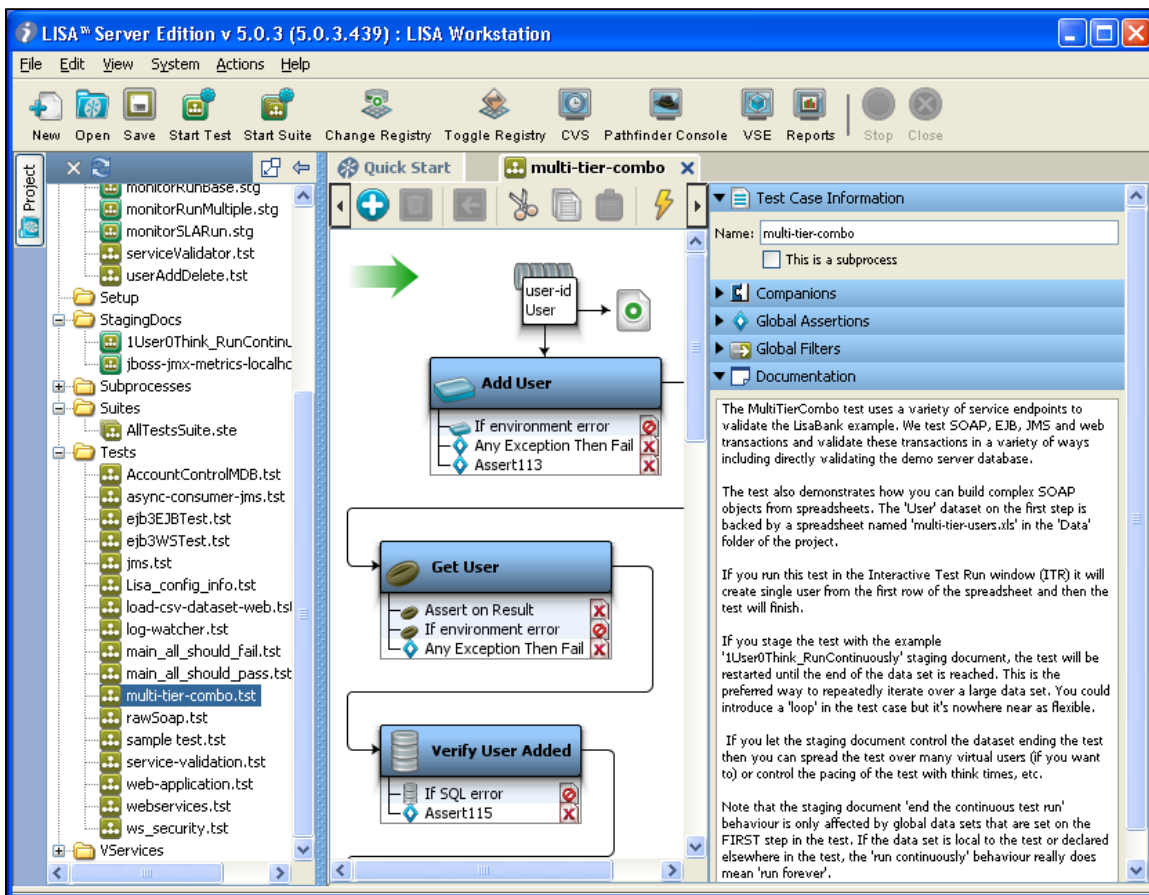
2.5 LISA Workstation Views

2.5 Different Views in LISA Workstation

To take a glimpse of all the windows/editors in the LISA workstation, let's take a look at the **multi-tier-combo** Test Case in the LISA examples directory (multi-tier-combo.tst).

LISA Workstation - Model Editor

Following screen shows the multi-tier-combo test case opened in the LISA Workstation.

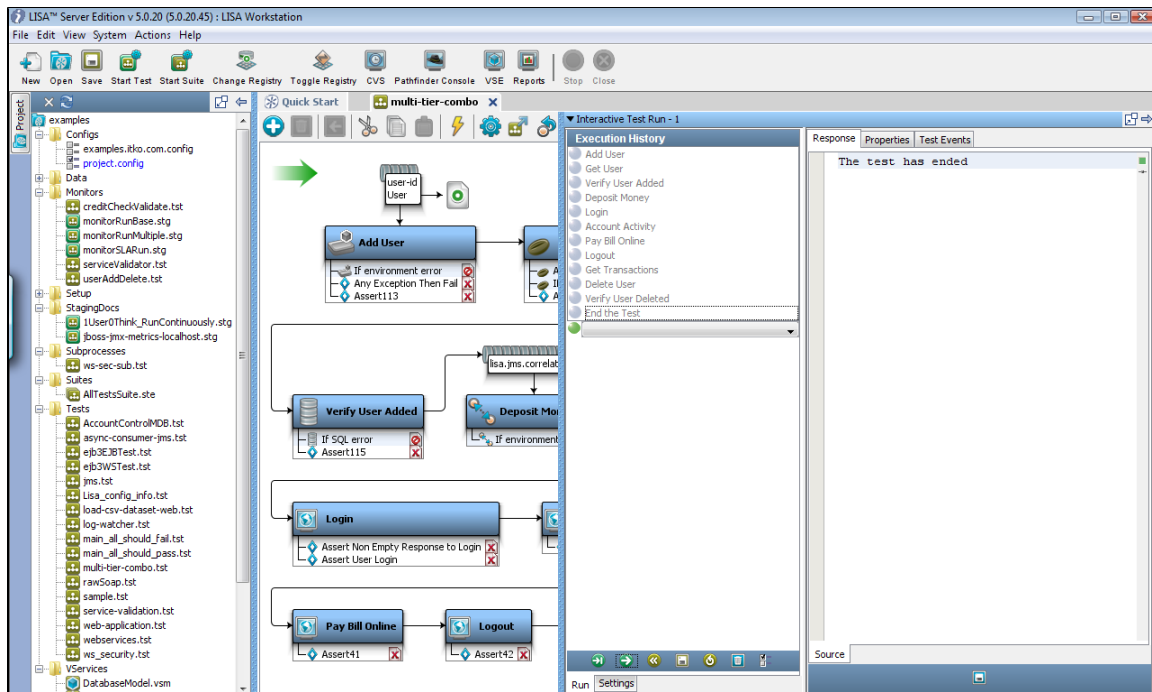


The **Element Tree** on the right is either Test case or Test Step specific and is discussed in depth in [Part Two](#) of the User Guide.

LISA Workstation - Interactive Test Run (ITR) Utility

Following screen shows the multi-tier-combo test case being executed in the ITR.

The ITR utility helps to run or execute the Test Cases in LISA.



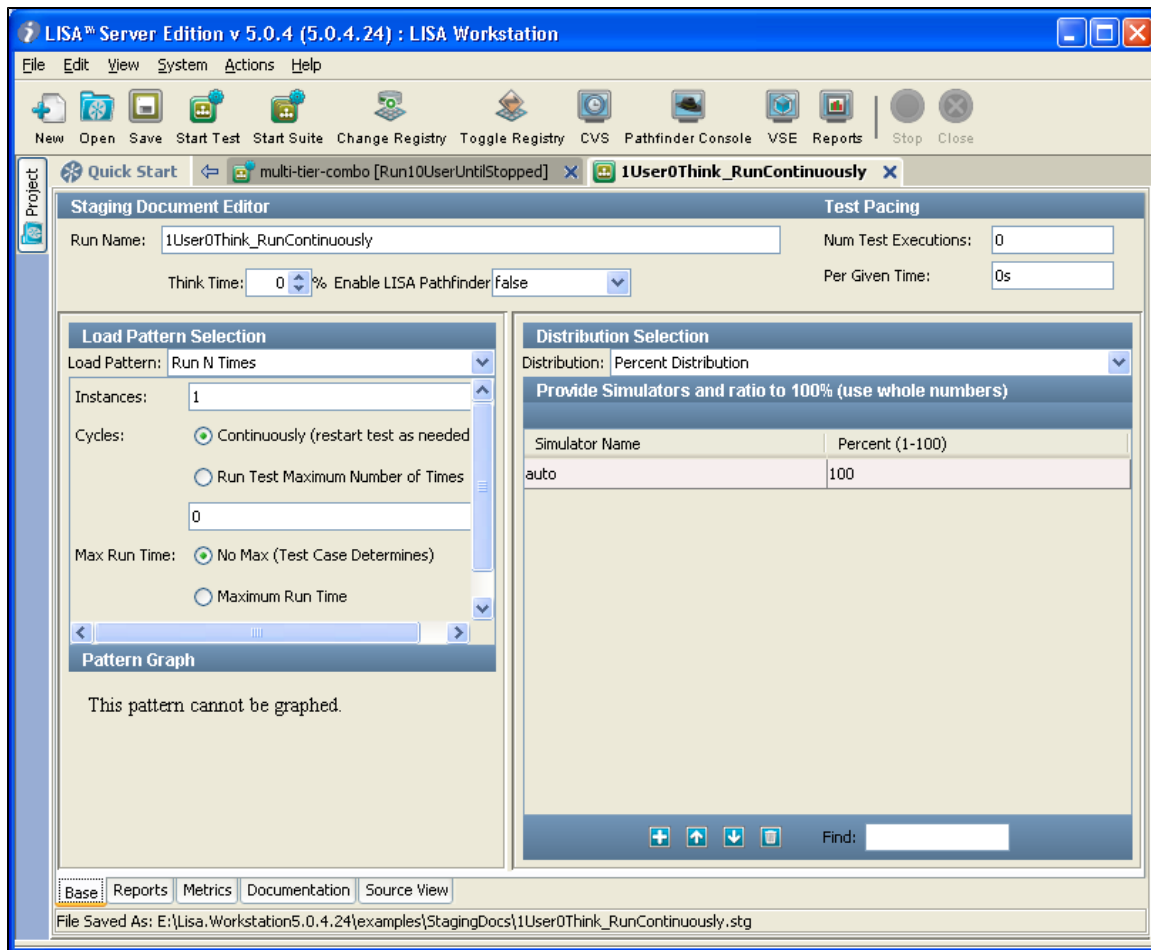
Notice the presence of an additional ITR toolbar and tabs (Run/Settings) at the bottom of the ITR screen on right.

For more information, refer to [Running a test case in ITR](#) section.

LISA Workstation - Staging Document Editor

Following screen shows the "1User0Think_RunContinuously" Staging document opened in the LISA Workstation. This staging document is available in the LISA examples directory.

The **Staging Document Editor** is a place where you define your settings for a test case run. The Staging document uses the full editor window in LISA Workstation.



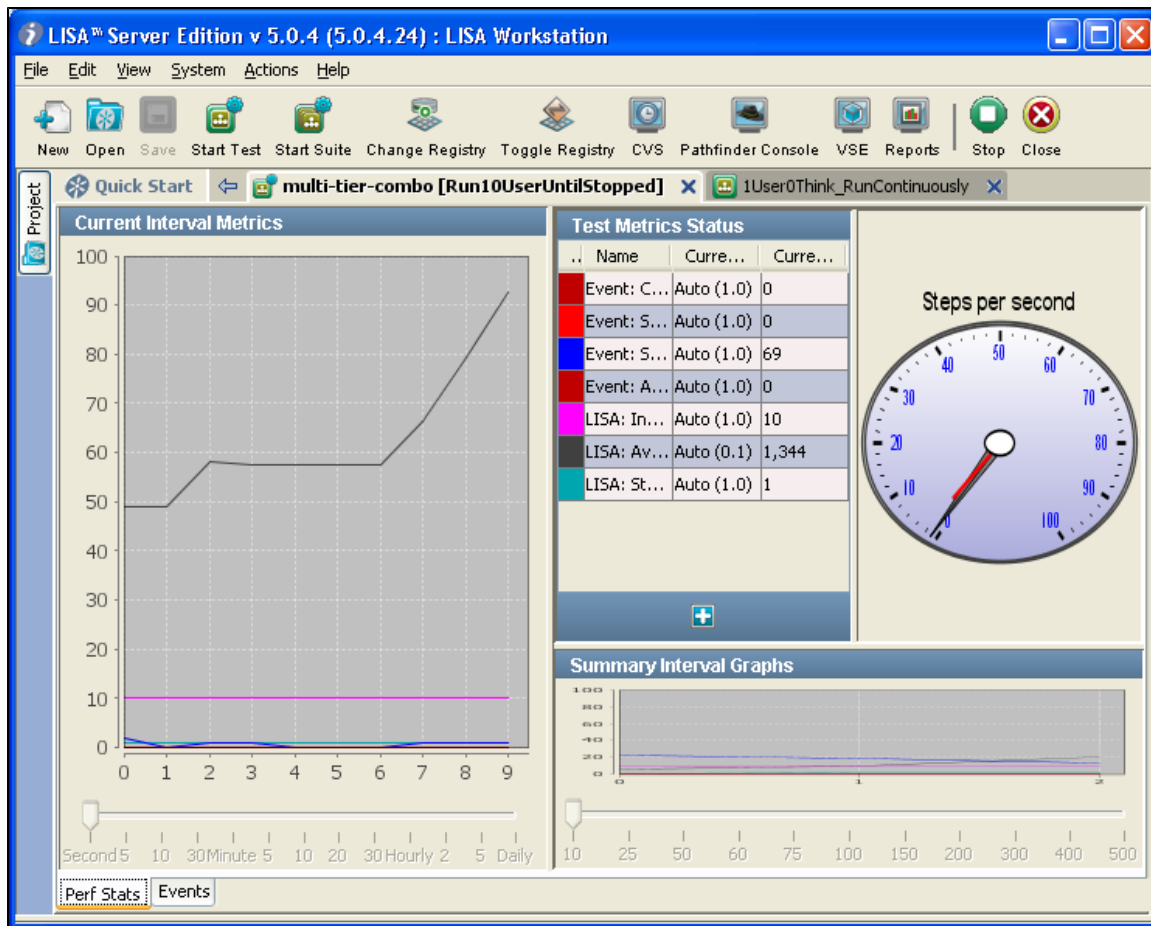
Note five different tabs at the bottom of the window – Base, Reports, Metrics, Documentation and Source View.

For more information, refer to [Building Staging Documents](#) section.

LISA Workstation – Quick Stage Run (Test Monitor)

Following screen shows the multi-tier-combo test case being staged with the **Quick Stage Run** utility in the LISA Workstation.

This utility will allow you to run the Test Case with minimum configuration.



This will show the **Performance statistics, i.e Metrics, Graphs and Events** generated in the test.

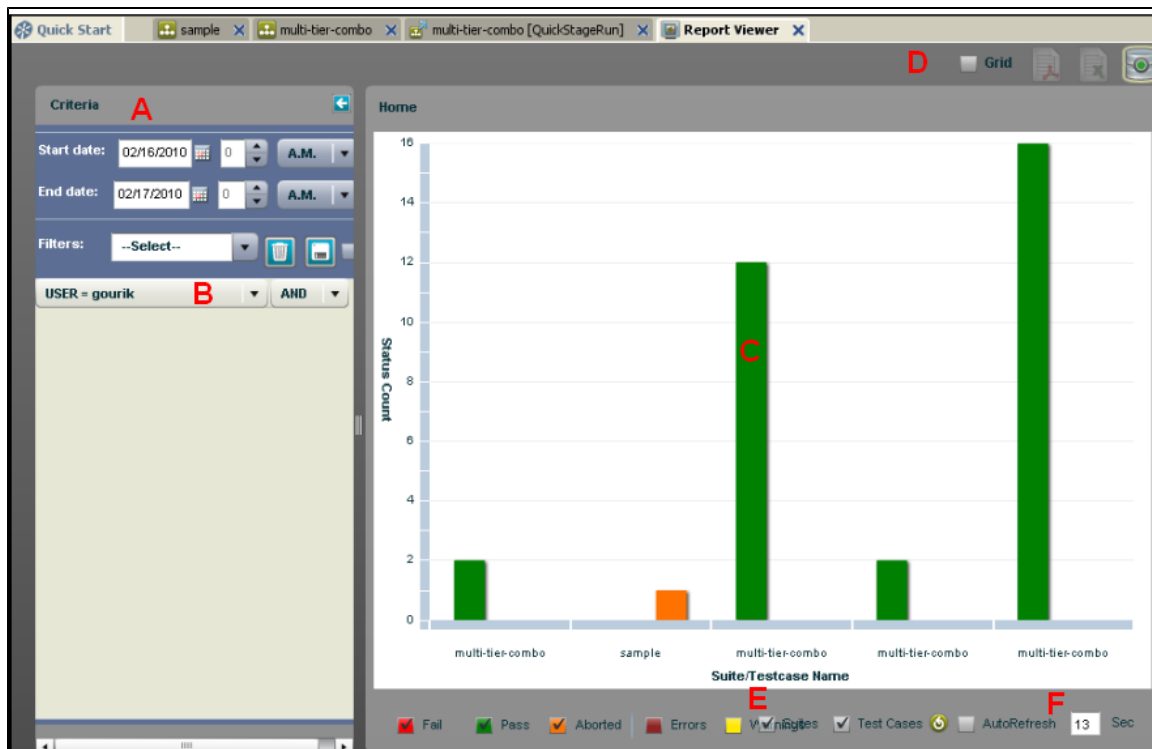
All these figures serve only to illustrate the many faces of LISA Workstation.

For more information, refer to [Running a Quick Test](#) section.

LISA Workstation - Report Viewer

Following screen shows the multi-tier-combo test case opened in the LISA Report Viewer.

This can also be opened in a Web Browser.

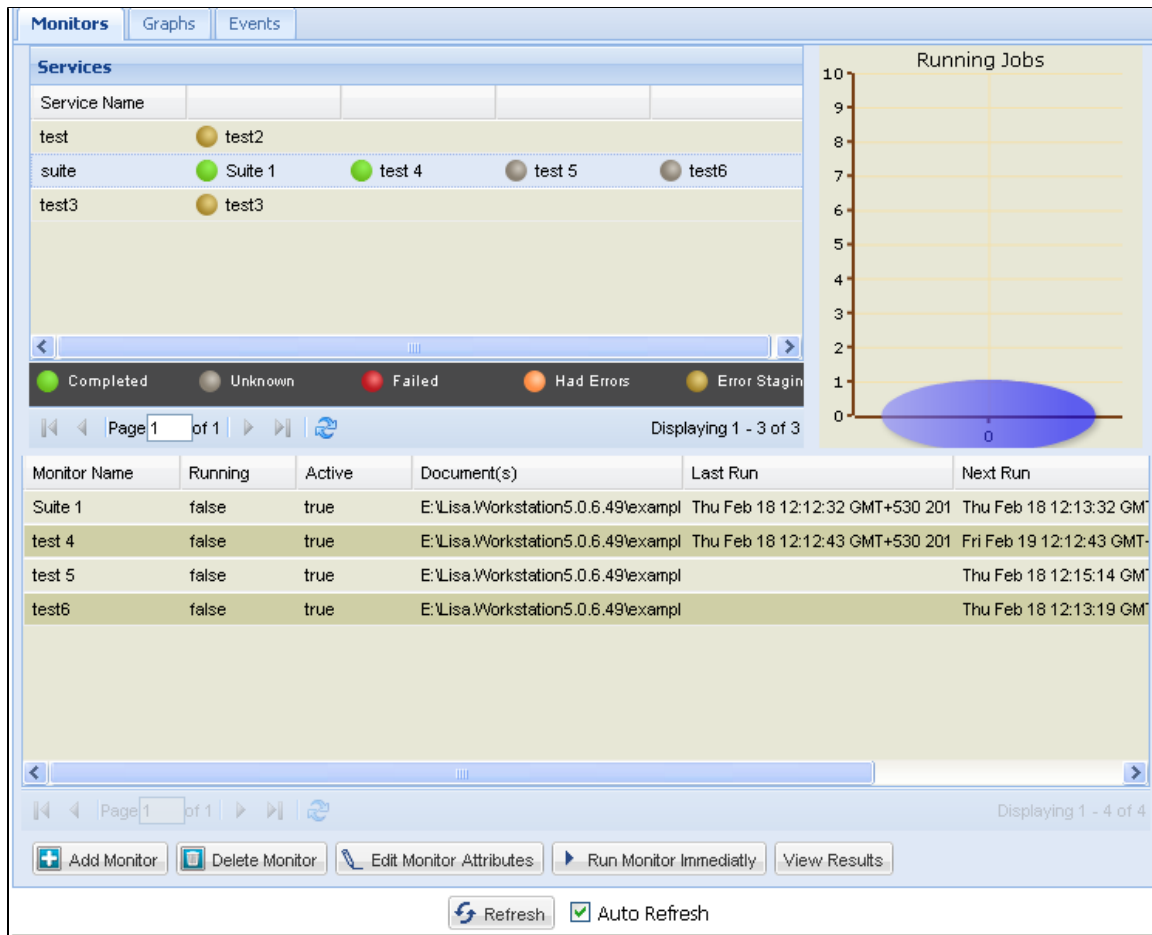


For more information, refer to [LISA Reports](#) section.

LISA Workstation - CVS Dashboard

Following screen shows the many tests in the LISA CVS Dashboard.

CVS Dashboard can also be opened in the Web Browser.

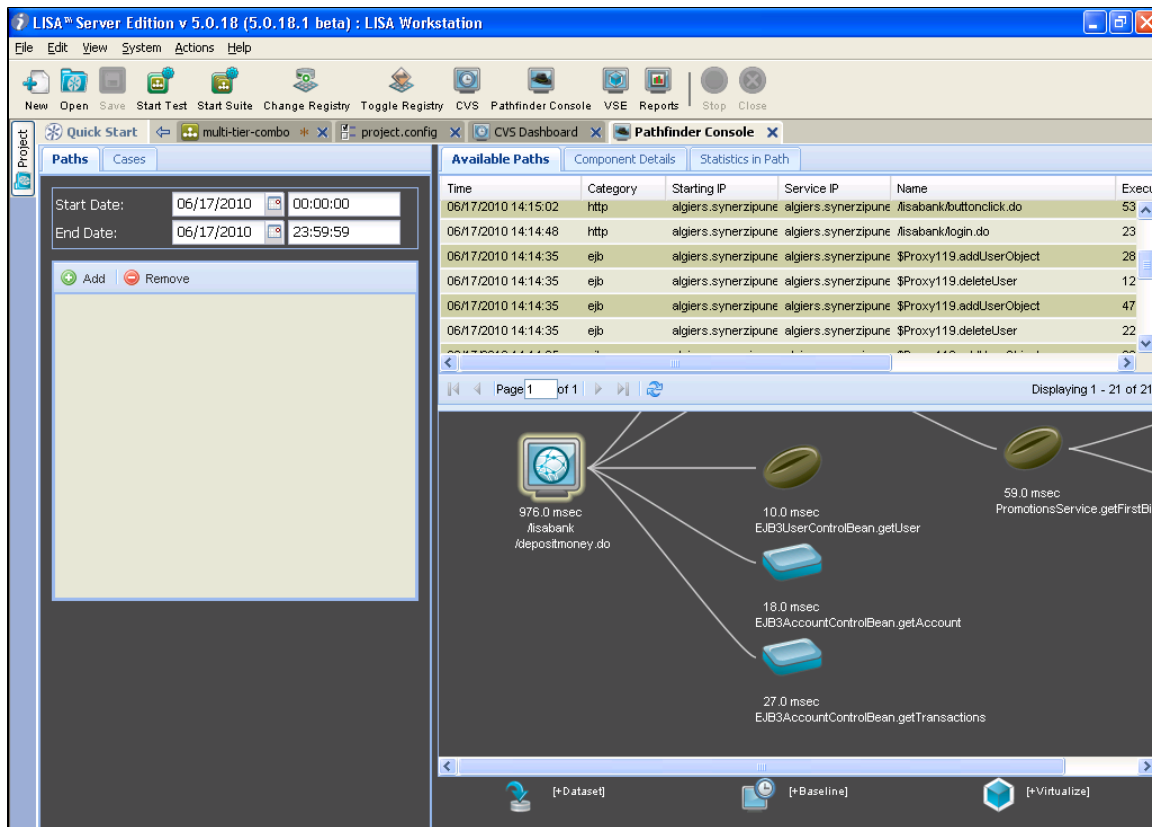


For more information, refer to [CVS Dashboard](#) section.

LISA Workstation - Pathfinder

Following screen shows the LISA Pathfinder Console.

Pathfinder can also be opened in a Web Browser.



For more information, refer to [LISA Pathfinder](#) section.

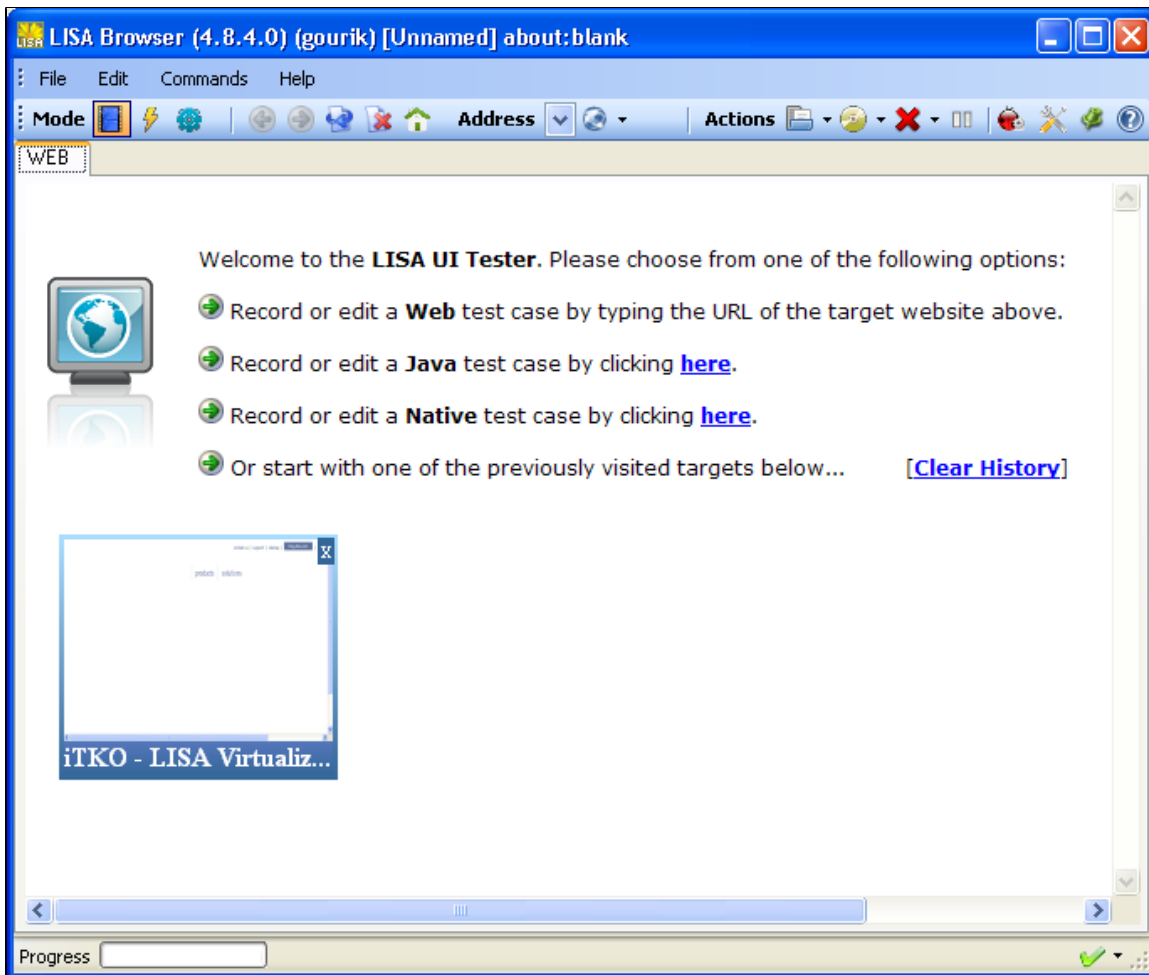
LISA Workstation - VSE Dashboard

Following screen shows the LISA VSE Dashboard:

For more information, refer to [LISA VSE guide](#) .

LISA Workstation - Web 2.0 Browser

Following screen shows the LISA Web 2.0 browser, which helps in the Web browser recordings.



For more information, refer to LISA web 2.0 user guide section .

2.6 LISA Runtime Information

2.6 LISA Runtime Information

To know more about the LISA runtime information,

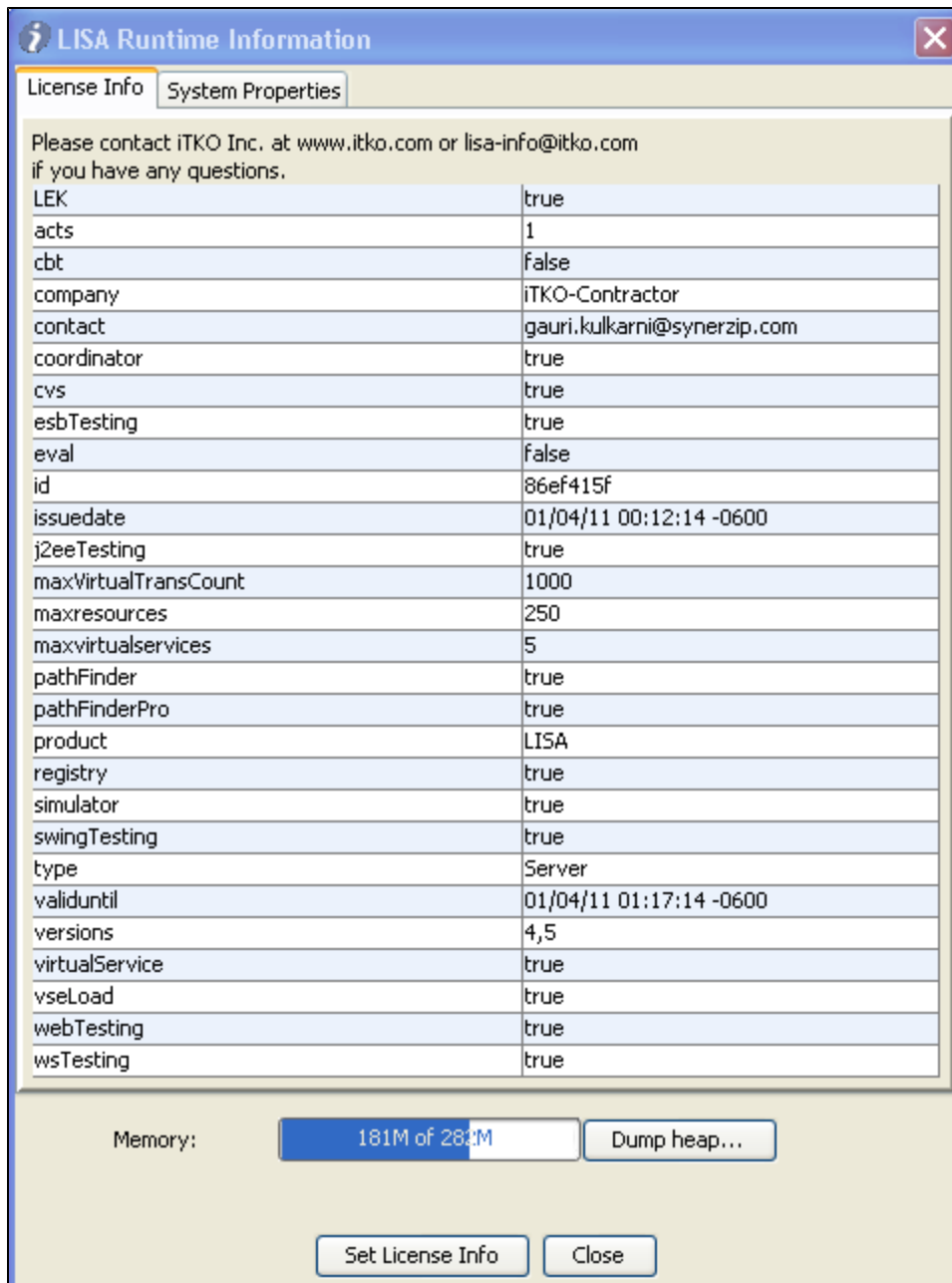
- Click Help menu > LISA Runtime information.

When selected, this will show current runtime information like current license information and the LISA system properties.

There are two tabs: License Info and System Properties

License Info:

This tab gives information related to LISA License



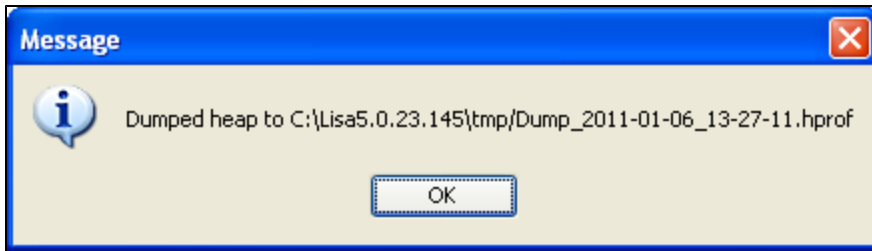
Dump Heap -

Since LISA 5.0, you can also generate a heap dump file (HProf) through LISA runtime panel.

- Click on the Dump Heat button to enable the heap dump.

Note - This functionality is available only on Sun java 6 java runtimes. It is a diagnostic tool which can help iTKO support determine the cause of OutOfMemory conditions. LISA will automatically dump the heap if an OutOfMemory condition occurs, this button is for manually triggering a heap dump.

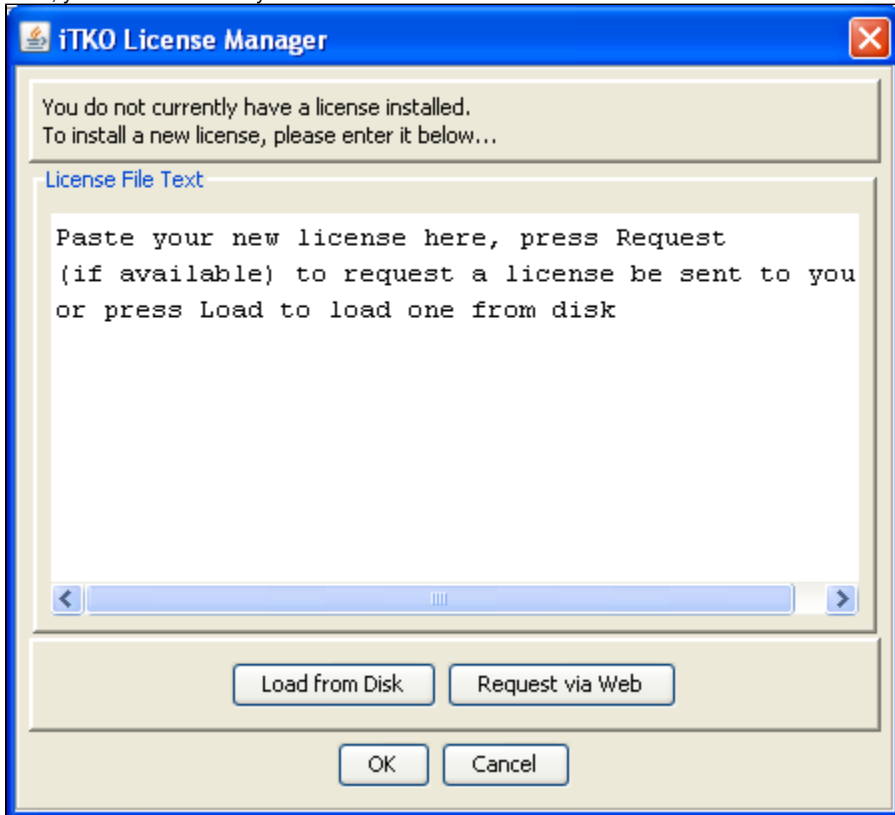
Once the Dump is created, it gives out a message as shown below:



Set License Info

Click on this button, to open the LISA License Manager.

Here, you can set or reset your LISA license as shown below:

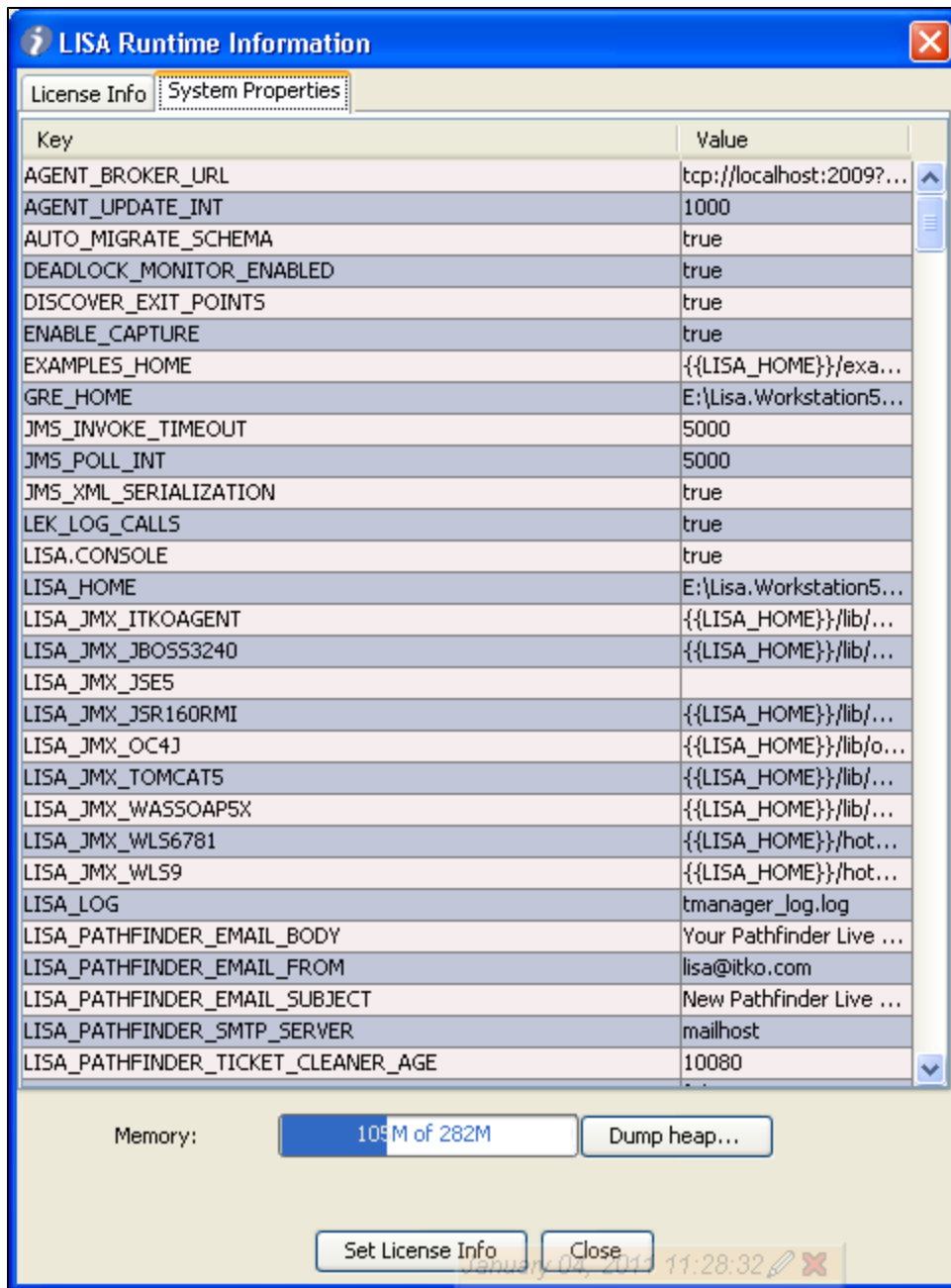


Load from Disk - If you have the license file available on your system, click this to open the file.

Request Via Web - Click to request the License via web from iTKO support.

System Properties:

This will give information regarding the LISA system properties:



3. Anatomy of a LISA Test Case

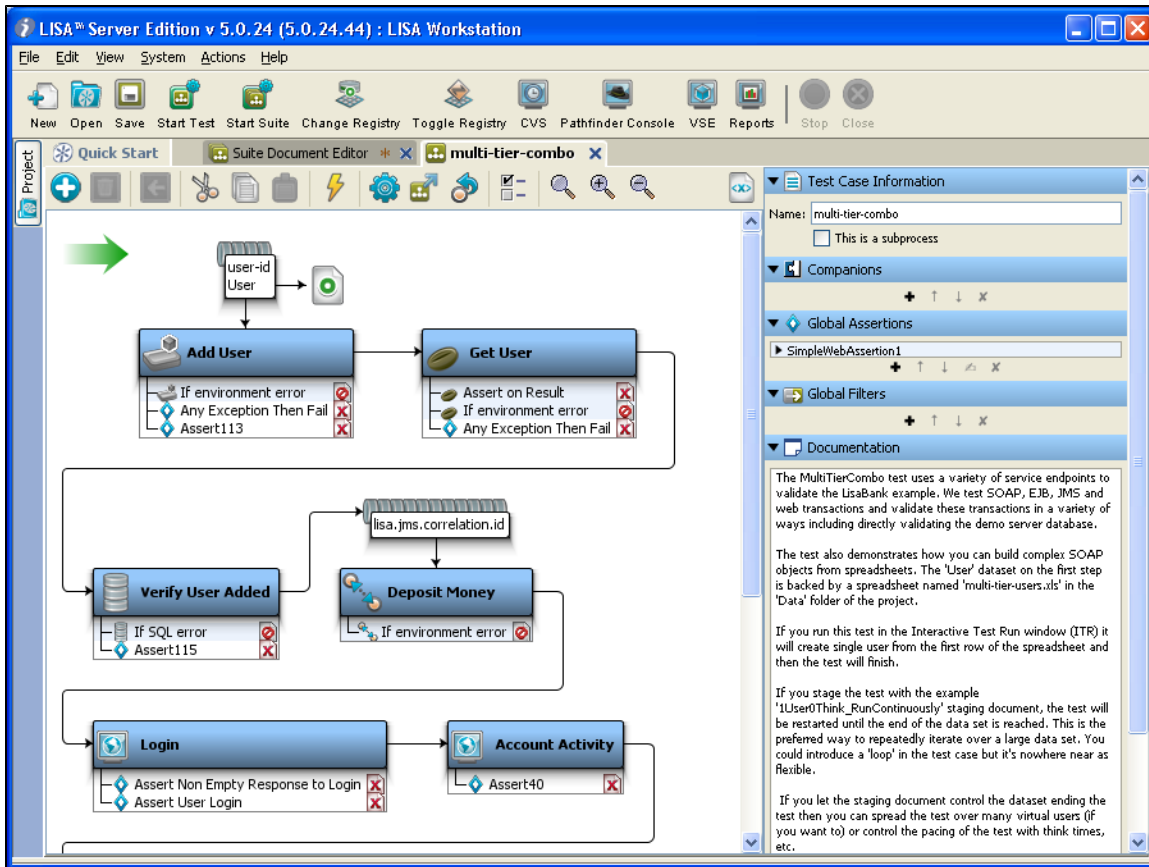
3. Anatomy of a LISA Test Case

A Test Case is a complete specification of how to test a business component in the 'system under test', or in some cases the complete 'system under test'.

A Test Case is persisted as an **XML** document, and contains all the information needed to test the given component or system.

A Test Case in LISA is a workflow with the test steps being connected by paths that represent successful and non-successful step conclusions. LISA Assertions may accompany the step, and different paths are provided based on the firing of any of the Assertions.

LISA Test Case in LISA Model Editor:



There are several LISA elements that facilitate the testing of each step. These elements are shown in the Element tree, which is on the right of the Model Editor in LISA Workstation.

This section includes the following topics:

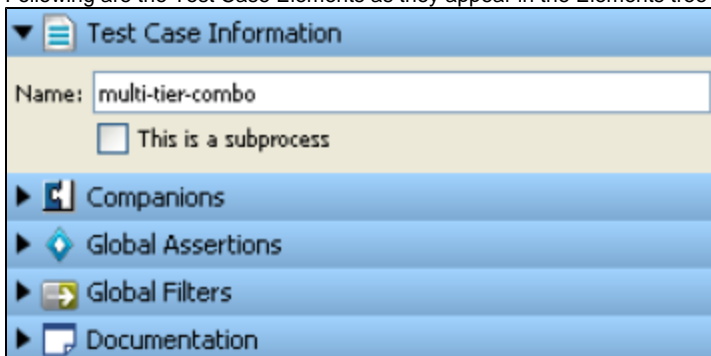
- [3.1 Elements of a Test Case](#)
- [3.2 Elements of a Test Step](#)
- [3.3 Test Case Quick Start](#)

3.1 Elements of a Test Case

3.1 Elements of a Test Case

The Elements of a Test Case help in building the Test Case as a whole.

Following are the Test Case Elements as they appear in the Elements tree on the right side of the LISA Workstation:



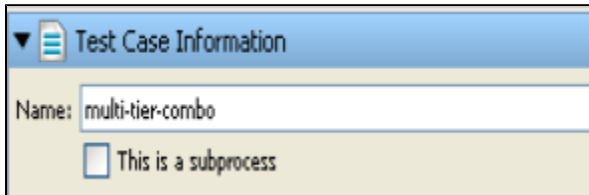
Test Case Information
Companions

Test Case Information

The **Test case information** tab in the right panel provides a place to document basic information about the Test Case, like Test Case **Name**.

More importantly, this tab is also used as an entry point for creating a sub-process, or converting a Test Case into a sub-process. A sub-process is a Test Case that is designed to be called by another Test Case rather than to be run as a standalone test.

The Test Case information editor is shown below:



In this editor,

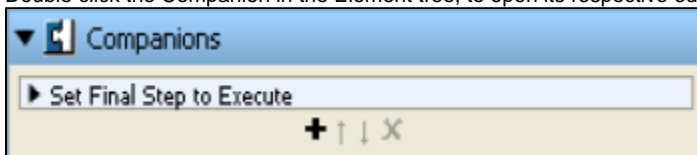
You can **Rename** the Test Case and /or and check the "**This is a subprocess**" box to make this Test Case work as a **Sub process**.

For more information on sub processes see [Building Sub Process](#).

Companions

A Companion is a LISA element that runs **before and after every Test Case** execution. Companions are used to configure global behavior in the Test Case. A restart causes the Companions to run again.

Double click the Companion in the Element tree, to open its respective editor.

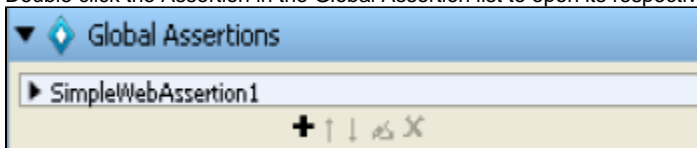


For more information on Companions see [Using Companions](#).

Global Assertions

An Assertion is a LISA code element that runs **after a step and all its Filters** have run, to verify that the results from running the step match expectations. Global Assertions are Assertions which are applied to the entire Test Case.

Double click the Assertion in the Global Assertion list to open its respective editor.

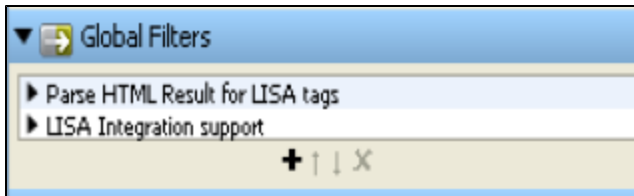


For more information on Assertions see [Adding Assertions](#).

Global Filters

A Filter is a LISA code element that runs **before and after a test step**, giving you the opportunity to massage the data in the result, or store values in properties. Global Filters are Filters which are applied to the entire Test Case.

Double click the Filter in the Global Filter list to open its respective editor.



For more information on Filters see [Adding Filters](#).

Documentation

The Documentation text area allows you to add documentation for your Test Case. This text is not used by LISA in any process, but it is a convenient place and more importantly a good practice to put a description of your Test Case, and notes for other users/testers, who will use this Test Case.

The information written here, is stored in the Test Case XML file. If this Test Case is used as a sub process, this information will be passed to the calling step.

3.2 Elements of a Test Step

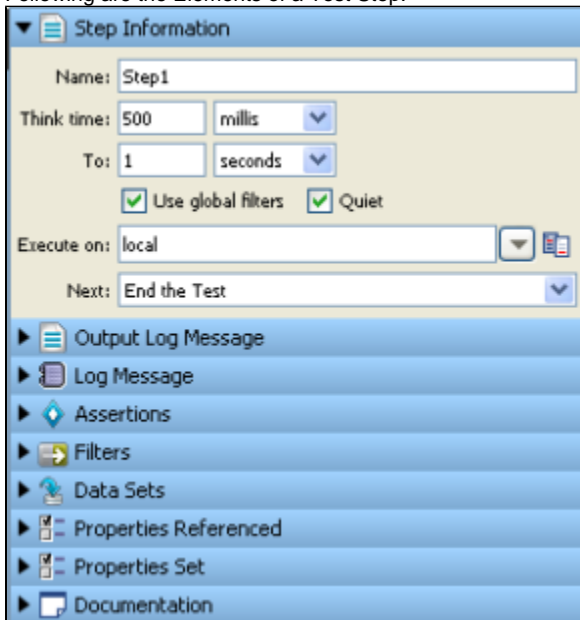
3.2 Elements of a Test Step

A **Test Step** is an element in the LISA workflow that performs a basic action to validate a business function in the 'system under test'.

Steps can be used to invoke portions of the system under test.; These steps will typically be chained together to build out workflows as Test Cases in the LISA Model Editor.

From within each step the user has the ability to create **Filters** to extract data or create **Assertions** to validate response data.

Following are the Elements of a Test Step:



Step Information
 Output Log Message
 Log Message
 Assertions
 Filters
 Data Sets
 Properties Referenced
 Properties Set
Documentation

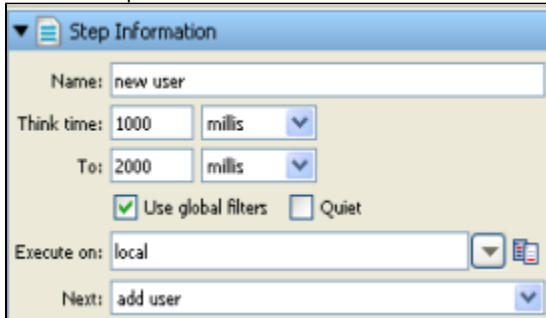
These LISA elements are described briefly in the sections below, and discussed in detail in [Building Test Steps](#).

Step Information

The Test step information tab in the Element tree provides the user a place to document basic information about the test step.

You can enter the Test Step specific information in this tab like the Step name, Think time, Execute on Step details and next step details etc. You can also run this step using Global Filters or run this step Quietly.

The Test Step Information editor is shown below:



The screenshot shows the 'Step Information' dialog box. It has a title bar with a dropdown arrow and the text 'Step Information'. Inside, there are several fields: 'Name:' with the value 'new user'; 'Think time:' with a text box '1000' and a dropdown 'millis'; 'To:' with a text box '2000' and a dropdown 'millis'; a checkbox 'Use global filters' which is checked, and a checkbox 'Quiet' which is unchecked; 'Execute on:' with a text box 'local' and a dropdown arrow; and 'Next:' with a text box 'add user' and a dropdown arrow.

For more information, see [Building Test Steps](#).

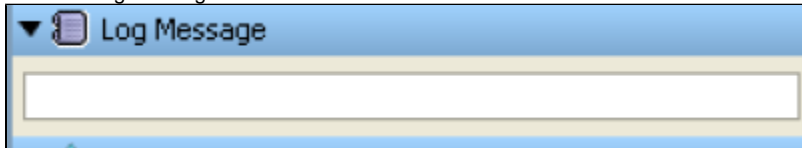
Output Log Message

For the purpose of illustration, we have added the "Output log Message" step from Utilities section below: Depending on the step added, the step related tabs (here -Output log message) will change.

Log Message

A Log message is a text field in which you can enter a message for a particular step. This message will be seen upon execution of the test step/case.

Enter the log message here...



The screenshot shows the 'Log Message' dialog box. It has a title bar with a dropdown arrow and the text 'Log Message'. Inside, there is a large text area for entering the log message.

Assertions

An **Assertion** is a LISA code element that runs after a step and all its Filters have run, to verify that the results from running the step match expectations. The result of an Assertion is Boolean - either true or false (there are no other possibilities).

The outcome determines whether the test step passes or fails, and the next step to run in the Test Case. That is, the Assertion can dynamically alter the Test Case workflow by introducing conditional logic (branching) into the workflow.

For more information on Assertions see [Adding Assertions](#).

Filters

A Filter is a LISA code element that runs before and after a test step, giving you the opportunity to massage the data in the result, or store values in properties.

For more information on Filters see [Adding Filters](#).

Data Sets

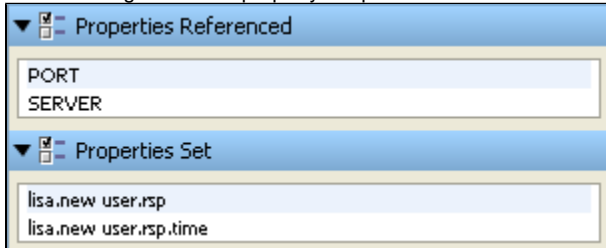
A Data Set is a collection of values that can be used to **set properties** in a Test Case while a test is running. This provides a mechanism to introduce external test data to a Test Case.

For more information on Data Sets see [Using Data Sets](#).

Properties Referenced

This gives a list of properties **used** or **Referenced** by the Test Step.

Select and right click the property to open the extended view and get its variable value.



For more information on Properties, see [Using Properties](#).

Properties Set

This gives a list of properties **Set** by this Test Step.

The Properties Referenced and Set are for a particular step and will change when another step is highlighted.

For more information on Properties, see [Using Properties](#).

3.3 Test Case Quick Start

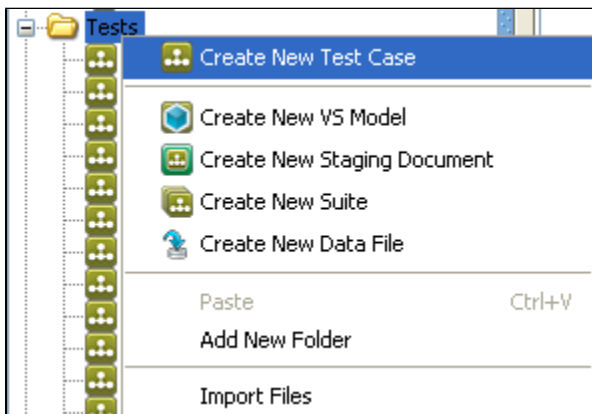
3.3 Test Case Quick Start

From LISA 5.0, you first need to set the LISA Registry to work in a LISA workstation.

Only when you select the Registry, you are allowed to enter the LISA workstation. See [LISA Registry](#).

To get started with the Test Cases - you will first need to create/open a LISA Project in the Workstation. See [Creating a Project](#).

Once you create a Project, you can create a Test Case within, by right clicking the "Tests" folder in the Project tree. See [Creating Test Cases](#).

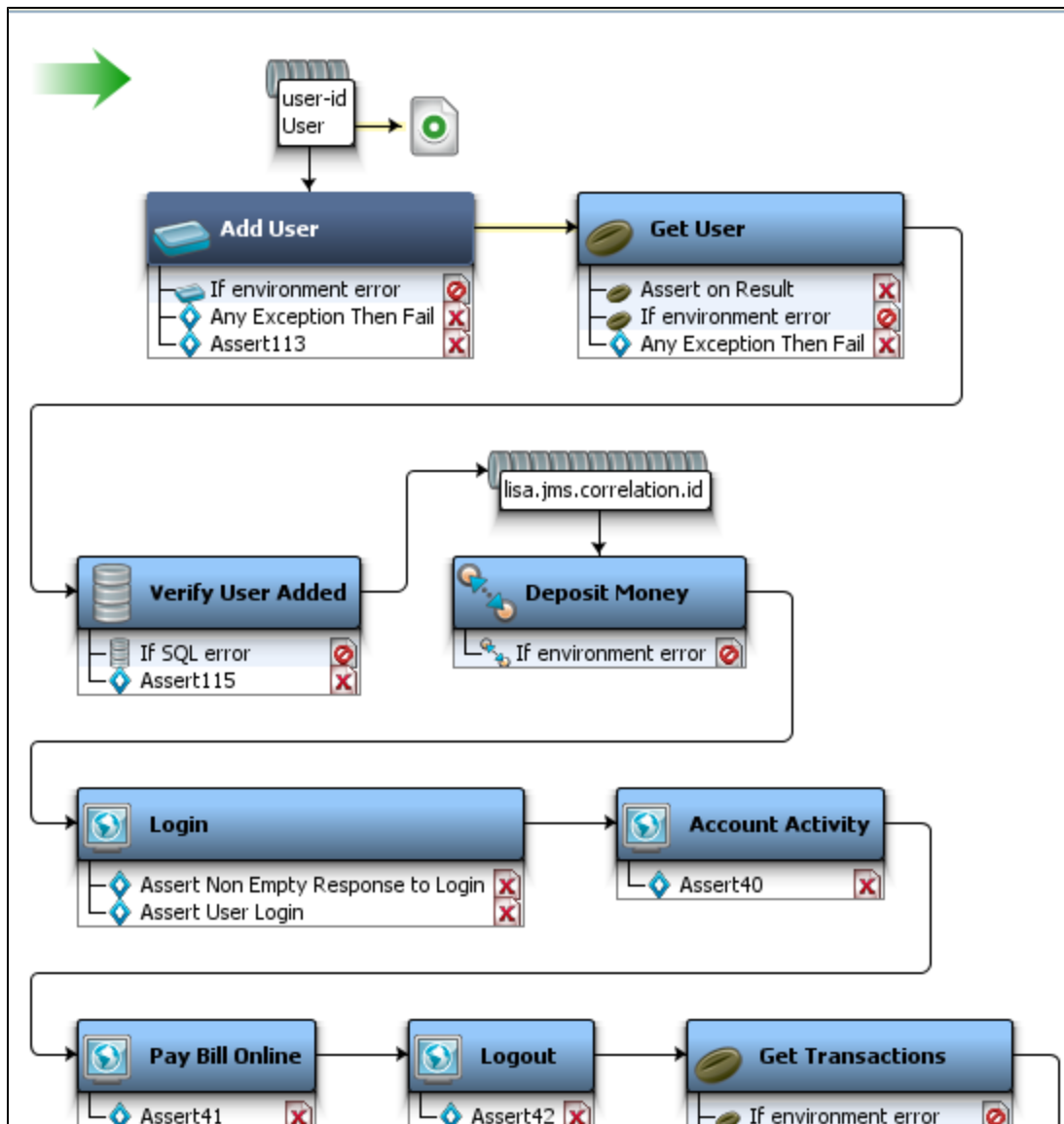


Note: You can open a Test Case either from a valid LISA Project or from outside a project, by using the main menu File > Open > Test Case menu. A non-project test case will open as it used to in earlier versions. If however, we choose a test case within a different project from this menu item, the current project will close, and open the project that the test case resides in, before opening the test case.

Once you open LISA Workstation for the first time, by default a LISA Project named "examples" is opened. You can also open a new Project and create new test cases in it.

For the purpose of illustration, we have created a new project named "sample" as shown below. The project structure contains folders for Configs, Staging Docs, Sub processes, Suites, Tests and Virtual Services. The same folder structure is replicated in the file system in %LISA_HOME% directory.

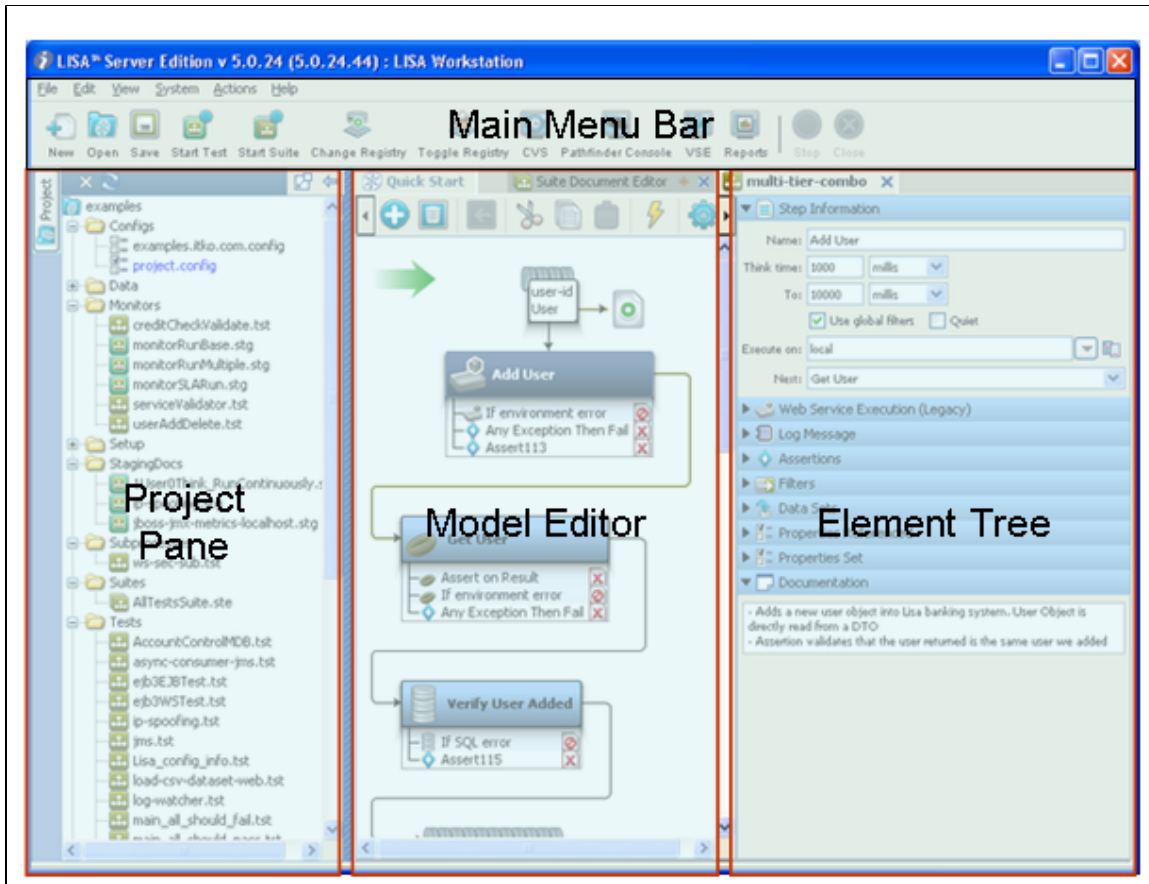
We have opened the "multi-tier-combo.tst" test case as an example Test Case in LISA Model Editor:



LISA Model Editor


LISA Model editor is a place where you can create and view the test case workflow. The test case workflow consists of all test steps, filters, assertions etc applied to a particular test case.

You can clearly see three panes in the LISA Workstation window, which will be referenced frequently in the chapters ahead and hence are explained in brief below:




The workstation is split in three parts:

The Project tree, The Model Editor and the Element Tree.

The **"Project tree"** is the first to be seen. The Project tree is docable and can be opened or closed from a Project button  on the left. At start, LISA opens a default Project "examples" in the Project tree.

For more information, refer to Creating a Project .

The **Model Editor** is the place to create and view Test cases.

The Green arrow  in the Model Editor marks the start of a Test Case.

The **Element Tree** holds the elements required for a Test Case or a Test Step. You can add/delete a Element by clicking on the required Test Case/Test Step Element.

There are some Elements which can be applied at the **Global level** (to an entire Test Case) and some can be applied at a **Step level** (only to a particular Test Step)

There are some tabs in which information should be entered at the starting of a Test Case. For example:

- **Test Case Information tab** -- enter the Test Case name and check if this is a sub process. By default it will be **Step n** -- where n is an incremental value depending on number of steps added.
- **Documentation tab** -- enter the documentation of the Test Case.

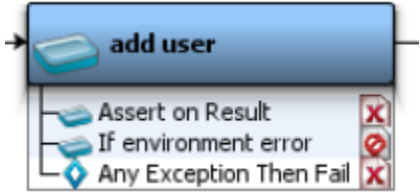
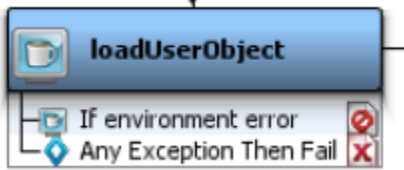



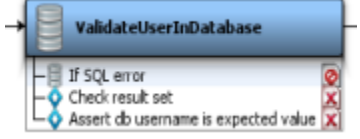

Once you add Steps to this Test Case, a workflow of steps begins to form and a whole new set of elements appear at a Test Step level in the Elements pane.

The Workflow in the Model Editor provides a graphical view of a Test Case. This is very helpful as it gives a quick visual check on the Test Case workflow.

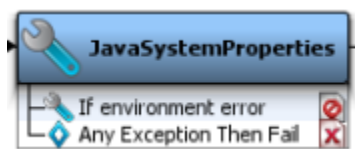
In the Model Editor, each step in the Test Case is represented by an icon in the workflow. The icons change according to the type of the test step.

For example, if you have a Database related step - a database icon and the associated Filters, Assertions will be attached to the step as shown below:

The following table depicts some of the LISA Test Step types and its associated graphical denotations:

Web Services steps	
Java/J2EE steps	
External Sub Process steps	
Virtual Service steps	
JMS steps	
Database steps	
Raw soap steps	

Java script steps



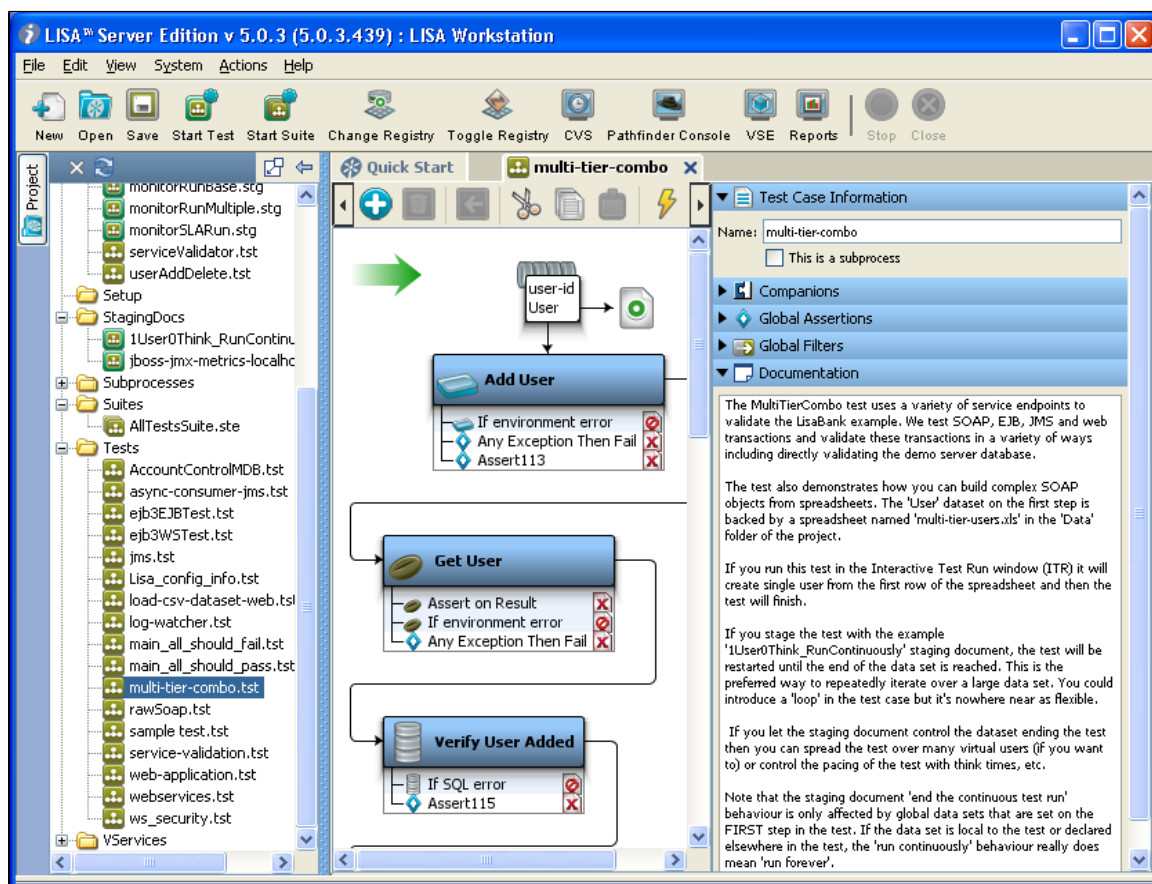
Note: A sample Test Case "multi-tier-combo" Test Case which lies in the LISA examples directory has been referred to mostly in this documentation.

3.4 Multi-tier-combo Testcase

3.4 Multi-tier-combo Testcase

For the purpose of illustration, through out the User guide, we will take a look at the **multi-tier-combo** Test Case, which is in the LISA examples directory.

Once you open this Test Case, it is seen as below:



The MultiTierCombo test uses a variety of service endpoints to validate the LisaBank example. We test SOAP, EJB, JMS and web transactions and validate these transactions in a variety of ways including directly validating the demo server database.

The test also demonstrates how you can build complex SOAP objects from spreadsheets. The 'User' dataset on the first step is backed by a spreadsheet named 'multi-tier-users.xls' in the 'Data' folder of the project.

If you run this test in the **Interactive Test Run window (ITR)** it will create single user from the first row of the spreadsheet and then the test will finish.

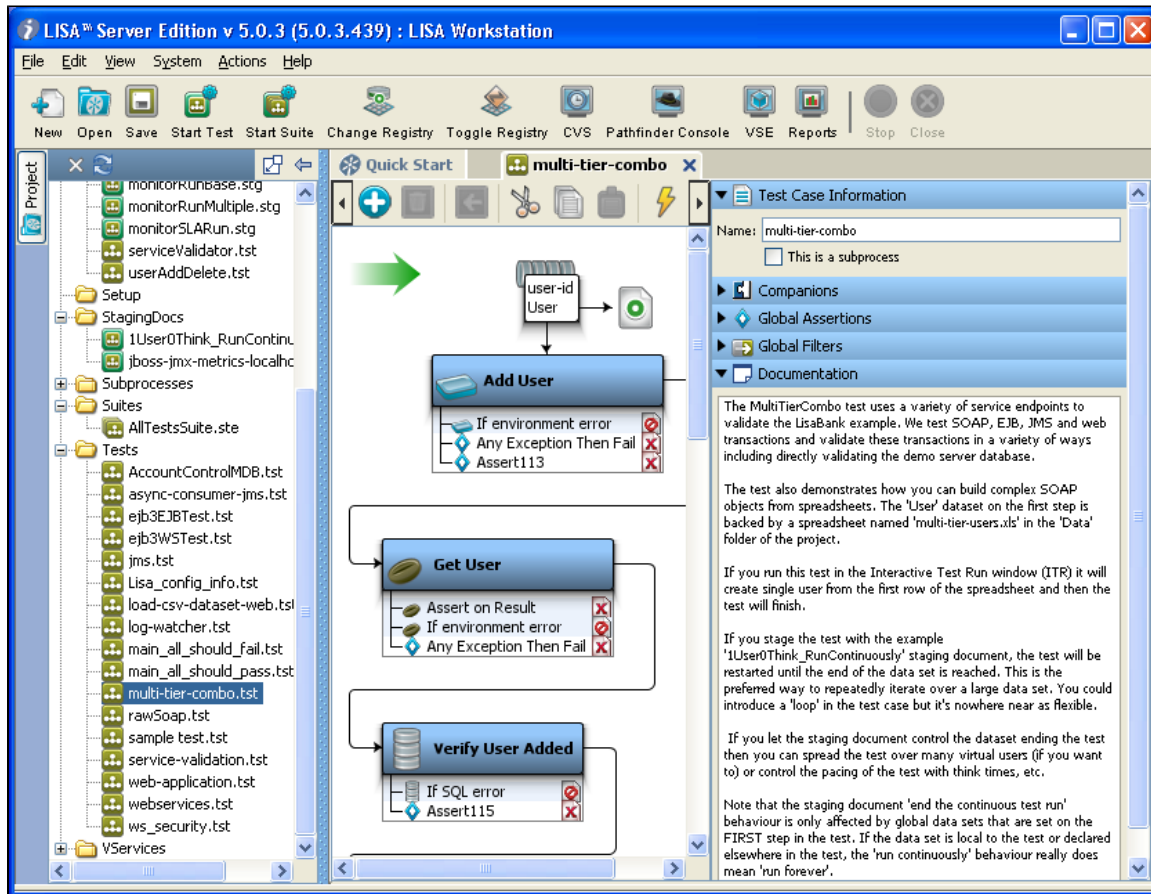
If you stage the test with the example '1User0Think_RunContinuously' **staging document**, the test will be restarted until the end of the data set is reached. This is the preferred way to repeatedly iterate over a large data set. You could introduce a 'loop' in the test case but it's nowhere near as flexible.

If you let the staging document control the dataset ending the test then you can spread the test over many virtual users (if you want to) or control the pacing of the test with think times, etc.

Note that the staging document 'end the continuous test run' behaviour is only affected by global data sets that are set on the FIRST step in the test. If the data set is local to the test or declared elsewhere in the test, the 'run continuously' behaviour really does mean 'run forever'.

Notice the "example" project folders being opened in the Project pane and a set of Test Case Elements in the Element Tree.

Here you can see a **Test Case Information** tab in which you can see the name of the Test Case. You can rename the Test Case here in case you need to do so.



Once you click on a Test Step (**Add User**), the panel on the right changes to show the test step related elements.

The **"Test case information"** tab will change to a **"Step Information"** tab in which you can write the name of the test step and set its properties.

Creating Test Cases reinforces the approach that LISA takes internally, and discusses how the various elements of a Test Case can be viewed in terms of a workflow.

4. LISA Tutorials

4. LISA Tutorials

Welcome to LISA Getting Started Tutorials!

This is the place where you should get started working and knowing LISA!

This section is made up of a series of simple tutorials illustrating various aspects of LISA. The tutorials are sequential and should be completed in order.

The first few tutorials walk you through using LISA Workstation, to build simple test cases and become familiar with very important LISA concepts like LISA Project, Properties, Data Sets, Filters and Assertions.

The subsequent tutorials illustrate deeper knowledge about how to set up test steps to interact with and test several common technologies including Web pages, Java objects (POJOs), Enterprise JavaBeans (EJBs), Web Services, and databases.

Several of these tutorials use the LISA Demo Server as the system under test (SUT). You can use the Demo Server installed with LISA (locally on your workstation), or you can reference a similarly configured demo server at <http://examples.itko.com/itko-examples/>.

For more information on installing the LISA Demo Server see [1.2.3 Downloading the Installer](#) and [2.1 Installing the JBOSS Demo Server](#).

After completing the tutorials, you should feel confident building test cases in your own LISA environment.

The following tutorials are available.

- [4.1 Tutorial 1 - Basic Concepts - LISA Project, Test Case & Properties](#)
- [4.2 Tutorial 2 - Basic Concepts - LISA Data Sets](#)
- [4.3 Tutorial 3 - Basic Concepts - Filters and Assertions](#)
- [4.4 Tutorial 4 - Manipulating Java Objects \(POJOs\)](#)
- [4.5 Tutorial 5 - Running a Demo Server Web Application](#)
- [4.6 Tutorial 6 - Testing a Web Site](#)
- [4.7 Tutorial 7 - Testing an Enterprise JavaBean \(EJB\)](#)
- [4.8 Tutorial 8 - Testing a Web Service](#)
- [4.9 Tutorial 9 - Examining and Testing a Database](#)
- [4.10 Tutorial 10 - Staging a Quick Test](#)

4.1 Tutorial 1 - Basic Concepts - LISA Project, Test Case & Properties

4.1 Tutorial 1 - Basic Concepts: LISA Project, Test Case & Properties

In this tutorial, you will learn how to create a **LISA Project** and a **LISA Test Case**.

You will also look at the use of **LISA properties** to understand the various places from which they can originate.

LISA concepts discussed

In this tutorial, do the following:

- Create and save a new LISA Project
- Create and save a new LISA test case
- Create LISA properties within LISA
- Add simple test steps
- Use the Interactive Test Run facility

Prerequisites


LISA Workstation is installed and LISA license credentials are entered.

You have read the [LISA Basic concepts](#) and are familiar with these concepts.

Steps

Step 1 - Start LISA Workstation

To start the LISA Workstation,

- Double-click the LISA desktop icon  ,
or Select **Start > Programs > LISA > LISAWorkstation**.

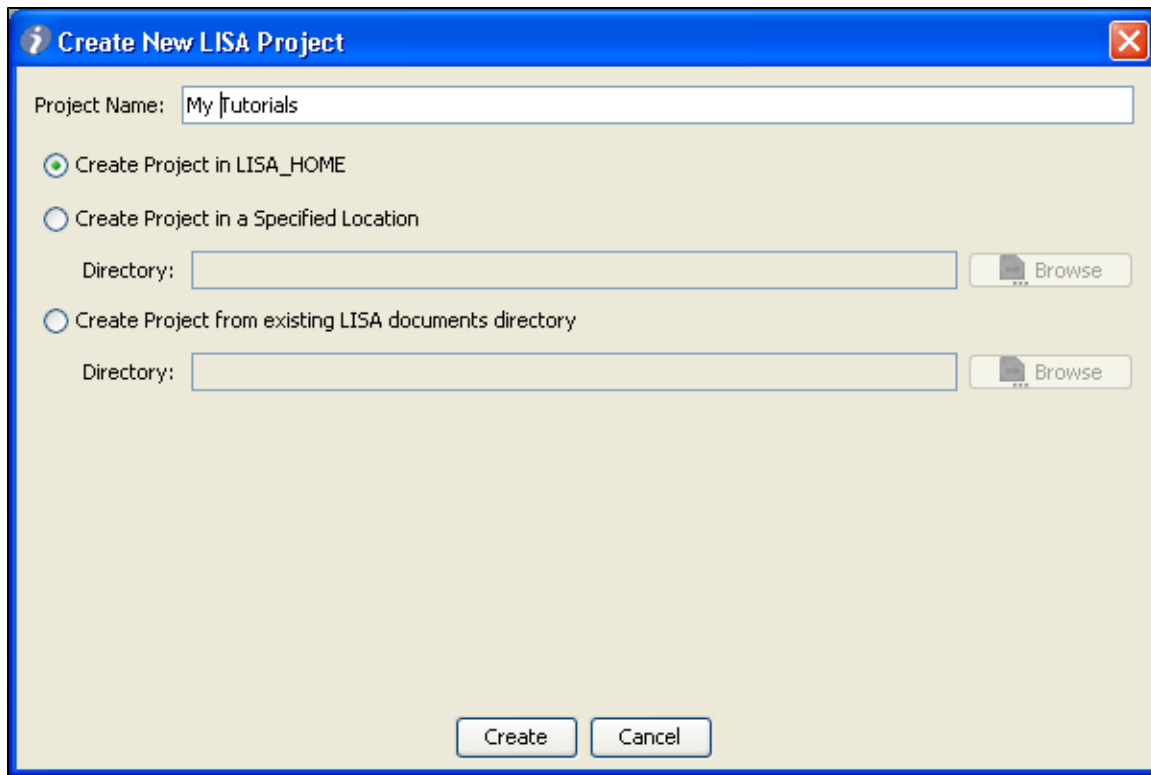
Step 2 - Create a LISA Project

A LISA project will hold all the test case example files which are required for the tutorials. All the files we create in this Getting Started Guide are stored in this Project.

Note - At a time, you can only open one Project in the LISA Workstation.

To create a New Project,

- Open LISA Workstation and click **New > Project**.




The image shows a Windows-style dialog box titled "Create New LISA Project". It has a blue title bar with a question mark icon on the left and a red close button on the right. The main area is light beige. At the top, there is a text field labeled "Project Name:" containing the text "My Tutorials". Below this, there are three radio button options. The first option, "Create Project in LISA_HOME", is selected. The second option, "Create Project in a Specified Location", is followed by a text field labeled "Directory:" and a "Browse" button with a folder icon. The third option, "Create Project from existing LISA documents directory", is also followed by a text field labeled "Directory:" and a "Browse" button with a folder icon. At the bottom of the dialog, there are two buttons: "Create" and "Cancel".

Create New LISA Project

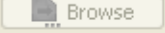
Project Name:

☒ Create Project in LISA_HOME

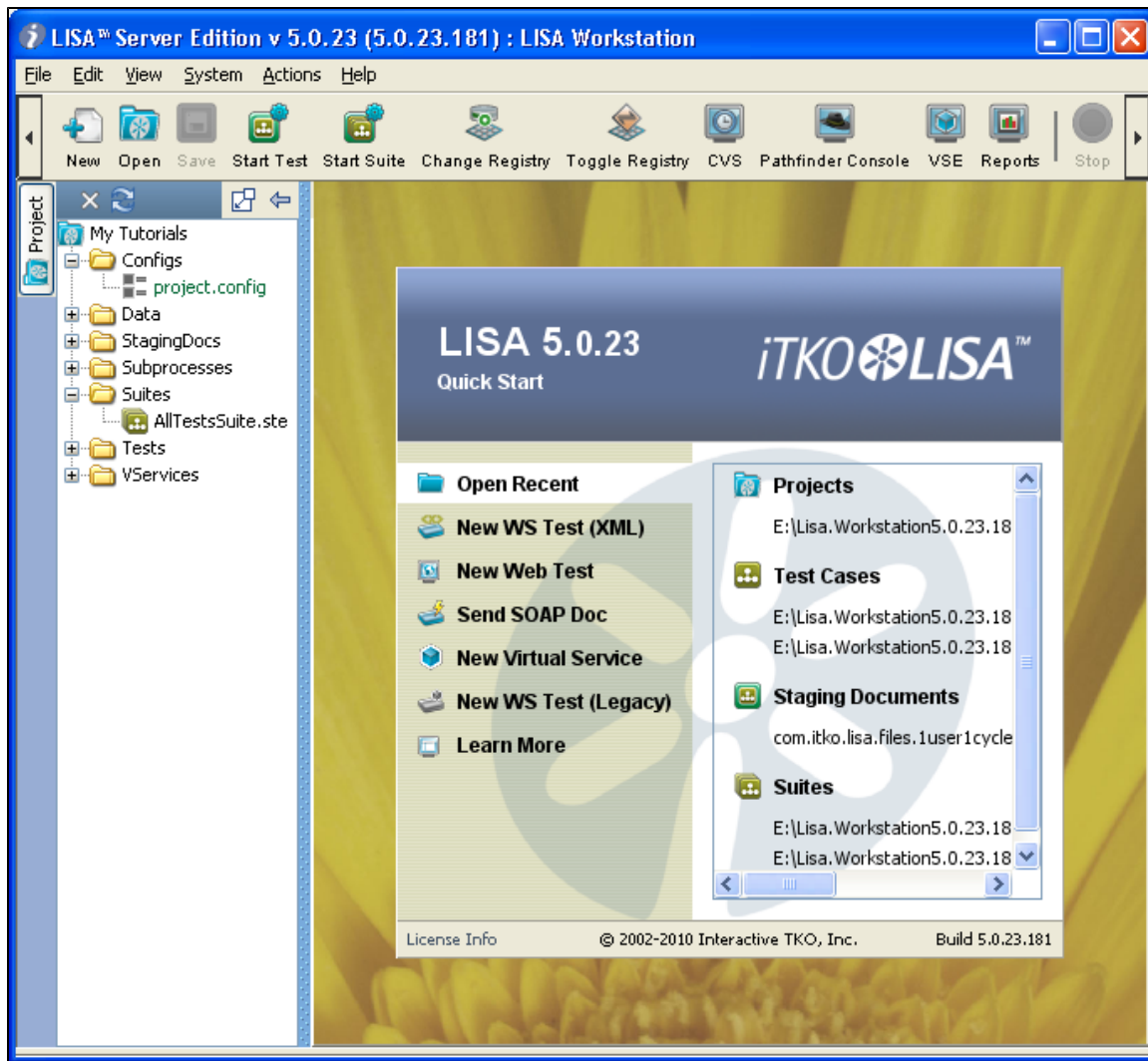
☐ Create Project in a Specified Location

Directory: 

☐ Create Project from existing LISA documents directory

Directory: 

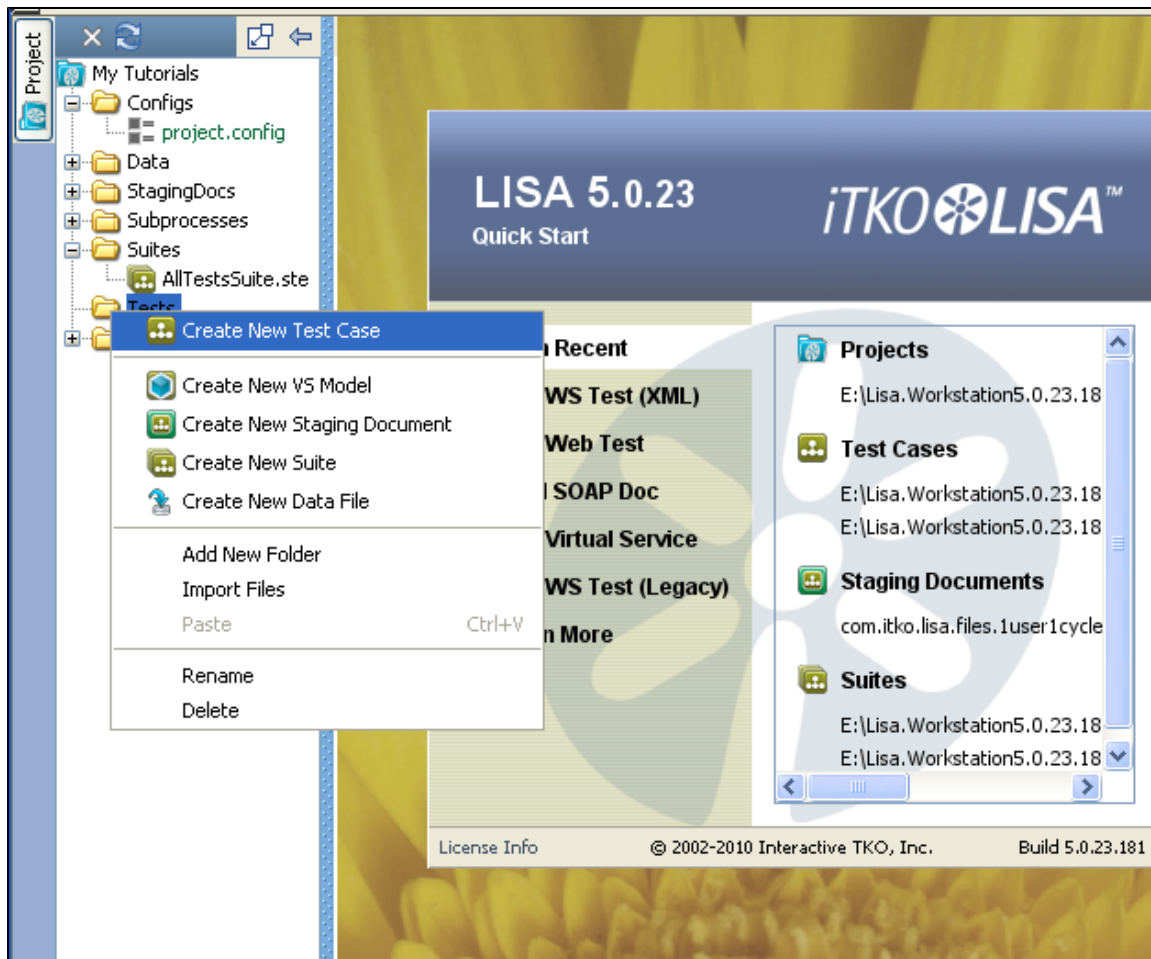
1. Enter the new project name as **My Tutorials**
2. Create the project in **LISA_HOME** directory.
3. Click **Create** to create the new project.



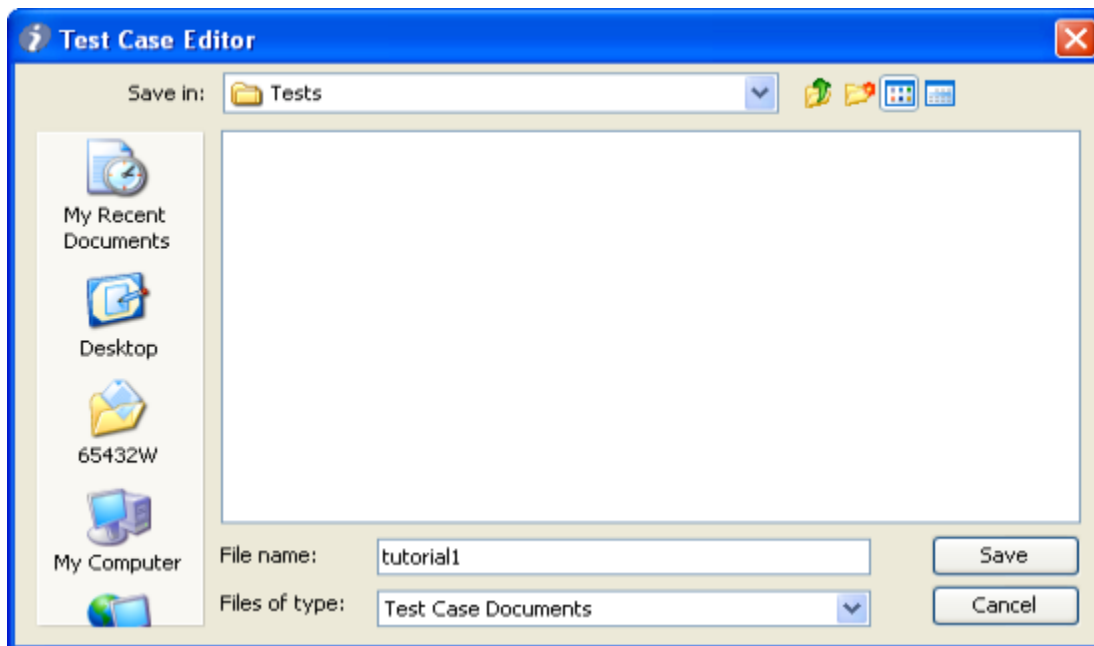
Step 3 - Create a new Test Case

To create a new test case,

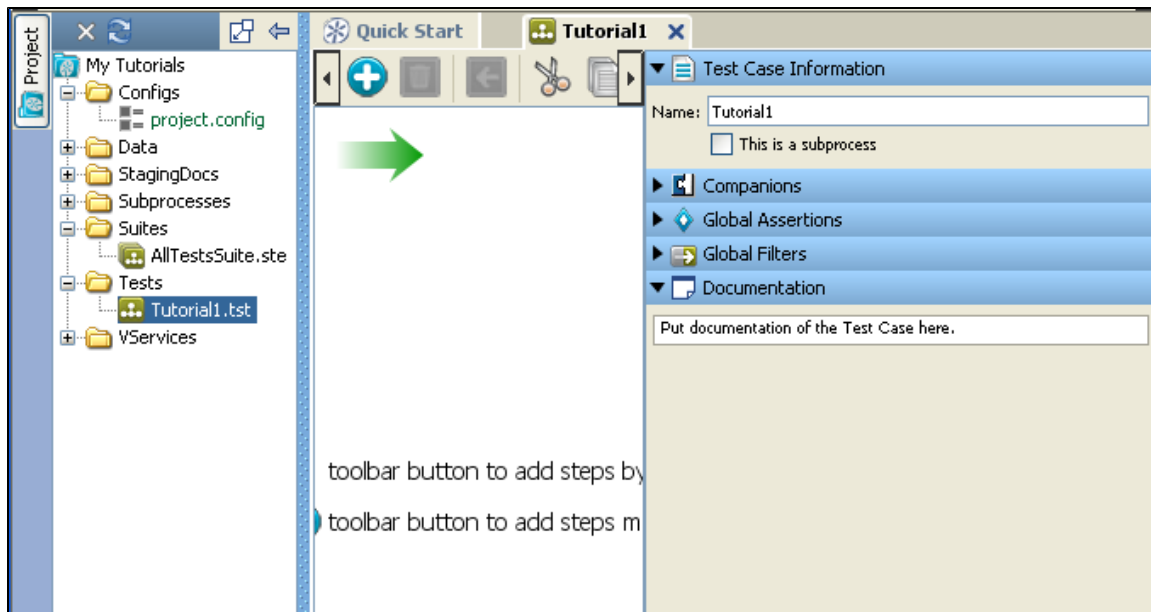
1. Open the newly created project - **My Tutorials**.
2. Select the **Tests** folder and right click to open a new menu.
3. Click **Create New test case**.



4. Name the new test case **tutorial1** and click **Save**.



The LISA Workstation Editor opens a new tab named **tutorial1**.



Initially, the **project.config** is Green in color and has no values assigned.

- Right click the project.config and select "**make active**" to make this as the active configuration.

Now it turns Blue in color as shown below:

[Return to previous step](#)

Step 4 - Adding a Property to Configuration

We need to set a global property in the configuration, so that you access it later in the tutorial.

The default LISA configuration is named as "**project.config**" and it is created automatically for a new project. It is seen under the **Configs** subfolder in the project pane.

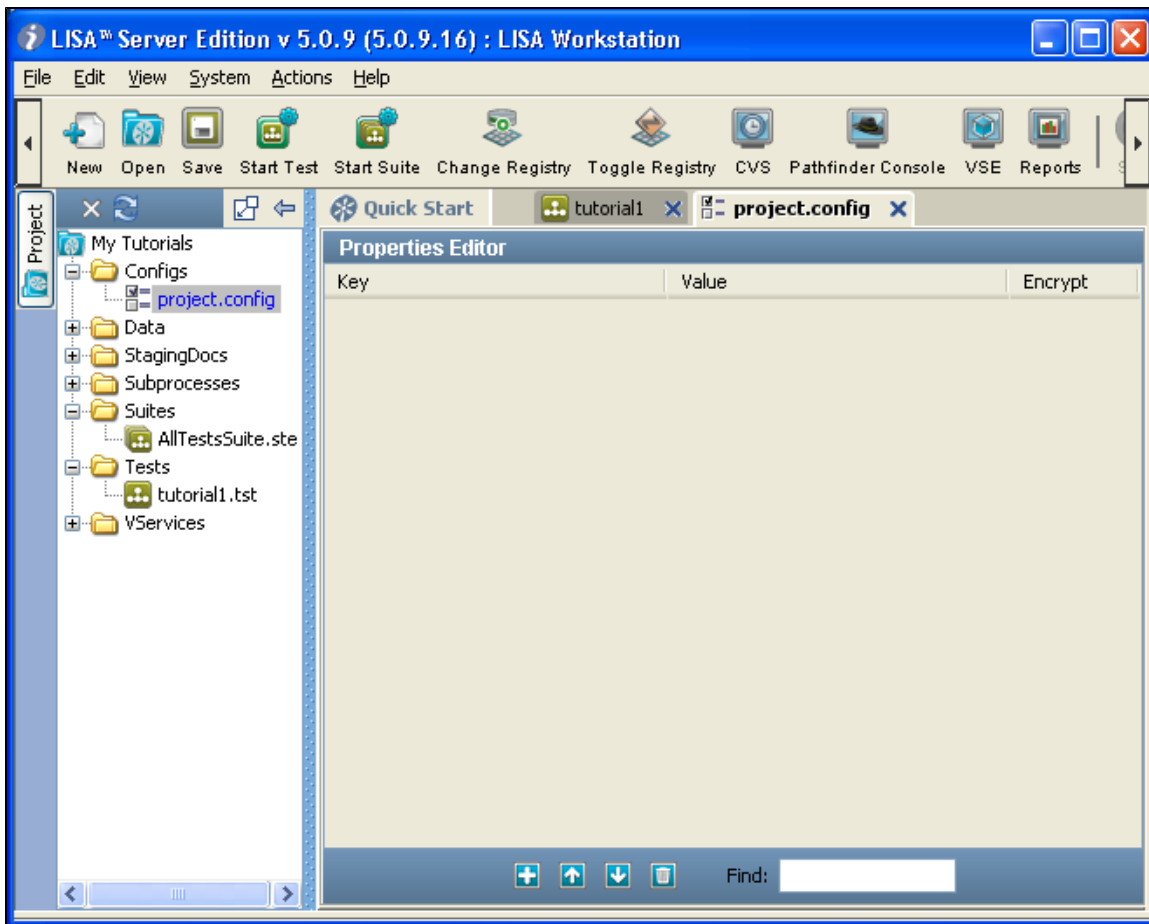
You can add the properties to the project.config file and if required can also create a new config file.


Note - It is mandatory to add properties to the project.config file.

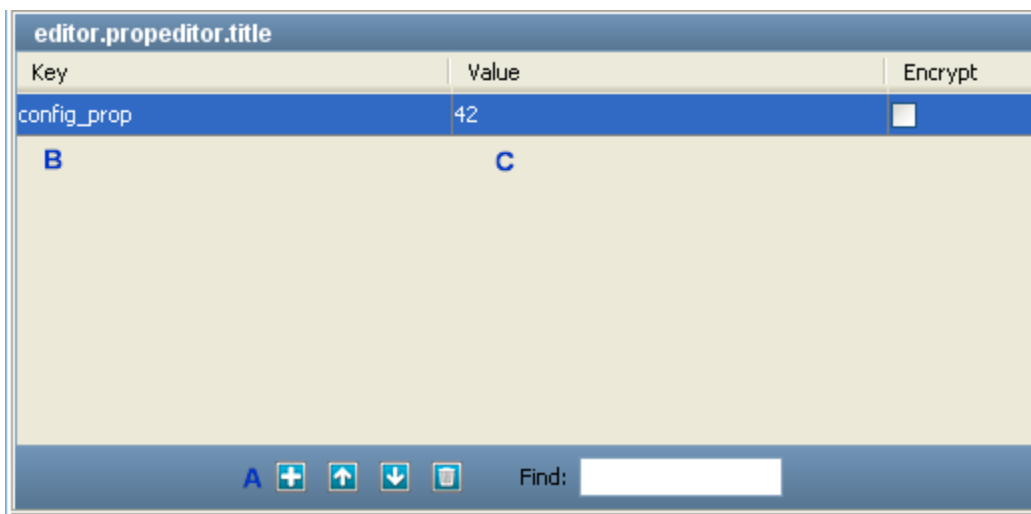
To add a property to project.config,

1. Double click on the "**project.config**", in **My tutorials > Configs** folder in the project pane.

This opens the configuration editor for Project.config.



2. Click the add icon  at the bottom, to add a new row (see A below).
3. In the new row **Key** field, enter **config_prop** for the properties name (see B below).
4. In the **Value** field, enter **42** (see C below).



5. Click Save

The property has been added to the configuration file.

[Return to previous step](#)

Step 5 - Add a Simple Test Step

We will add a simple test step - Output Log Message step, to write text out to the log file.
The text however, can include properties.

To add a Test Step,

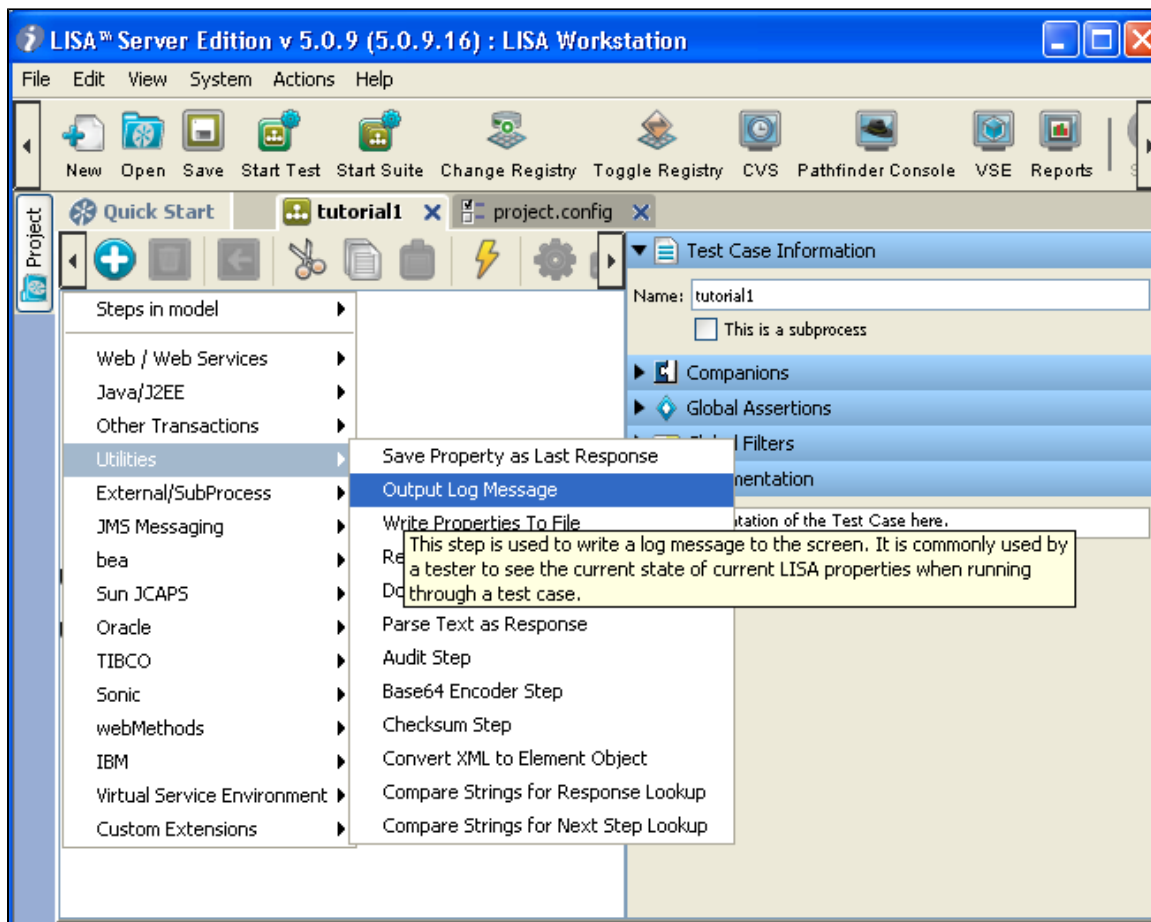
1. Click on the **tutorial1** Tab.



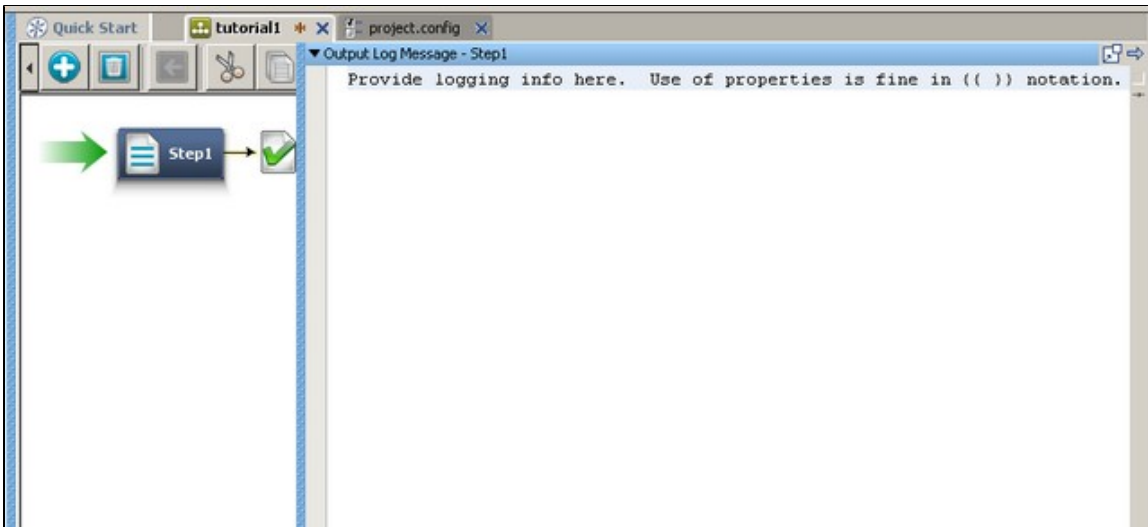
2. Click the **Add Step** button

The **Add Step** menu is displayed.

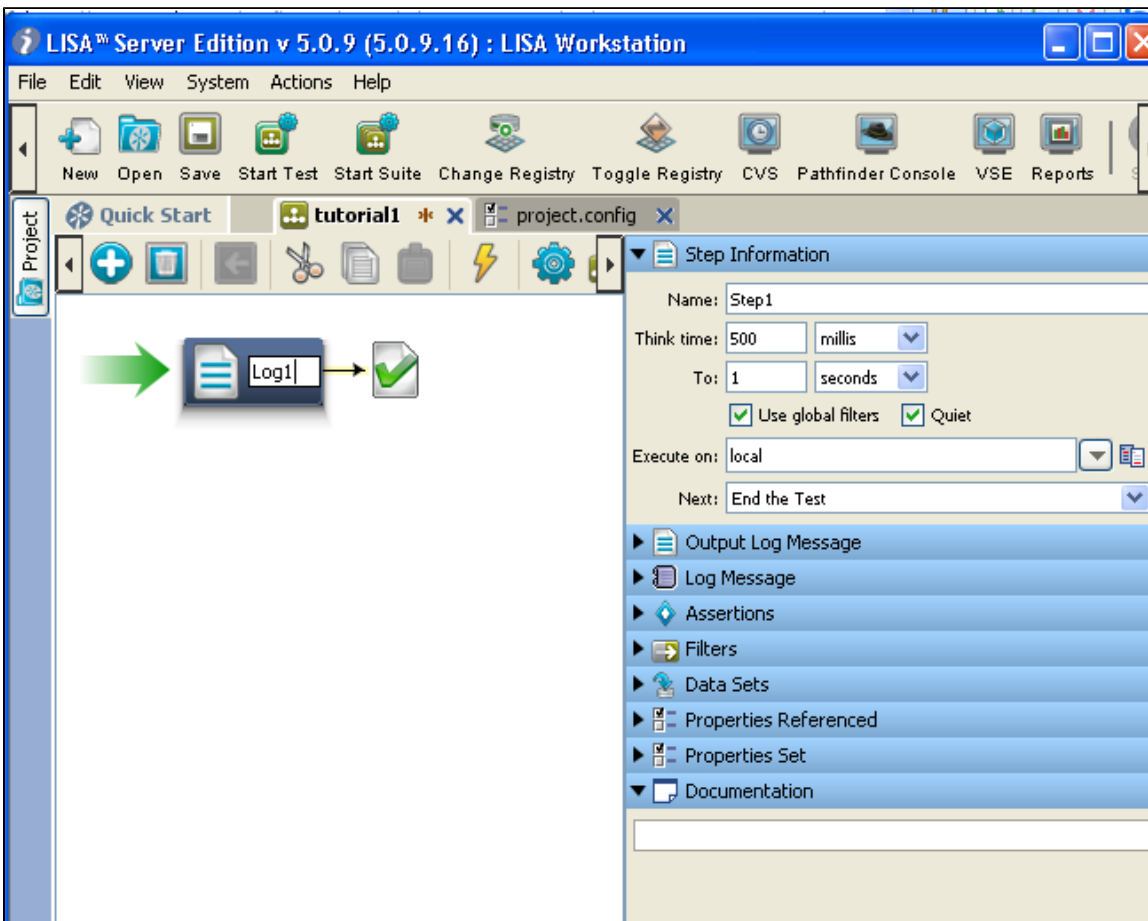
3. Click **Utilities**, and select **Output Log Message**.



Step1 has now been added to the model editor.



4. Click on Step1 and rename it by right clicking and selecting Rename. Rename it as **Log1**.



5. Make sure Log1 is still selected and click on the tab **Output Log Message** in the step element pane to open the log editor and enter a log message.

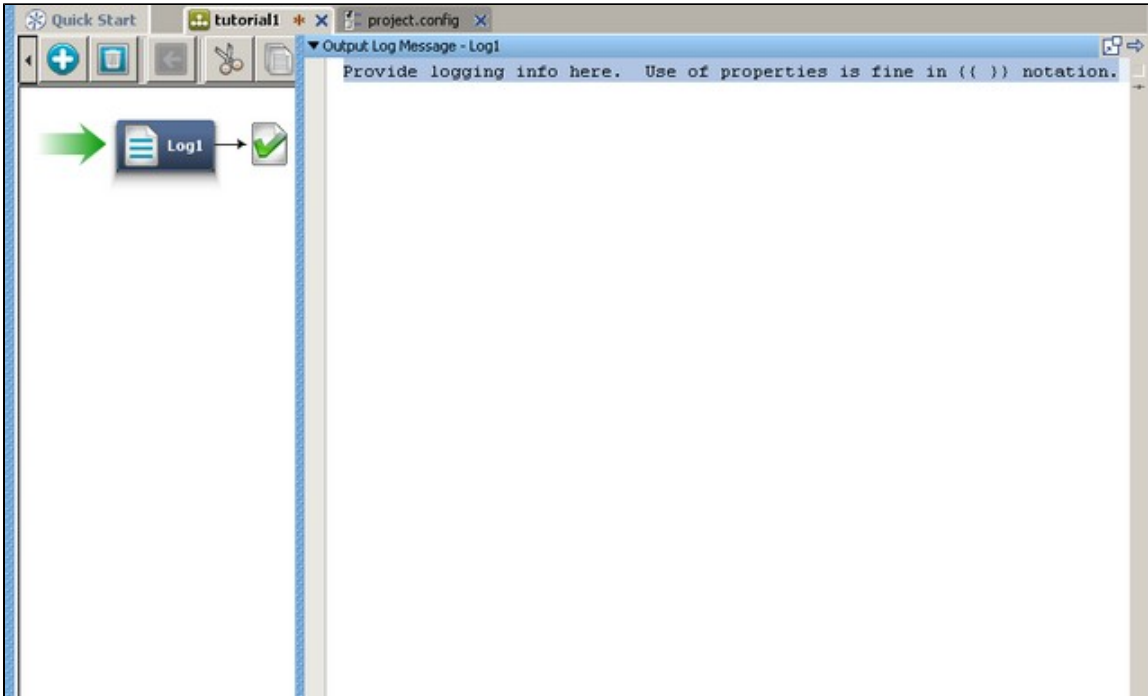
[Return to previous step](#)

Step 6 - Add a simple Log message

In this step, add a simple log message that contains text and properties and set LISA properties.

To add a log message:

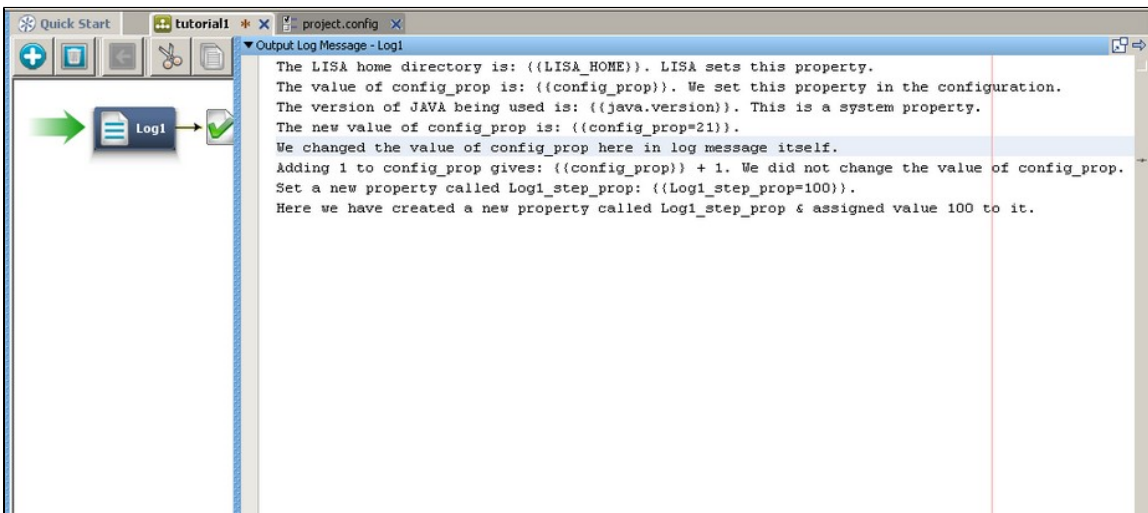
1. In the **Log Editor**, highlight the place-holder text and press **<Delete>** to clear the field.



2. In the **Log Editor** on the right, copy and paste the following text:

The LISA home directory is: {{LISA_HOME}}. LISA sets this property.
The value of config_prop is: {{config_prop}}. We set this property in the configuration.
The version of JAVA being used is: {{java.version}}. This is a system property.
The new value of config_prop is: {{config_prop=21}}. We changed the value of config_prop here in log message itself.
Adding 1 to config_prop gives: {{config_prop}} + 1. We did not change the value of config_prop.
Set a new property called Log1_step_prop: {{Log1_step_prop=100}}.
Here we have created a new property called Log1_step_prop & assigned value 100 to it.

Now it will look like:



The log message is plain text and LISA properties are used with LISA syntax double curly braces ({{ }}).

The properties being used can originate from several sources as mentioned below:

- The installation set up **LISA_HOME**.
- You set up **config_prop** in the configuration for this test case.

- **java.version** is a system property.
- You set up **Log1_step_prop** in this step by creating it in the log message.

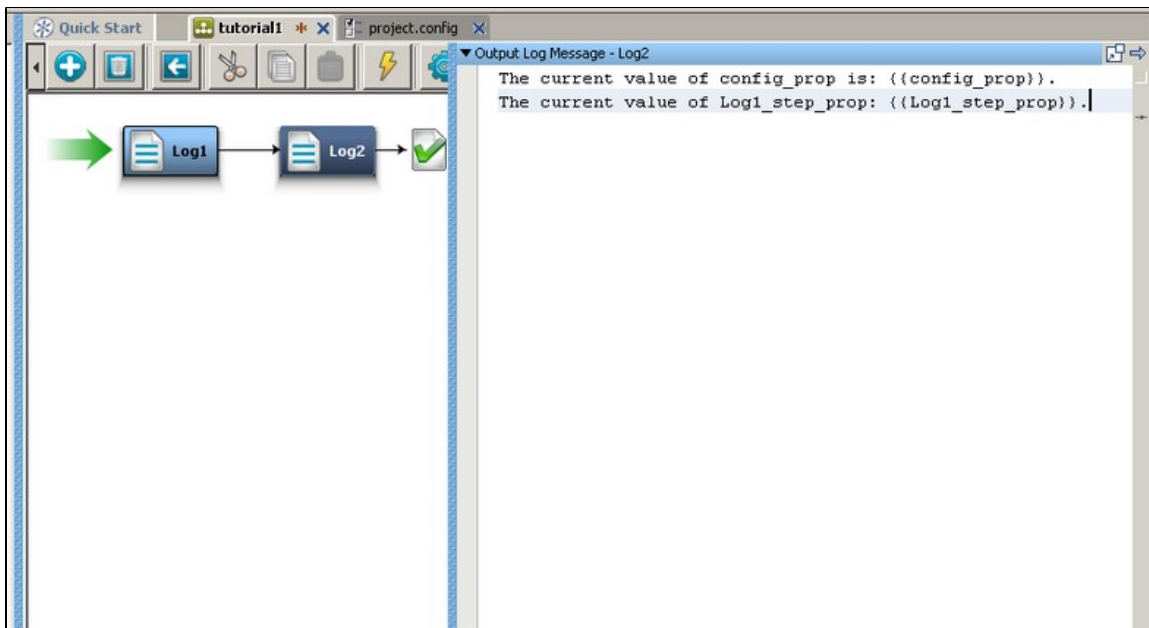
[Return to previous step](#)

Step 7 - Add another Output Log Message

To add a Second Output Log Message step:

1. Repeat [Step 5](#), and name the step **Log2**.
2. In the **Log Information** area, copy and paste the following text into the editor:

The current value of config_prop is: {{config_prop}}.
The current value of Log1_step_prop: {{Log1_step_prop}}.



[Return to previous step](#)

Step 8 - Save the Test Case

To save the Test Case:




From the main toolbar, click the Save icon, or choose **File > Save** menu.

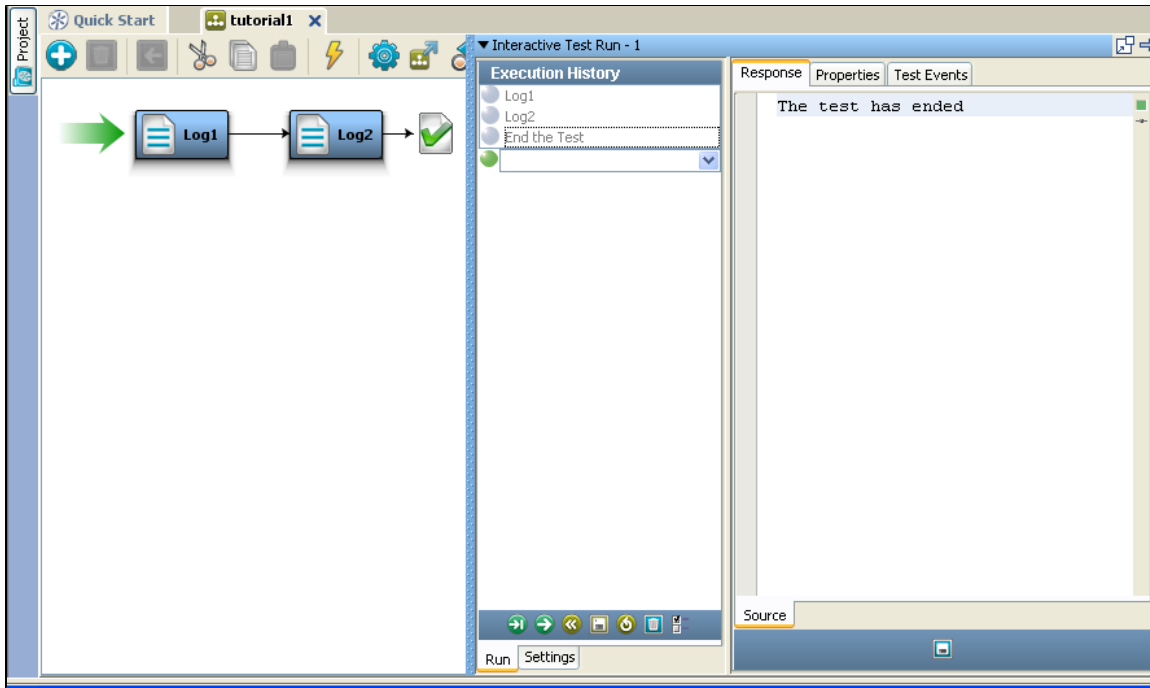
[Return to previous step](#)

Step 9 - Run the Test Step

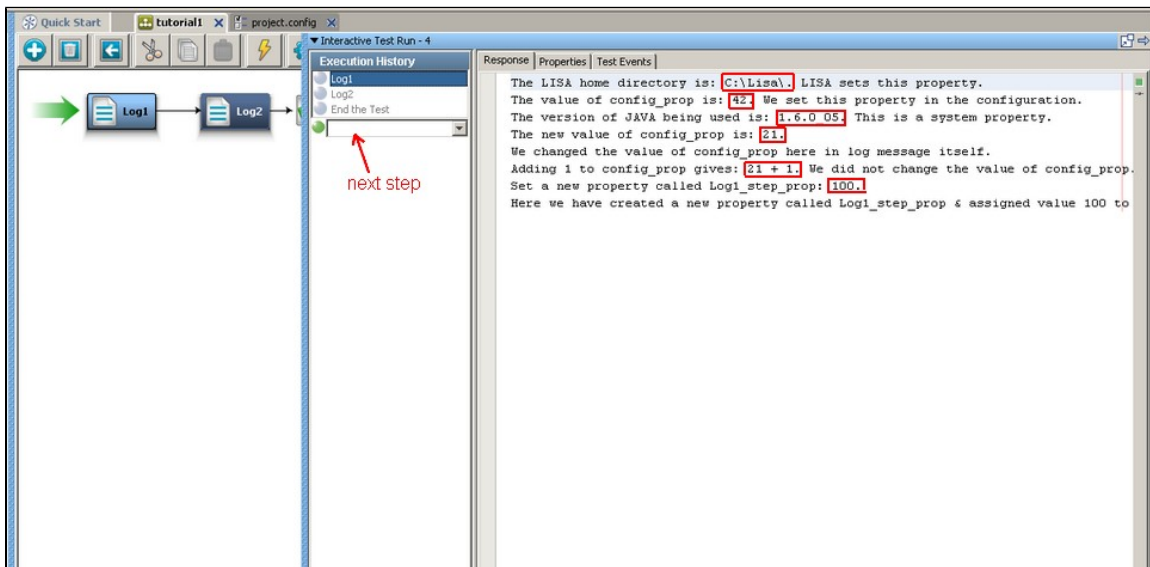
In this step, run the test case, step by step to observe the behavior of the properties that have been set.

To run the Interactive Test Run (ITR) feature:

1. From the toolbar, click **Start ITR** .



2. In the Execution History pane, click **Execute Next Step**  to run Log1 step.
The Log1 step has run. The **Response** tab displays the response from the Log1 step. The properties have been replaced by the actual values, as shown in the red boxes:



[Return to previous step](#)

Step 10 - To Observe Property values

To Observe the Property values in ITR,

1. Click the **Properties** tab in the ITR.
You can see the value of each property before and after the execution of the Log1 step.
A value created by the step is highlighted in **green**. A value modified in the step is highlighted in **yellow**.

Interactive Test Run - 1

Execution History

- Log1
- Log2
- End the Test

Log1

Response Properties Test Events

Initial property values may be changed prior to executing the step. They are read-only after execution. Create a new property by editing an existing key.


Key	Value	Previous Value
LISA_LAST_STEP	Log1	
LISA_DOC_PATH	C:\Lisa\Projects\my tutorials\Tests	C:\Lisa\Projects\my tutorials\Tests
Log1_step_prop	100	
lisa.scriptEngine	NotSerializableStateWrapper >> bsh.Inter...	
lisa.Log1.rsp.time	15	
LISA_TC_PATH	C:\Lisa\Projects\my tutorials\Tests	C:\Lisa\Projects\my tutorials\Tests
robot	0	0
LISA_HOST	Haridwar	Haridwar
LISA_PROJ_NAME	my tutorials	my tutorials
lisa.Log1.rsp	The LISA home directory is: C:\Lisa\, LISA ...	
LISA_USER	snehalg	snehalg
instance	0	0
testCaseId	62303337383862642D623666302D3439	62303337383862642D623666302D3439
LISA_TC_URL	file:/C:/Lisa/Projects/my%20tutorials/Tests	file:/C:/Lisa/Projects/my%20tutorials/Tests
LASTRESPONSE	The LISA home directory is: C:\Lisa\, LISA ...	
config_prop	21	42
testCase	tutorial1	tutorial1
LISA_PROJ_ROOT	C:\Lisa\Projects\my tutorials	C:\Lisa\Projects\my tutorials
LISA_DOC_URL	file:/C:/Lisa/Projects/my%20tutorials/Tests	file:/C:/Lisa/Projects/my%20tutorials/Tests

2. Check these values by looking at the response in Step 9.

[Return to previous step](#)

Step 11 - Execute the Log2 step

To execute the Log2 step,

1. In the ITR, click **Execute Next Step**  to run the **Log2** step.
2. Click the **Response** tab to view the response.
3. Click the **Properties** tab to view the property values.

Interactive Test Run - 1

Execution History

- Log1
- Log2
- End the Test

Log1

Response Properties Test Events

Initial property values may be changed prior to executing the step. They are read-only after execution. Create a new property by editing an existing key.

Key	Value	Previous Value
LISA_LAST_STEP	Log2	Log1
LISA_DOC_PATH	C:\Lisa\Projects\my tutorials\Tests	C:\Lisa\Projects\my tutorials\Tests
Log1_step_prop	100	100
lisa.scriptEngine	NotSerializableStateWrapper >> bsh.Interpreter@1aac075	NotSerializableStateWrapper >> bsh.Interpreter@1aac...
lisa.Log1.rsp.time	15	15
LISA_TC_PATH	C:\Lisa\Projects\my tutorials\Tests	C:\Lisa\Projects\my tutorials\Tests
robot	0	0
LISA_HOST	Haridwar	Haridwar
LISA_PROJ_NAME	my tutorials	my tutorials
lisa.Log1.rsp	The LISA home directory is: C:\Lisa\, LISA sets this prop...	The LISA home directory is: C:\Lisa\, LISA sets this prop...
LISA_USER	snehalg	snehalg
instance	0	0
lisa.Log2.rsp	The current value of config_prop is: 21. The current valu...	
testCaseId	62303337383862642D623666302D3439	62303337383862642D623666302D3439
LISA_TC_URL	file:/C:/Lisa/Projects/my%20tutorials/Tests	file:/C:/Lisa/Projects/my%20tutorials/Tests
LASTRESPONSE	The current value of config_prop is: 21. The current valu...	The LISA home directory is: C:\Lisa\, LISA sets this prop...
config_prop	21	21
testCase	tutorial1	tutorial1
LISA_PROJ_ROOT	C:\Lisa\Projects\my tutorials	C:\Lisa\Projects\my tutorials
LISA_DOC_URL	file:/C:/Lisa/Projects/my%20tutorials/Tests	file:/C:/Lisa/Projects/my%20tutorials/Tests
lisa.Log2.rsp.time	0	

[Return to previous step](#)

Step 12 - Save the Test Case

To save the test case,



From the main toolbar, click the Save icon or choose **File > Save** menu.

[Return to previous step](#)

Review

In this tutorial, you took a first look at LISA properties. You saw that LISA properties are denoted using a special LISA syntax, **property name**. You can also set properties using a variation of this syntax **property name=value**. After you set a property, use or modify it in subsequent steps in a test case.

In this tutorial, you did the following:

- Learned how to create and save a new test case.
- Learned how to add a simple test step (Output Log Message).
- Used a configuration to store properties.
- Saw a brief glimpse of the Interactive Test Run utility.

4.2 Tutorial 2 - Basic Concepts - LISA Data Sets

4.2 Tutorial 2 – Basic Concepts - LISA Data Sets

In this tutorial, you will learn how to create and use a **simple LISA Data Set**.

You will also know how to provide this Data set data to a test case.

LISA Concepts Discussed

In this tutorial, do the following:

- Create a simple data set.
- Use the data set in a variety of ways.
- Iterate through a series of test steps using a data set.

Prerequisites

Complete Tutorial 1. Open LISA Workstation if not already open.

Steps

Step 1 - Creating a Simple Data Set

We will use a **simple comma delimited text file** as the data set.

(This is just one of several options available to create a Data Set).

To create the data set,

1. In a text editor like Notepad, create a text file as shown below.

Copy and paste the following properties and values into Notepad:

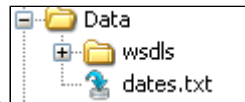
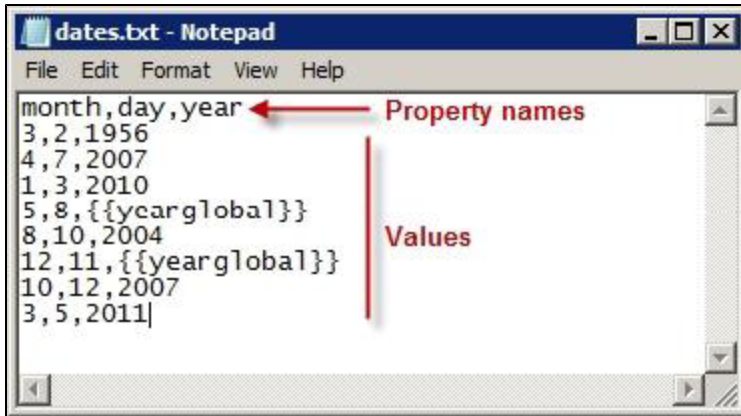
```
month,day,year
3,2,1956
4,7,2007
1,3,2010
5,8,yearglobal
8,10,2004
12,11,yearglobal
10,12,2007
3,5,2011
```

Note - Do not use spaces in the text file.

The first row specifies the names of the properties to which this data is assigned (month, day, year).

The remaining rows represent the data that is read and used in the test case. You can also use properties in the data set.

2. Save the file as **dates.txt** in the "**Data**" subfolder of the "**My tutorials**" project.



You can see the saved text file, added in the Data folder in the left project pane.

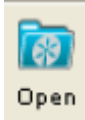
Step 2 - Open LISA Workstation

Step 3 - Create a Test Case

To create a new test case within the project "My tutorials", we will first need to open the Project and then the Test Case.

To open the Project,

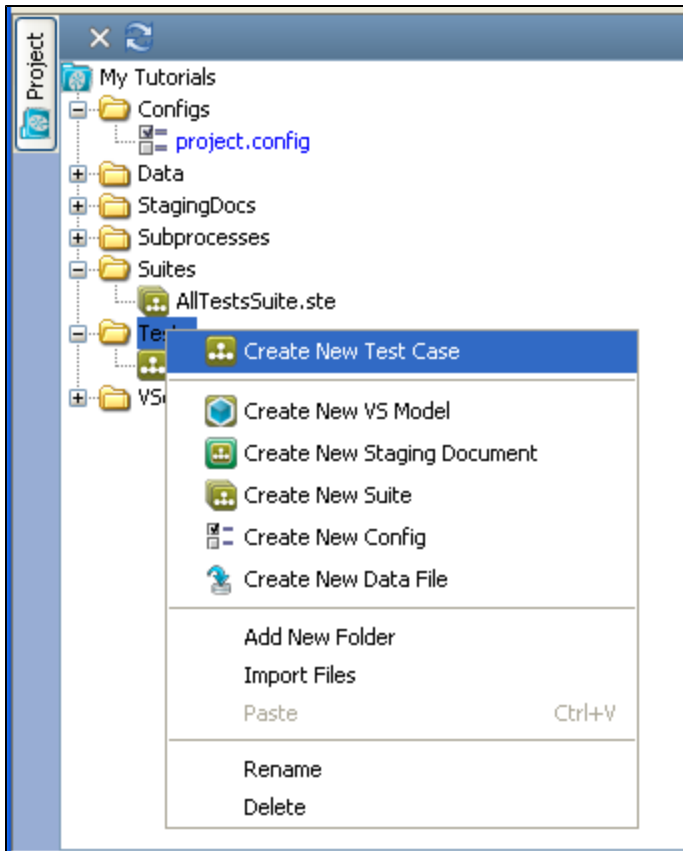
1. Open the project **My tutorials** if it is not already open.



Use the Open icon on the toolbar for the same and browse to the project location.

To create a new Test Case,

2. Right click on the **Tests** subfolder and choose **Create New Test Case** option.



3. In the Test Case Editor window, navigate to the subfolder **Tests**.

4. Name the new test case as **tutorial2** and click **Save**.

Step 4 - Adding a Property to the Configuration


You can Add a property to the test case, by modifying the project.config file.

You can also add the property in a new configuration file, having key-value pairs.

Note - These property values must also be added in the project.config file to be effective.

To add a property to the project.config file,

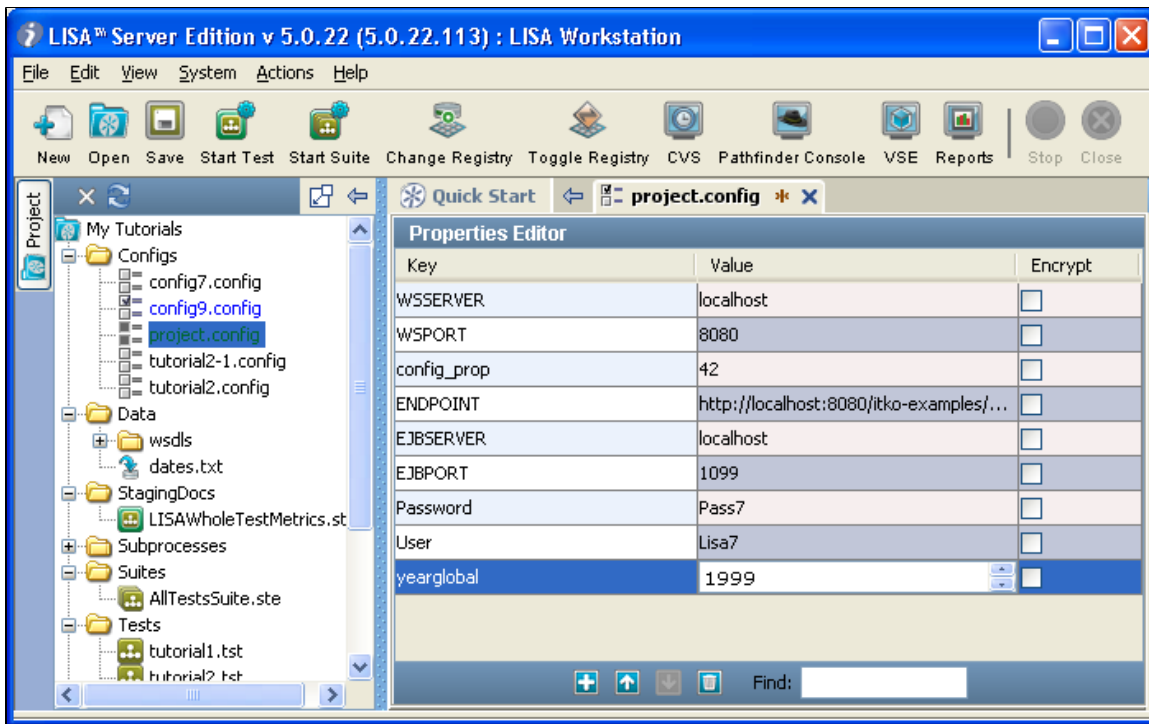
1. Double click and open the project.config editor.

2. In the configuration editor, click the add icon  to add a new row.

3. Enter the property as key-value pair.

- In the new row **Key** field, enter **yearglobal** for the properties name.

- In the **Value** field, enter **1999**.



Step 5 - Add Test Step for Output Log

In this step, use a simple test step, the **Output Log Message** step, to write text out to log. The text, however, can include properties.

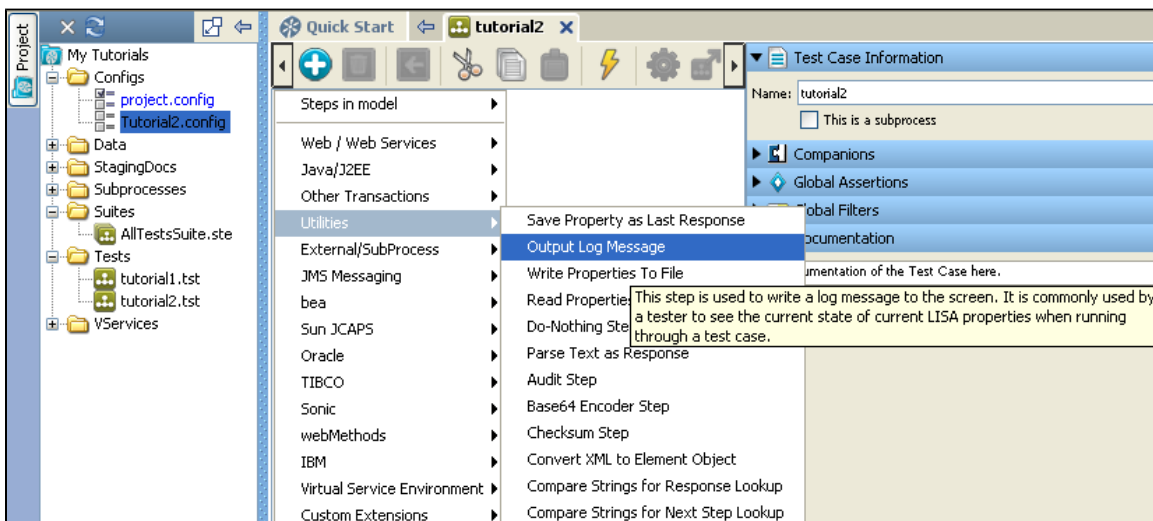
To add a Test Step,



1. Right click in the model editor and click **Add Steps**.

The menu of **Add Steps** is displayed.

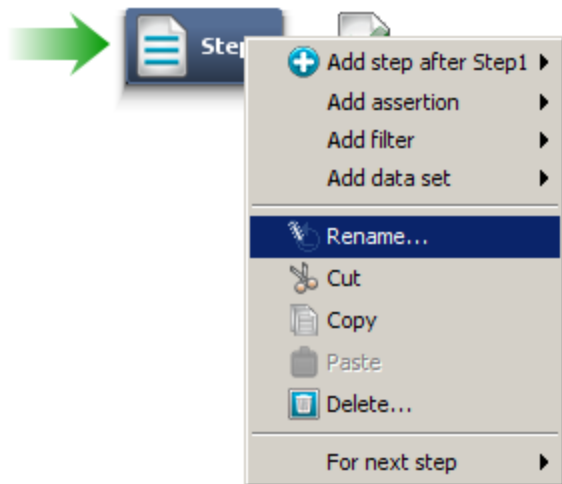
2. Click **Utilities**, and select the **Output Log Message** step type to add the test step.



3. **Step1** has been added to the model editor.

4. Select **Step1** and rename it as **DSstep1** in the Step Information element tab.

You can also rename by right-clicking on Step 1 in Model Editor and selecting Rename from the pop-up menu.



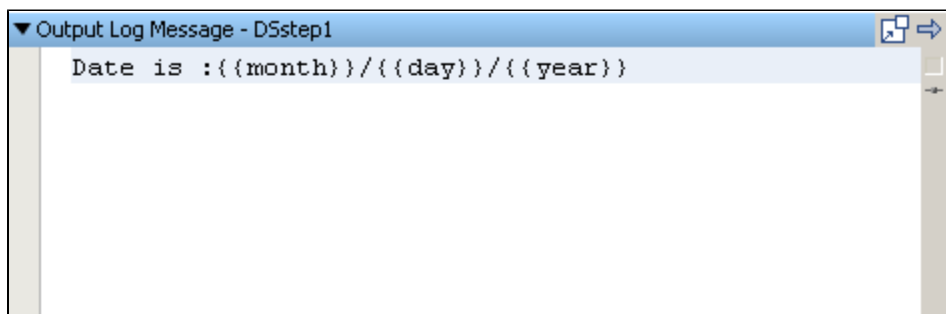
5. Click on the arrow next to **Output Log Message** in the Step Editor section.

Note - The available choices in the Step Editor section will vary based on the type of step selected.

6. In the Log Information area that will slide open when you click the arrow, enter the following log message:

Date is :{{month}}/{{day}}/{{year}}

Note - The curly brackets are very important, else the test case is not executed.



Step 6 - Create Another Output Log Message Step

Create another test step similar to the above step,

1. Create another **Output Log Message** test step and name it **DSstep2**.

To do this repeat [Step5](#).

2. In the Log Information area, enter the following log message:

Date is : {{month}}/{{day}}/{{year}}




3. Save the test case by clicking the Save icon **Save** .

Step 7 - Execute the Test

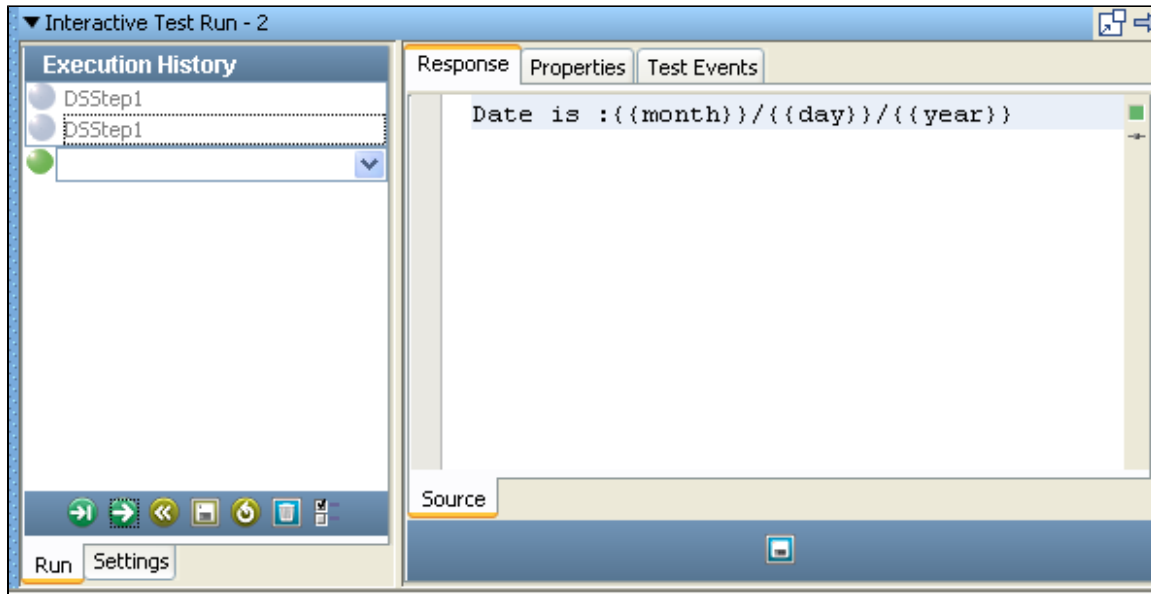
In this step, run the **Interactive Test Run** (ITR) to execute the test at run time and see if you get any responses.

To run the test in the ITR,

1. From the toolbar, click **Start ITR**

2. In the Execution History pane that opens up, click **Execute Next Step** .

The properties in the **Response** tab only show the **property name and the empty properties {}**.




This indicates that LISA was unable to find a value for the properties month, day, and year.

This is an expected result since you have not set the properties yet.

Step 8 - Add the Data Set

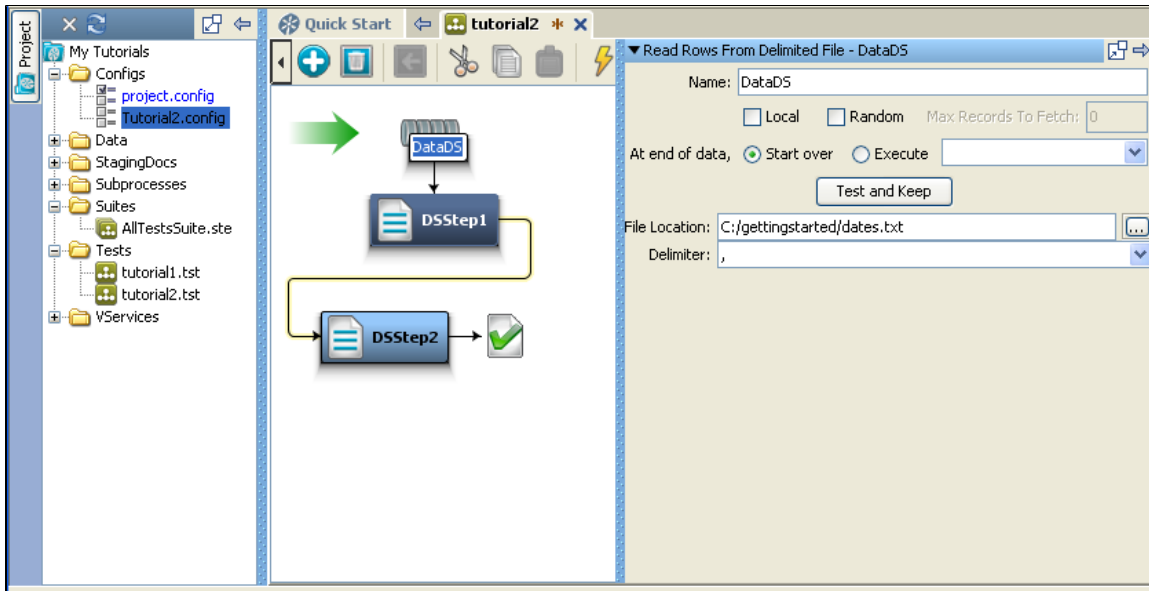
In this step, add the data set **dates.txt** that was created in Step1 to the test step.


To add a data set to the test case,

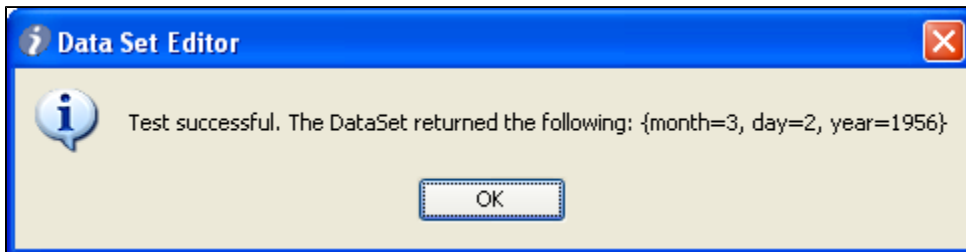
1. Select **DSstep1** in the model editor.
2. Click on the step element tab **Data Sets**.
3. Click on the  icon below the Data Sets element.
4. From the **Common Datasets** list, select **Read rows from a delimited data file**.

The Data set gets added to the test step. Double click on the Data Set to open the data set editor.

5. In the Data Set Editor, enter **DataDS** for the name of the data set.



6. Click on the **File Location** browse button  to navigate to and select the **dates.txt** file.
7. Accept the rest of the default settings.
8. Click the **Test and Keep** button.
9. If the test is successful, the following message window opens:



10. Click **OK**.
11. In ITR, click **Restart test** button.
12. Execute the test. The first row of data in the data set is displayed in the **Response** tab.

Hint Both step responses display the same date because we read only from the data set in **DSstep1**.

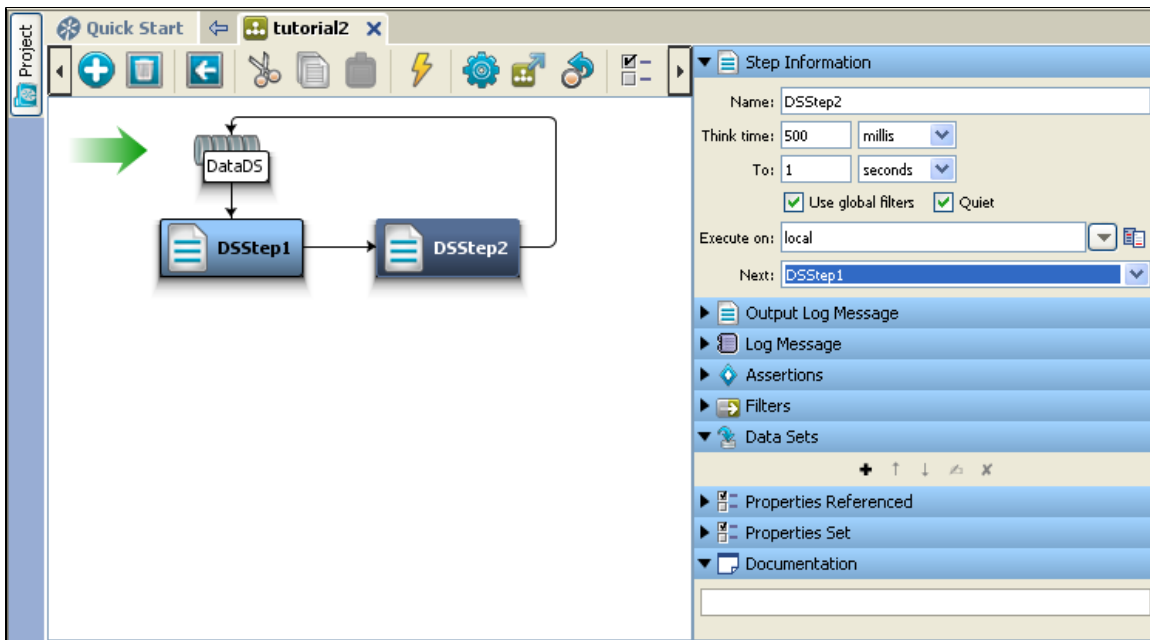
Step 9 - Change Data Set Behavior


Modify how the data set behaves to loop through the test step, until all the data rows in the data set are used.

To use the data set to loop through the test step,

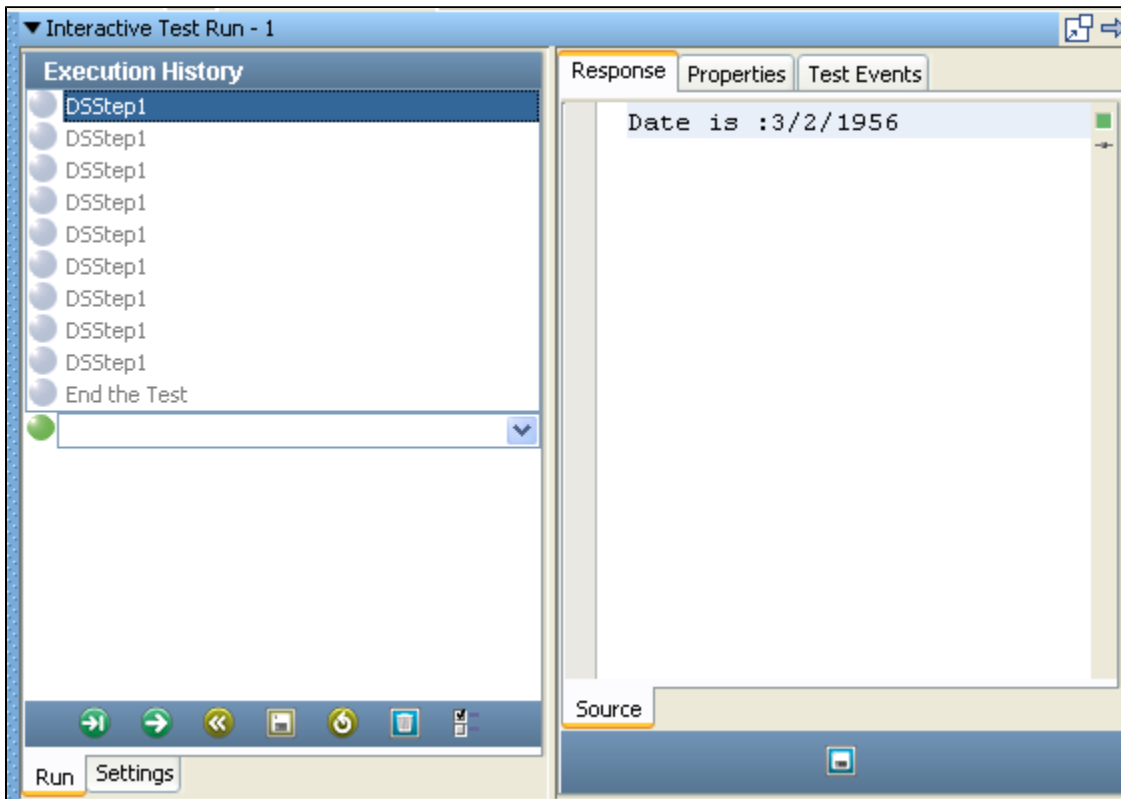
1. In the step element pane of **DSstep1**, click on the arrow next to **DatesDS** under the Data Sets element.
2. In the **At end of data** field, select **Execute** option.
3. Click the drop-down arrow on the **Execute** field and select **End the Test** from the list of choices that appear. This will make the test to end when all the data rows have been consumed.
4. Click **Test and Keep**.
5. Click **OK** to close the test success message.
6. In the model editor, select the **DSstep2**.
7. Click and open the **Step Information** element tab (see A).
8. In the **Next** list, select **DSstep1** to cause the two test steps to loop (see B).

Note that visually the model editor shows the arrow going to data set **DatesDS** and not the test step **DSstep1**, which shows the order of execution that will happen (The Data set will be executed before the test step).



9. In the ITR, click **Restart test** .
10. Execute the test.

Notice that the test runs in a loop until there are no more data rows in the data set and you can see both the steps executed in a loop.



Step 10 - Save the Test Case

Review

In this tutorial, you did the following:

- Created a Data Set and used it as data for running a simple test case.
- Created a comma-delimited Data Set.
- Added a Data Set to a test case.

- Learned how test step accesses the data in the Data Set.

4.3 Tutorial 3 - Basic Concepts - Filters and Assertions

4.3 Tutorial 3 - Basic Concepts - Filters and Assertions

In this tutorial, you will modify the test case created in Tutorial 2 to include a simple Assertion and a Filter.

To know about Filters and Assertions, see [LISA Basics](#).

LISA Concepts Discussed

In this tutorial, do the following:

- Open an existing Test Case.
- Add a Filter to the test step.
- Add an Assertion to the test step.

Prerequisites

Complete Tutorial 2. Open LISA Workstation if not already open.

Steps

Step 1 - Renaming a Test Case

In this step, you will open **tutorial2.tst** and save it as **tutorial3.tst**.

1. Open the **tutorial2.tst** test case in **My tutorials** project.
2. Double click on the **tutorial2.tst** in **Tests** subfolder.
3. From the menu bar, select **File > Save As**.
4. In the **File name** field, enter **tutorial3** and click **Save**.

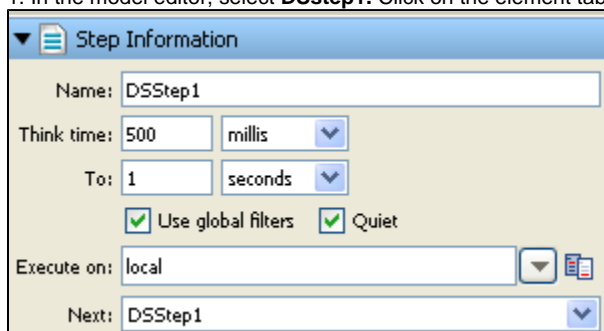
A new **tutorial 3** is created and saved under Project **My Tutorials**.

Step 2 - Change Action of Test Step

Change the "Next Steps" action of both test steps so that **DSStep1** is the next step, and only the first step reads from the Data Set.

To modify the existing test case,

1. In the model editor, select **DSStep1**. Click on the element tab Step Information, and change the **Next** step to **DSStep1**.



Step Information

Name: DSStep1

Think time: 500 millis

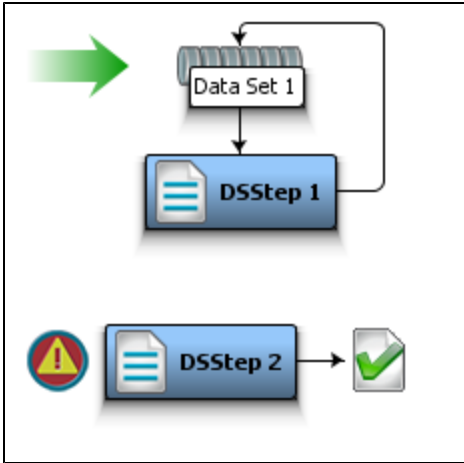
To: 1 seconds

☒ Use global filters ☒ Quiet

Execute on: local

Next: DSStep1

With this action we will go back to the same step **DSStep1** as shown below:



2. In the model editor, select **DSstep2** and change the Log Information as follows:

Date contains 1999. It is: {{month}}/{{day}}/{{year}}.

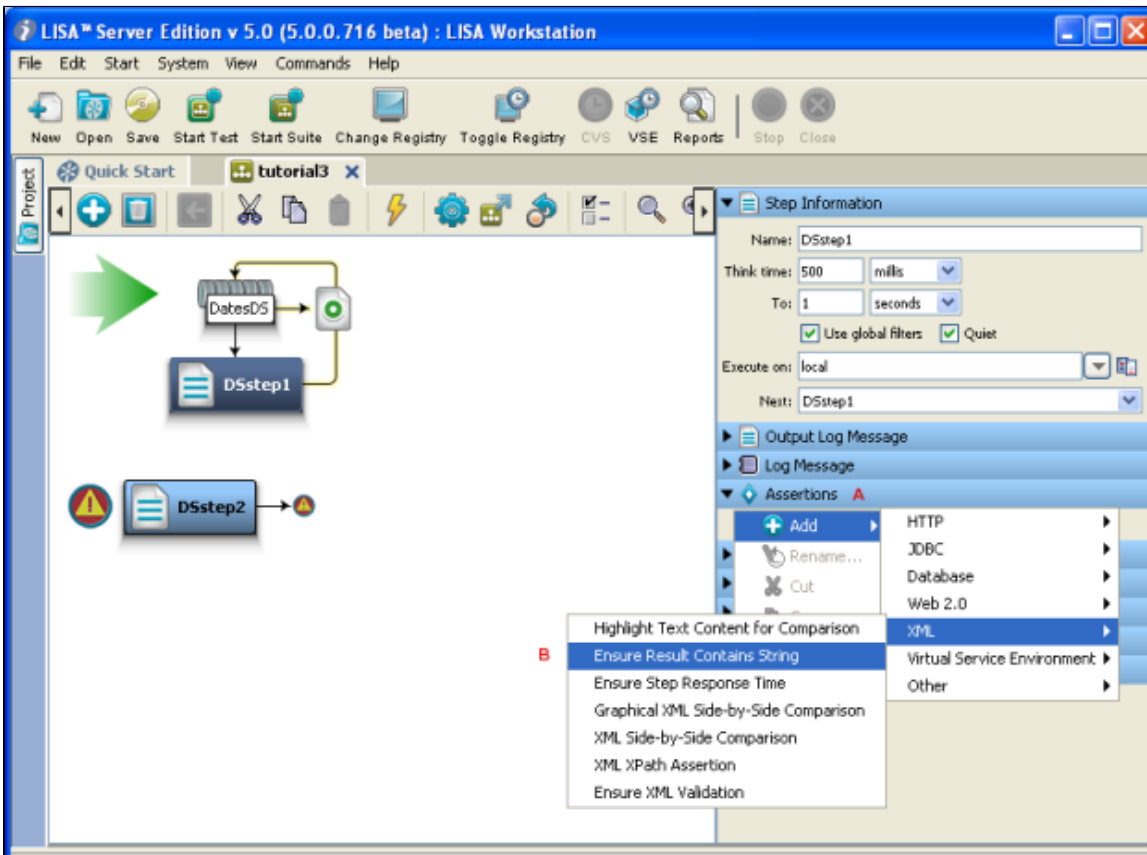
Note - The curly brackets are very important, else the test case is not executed with correct values.

Step 3 - Add an Assertion

Add an assertion to test whether the date contains the year **1999**.

To add an Assertion to DSStep1,

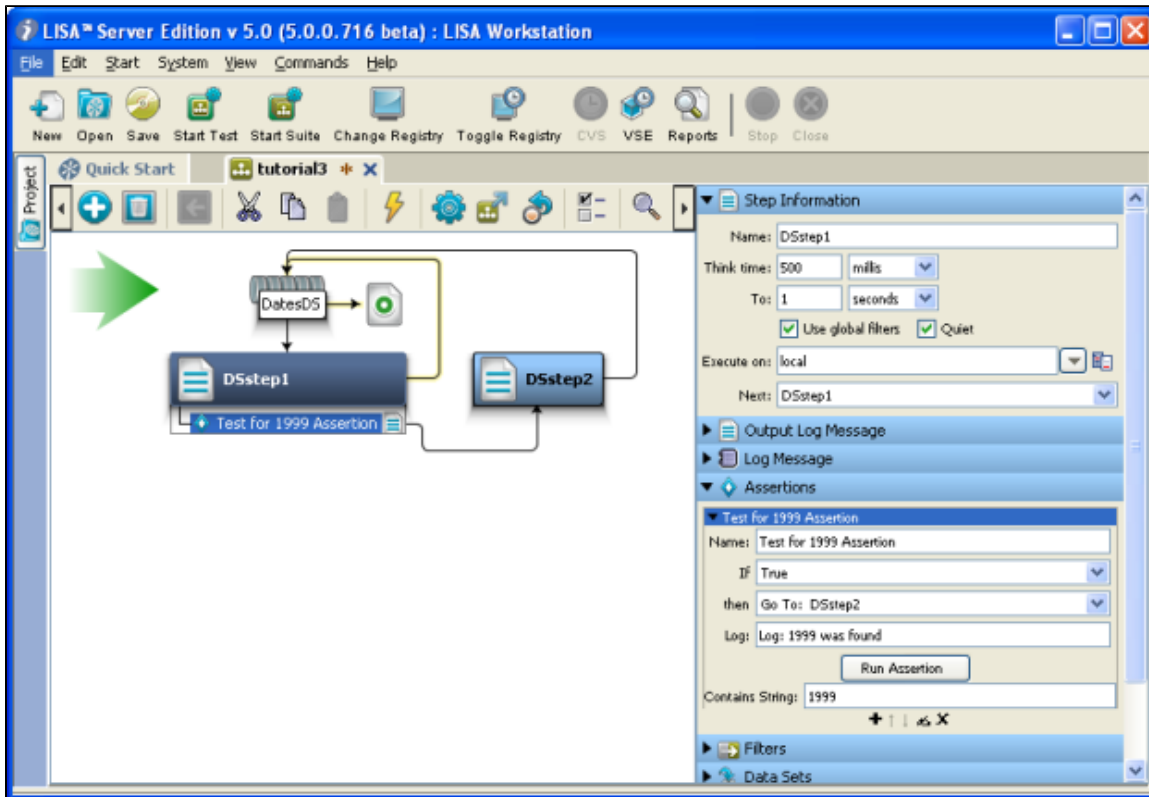
1. Select **DSstep1** in the model editor, and click on the **Assertions** element tab (see A).
2. Click on **+** icon, and from **Assertion/XML** submenu, select **Ensure Result Contains String** (see B).



The new assertion applied to DSStep1 is added to the **Assertions** tab, and the assertion editor opens.

3. In the Assertion Editor,

- In the **Name** field, enter **Test for 1999 Assertion**.
- In the **IF** list, select **True**.
- In the **then** list, select **Go To: DSstep2**.
- In the **Log** field, enter **Log: 1999 was found**.
- In the **Contains String** field, enter **1999**.



The assertion tests for the string **1999** in the response.

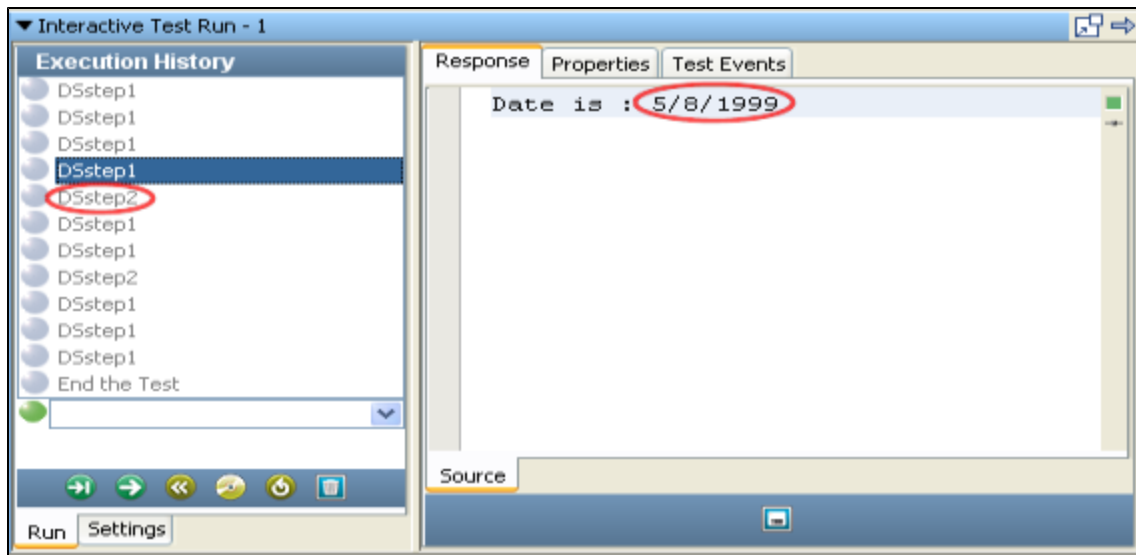
- If the string (1999) is present, the assertion redirects to the **DSstep2** step.
- If the string (1999) is not present, the next step, **DSstep1**, is executed.

Step 4 - Test the Assertion

To test the assertion in ITR,

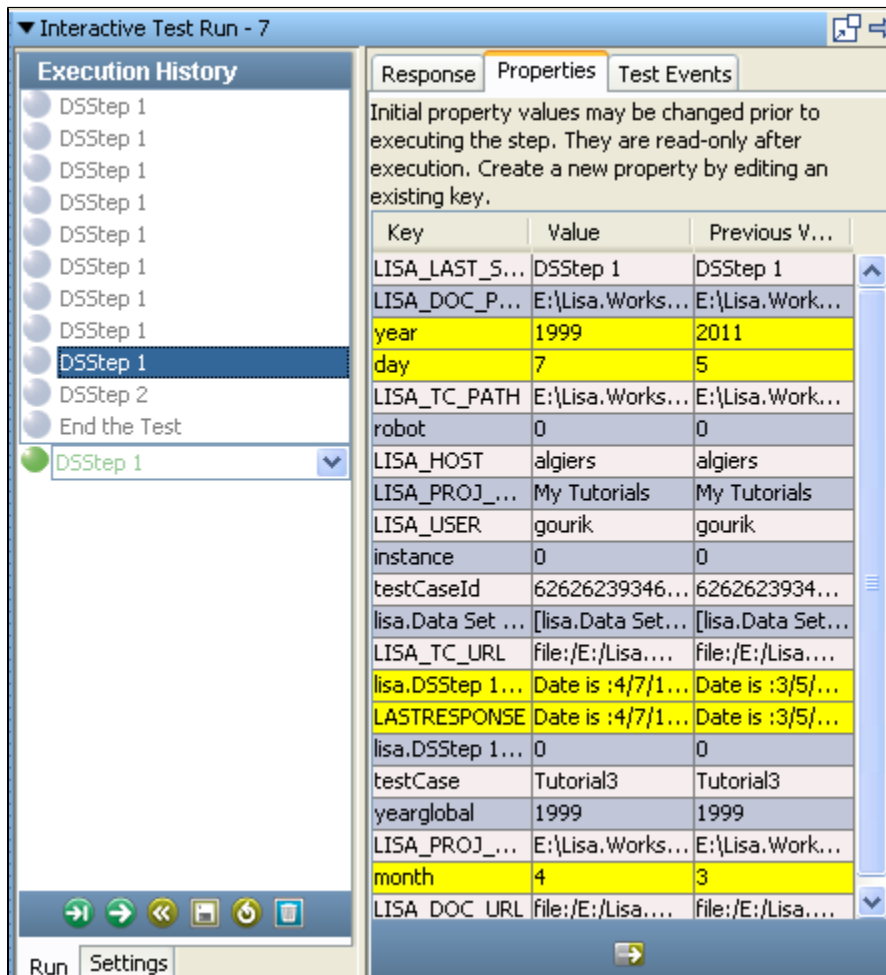
1. Run the ITR until the end step is reached.

When the date in the **Response** tab contains **1999**, the **DSstep2** is executed.

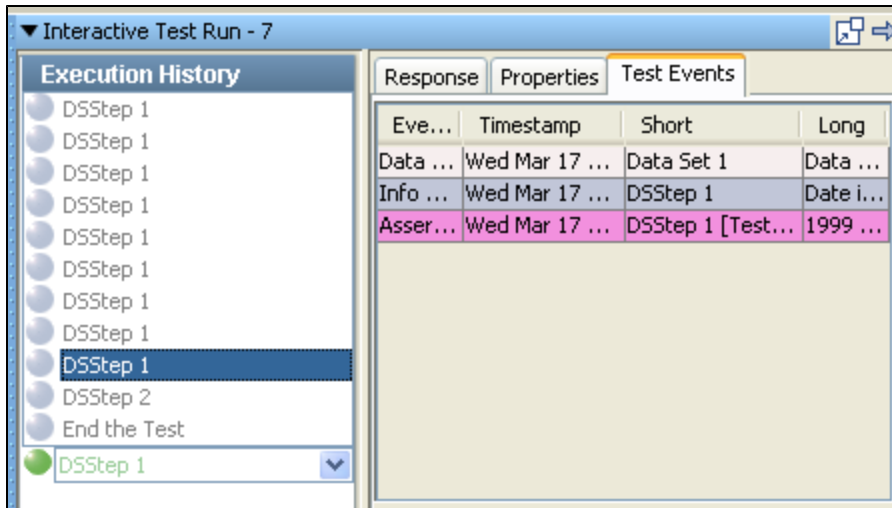


2. Open the **Properties** and **Test Events** tabs for the DSstep1 steps that are executed immediately before a DSstep2 step to see what was reported.

Properties Tab -



Test Events Tab -

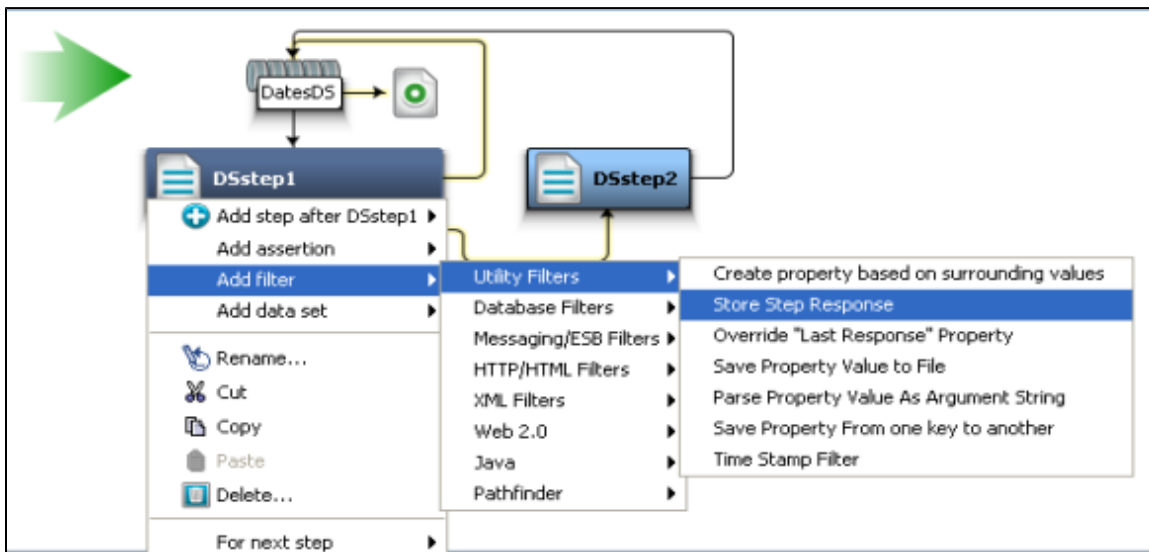


Step 5 - Add a Filter

Add a Filter to DSstep1 step to capture the response in a property and display the value of this property in DSstep2.

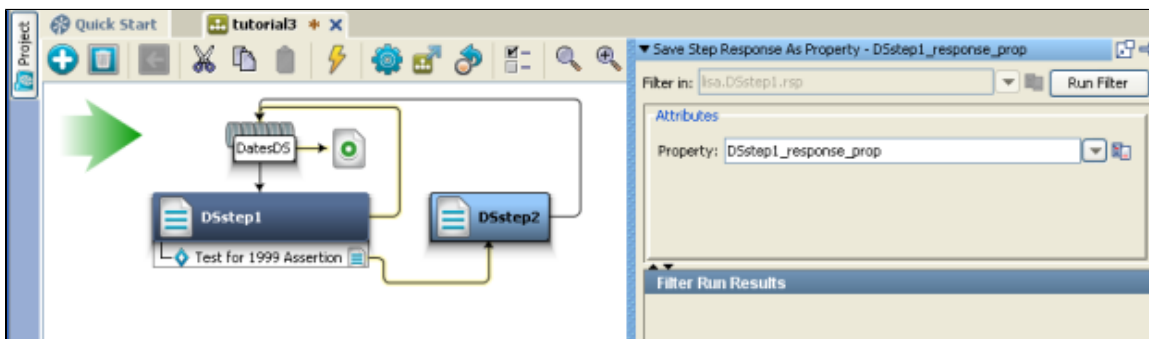
To add a Filter,

1. In the model editor, right click on **DSstep1** and select **Add Filter** or select **DSstep1** in the model editor and click on the **Filters** tab in the step element pane.
2. From the **Utility Filters** list, select **Store Step Response**.



The new filter is added to the Filters tab.

3. In the filter editor, enter the name of the property to store the response to **DSstep1_response_prop**.



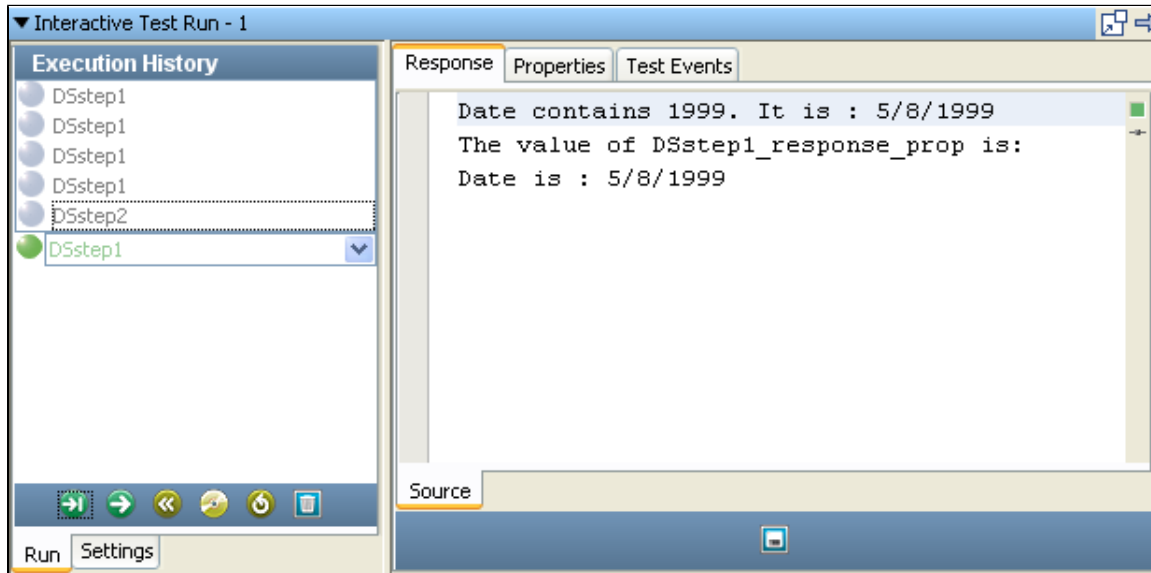
4. In the model editor, select **DSstep2**, and add the following to the end of the log message:

The value of DSstep1_response_prop is: {{ {{ {{ }} DSstep1_response_prop }}

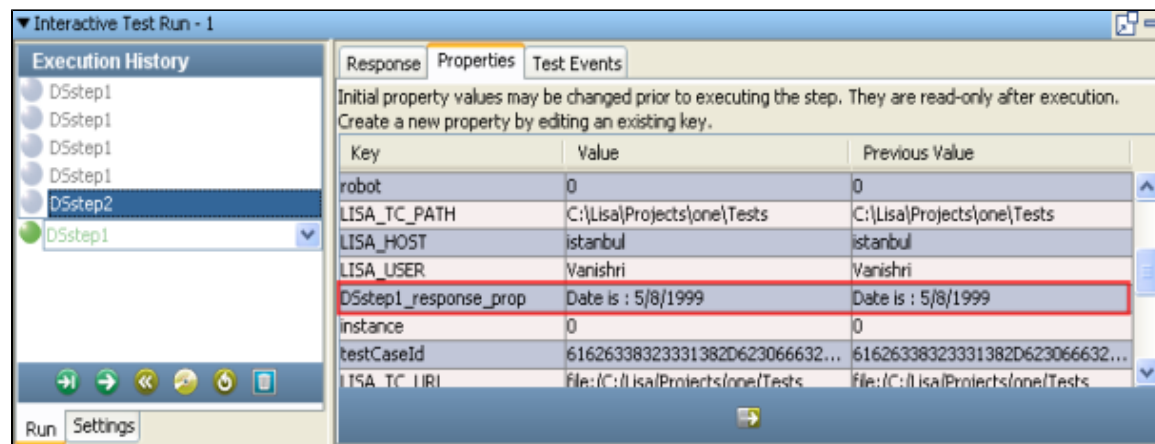
Step 6 - Test the Filter

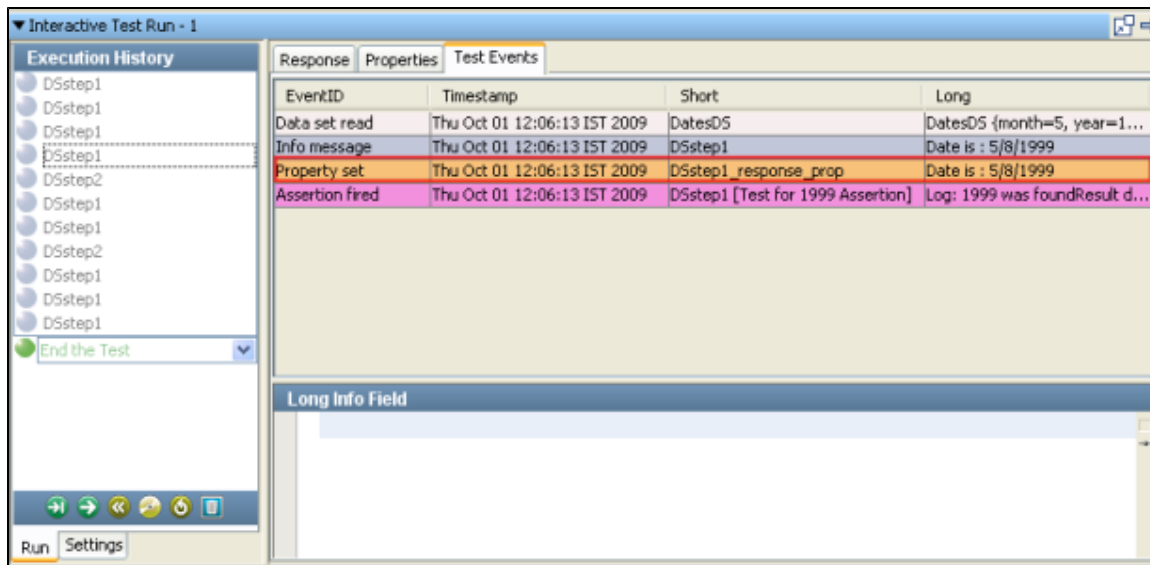
To test the Filter,

1. Run the ITR until the end step is reached.
2. The response from DSstep1 is displayed in the log for DSstep2.



3. Open the **Properties** and **Test Events** tabs, and you see the property **DSstep1_response_prop**.





Step 7 - Save the Test Case

Click Save to save the test case.

Review

In this tutorial, you did the following:

- Took a first look at LISA filters and assertions.
- Opened and modified an existing test case.
- Learned how to add a simple filter.
- Learned how to add a simple assertion.
- Used the Interactive Test Run utility to check if the assertion worked as planned.

More Information

LISA provides filters and assertions to cover most of the situations you will come across in your test case development. If there is not an appropriate filter, LISA provides a mechanism, through the LISA Extension Kit, to allow custom filters and assertions to be developed see the LISA Developers Guide for more information.

4.4 Tutorial 4 - Manipulating Java Objects (POJOs)

4.4 Tutorial 4 – Manipulating Java Objects (POJOs)

In this tutorial, you will **create and manipulate a simple Java object** and;use the **Java Date** class to create a date object.

The **Dynamic Java Execution** step allows you to create a Java object from a class in the LISA classpath.

First, construct the object and look at how to call methods on the object and then incorporate the object into a simple LISA model editor.

LISA Concepts Discussed

In this tutorial, do the following:

- Use the Dynamic Java Execution step.
- Use the Complex Object Editor for simple objects.
- Use inline filters and save the results into a property.

Prerequisites

Complete Tutorial 3. Open LISA Workstation if not already open.

Steps

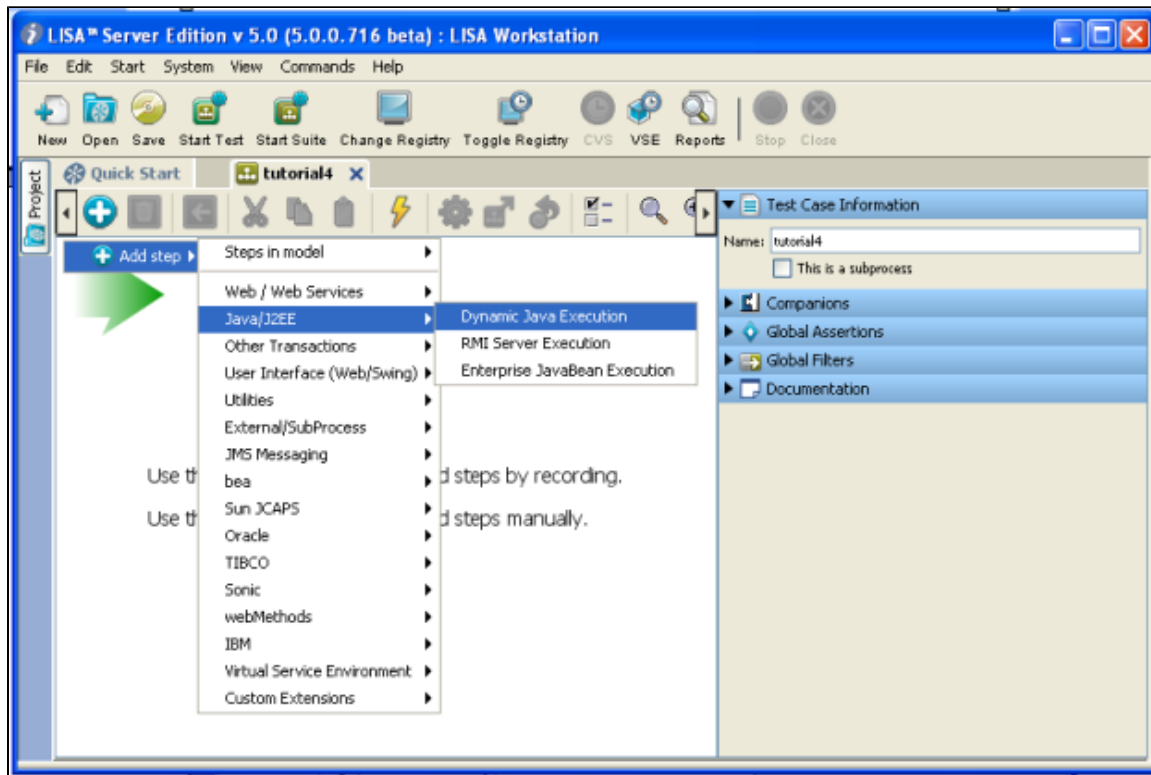
Step 1 - Create a New Test Case

Create a new test case called **tutorial4**, within the Project **My Tutorials**.

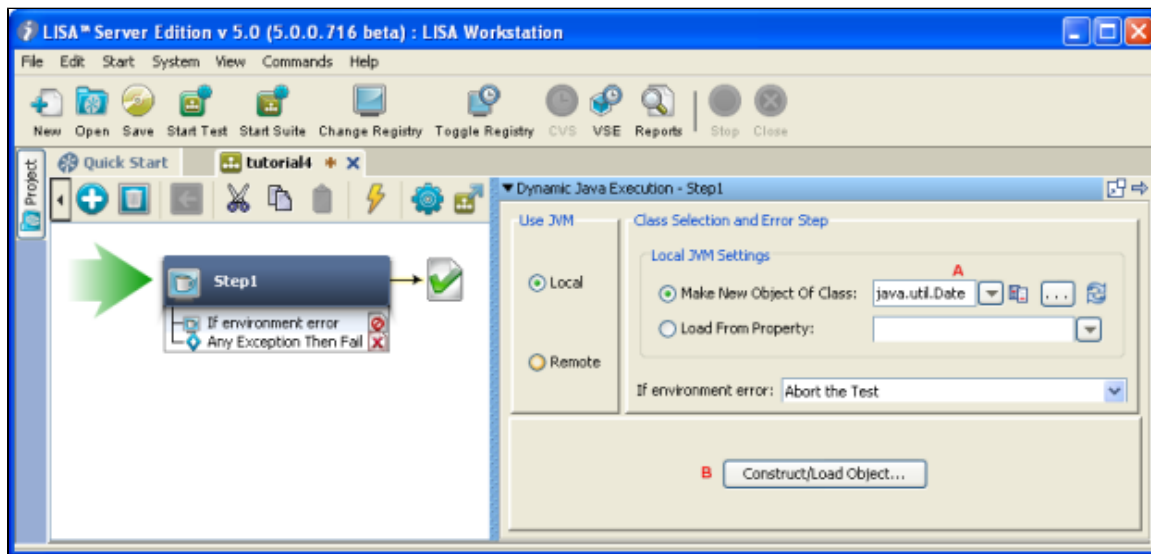
Step 2 - Create a New Test Step

To create a new test step,

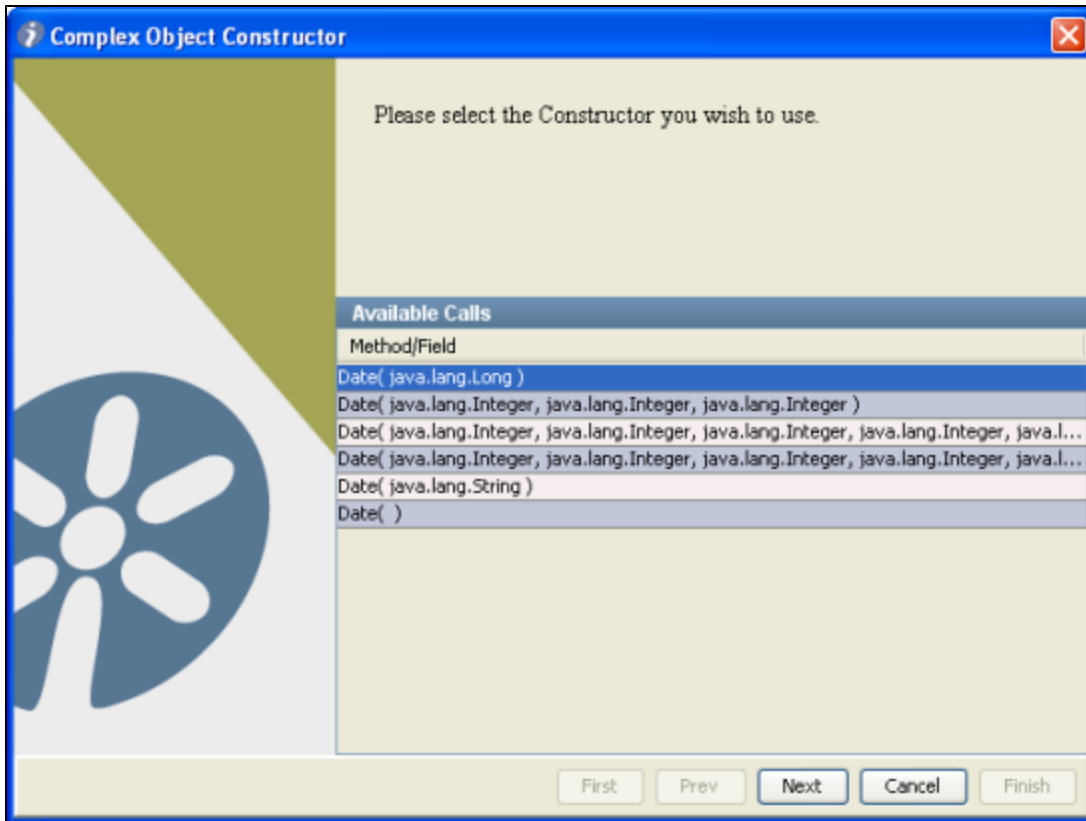
1. Right click in the model editor, and click **Add Steps** or
From the test case toolbar you can click the **Add Steps** button.
The **Add Steps** menu is displayed.
2. Click **Java/J2EE**, and select **Dynamic Java Execution** step type.



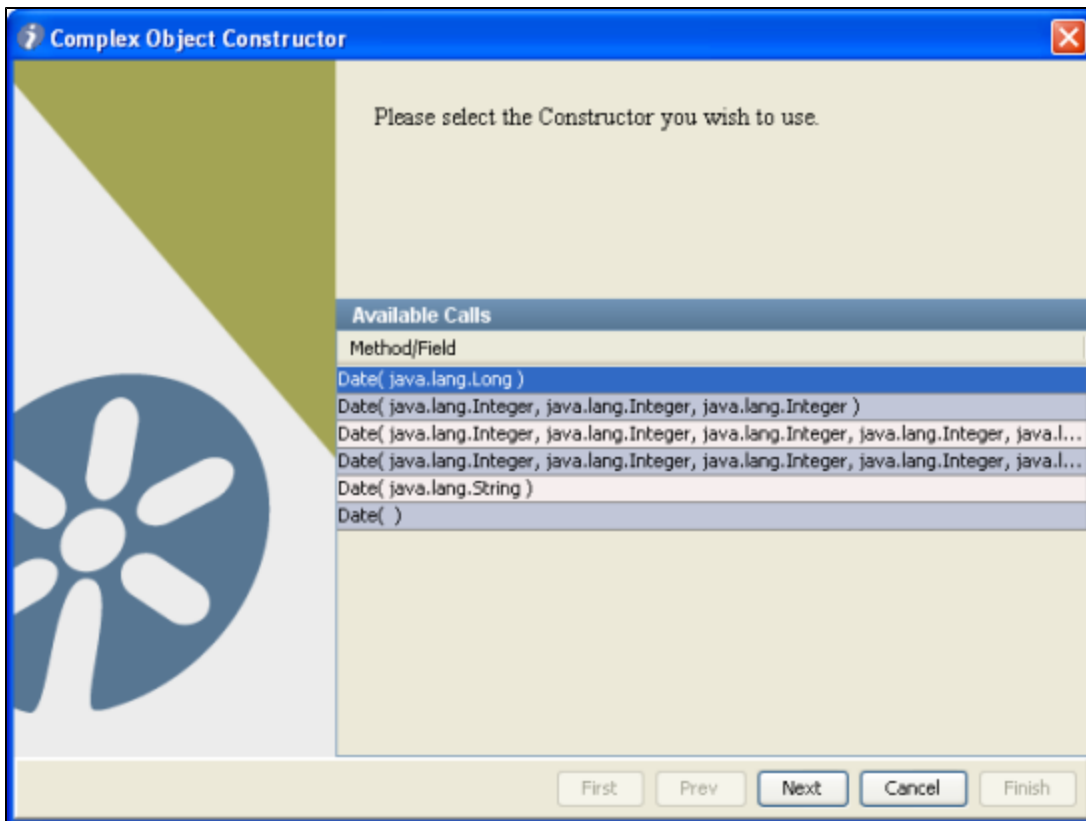
The Step editor opens.



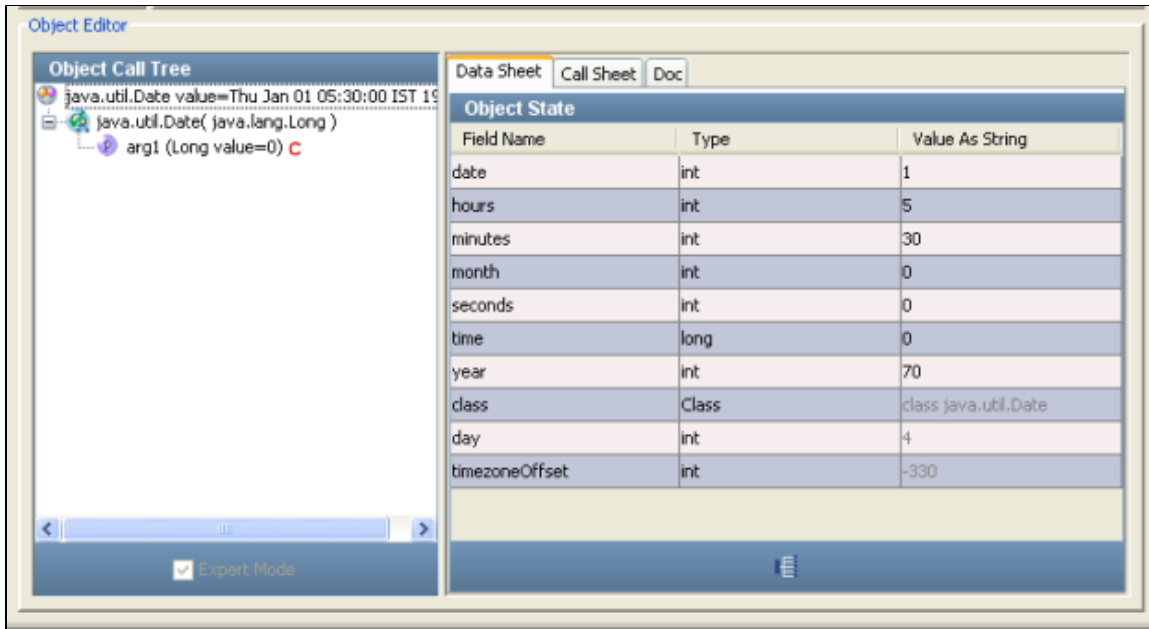
3. In the "Class Selection and Error Step" area, click "**Make New Object of Class**" and select the package name of the Date class, **java.util.Date** (see A above).
4. Click **Construct/Load Object** (see B) to open the **Complex Object Constructor** wizard.








5. From the available methods list, select the **Date** constructor.



6. Click **Next** and **Finish** to open the **Complex Object Editor**.



The **Object Call Tree** displays the full context of the method calls. It uses some icons for denotation which include:

-  - The object that provided the original method call (see A)
-  - The method you wish to invoke
-  - The parameters of this method (see c).
-  - The return value of this method
-  - The method complete, when you have called the method.

Now you have a Java object to manipulate it in the Complex Object Editor.


Note - Please note that you do not have to write any Java code here.

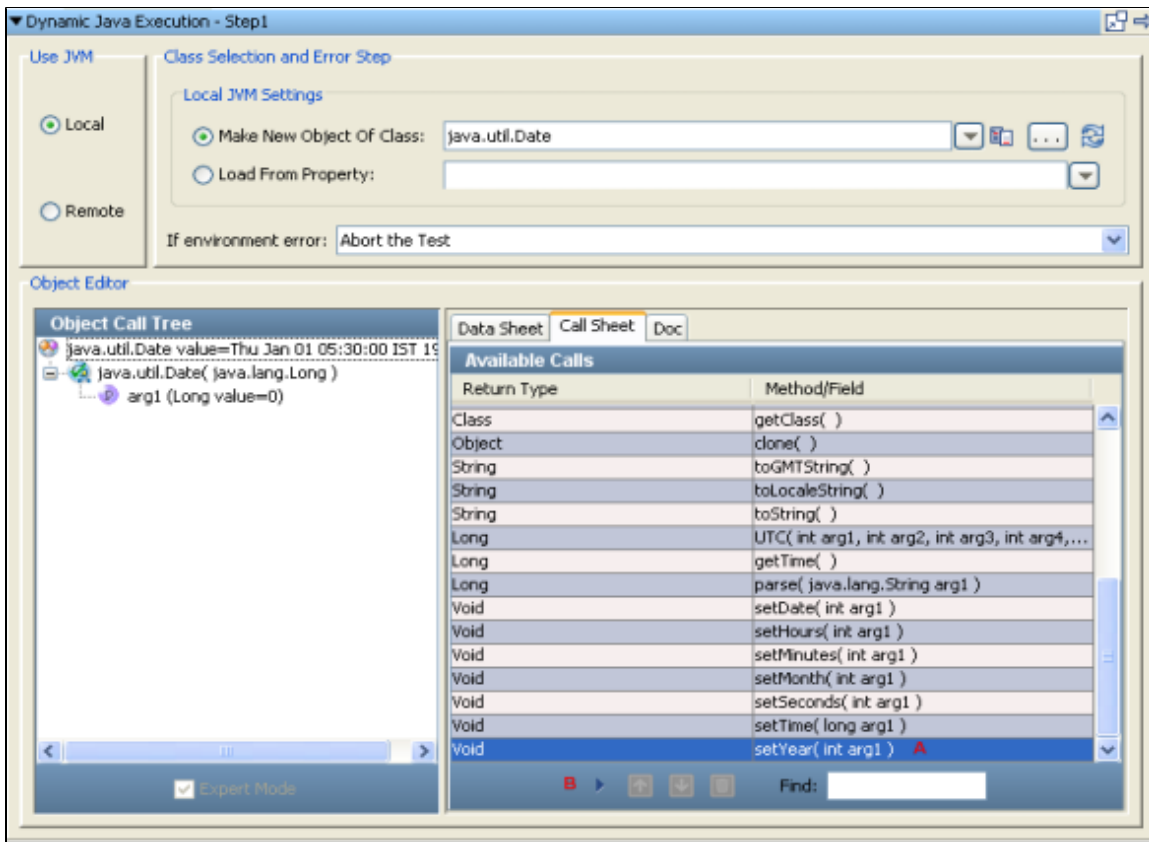
Step 3 - Make a Call on Object

To make a simple call on the Date object,

1. In the **Object Call Tree** list, click the **Date** object, and click the **Call Sheet** tab.

You can execute any of the methods (listed in the **Method/Field** column) exposed by the object.

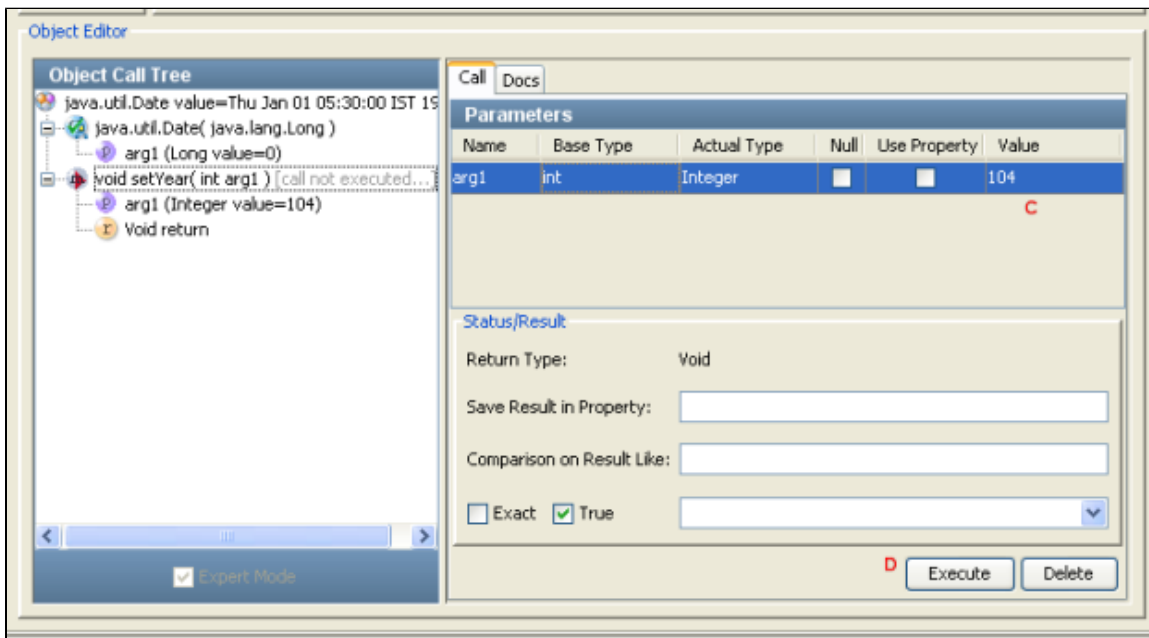
2. Double click the **setYear()** method (see A), or select the method and click the **Invoke Method** icon  (see B) to invoke the method.



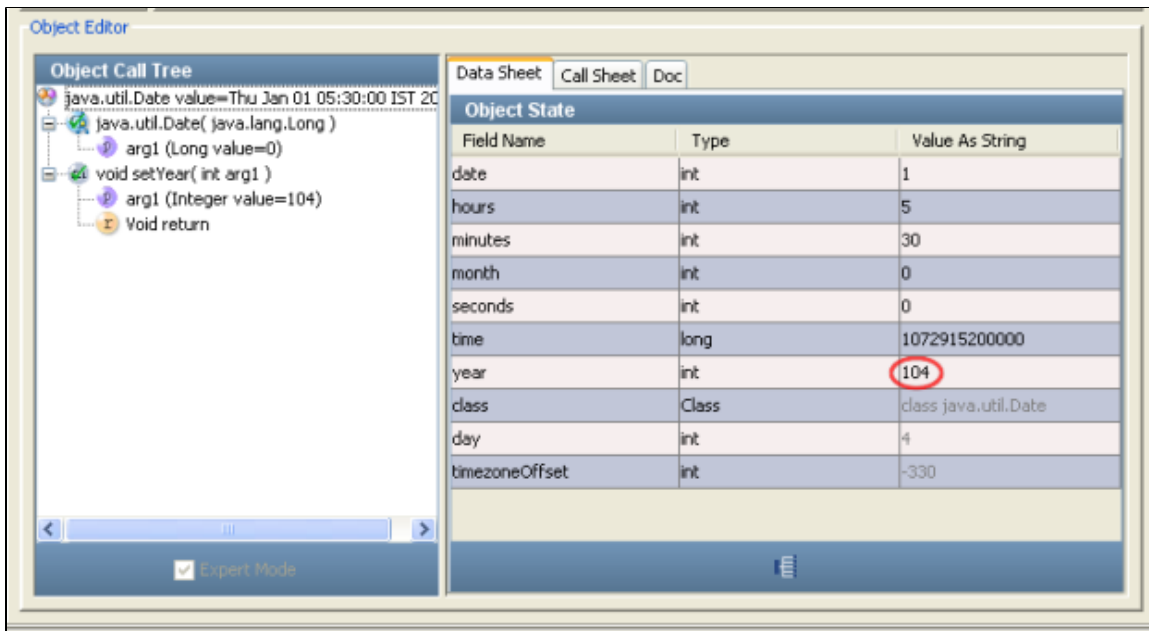
Method invocation will show the method invoked and open the Call Tab and Doc tab as shown below:

3. The **Call** tab lists the argument information. In the **Value** field for **arg1**, enter **104** (see C below).

4. Click **Execute** (see D below).



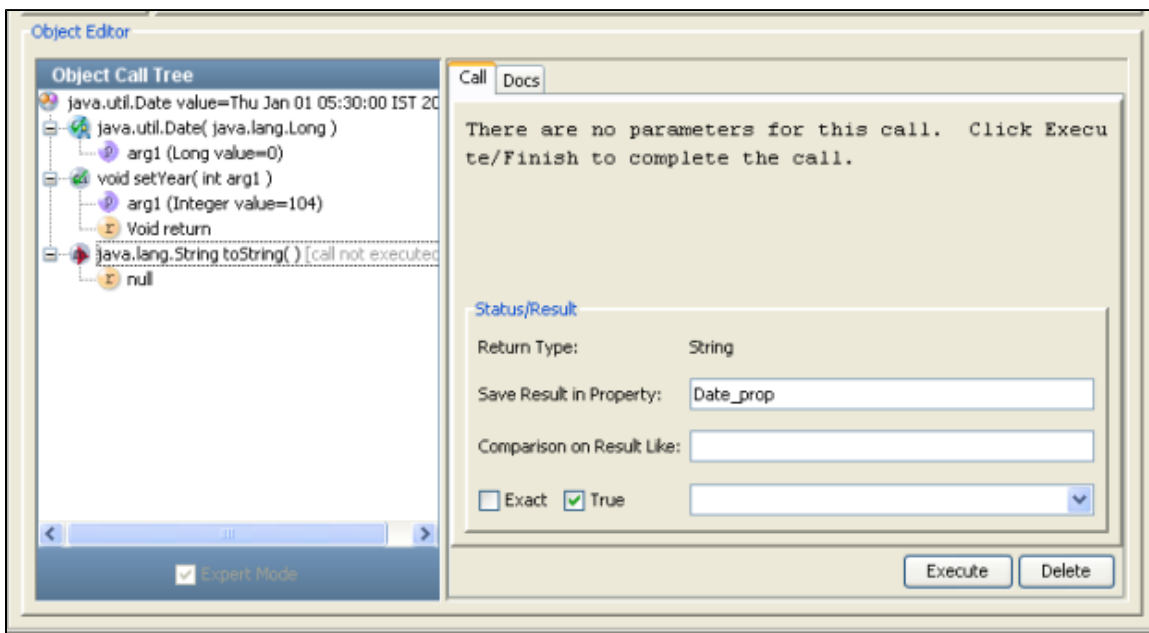
5. Verify that the date returned changed, by returning to the **Date** object **Data Sheet** tab, as shown below:



Step 4 - Add an Inline Filter

To add an Inline Filter,

1. Click the **Date** object in the **Call Sheet** tab.
2. Invoke the **toString()** method to retrieve the date to be placed in a property.
3. In the **Status/Result** window, in the **Save Result in Property** field, add an inline filter by entering **Date_prop** as the property name.



4. Click **Execute**.

Step 5 - Verify the Property Created

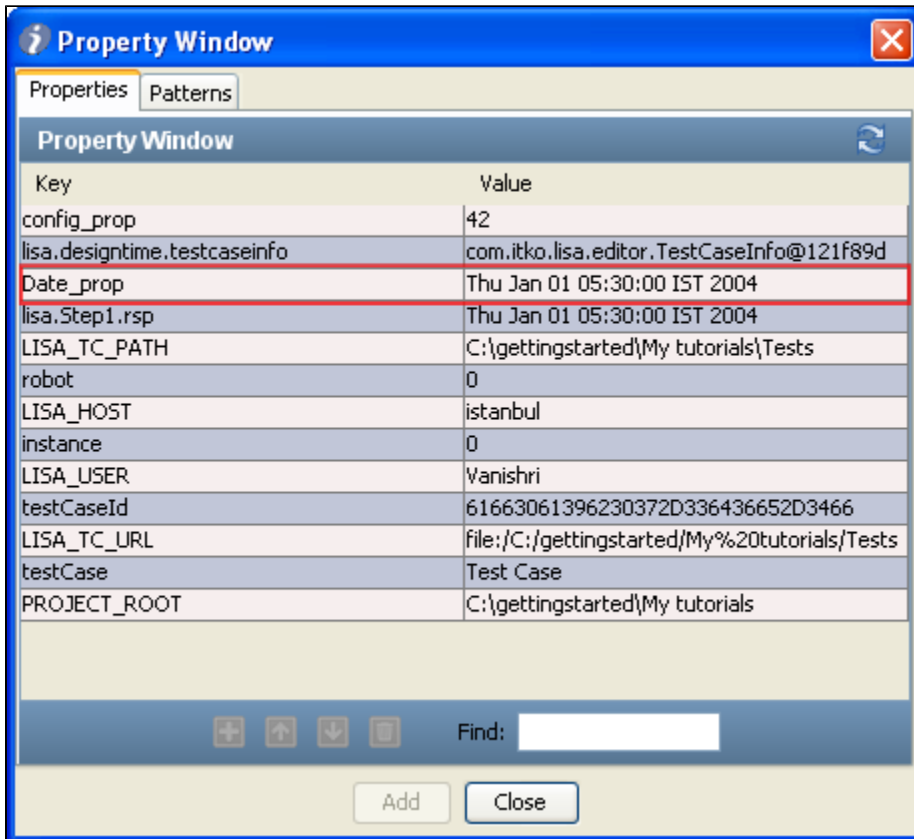
To verify that the **Date_prop** was created,



1. Click **Show Model Properties** button on the test case tool bar:

Or Click **Help** main menu and open the **Property** window.

2. Find the **Date_prop** property in the property window.



Step 6 - Save the Test Case

Review

In this tutorial, you did the following:

- Created a test step to manipulate a Java object of **Date** type.
- Inserted the test step into a LISA model editor driven by a data set containing test dates.
- Used the Complex Object Editor to manipulate Java objects.
- Learned how to add inline filters to objects and save results into a property.

4.5 Tutorial 5 - Running a Demo Server Web Application

4.5 Tutorial 5 – Running a Demo Server Web Application

In this tutorial, you will step through a simple Web application that accompanies LISA.

The LISA bank application is a simple front-end that is connected to a database table containing financial account information. The application business logic consists of Enterprise **JavaBeans** and **Web Services**. From the Web site, you can view the profile of the user, create/close/delete an account, and add/delete addresses etc.

The goal of this tutorial is to become familiar with the application. This example application is used in all subsequent tutorials as the system under test.

LISA Concepts Discussed

No new concepts are introduced.

Prerequisites

Read [LISA Basic](#) concepts.

The LISA demo server should be running.

Steps

Step 1 - Launching the Web Application

To launch the LISA bank Web application,

1. Open a browser window and enter the following URL to display the home page.

- <http://localhost:8080/lisabank/> (local Demo Server)
(Replace **localhost** in the above path, by your machine **IP** address)

Note - To open the LISA Financial Online using local Demo Server, keep the local Demo Server running.

The LISA Financial Online main page opens:

Monday, Oct 25, 2010

[MyMoney?](#) Login

Name

Password

Login

Tested by
iTKO LISA

[Help](#) | [Log In](#)

LISAfinancial Online

[Home](#) | [Location Finder](#) | [Contact Us](#) | [Printable](#)

Welcome!

Sound financial footing
for the
things that
matter

New at LISAfinancial

[New Flexible-Interest Loan Plans](#)
[Just added to MyMoney?: All-in-One Planning Dashboards](#)
[Q1 2007 Financial Results for LISAfinancial](#)

Market Rates

Mortgage	Rate	APR	Points
1-Yr ARM	5.75%	7.02%	0.00%
3-Yr ARM	5.75%	6.79%	0.00%
5-Yr ARM	5.75%	6.60%	0.00%
30-Yr ARM	6.62%	6.65%	0.00%

Step 2 - Logging in to LISA Financial

To log into LISA financial Online site,

1. In the **Name** field, enter the user name **lisa_simpson**.
2. In the **Password** field, enter **golisa**.

The **Welcome lisa** page opens as below:



The **MyMoney Home** page lists MyMoney tools.
On the left-hand side, all the actions a user can take are listed:

1. View Profile
2. New Account
3. Close Account
4. Add Address
5. Delete Address
6. Log Out

Notice that there are no accounts established for user **lisa_simpson**.

Step 3 - Adding an Account

To create a new account for a user,

1. Click **New Account**.



2. From the **Account Type** list, select **SAVINGS**.
3. In the **Account Name** field, enter **MySavings**
4. In the **Initial Balance** field, enter **100.00**.
5. Click **Add Account**.

New Account Details	
Username	lisa_simpson
Account Type	SAVINGS
* Account Name Account Name	MySavings
* Initial Balance Initial Balance	100.00
<input type="button" value="Add Account"/>	

The new savings account is added under the **Accounts** section as shown below:

[Help](#) | [Log Out](#)

LISAfinancial Online

[Home](#) | [Location Finder](#) | [Contact Us](#)

Welcome lisa simpson (lisa_simpson)

Monday, Oct 25, 2010

Welcome lisa !

Tested by



MyMoney Home >

Accounts

This is to test the modified jsp file.

Account Number	Account Type	Name	Current Balance	Available Balance
25839954210	SAVINGS	MySavings	\$100.00	\$100.00
Total			\$100.00	\$100.00

MyMoney Tools

- [Mortgage Planner](#)
- [Retirement Planner](#)
- [Online Transactions](#)
- [Fund Allocation](#)
- [Identity Security](#)

Fund Statistics



Quarter	East	West	North
1st Qtr	20	30	45
2nd Qtr	25	35	48
3rd Qtr	30	40	50
4th Qtr	22	32	46

6. Repeat this step to create additional **Checking** and **Auto_Loan** accounts.
Do not use commas in the **Initial Balance** field.

[Help](#) | [Log Out](#)

LISAfinancial Online

[Home](#) | [Location Finder](#) | [Contact Us](#)

Welcome lisa simpson (lisa_simpson)

Monday, Oct 25, 2010

Welcome lisa !

View Profile
New Account
Close Account

Add Address
Delete Address

Log Out

Tested by

MyMoney Home >

Accounts

This is to test the modified jsp file.

Account Number	Account Type	Name	Current Balance	Available Balance
25839954210	SAVINGS	MySavings	\$100.00	\$100.00
25961335821	CHECKING	Checking	\$100.00	\$100.00
25999730929	AUTO_LOAN	AutoLoan	\$100.00	\$100.00
Total			\$300.00	\$300.00

MyMoney Tools

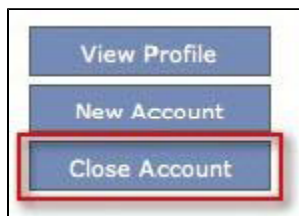
- [Mortgage Planner](#)
- [Retirement Planner](#)
- [Online Transactions](#)
- [Fund Allocation](#)
- [Identity Security](#)

Fund Statistics

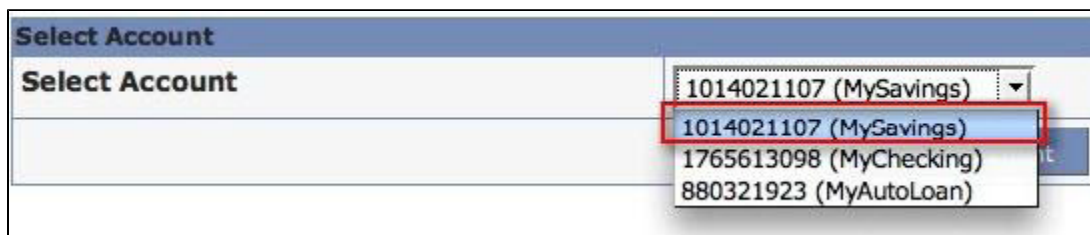
Step 4 - Deleting an Account

To delete or close an account,

1. Click **Close Account**.



2. From the **Select Account** list, select **MySavings**.



Select Account

Select Account

25839954210 (MySavings) ▼

Select Account

Account Details

Username

lisa_simpson

Account ID

25839954210

Account Type

SAVINGS

Account Name

MySavings

Balance

100.0

Confirm Delete

3. Click **Confirm Delete**.

MySavings account is no longer displayed in the Accounts list as shown below:

Monday, Oct 25, 2010

Welcome lisa !

View Profile

New Account

Close Account

Add Address

Delete Address

Log Out

Tested by

ITKO LISA™

MyMoney Home >

Accounts

This is to test the modified jsp file.

Account Number	Account Type	Name	Current Balance	Available Balance
25961335821	CHECKING	Checking	\$100.00	\$100.00
25999730929	AUTO_LOAN	AutoLoan	\$100.00	\$100.00
Total			\$200.00	\$200.00

MyMoney Tools

Mortgage Planner

Retirement Planner

Online Transactions

Fund Allocation

Identity Security

Fund Statistics

Period	East	West	North
1st	20	30	40
2nd	25	35	45
3rd	30	40	50
4th	25	35	45

Step 5 - Logging Out

Log out of the LISA Financial Online site by clicking **Log Out**.

Review

In this tutorial, you did the following:

- Logged into LISA Financial Online.
- Created new accounts.
- Deleted an account.

4.6 Tutorial 6 - Testing a Web Site

4.6 Tutorial 6 - Testing a Web Site

In this tutorial, you will use the LISA Web recorder to record the path through a Web site and create test steps of HTTP/HTML Request for each HTTP request/response pair.

The HTTP/HTML Request step allows you to make requests against a Web server and receive results from within a test case. Test a simple Web site to ensure that the pages work as expected.

LISA Concepts Discussed

In this tutorial, do the following in parts:

- A) Use the LISA Web recorder to create a test case containing HTTP/HTML Request steps
- B) Edit and run the test case that the recorder produces
- C) Add an assertion to an HTTP/HTML Request step

Prerequisites

Complete Tutorial 5. Open LISA Workstation if not already open. Make sure you have access to the demo server (either the Local demo server, or the iTKO demo server).

Tutorial Parts

Part A - Record and Run the LISABank Test Case

Part B - Running the Test Case

Part C - Modifying HTTP/HTML Request Test Steps (optional)

Part A - Record and Run the LISABank Test Case

Part A - Record and Run the LISABank Test Case

Use the LISA Web recorder to create a test case containing HTTP/HTML Request steps.

Steps

Step 1 - Create a new test case

Create a new test case **tutorial6a** in the **Tests** subfolder under **My tutorials** project.

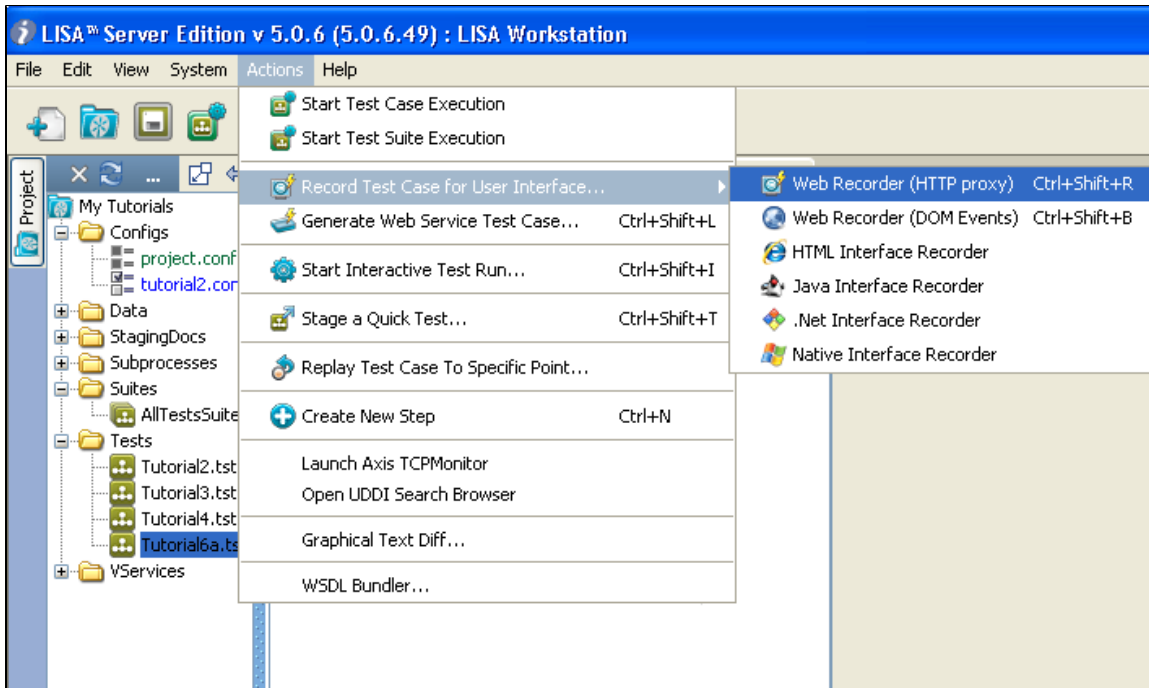
The model editor is opened.

Step 2 - Start the Web Recorder

To start the Web recorder,

1. Select **Record Test Case for User Interface... > Web Recorder (HTTP proxy)** from the Actions main menu.

Or click **Recorder**  icon and select **Record Test Case for User Interface... > Web Recorder (HTTP proxy)**.

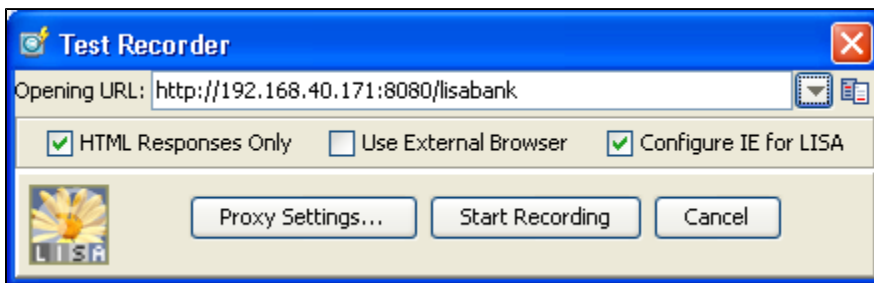


2. In the **Test Recorder** window, in the **Opening URL** field - enter the URL of the site to browse.

Here, we want to record transactions on the Demoserver example site for iTKO, so we enter the following address:

<http://localhost:8080/lisabank>.

Note - Replace **localhost** in this path by your machine **IP** address.



3. Click **Start Recording** button to open the LISA Browser window.

The browser loads the LISA Bank web page and you can start recording.

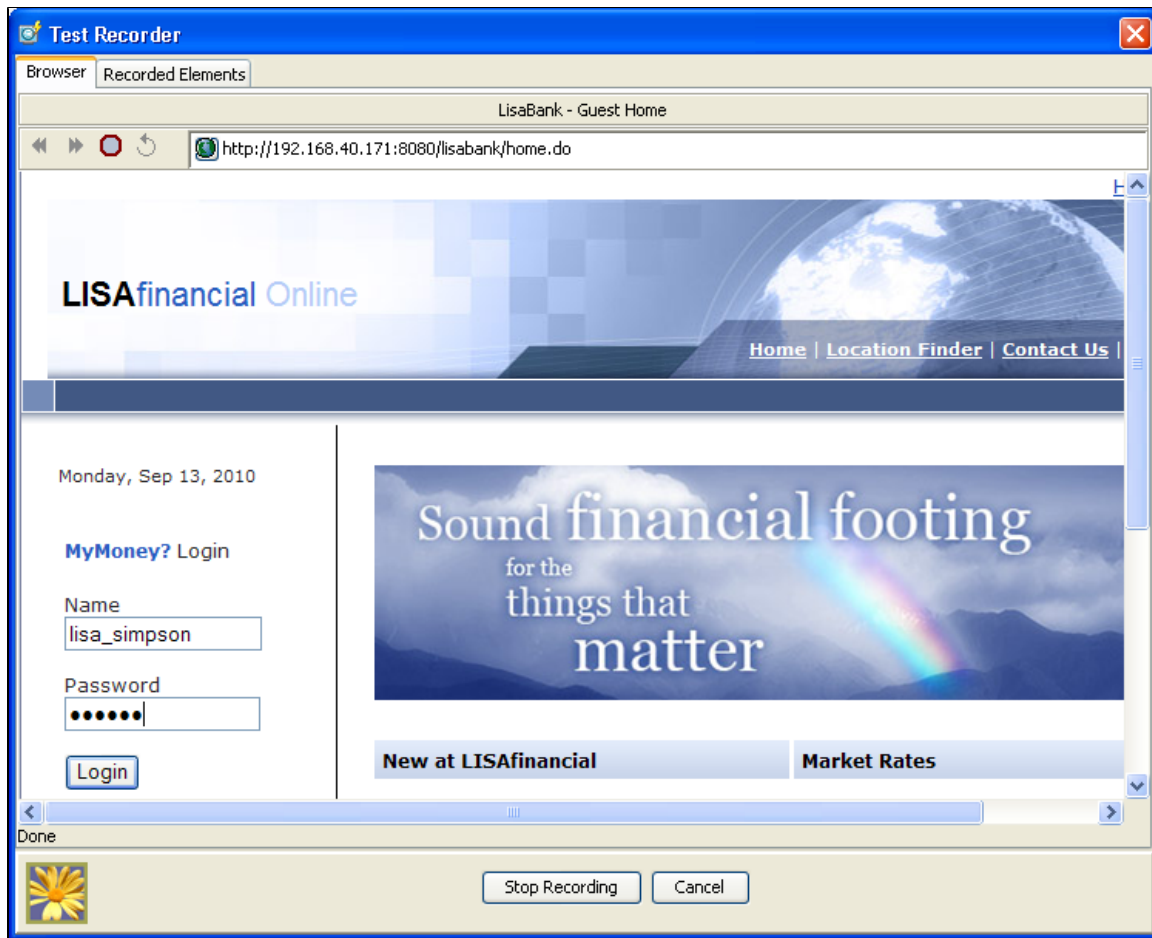
Step 3 - Record Pages on LISA Bank

While you visit several Web pages, LISA records the request and response information for each page visited.

To Login to LISA Bank,

1. Enter Name as **lisa_simpson** with the password **golisa**.
2. Click login button.

Once you login, you can do various transactions within LISA bank.



2. Click the **CHECKING Account** to see account activity.

Test Recorder

Browser
Recorded Elements

LisaBank - Home

http://192.168.40.171:8080/lisabank/login.do

Monday, Sep 13, 2010

Welcome lisa !

View Profile

New Account

Close Account

Add Address

Delete Address

Log Out

MyMoney Home >

Accounts

This is to test the modified jsp file.

Account Number	Account Type	Name	Current Balance	Available Balance
339369443372	CHECKING	Gauri	\$10000.00	\$1000
Total			\$10000.00	\$1000

MyMoney Tools

Fund Statistics

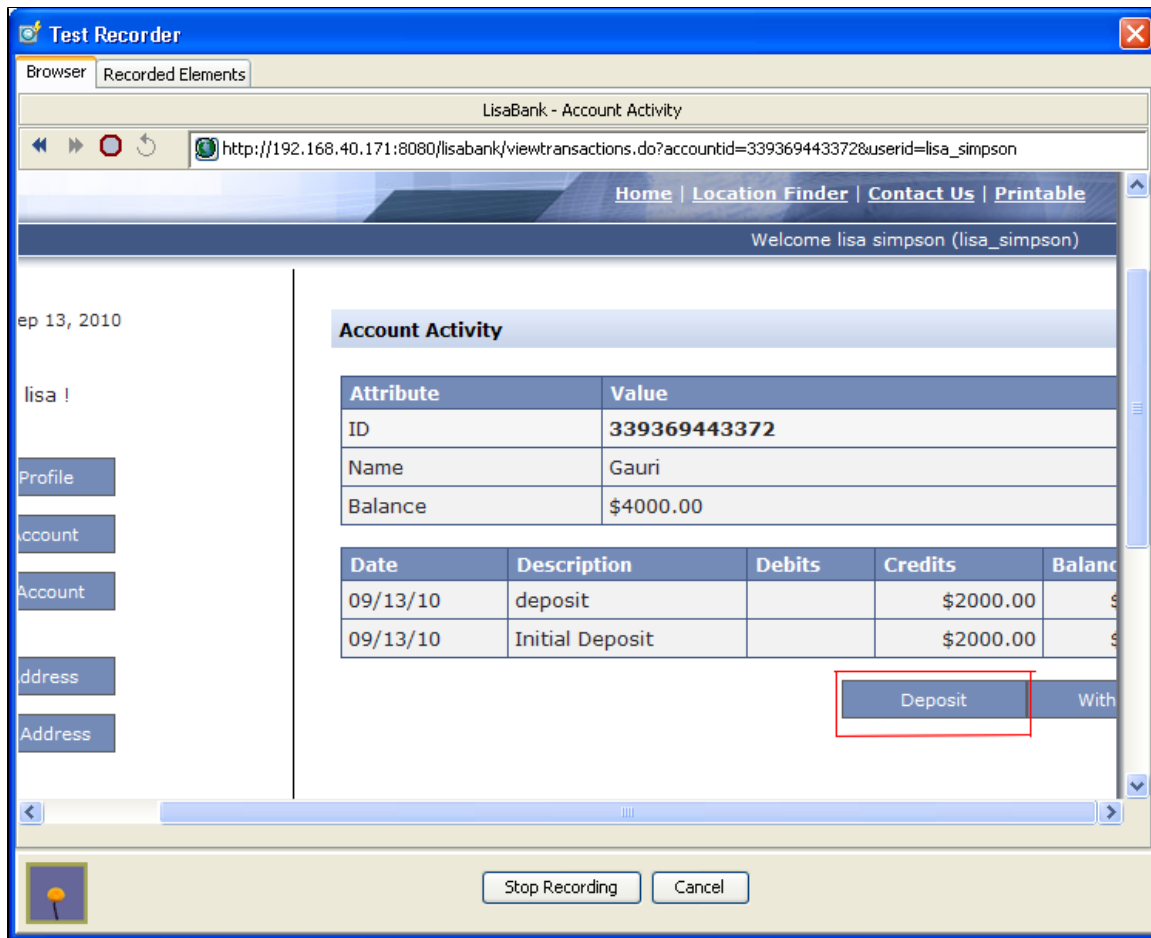
- Mortgage Planner
- Retirement Planner
- Online Transactions
- Fund Allocation
- Identity Security

A bar chart titled 'Fund Statistics' showing performance across four categories (E, V, N, and an unlabeled one) with values ranging from 10 to 90. The chart has a legend on the right with colored squares corresponding to the bars.

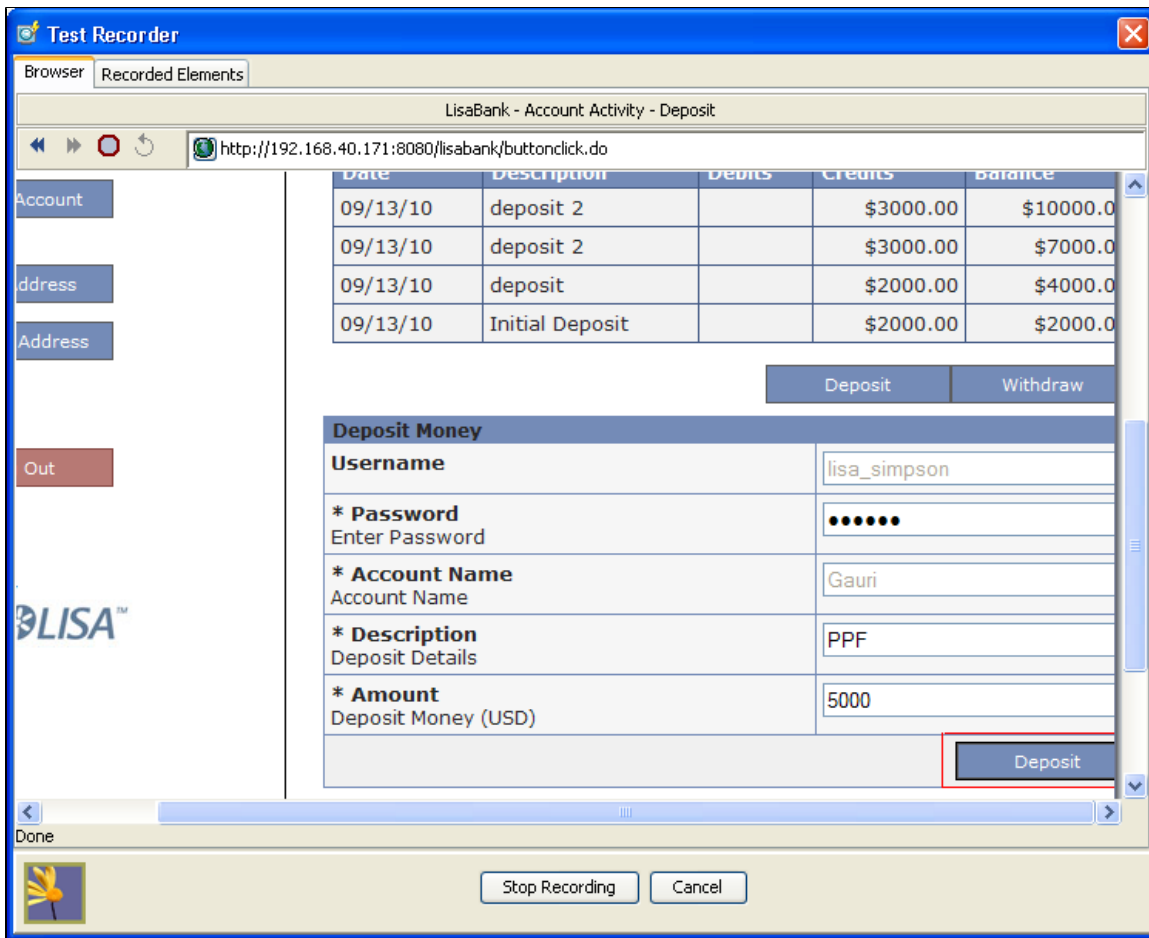
Stop Recording

Cancel

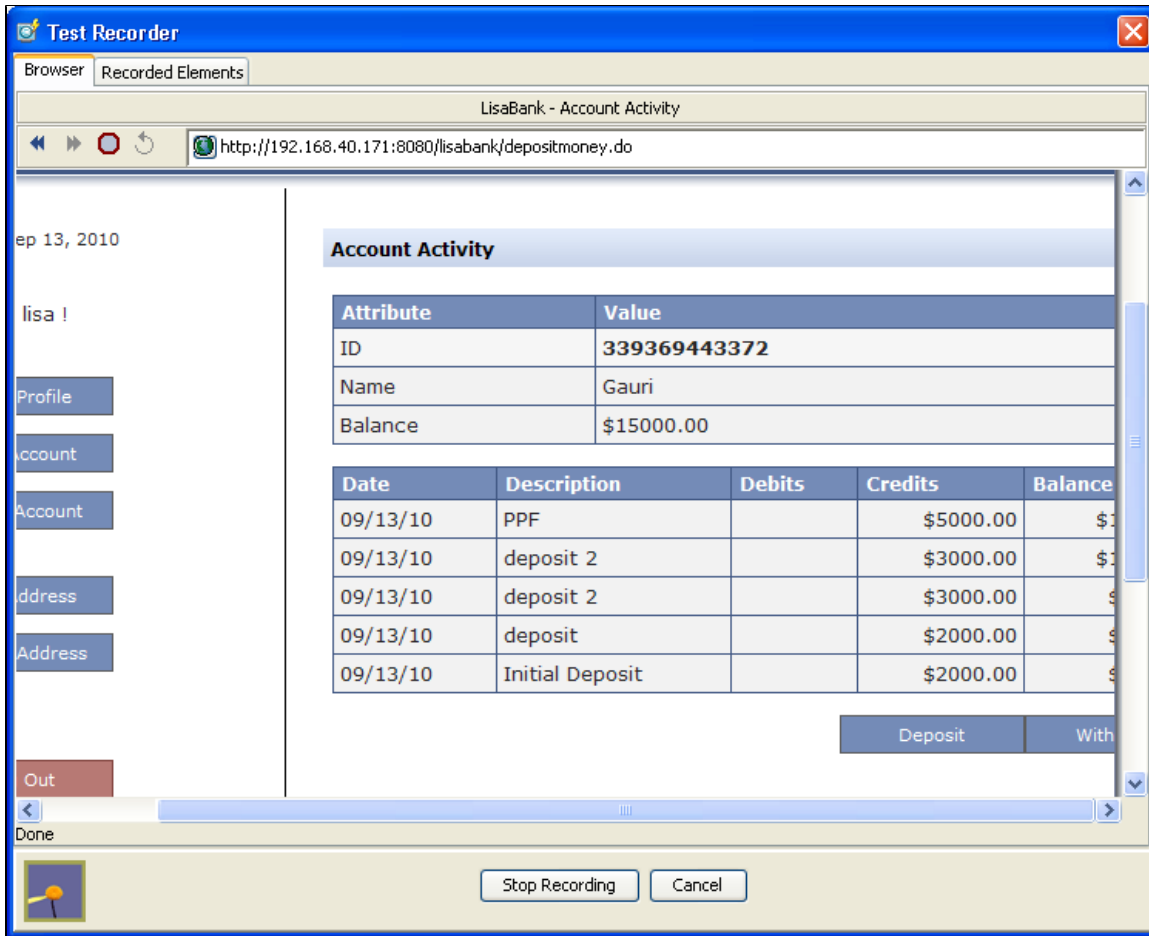
The Account activity is seen as below:



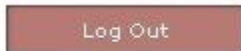
3. Click **Deposit**. To deposit or withdraw money, it will ask you for password again.
4. Enter the password **golisa**, a description of the transaction and the **amount** for this deposit.
5. Click the **Deposit** button just below the Deposit Money section.



You can see the amount being deposited -

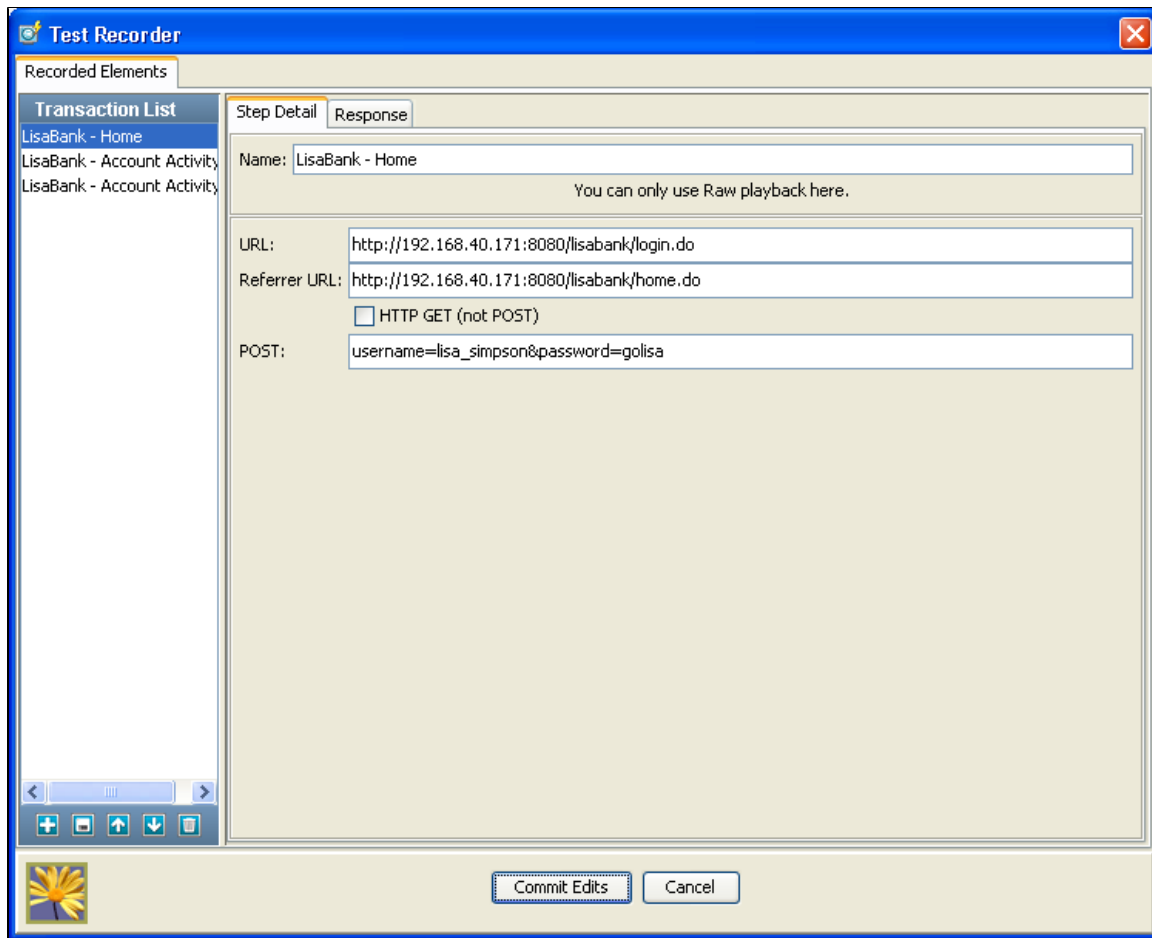


6. Click **Log Out** from the left navigation pane.

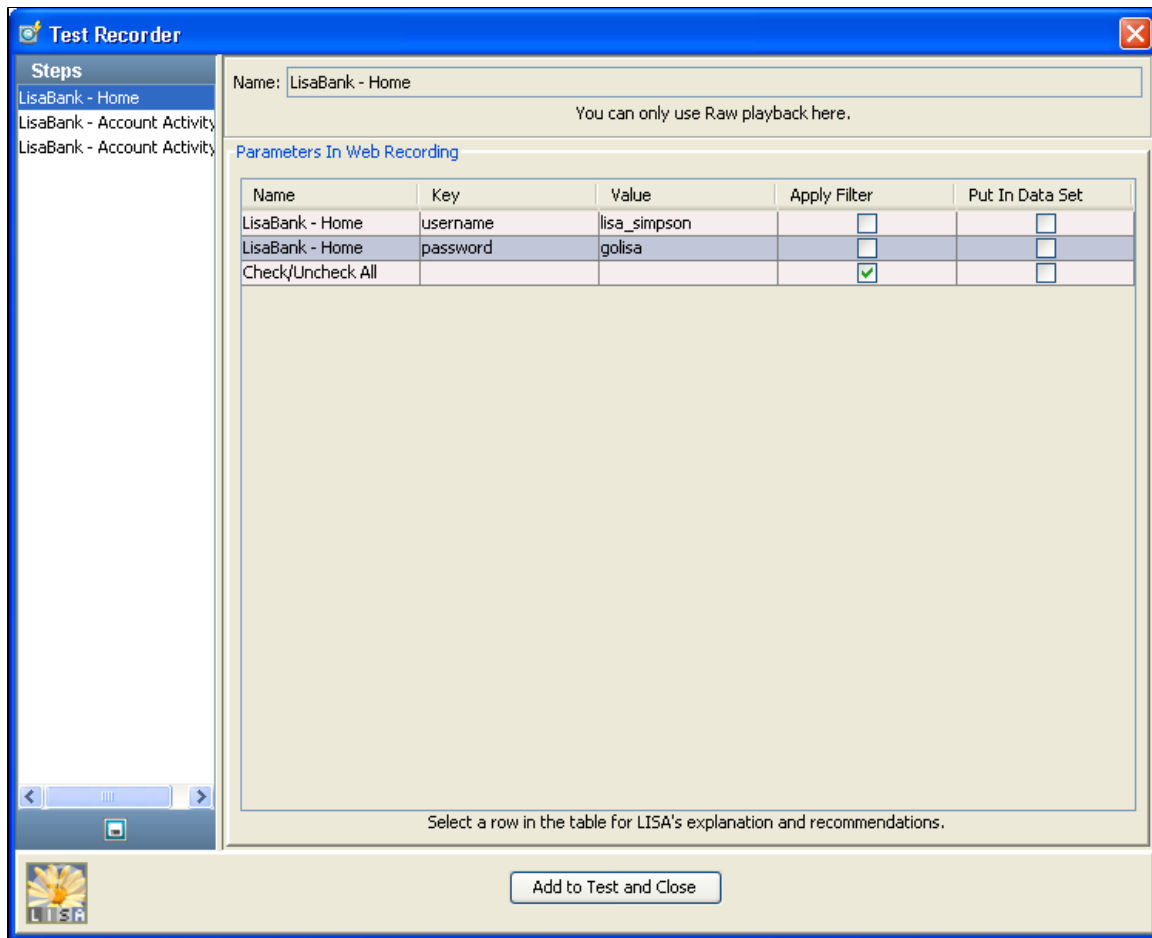


7. Click **Stop Recording** at the bottom of the Test Recorder window.

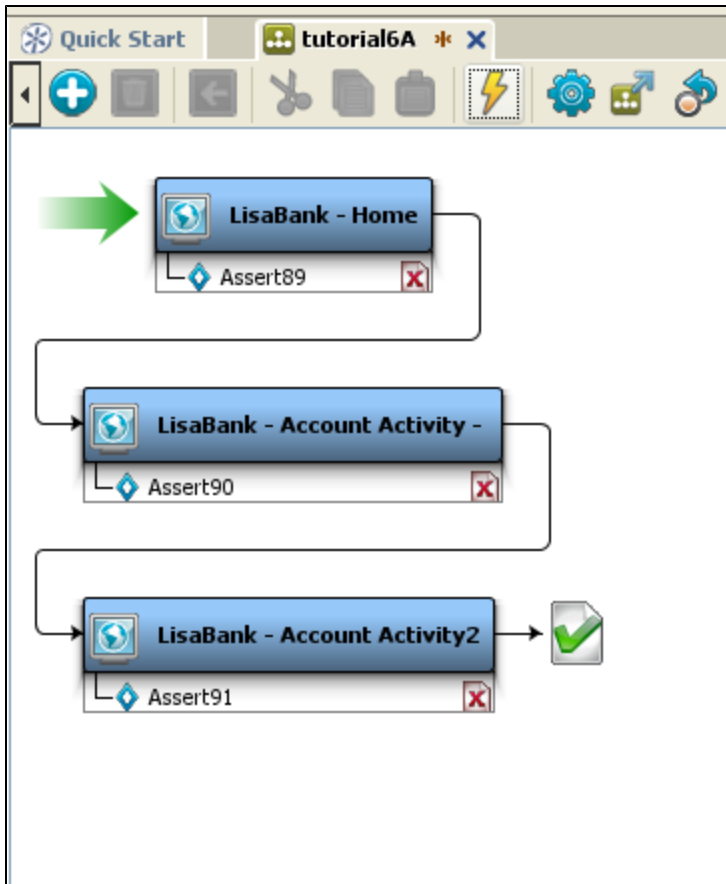
LISA automatically creates filters and properties for the web page references. The form fields for the deposit are also displayed. You can see the Transactions (steps) list on the left and each step detail on the right.



8. Click **Commit Edits** to save the recording.
9. In the Test Recorder window, click **Add to Test and Close**.



LISA populates the Model Editor with a new test case, having all the transaction information from the saved recording.





Step 4 - Save the Test Case

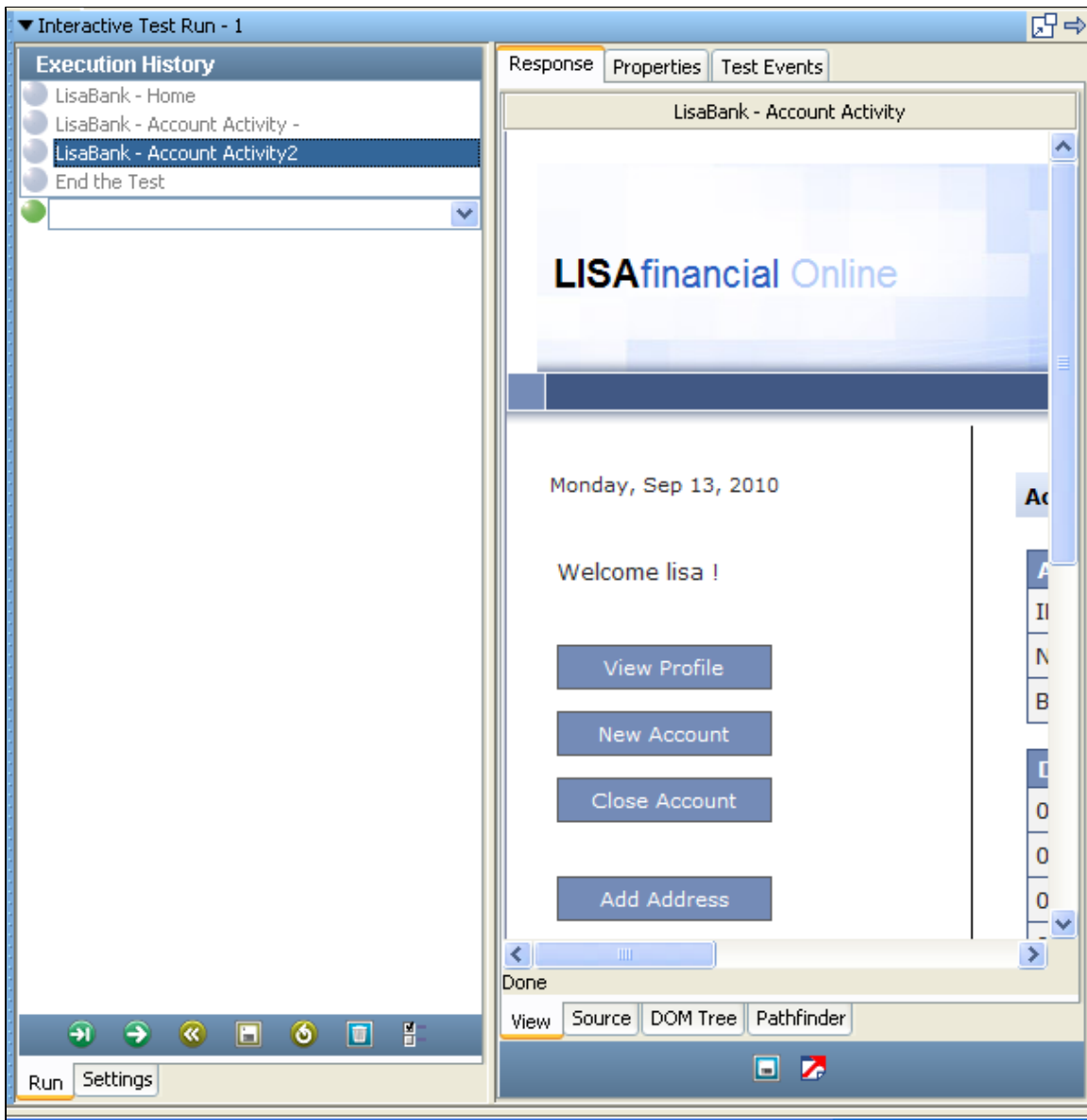
Part B - Running the Test Case

Part B - Running the Test Case

In this section, continue from [Part A - Record and Run the LISABank Test Case](#) to run the saved test case in the Interactive Test Run utility and view the results.

To run the test case in the ITR,

1. From the toolbar, click **Start ITR** .
2. In the ITR **Execution History** pane, click **Execute Next Step** .



The **Response > View** tab shows the rendered pages as the ITR replays the deposit into the MyChecking account. The **Source** tab shows the HTML code for the page captured in the step.

The deposit is the same as the way you recorded it.

LISA acts as the browser and sends the same HTTP requests to the Web server.

Part C - Modifying HTTP_HTML Request Test Steps (Optional)

Part C - Modifying HTTP_HTML Request Test Steps (Optional)

The Web recorder produces HTTP/HTML Request test step for each request.

You can edit and modify these test steps just like the other test steps in LISA. The recorder uses the parameters that you entered during the recording as values for the Post and Get parameters in the request.

To generalize your test, replace these hard-coded description and deposit values ('cash' and '1000.00') with properties from a data set. For more information see [Tutorial 2 - Step 8](#).

The figure below shows the **Account Activity3** step from our recording results. LISA has parameterized the **Host Name** and **Port** parameters and added these properties to the configuration. The values for **description** and **amount** are hard coded.

The objective in this part of the tutorial is to parameterize the test case to deposit different amounts of money by utilizing a numeric counting data set. When you run the test case, it uses deposit values different from the ones recorded.

Steps

Step 1 - Copy a Test Case

Save a copy of **tutorial6a** test case and name it **tutorial6c**.

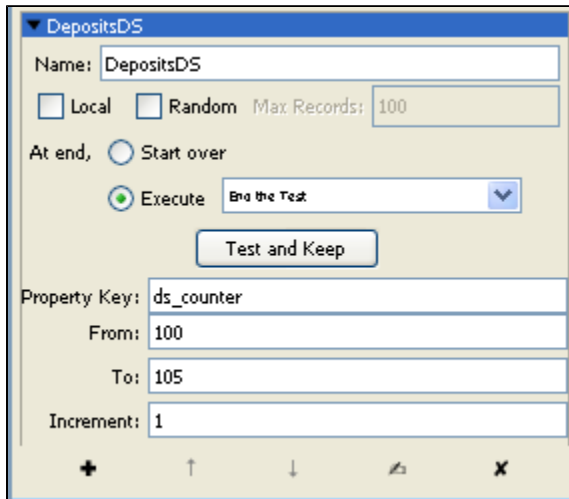
Hint: If you need a reminder, the procedure for saving a test case as a copy with a new name is explained in [Tutorial 3 - Step 1](#)

Step 2 - Add a Data Set

To add a data set to the test case:

1. Select the first step.
2. In the step element pane, click on the **Data Sets**.
3. From the **Common Datasets**, select **Create a numeric counting data set** to add this data set.

The Data Sets editor opens:



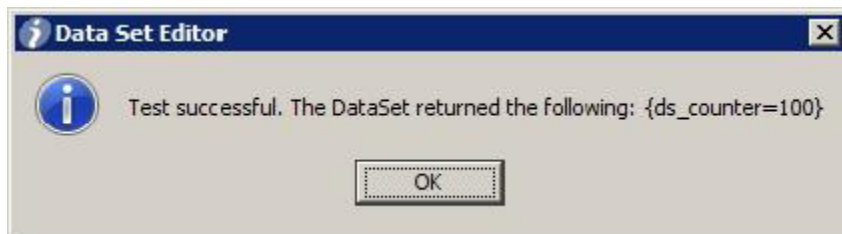
The screenshot shows the 'DepositsDS' dialog box. It has a 'Name' field with 'DepositsDS'. Below it are 'Local' and 'Random' checkboxes, and a 'Max Records' field with '100'. The 'At end' section has 'Start over' and 'Execute' radio buttons, with 'Execute' selected. A dropdown menu next to 'Execute' shows 'End the Test'. There is a 'Test and Keep' button. The 'Property Key' field has 'ds_counter'. Below it are 'From' (100), 'To' (105), and 'Increment' (1) fields. At the bottom are icons for adding, moving up, moving down, and deleting.

4. Enter the following in respective fields:

- In the **Name** field, enter a unique name **DepositsDS**.
- In the **At end** field, select the **Execute** option and select **End the Test** from the list.
- In the **Property Key** field, enter **ds_counter**.
- In the **From** field, enter **100**.
- In the **To** field, enter **105**.
- In the **Increment** field, enter **1**.

5. Click **Test and Keep** button to test the Dataset.

You should see a success message that shows the first row of data in the data set:



6. Click **OK**.

Step 3 - Change Parameters

To modify the POST parameters for the recorded deposit:

1. In the Test Case pane, select **LISABank - Account Activity3**.
2. In the **POST Parameters** area, change the **description** value to "**deposit ds_counter**".
3. Change the amount value to "**ds_counter**".

HTTP/HTML Request - LisaBank - Account Activity2

☒ Specify URL in parts ☐ Use property

Protocol: User:

Host Name: Password:

Port (opt):

Path:

URL Parameters	
Key	Value
Find: <input type="text"/>	

POST Parameters		
Key	Value	Encrypt
userid	{{useridLisaBank_-_Accou...	<input type="checkbox"/>
username	{{usernameLisaBank_-_Ac...	<input type="checkbox"/>
accountId	{{accountIdLisaBank_-_Ac...	<input type="checkbox"/>
password	golisa	<input type="checkbox"/>
accountname	{{accountnameLisaBank_-_...	<input type="checkbox"/>
description	deposit {{ds_counter}}	<input type="checkbox"/>
amount	{{ds_counter}}	<input type="checkbox"/>
Find: <input type="text"/>		

Form Encoding:

☐ Download files referenced (img, link, script, embed)

URL Transaction Info HTTP Headers Response

All Known St...	
Key	Value
Pass...	Pass7
User	Lisa7
DBDri...	org....
DBCo...	jdbc:...
DBPwd	sa
DBUs...	sa
EJBS...	local...
LISA...	E:\Li...
LISA...	
ENDP...	http:...
lisa.d...	com.i...
WSS...	192....
robot	0
LISA...	E:\Li...
LISA...	algiers
LISA...	My T...
LISA...	gourik
insta...	0
wsp	open

Step 4 - Save the test case Step 5

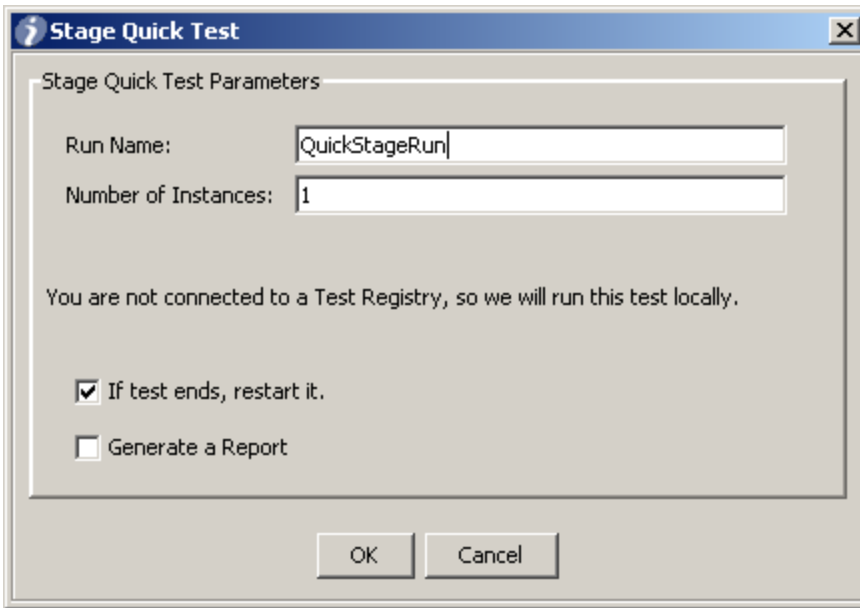
- Stage the Test case

To stage (or run) a Quick Test:

1. From the toolbar, click **Stage Test**.

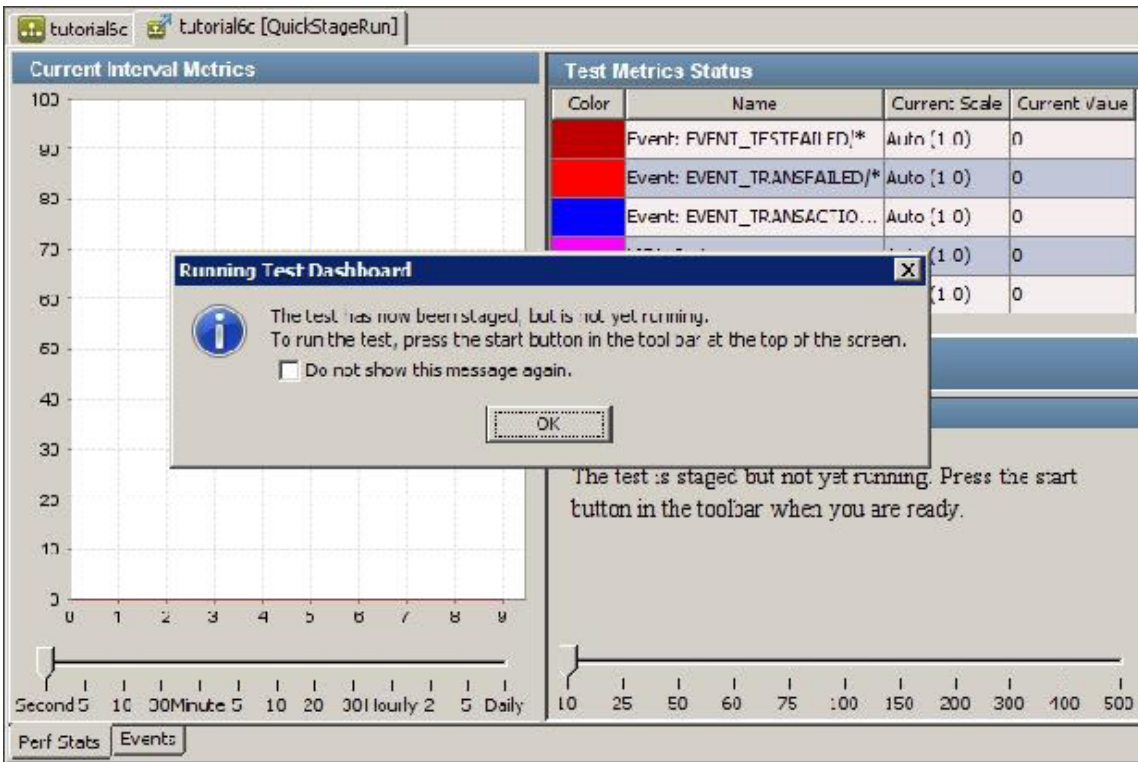


2. In the Stage Quick Test window, verify **If test ends, restart it** is checked (default).



3. Click **OK**.

4. The **Test Monitor** is displayed, but the test has not been started yet.

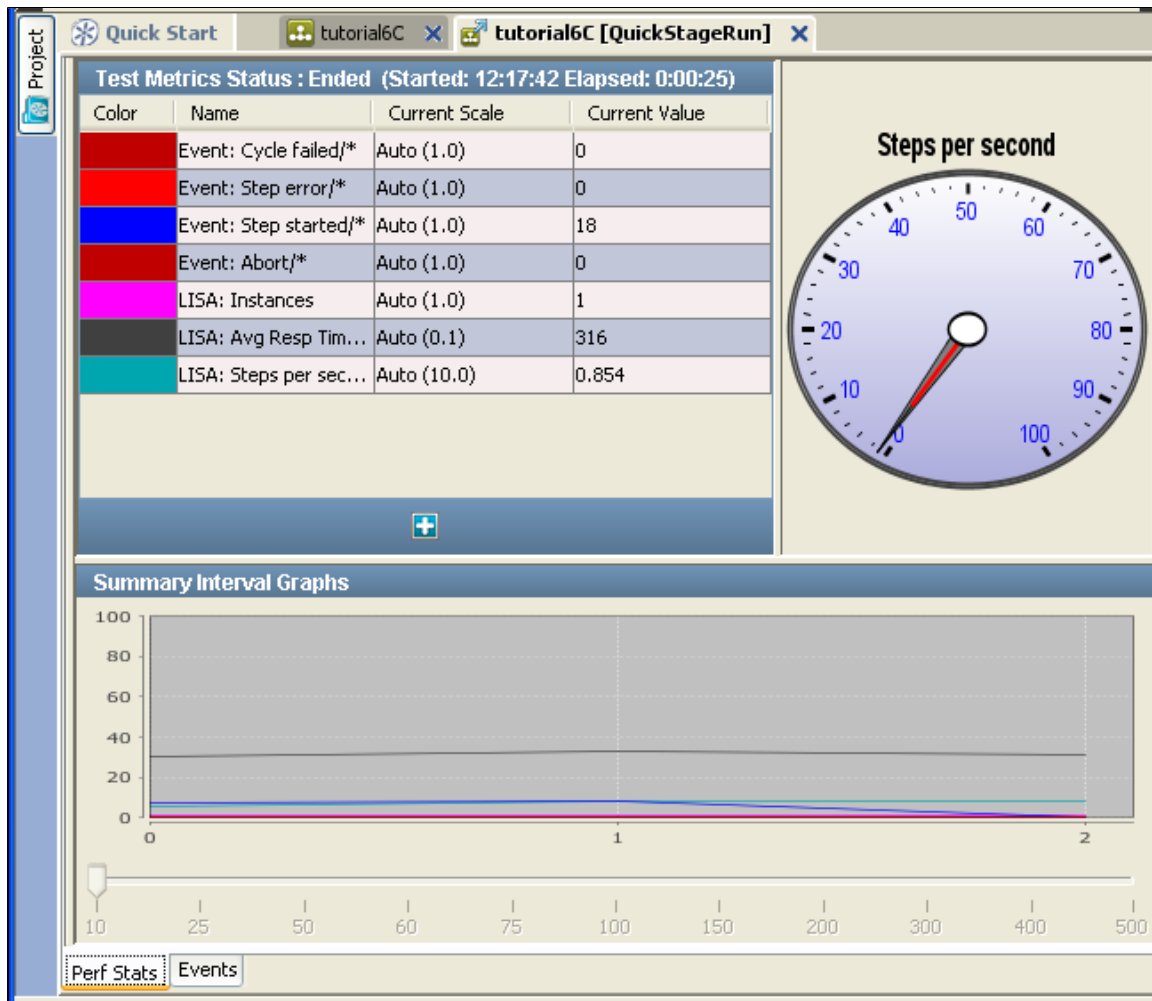


5. Click **OK**.

6. From the toolbar, click **Play**.



The line graphs show the progress of the test:



Step 6 - View the change in LISABank online

To view the deposits online,

1. Log into the LISAFinancial Online site using the user **lisa_simpson** and password **golisa**.
2. Click the account number link for checking account to view the deposits.


Monday, Sep 13, 2010

Welcome lisa !

View Profile
New Account
Close Account

Add Address
Delete Address

Log Out

Tested by


Account Activity

Attribute	Value
ID	339369443372
Name	Gauri
Balance	\$21230.00

Date	Description	Debits	Credits	Balance
09/13/10	deposit 105		\$105.00	\$21230.00
09/13/10	deposit 104		\$104.00	\$21125.00
09/13/10	deposit 103		\$103.00	\$21021.00
09/13/10	deposit 102		\$102.00	\$20918.00
09/13/10	deposit 101		\$101.00	\$20816.00
09/13/10	deposit 100		\$100.00	\$20715.00
09/13/10	deposit 105		\$105.00	\$20615.00
09/13/10	deposit 104		\$104.00	\$20510.00
09/13/10	deposit 103		\$103.00	\$20406.00
09/13/10	deposit 102		\$102.00	\$20303.00
09/13/10	deposit 101		\$101.00	\$20201.00
09/13/10	deposit 100		\$100.00	\$20100.00
09/13/10	PPF		\$5000.00	\$20000.00
09/13/10	PPF		\$5000.00	\$15000.00
09/13/10	deposit 2		\$3000.00	\$10000.00

Review

In this tutorial, you used the LISA Proxy Web recorder to record activity on the user's example Web site on the Demo Server.

You did the following:

- Set up and start the web recorder.
- Edit the resulting test case.
- Add inline filters and to objects.
- Ran the test case with the ITR.
- Staged a quick test.

4.7 Tutorial 7 - Testing an Enterprise JavaBean (EJB)

4.7 Tutorial 7 – Testing an Enterprise JavaBean (EJB)

LISABank provides a full set of EJBs to interact with an account, get the user and account information from the java interface.

In this tutorial, you will use the **Enterprise Java Bean Execution** step to call EJB methods within a test case and test the response with an assertion.

This is to test a simple EJB to ensure that the **addUser** and **deleteUser** methods work as expected.

LISA Concepts Discussed

In this tutorial, do the following:

- Use the Enterprise Java Bean Execution step.
- Use the Complex Object Editor with EJB objects.

Prerequisites

Complete Tutorial 6. Open LISA Workstation if not already open. Make sure you access to the demo server (either the local Demo Server, or the iTKO demo server).

Steps

Step 1 - Create a New Test Case

Create a new test case called **tutorial7**.

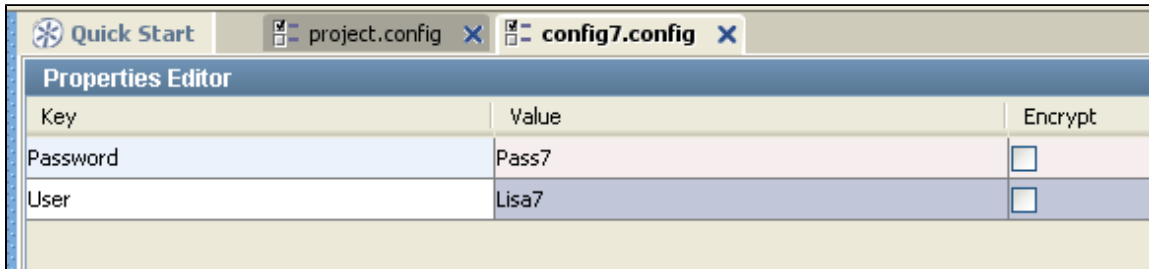
Step 2 - Create a new Configuration

To create a new configuration,

1. Refer to [Tutorial 2](#) , Step 4.
2. Create a new configuration **config7** and make it active for this test case.

Add two properties to it:

- Property named **User** with value **Lisa7**.
- Property named **Password** with value **Pass7**.

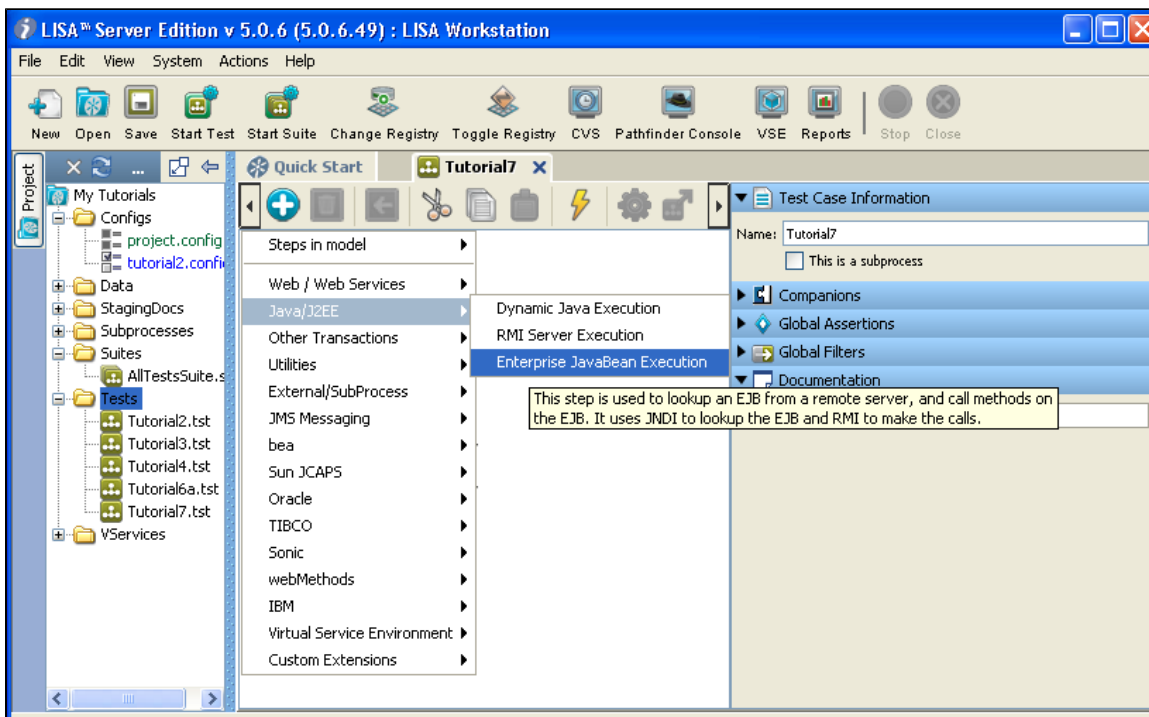


Step 3 - Add a EJB Test Step

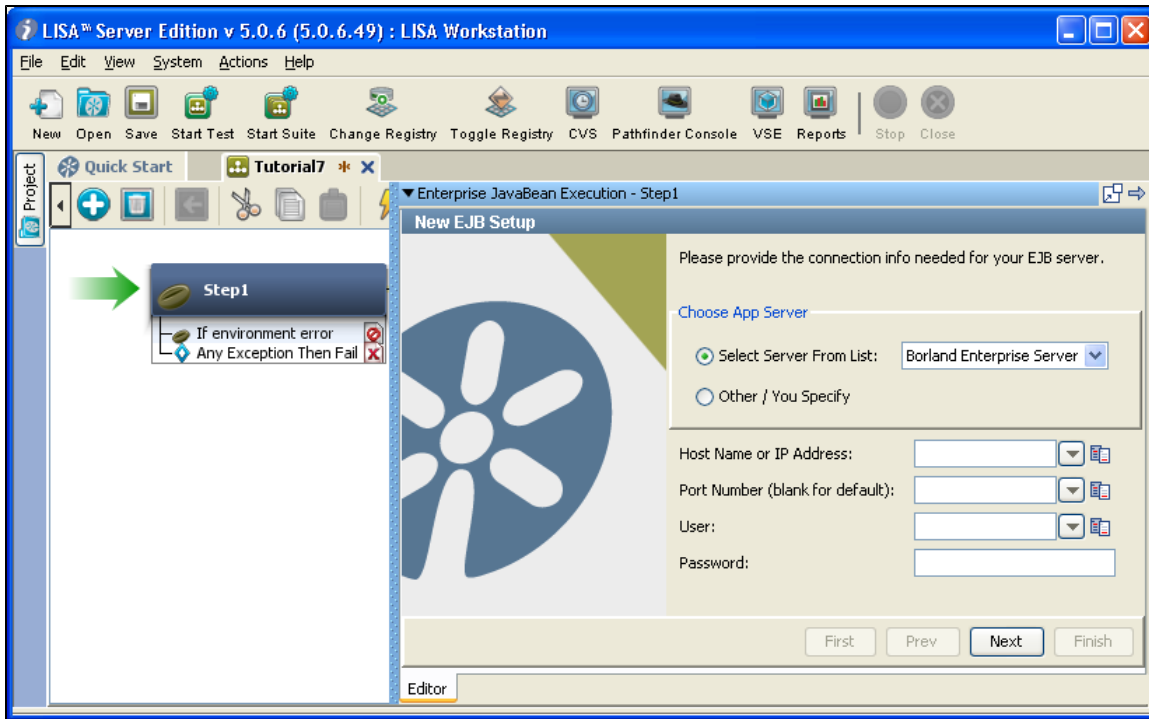
To add a EJB test step,



1. Right click in the model editor, and click **Add Steps**.



2. Click **Java/J2EE**, and select **Enterprise JavaBean Execution** step type.



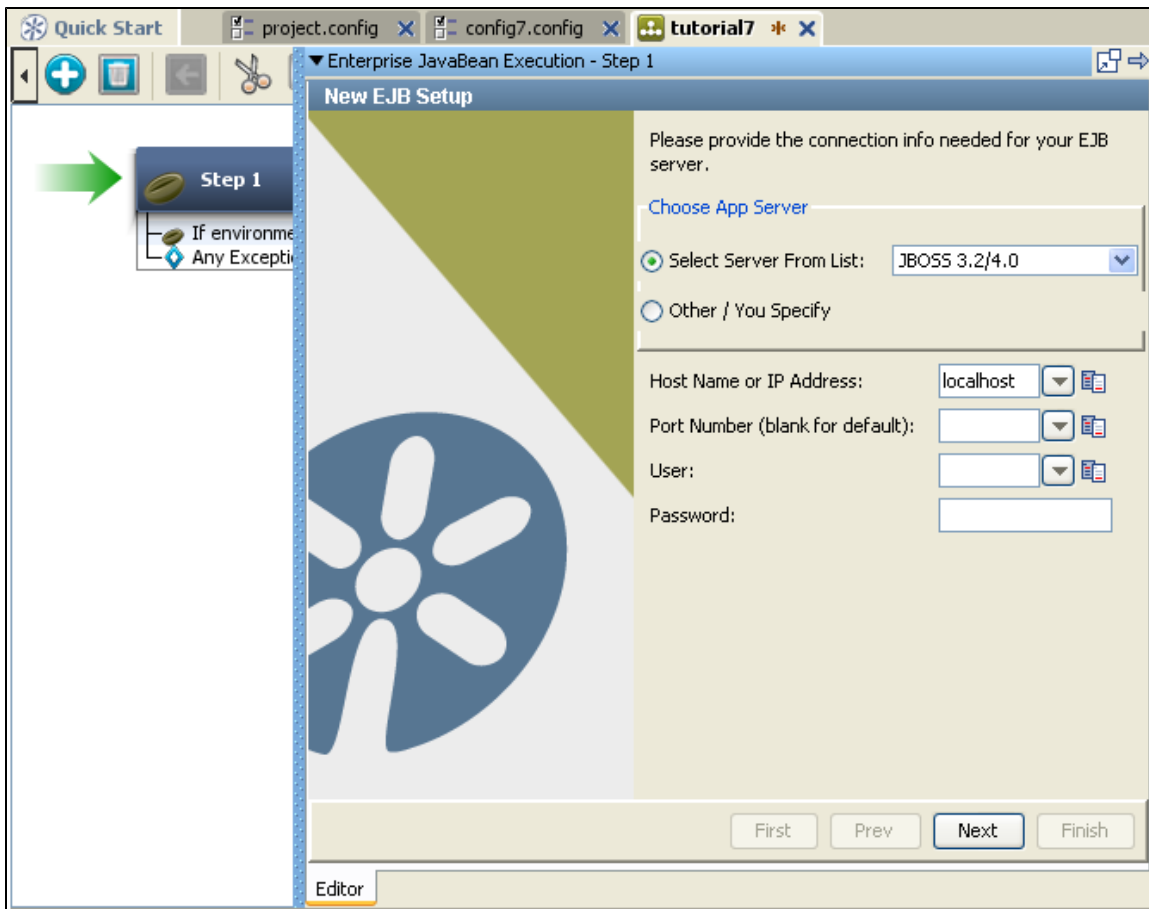
Step 4 -Locating the Demo Server

To identify and locate the demo server,

1. From the **Select Server From list**, select **JBOSS 3.2/4.0**.

Note - LISA supports many types of application servers.

2. In the **Host Name or IP Address** field, enter **localhost** if you are using the local demo server.
Enter **examples.itko.com** if you are running against the iTKO demo server.



3. Click **Next**.

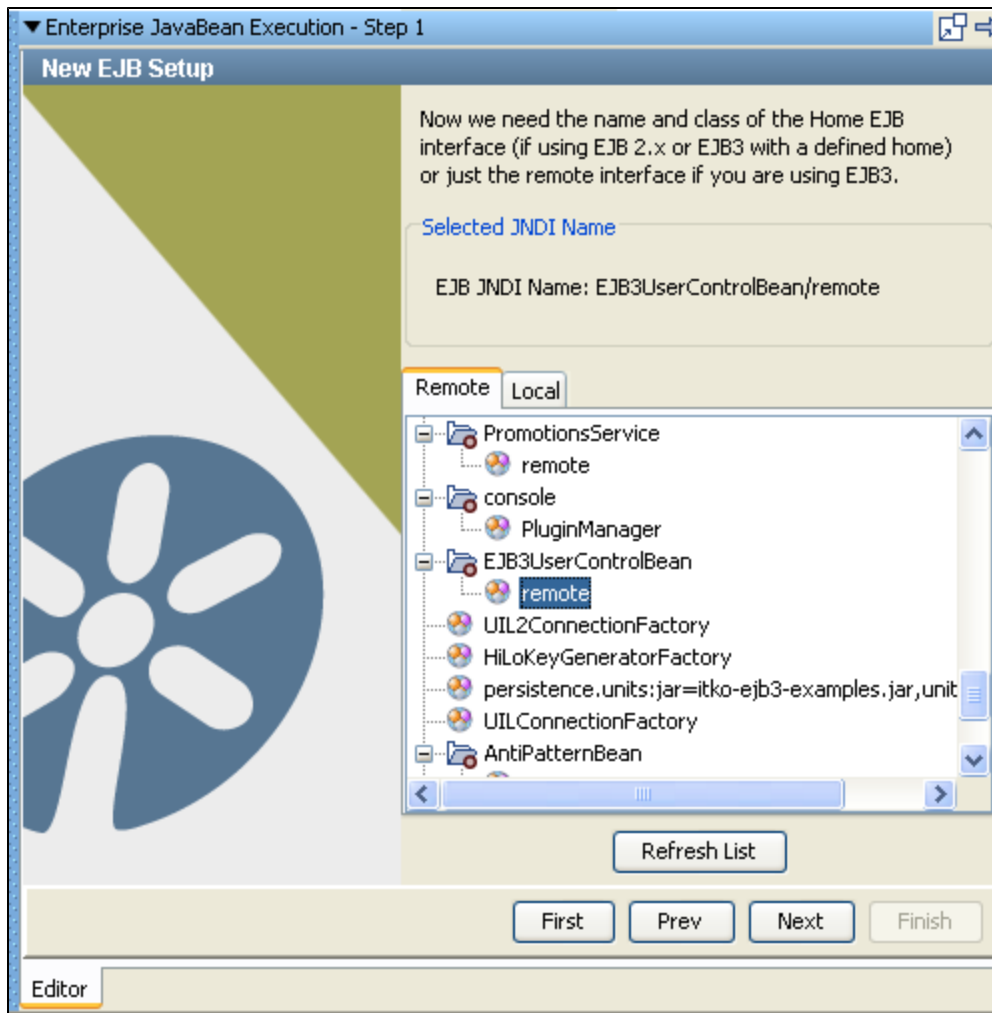
LISA fetches the name list from the EJB server.

Step 5 - Locating EJB Interface

To locate the EJB interface,

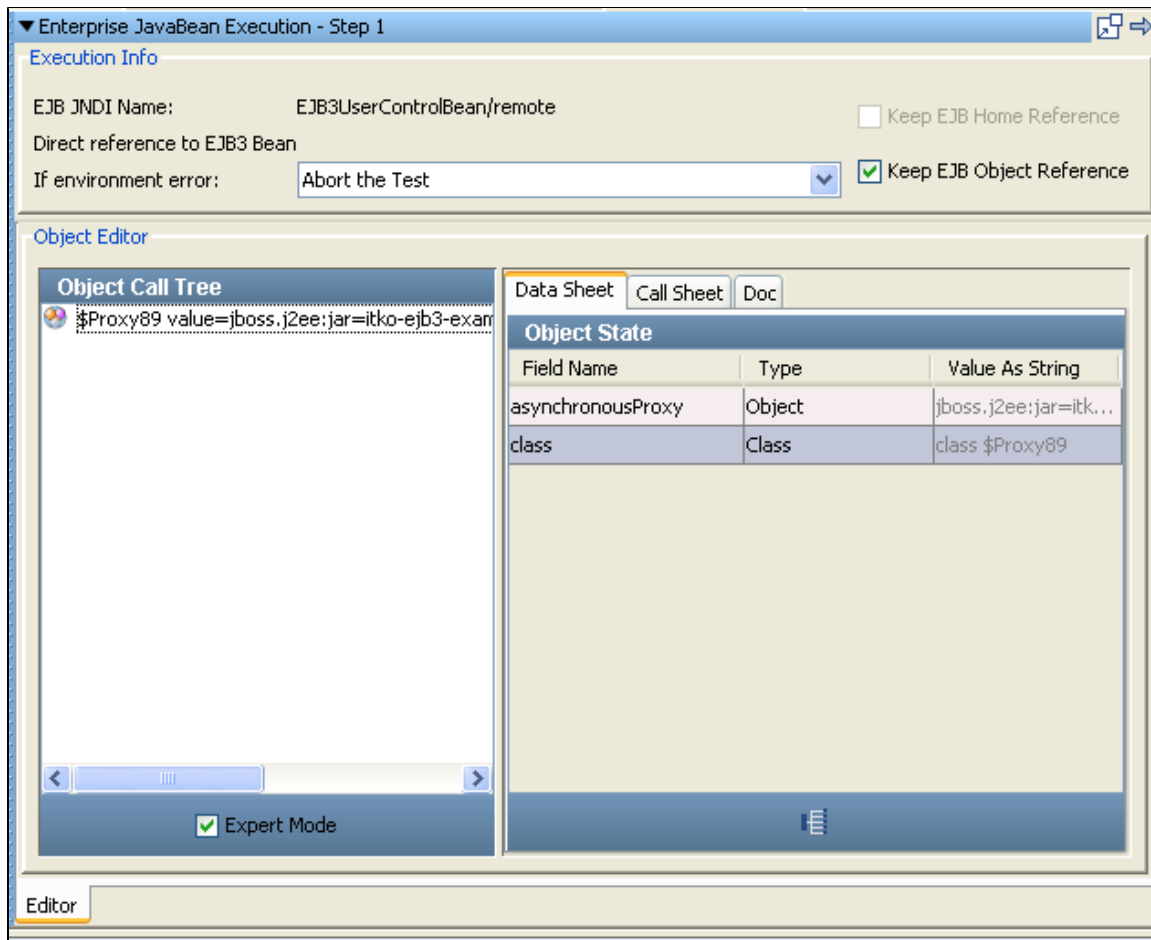
1. In New EJB Setup wizard, click the **Remote** tab.
2. Specify the fully qualified Java name of the home EJB interface.

This example uses **EJB.UserControlBean**. Select the name from the list.



3. Click **Next**.

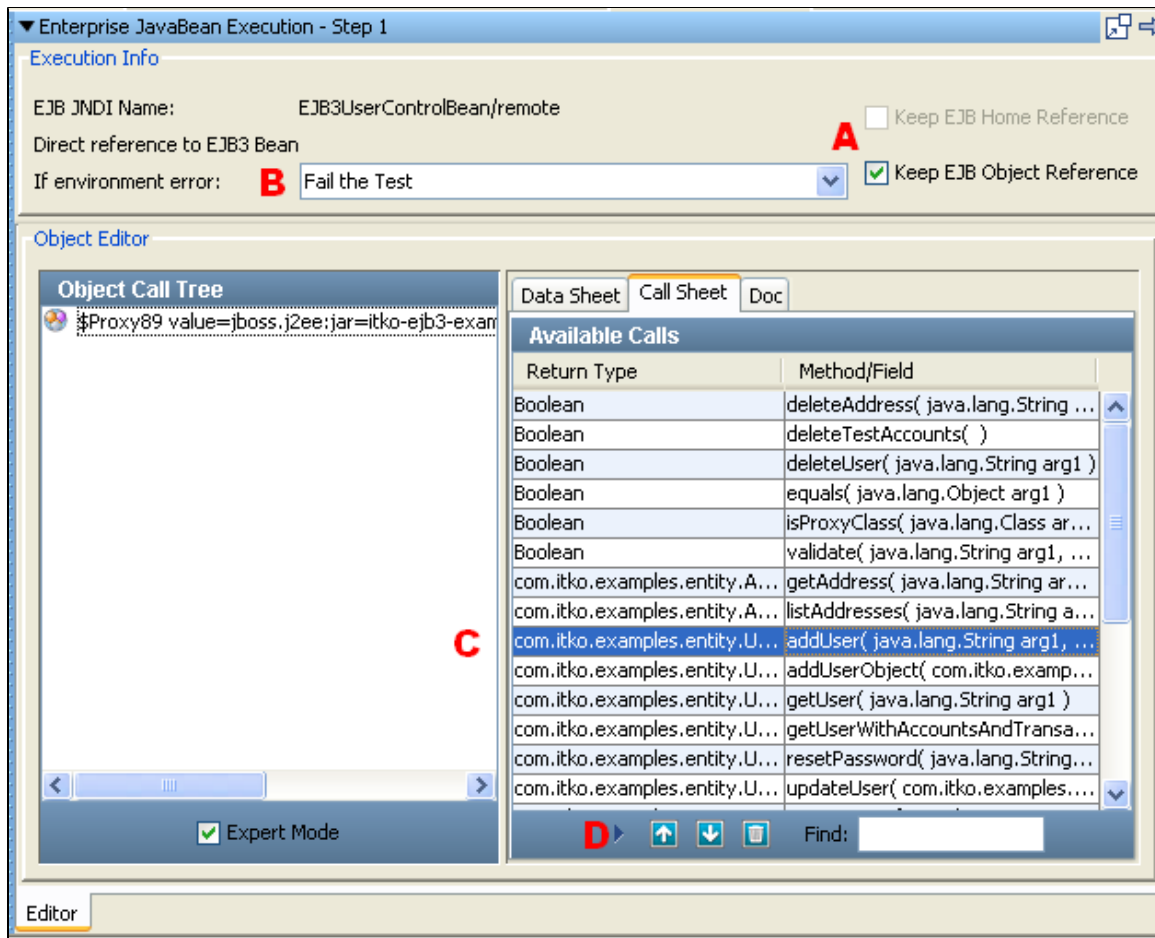
This will open the Complex Object Editor as below:



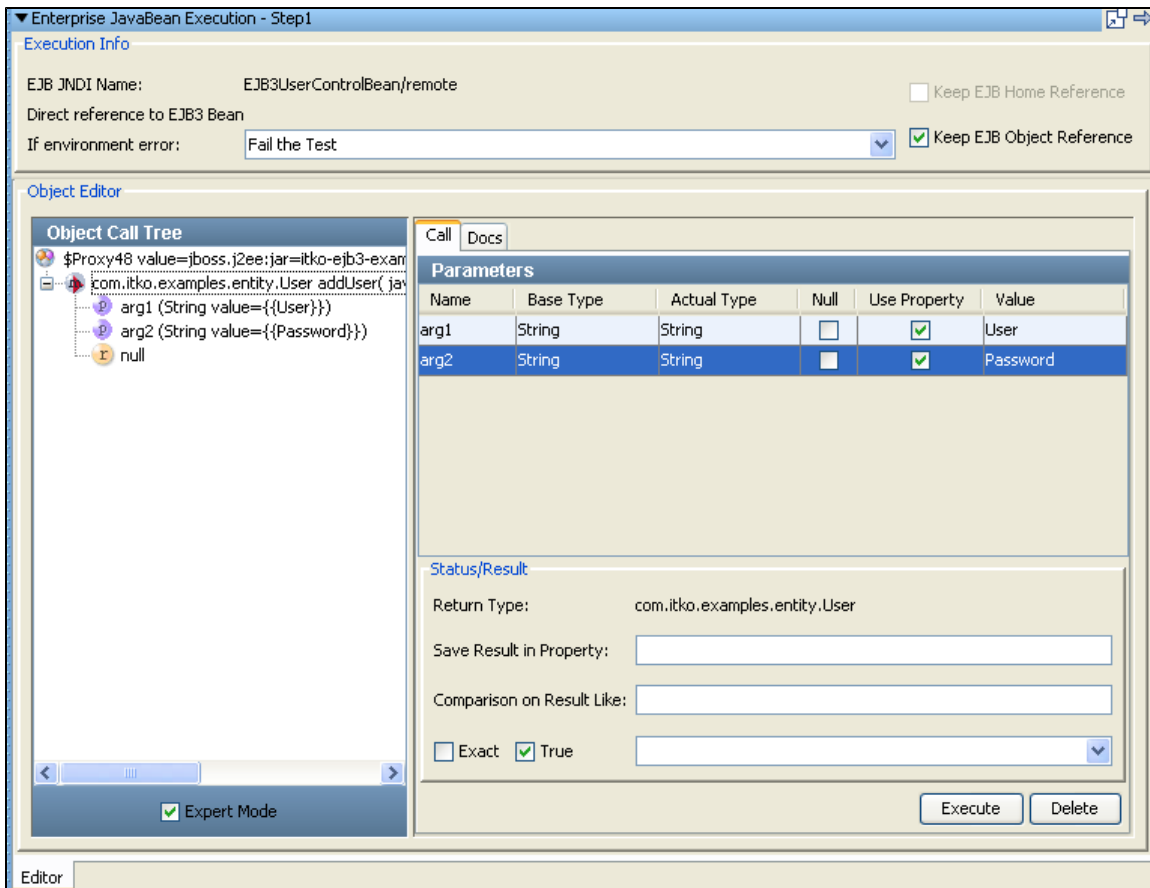
Step 6 - Configuring the EJB

To configure the EJB,

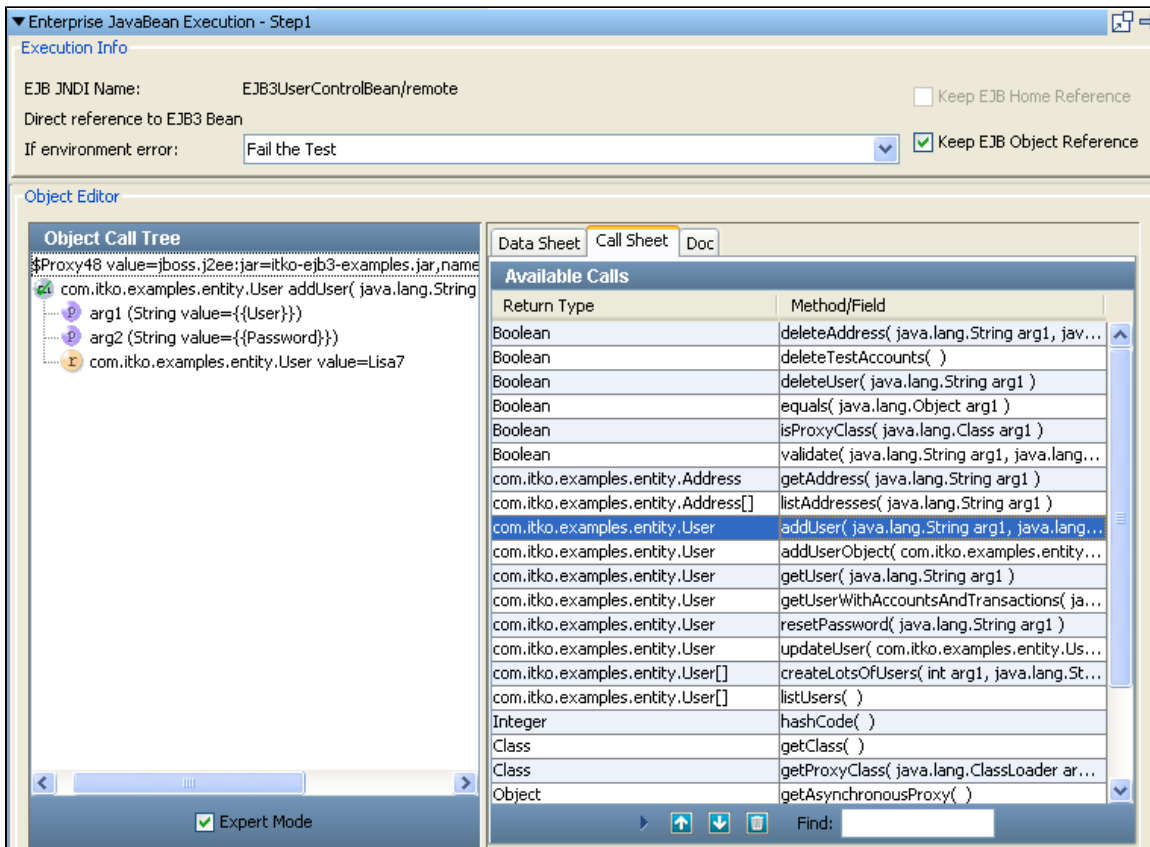
1. In the **Execution Info** area, if you use the same EJB object and home object repeatedly, check the **Keep the EJB Home Reference** and the **Keep the EJB Object Reference** check boxes, if they are not already selected by default (see A).
2. Set the **If Environment error** field to the step to execute if an exception occurs while executing this EJB Step.
Select **Fail the Test** from the list (see B).
3. In the Object Editor area, select the **Call Sheet** tab and select the **addUser** method (see C).
4. Click the **Invoke Method** icon (see D).



After Invoking the method, the Complex object editor looks like this -



- Click on **Execute** button.



Step 7 - Add an Assertion

To enter the method parameters and add an inline assertion:

1. In the Object Call Tree pane, check **Expert Mode** if it is not already enabled (see A).
2. Enter the property values for the **addUser()** method parameters (arg1 and arg2).
Enter **User** and **Password** that you added to the configuration in Step 3 (see B).
3. In the **Status/Result** area, add the inline assertion by checking **Exact** and un-checking **True**. (see C).
4. In the **Comparison on Result NOT Exactly** field, enter **True** (see D).
(The addUser method returns a Boolean.)
5. From the Exact list, select **Fail the Test**. (see E)
6. Click **Execute** (see F).

Enterprise JavaBean Execution - Step2

Execution Info

EJB JNDI Name: EJB3UserControlBean/remote ☐ Keep EJB Home Reference

Direct reference to EJB3 Bean

If environment error: Fail the Test ☒ Keep EJB Object Reference

Object Editor

Object Call Tree

- \$Proxy48 value=jboss.j2ee:jar=itko-ejb3-exan
- com.itko.examples.entity.User addUser(java.lang.String arg1 (String value={{User}}), java.lang.String arg2 (String value={{Password}}), boolean null)

Parameters

Name	Base Type	Actual Type	...	Use Prop...	Value
arg1	String	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>	User B
arg2	String	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Password

Status/Result

Return Type: com.itko.examples.entity.User

Save Result in Property:

Comparison on Result NOT Exactly: True **D**

☒ Exact ☐ True

Fail the Test **E**

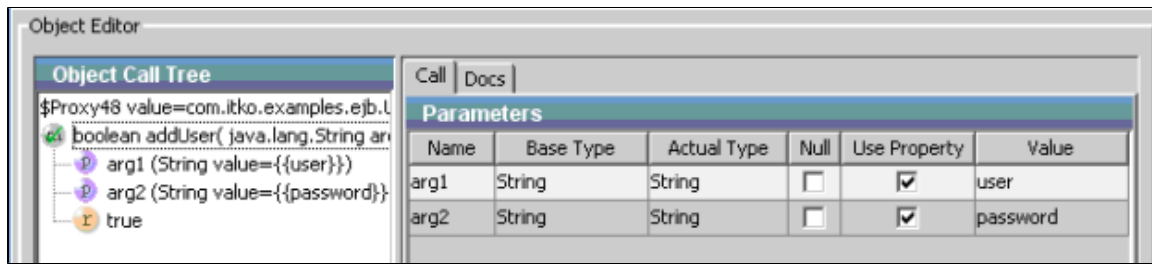
C **F** Execute Delete

Editor


The Object Call Tree displays the full context of the method calls, including:

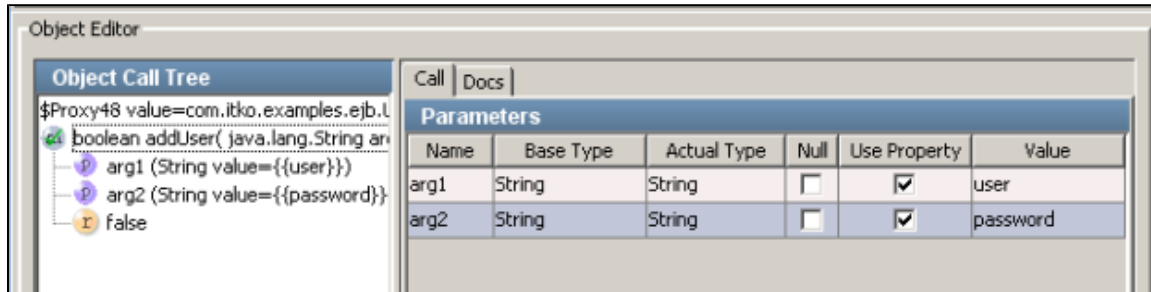
- - the method you wish to invoke
- - the parameters of this method
- - the return value of this method
- - the method complete, when you have called the method

The parameters to the method are displayed in the Object Call Tree, next to the parameter return value of this method is **true** in the Object Call Tree.



Test the addUser method again by clicking **Execute**.

The return  changes from true to **false** because the user has already been added.



Step 8 - Verify the Method Execution

To verify that the method executed properly:

1. Go to the LISAFinancial Online site.
2. Login as user **admin** with the password **admin**.

Scroll through the View Users list to see if **Lisa7** is listed.

It is Added as shown below:

[Help](#) | [Log Out](#)

LISAfinancial Online

[Home](#) | [Location Finder](#) | [Contact Us](#)

Welcome iTKO Admin (admin)

Friday, Oct 22, 2010

Welcome iTKO !

Add User

Modify User

Delete User

List Users

Add Account

Delete Account

Add Address

Delete Address

Log Out

Tested hv

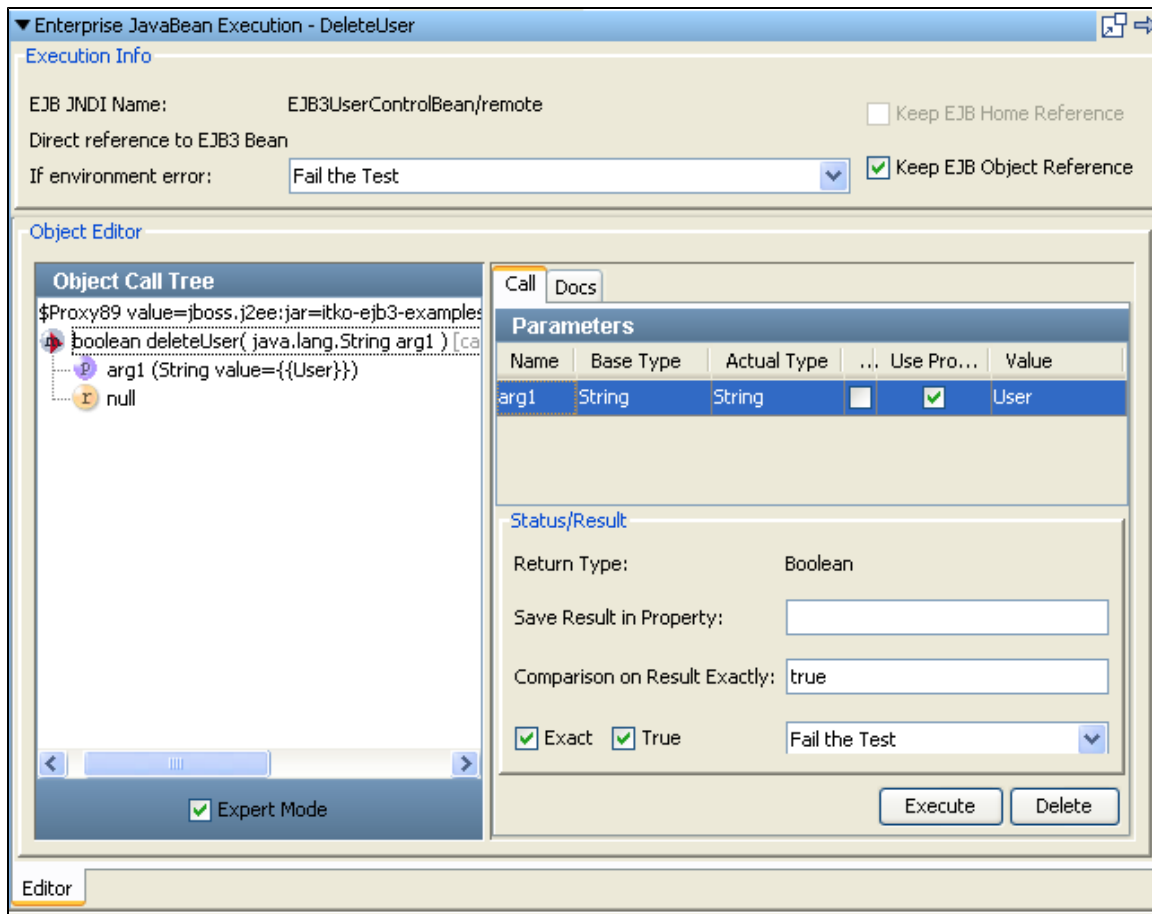
View Users

UserID	Name	Email
txzYiI7ik	null null	null
admin	iTKO Admin	lisabank-admin@itko.com
sbellum	Sara Bellum	sbellum@mycompany.com
wpiece	Warren Piece	wpiece@mycompany.com
areck	Amanda Reckonwith	areck@mycompany.com
boatv	Boaty Rabbit	boatv@rabbit.org
itko	itko test	itko.test@itko.com
lisa_simpson	lisa simpson	lisa.simpson@itko.com
LisaZ		
User		

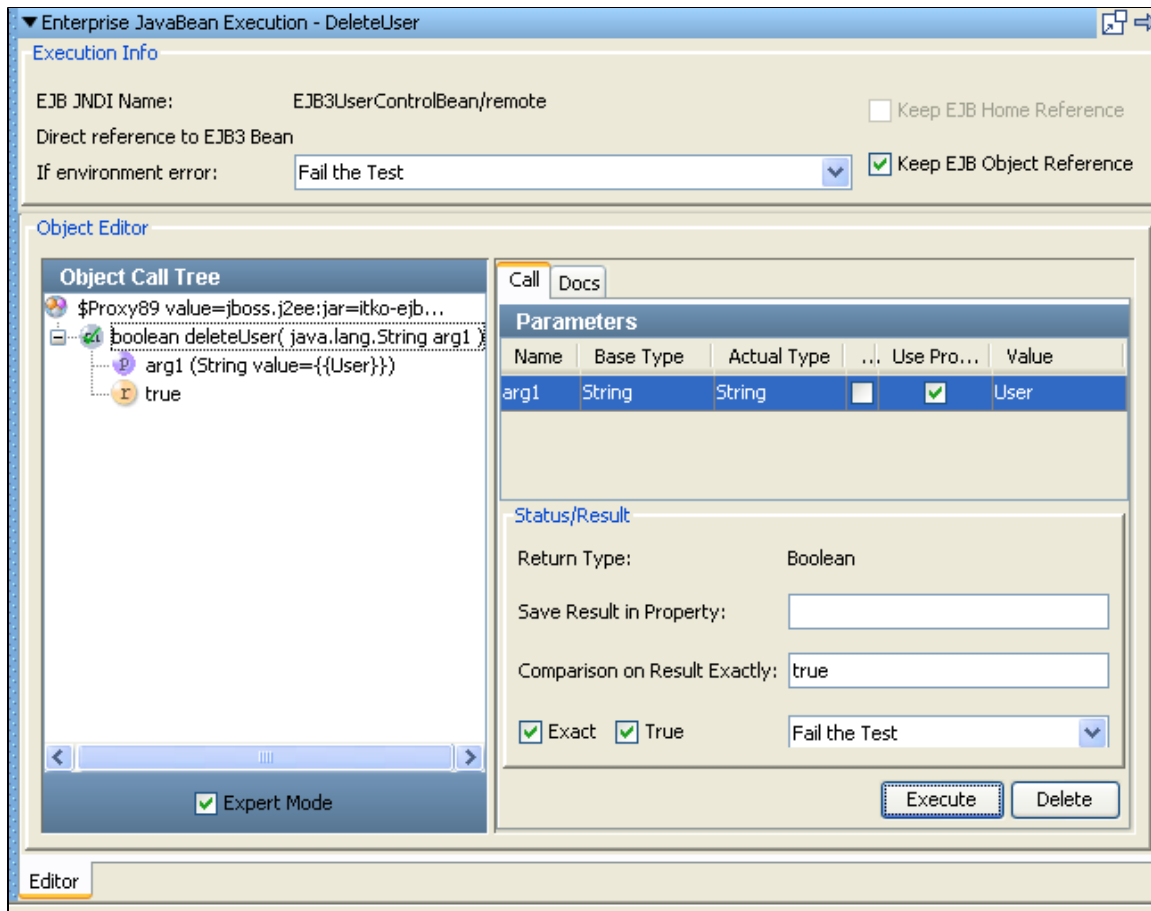
Step 9 - Add Another Test Step

To add another EJB step,

1. Repeat the tutorial beginning with Step 3 to add an EJB step named **DeleteUser**.
2. Use the method parameter property **User**.



3. Click Execute to execute this method and get results.



Note: The return  is **true**, indicating the user has been deleted.

Step 10 - Save the Test Case

Review

In this tutorial, you did the following:

- Created a test case consisting of two EJB test steps. The EJB object was loaded from the user's example application on the demo server.
- Created an EJB test step and load an EJB.
- Used the Complex Object Editor to manipulate EJB objects.
- Added inline filters and assertion to objects.

4.8 Tutorial 8 - Testing a Web Service

4.8 Tutorial 8 - Testing a Web Service

The following topics are available for this Tutorial.

- [Tutorial 8A - Testing a Web Service \(XML\)](#)
- [Tutorial 8B - Testing a Web Service \(Legacy\)](#)

Tutorial 8A - Testing a Web Service (XML)

Tutorial 8A – Testing a Web Service (XML)

In this tutorial, you will test a Web service (XML). You will use the **Web Service Execution** step to call Web service operations in a test case and test the response and request. These Web service operations provide the same functionality as the equivalent method calls in the EJB used in Tutorial 7.

LISA Concepts Discussed

In this tutorial, do the following:

- Use the Web Service Execution step.
- Execute the Method

Prerequisites

Complete Tutorial 5. Open LISA Workstation if not already open. Make sure you are running the Demo server (either the local Demo Server, or the iTKO demo server).

Steps

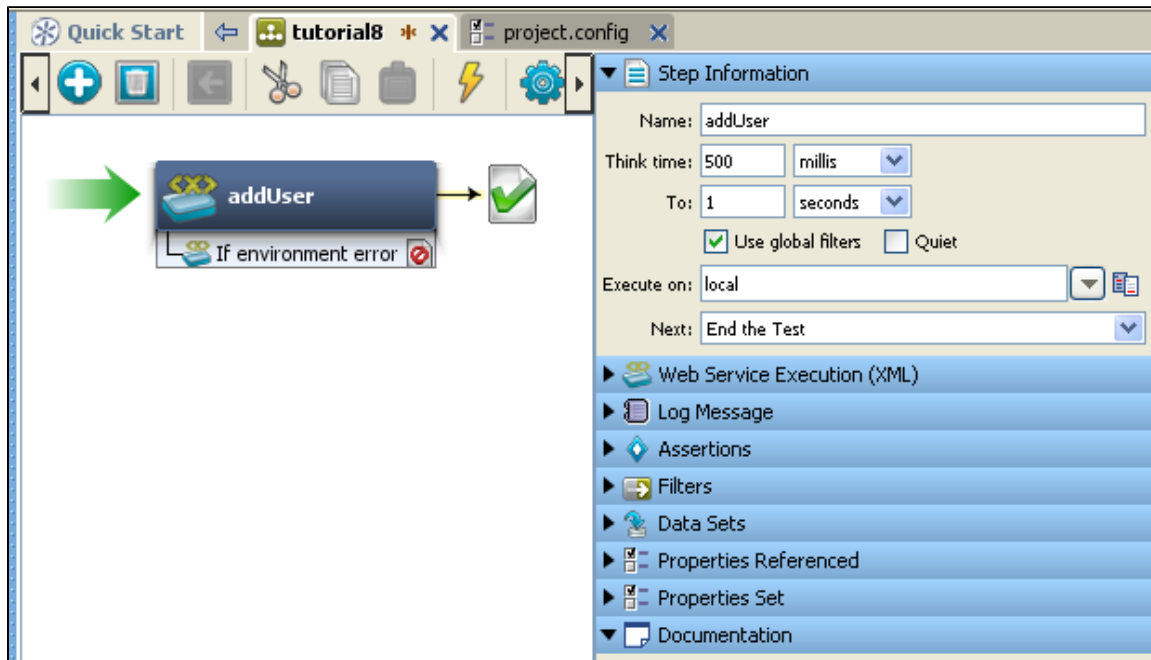
Step 1 - Create a New Test Case

Create a new test case called **tutorial8**.

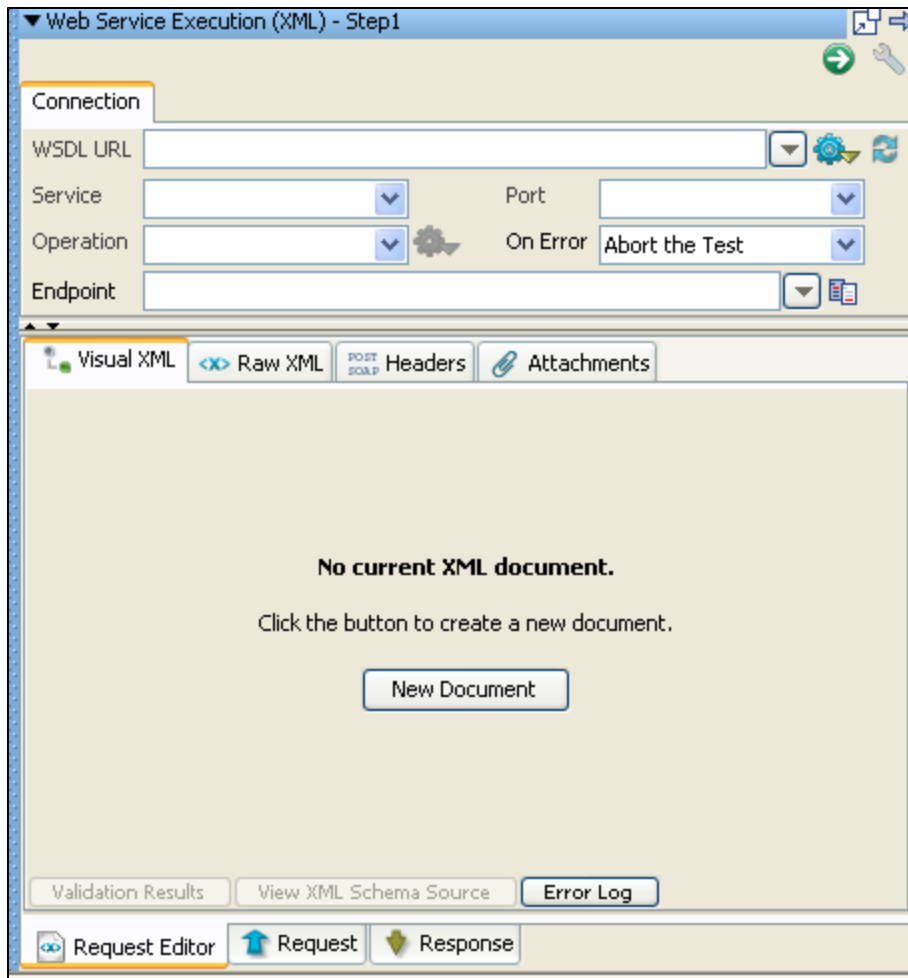
Step 2 - Add a Web Service Test Step "addUser"

To add a test step,

1. Right-click in the model editor, and click **Add Steps** or from the test case toolbar, you can click the Add Steps icon.
2. Click **Web/Web Services**, and select **Web Service Execution (XML)** step type.
3. **Step1** has been added to the model editor area.
4. Select Step1 and name it as **AddUser** in the Step Information area.



5. Double click the step to open the **Web Service Execution editor**.



6. Click on the **New Document** button to create a new XML document.

▼ Web Service Execution (XML) - Step1

Connection

WSDL URL

Service Port

Operation On Error

Endpoint

Visual XML Raw XML POST SOAP Headers Attachments

Node	Occurs	Nil	Nilable	Value

Validation Results View XML Schema Source Error Log

Request Editor Request Response

Step 3 - Create a Web Service Client

To create the Web Service Client,

1. In the WSDL field, enter the location of the WSDL.

```
http://WSSERVER:WSPORT/itko-examples/services/UserControlService?wsdl
```

2. In the **Service Name** field, enter **UserControlServiceService**.

Note - Do not use spaces within the name of the web service.

3. In the **Port** field, enter **UserControlServiceService**.

4. In the Operation field, select the operation to be tested. Here we have selected "**addUser**"

5. In the On Error field, select the action to be taken on test Error - **Abort the Test**.

Connection

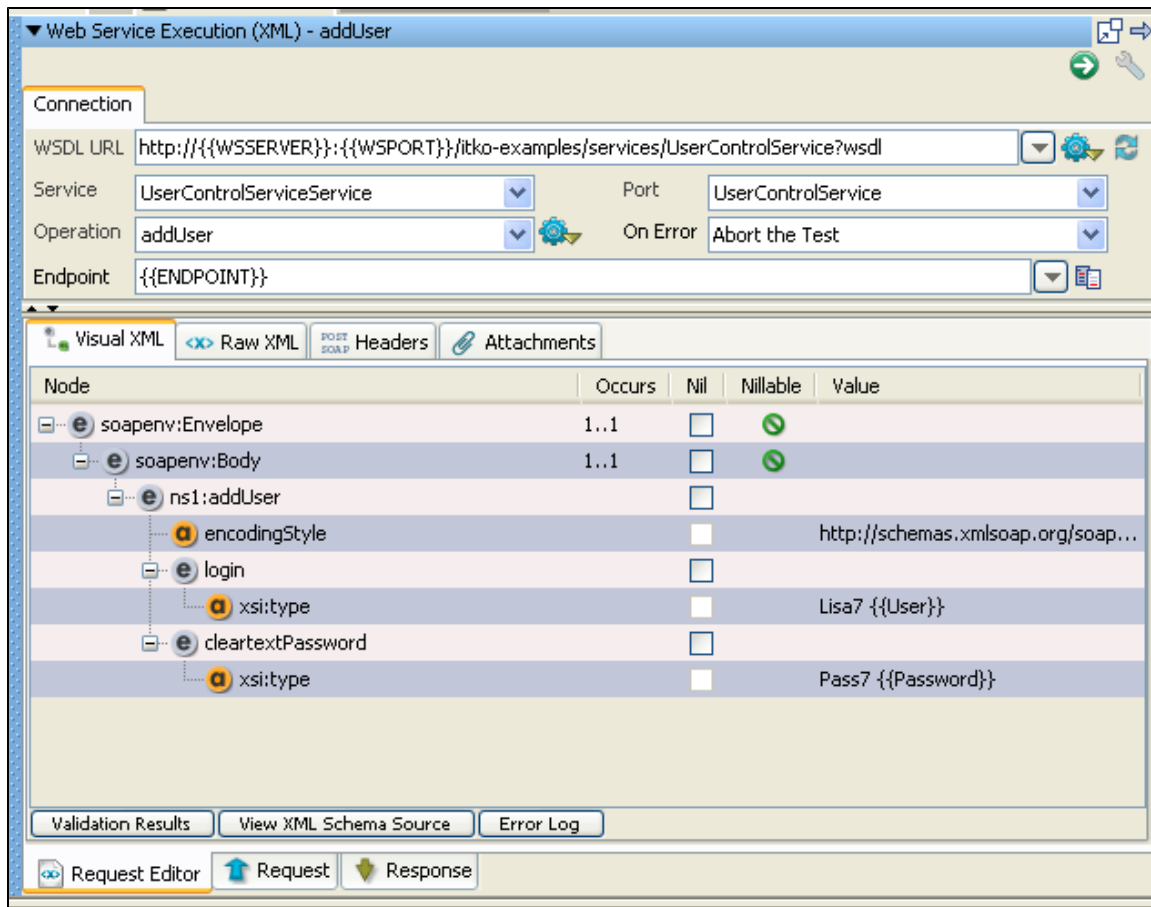
WSDL URL

Service Port

Operation On Error

Endpoint

LISA builds the Web Service client on this criteria.



Step 4 - Save the Test Case Step 5 - Execute the Test Case

To execute the test case,

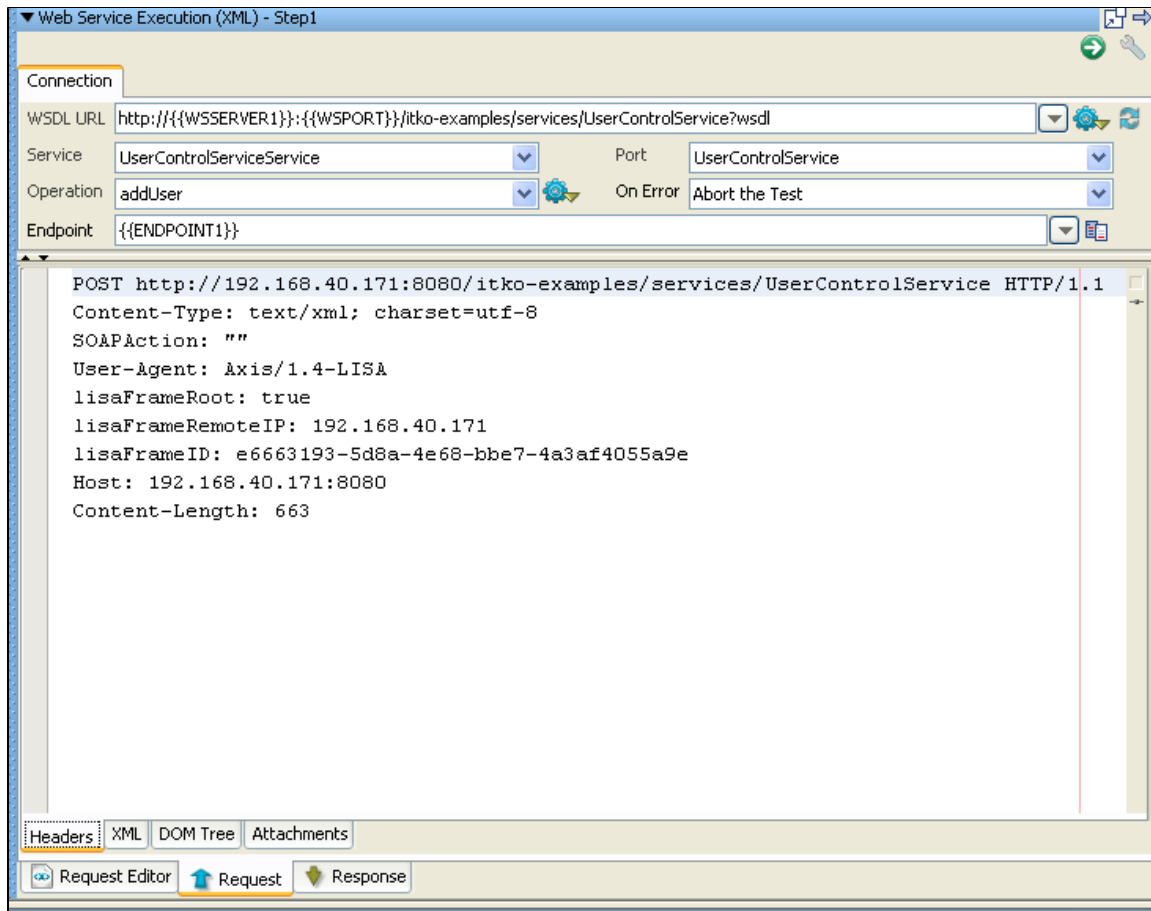
1. Click on the  button on top right.

This will Execute the test and show us the Request and Response.

Step 6 - Viewing the Request

To view the request upon execution,

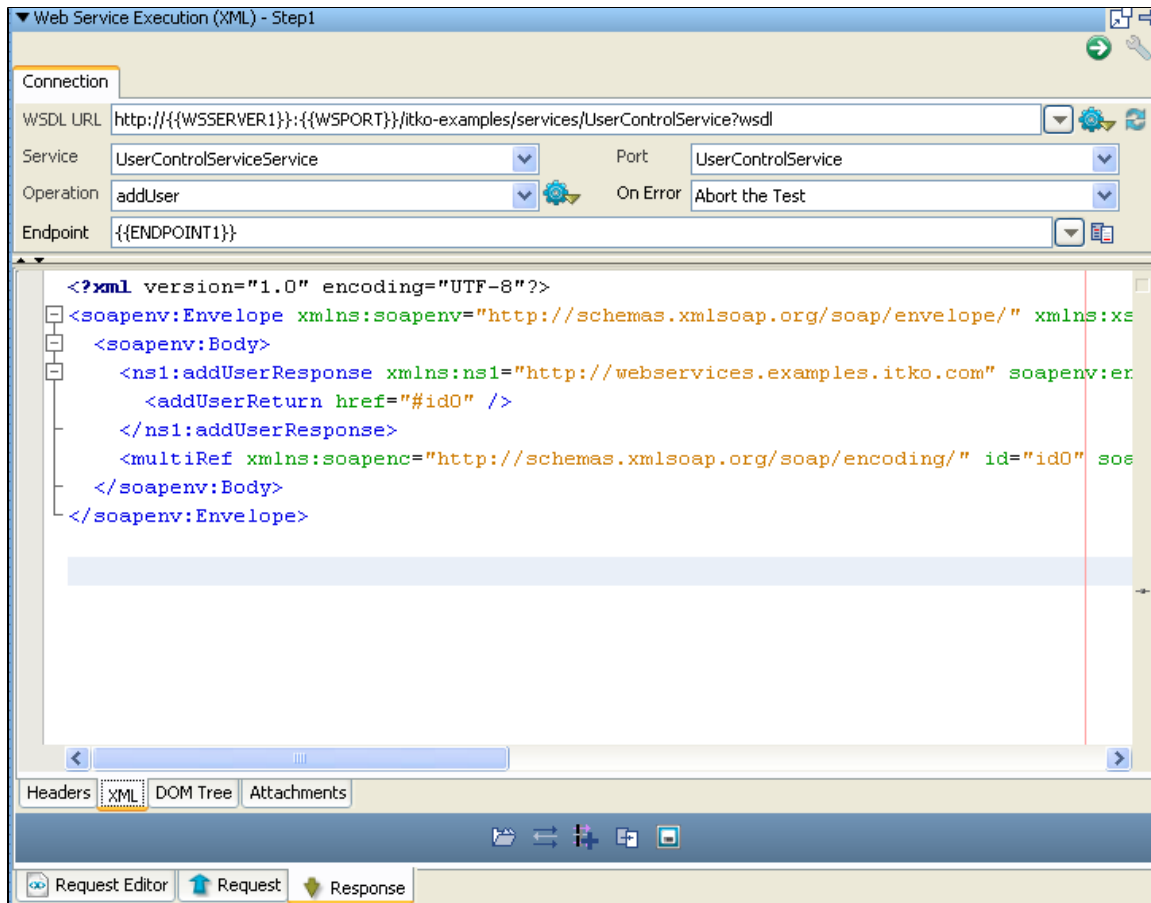
Click on the **Request** tab



Step 7 - Viewing the Response

To view the request upon execution,

Click on the **Response** tab



Review

In this tutorial, you did the following:

- Created a test case of Web Service XML test step.
The Web Service WSDL was available as part of the users example application on the Demo Server.
- Executed the `addUser` Method
- View the Request and Response for this operation.

Tutorial 8B - Testing a Web Service (Legacy)

Tutorial 8B – Testing a Web Service (Legacy)

In this tutorial, you will test a Web service that is part of the Web application in Tutorial 5, to ensure that the **addUser** and **deleteUser** operations work as expected.

Use the Web Service Execution (Legacy) step to call Web service operations in a test case and test the response with an assertion. These Web service operations provide the same functionality as the equivalent method calls in the EJB used in Tutorial 7.

LISA Concepts Discussed

In this tutorial, do the following:

- Use the Web Service Execution (Legacy) step.
- Use the Complex Object Editor for Web Service client objects.

Prerequisites

Complete Tutorial 5. Open LISA Workstation if not already open.

Make sure you are running the Demo server (either the local Demo Server, or the iTKO demo server).

Steps

Step 1 - Create a New Test Case

Create a new test case called **tutorial8**.

Step 2 - Create a New Configuration

Create a new configuration **config8** and make it active for this test case.

Add two properties to it:

- Property named **User** with value **Lisa8**.
- Property named **Password** with value **Pass8**.

Note - Make sure you also add these properties to the project.config file also.

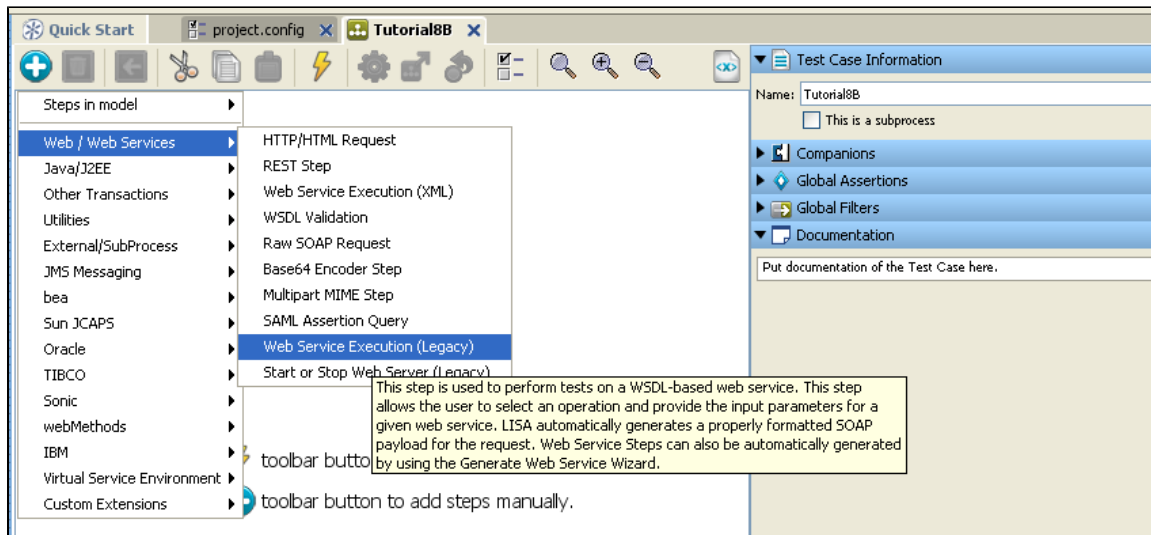
Step 3 - Add a Web Service Test Step "addUser"

To add a test step:

1. Right-click in the model editor, and click **Add Steps**.

or from the test case toolbar, you can click the Add Steps icon.

The menu of **Add Steps** is displayed.



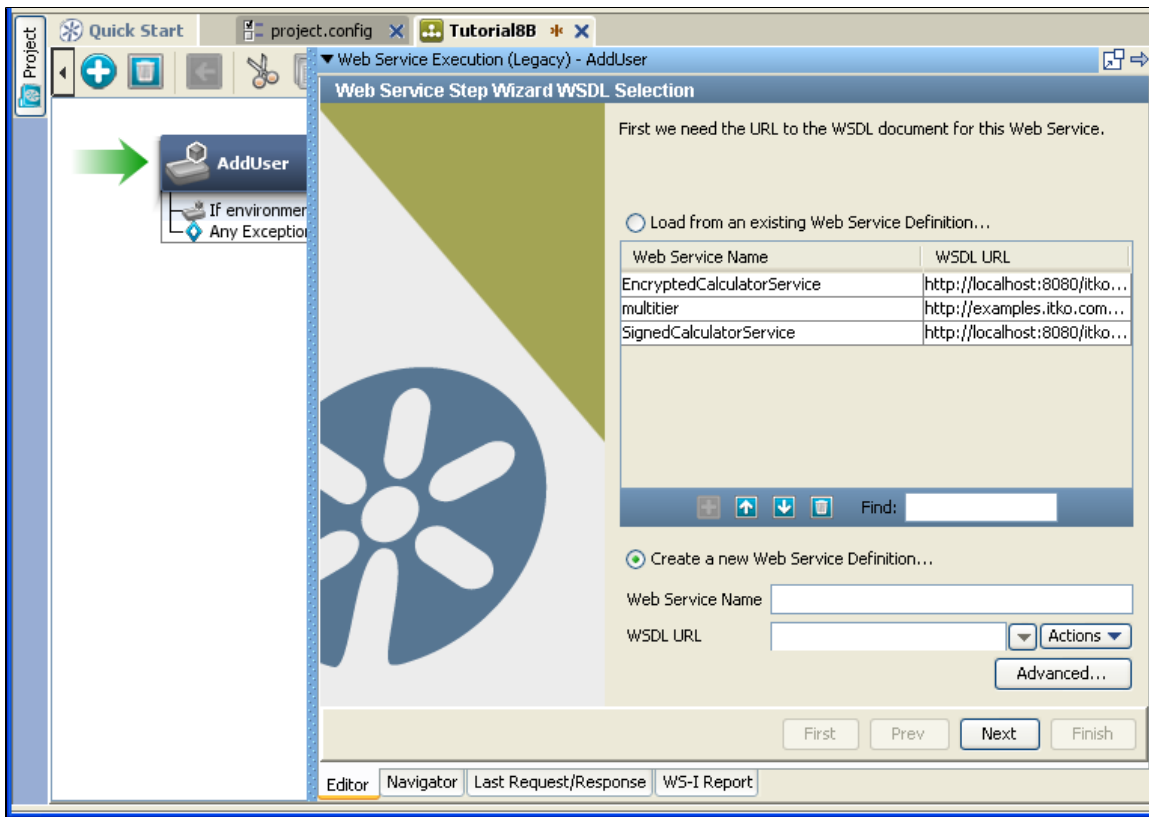
2. Click **Web/Web Services**, and select **Web Service Execution (Legacy)** step type.

3. **Step1** has been added to the model editor area.

4. Select Step1 and rename it as **AddUser** in the Step Information area.

5. Double click the step to open the **Web Service Execution editor**.

The **Web Service Step Wizard WSDL Selection** wizard opens.

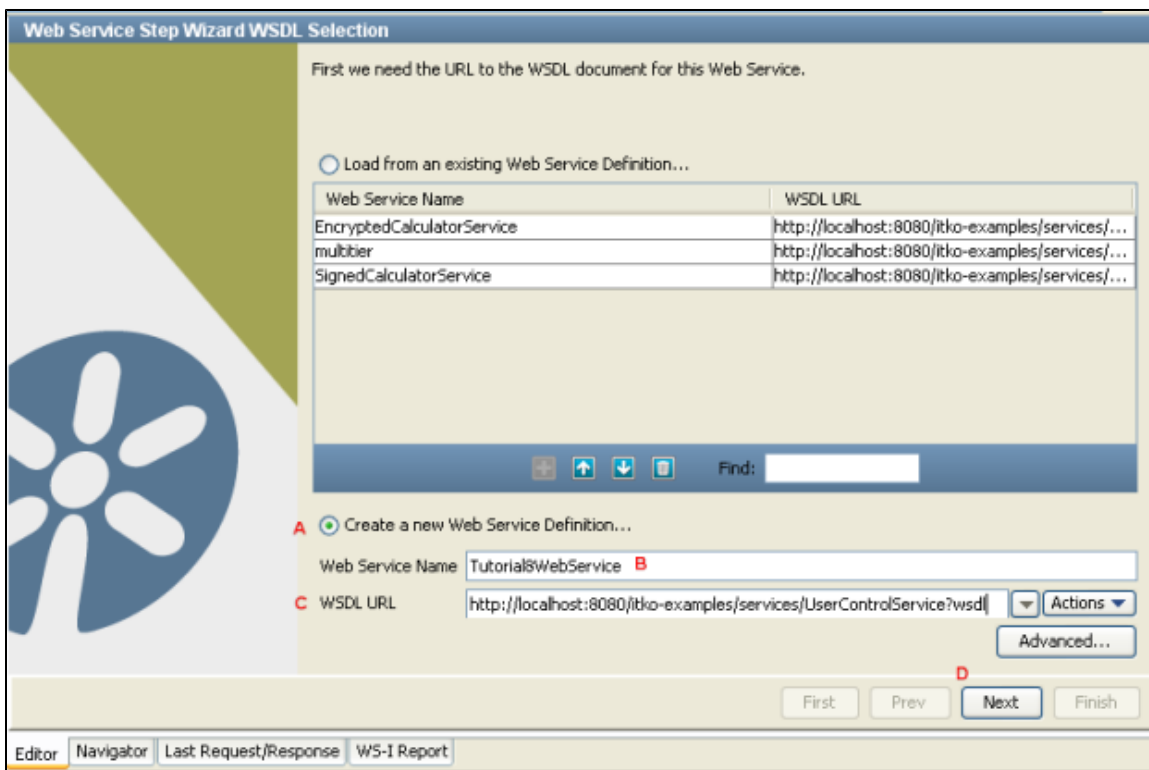


Step 4 - Create a Web Service Client

To create the Web Service Client,

1. In not already selected by default, select **Create a new Web Service Definition** (see A below).
2. In the **Web Service Name** field, enter **Tutorial8WebService** (see B below).

Note - Do not use spaces within the name of the web service.



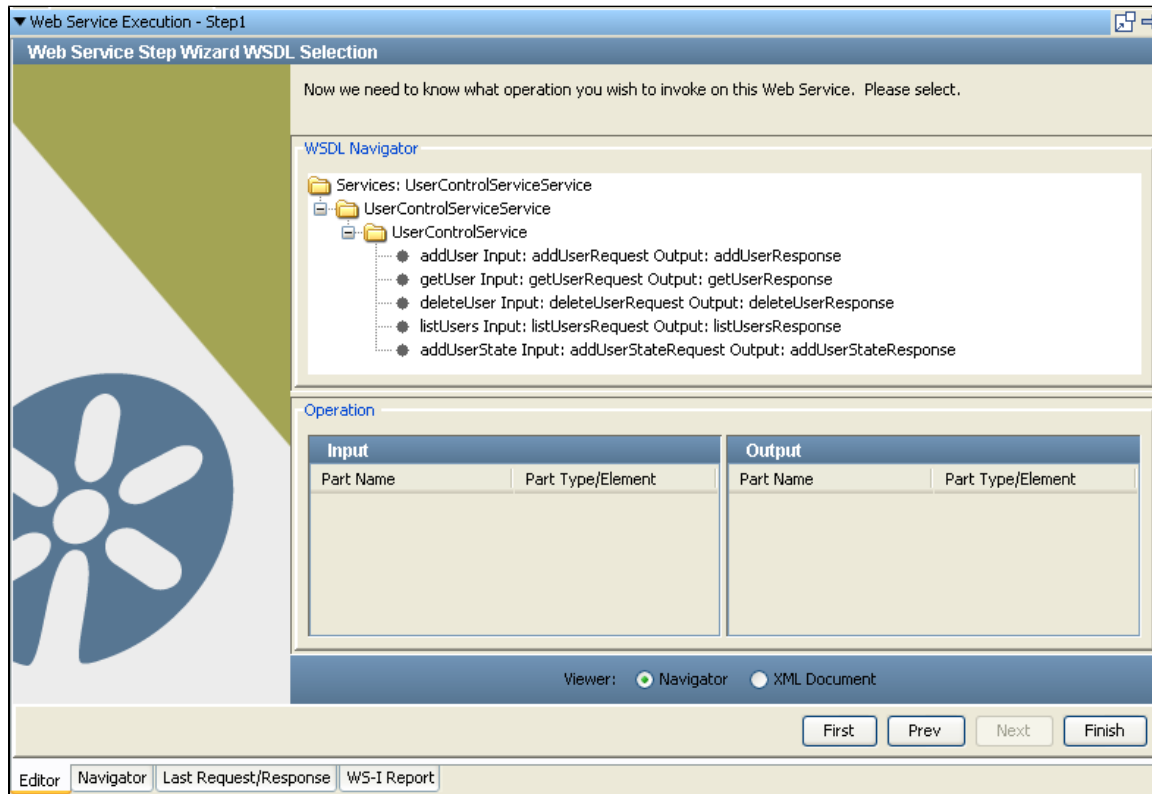
3. In the **WSDL URL** field, enter the location of the WSDL (see C above), either:

- Local---{+}http://localhost:8080/itko-examples/services/UserControlService?wsdl
- iTKO---{+}http://examples.itko.com/itko-examples/services/UserControlService?wsdl

Note - Replace localhost in the above path, by your machine IP address.

4. Click **Next** (see D above).

LISA builds the Web Service client.



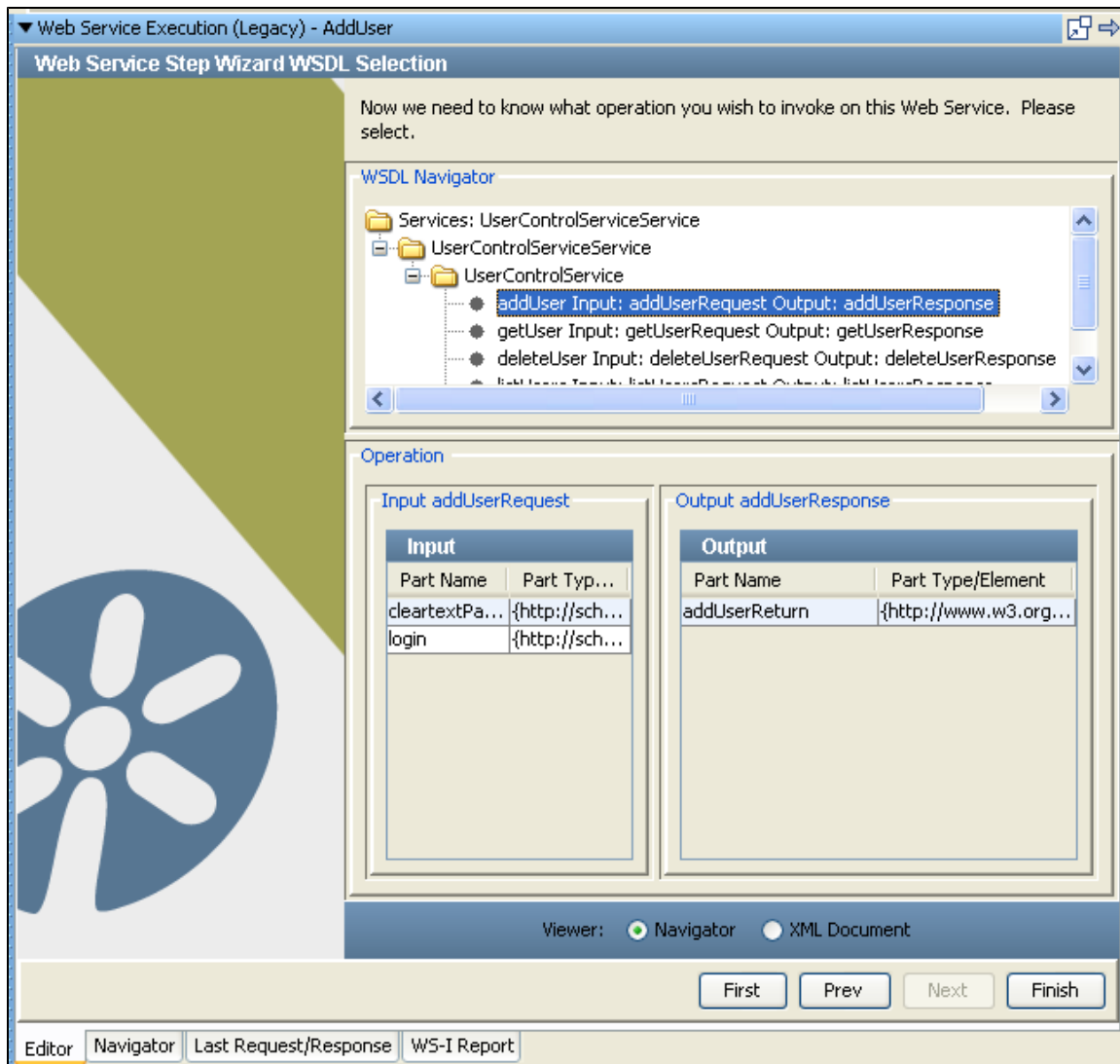
Step 5 - Test the addUser Method

To choose the operation to test,

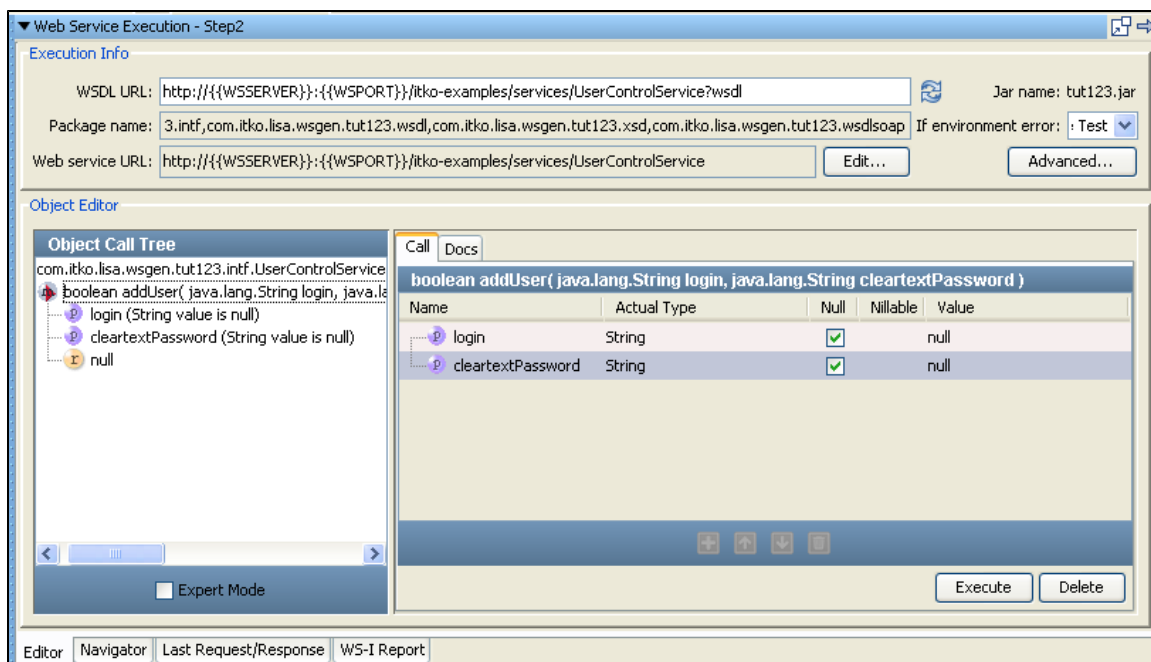
1. In the WSDL Navigator, select **addUser** from the list of operations.

The input and output parameters are shown in the **Operation** area below.

The **Input** **adduserRequest** and **Output** **adduserResponse** is listed.



2. Click **Finish** to open the **Complex Object Editor**.



Step 6 - Execute addUser Method

In this step, enter the method parameters, and add an inline assertion.

To execute the **addUser** method,

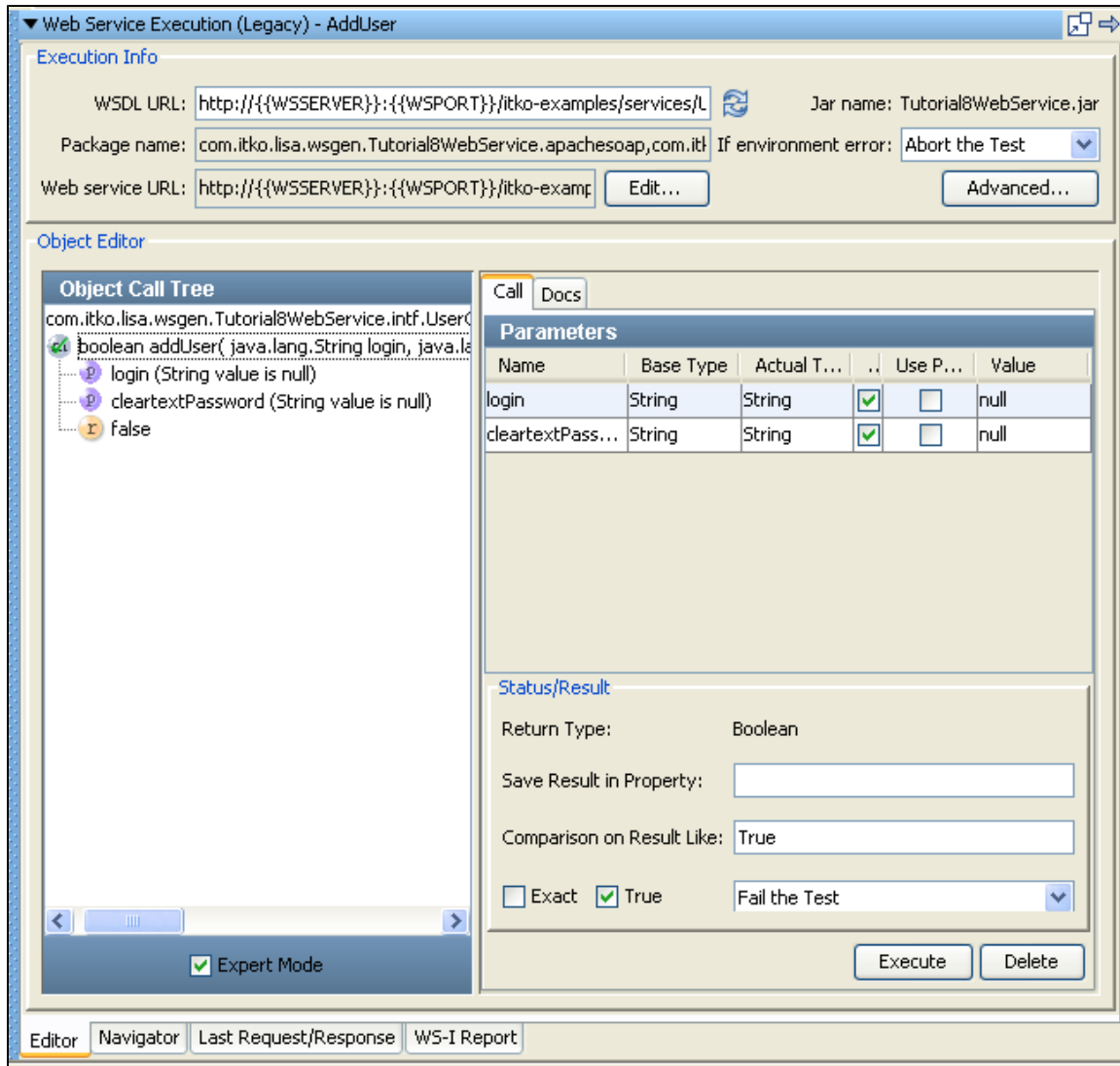
1. Enter the property values for the **addUser()** method parameters (login and cleartextPassword).

Enter **User** and **Password** that you added to the configuration (Config8).

Note: If the Status/Result portion of the window does not display, check the **Expert Mode** checkbox.

2. In the **Status/Result** area, add the inline assertion by checking the **Exact** checkbox and unchecking **True** checkbox.




3. In the **Comparison on Result NOT Exactly** field, enter **true**. (The addUser method returns a Boolean.)



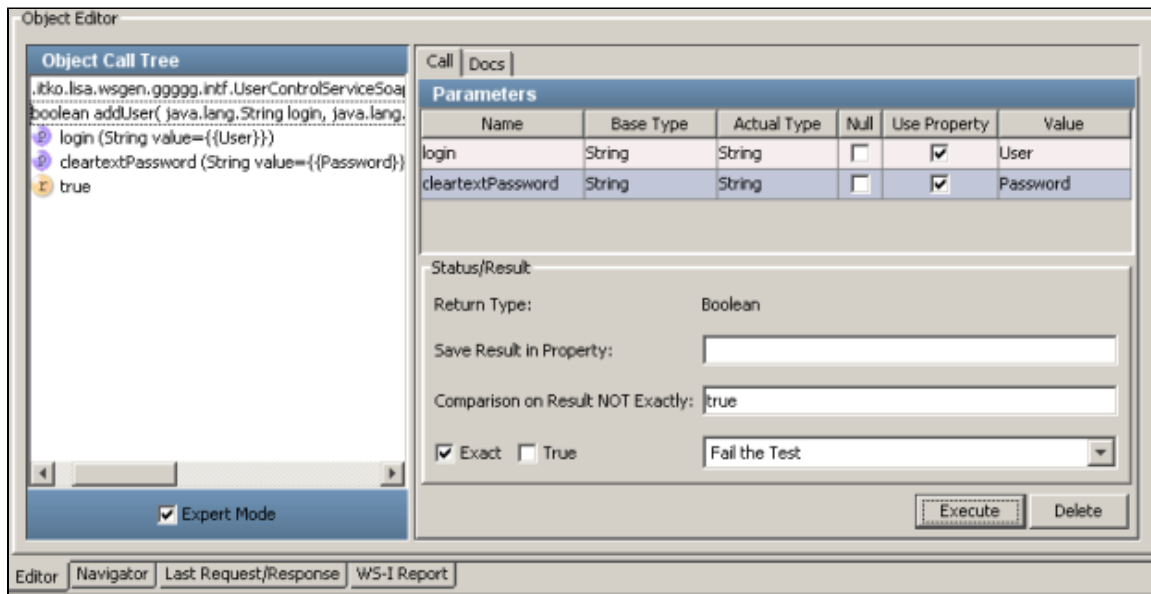
4. From the Exact list, select **Fail the Test**. (Meaning, If the addUser method returns anything but true, it executes the **fail** step).

5. Click **Execute**.

The Object Call Tree displays the full context of the method call, including:

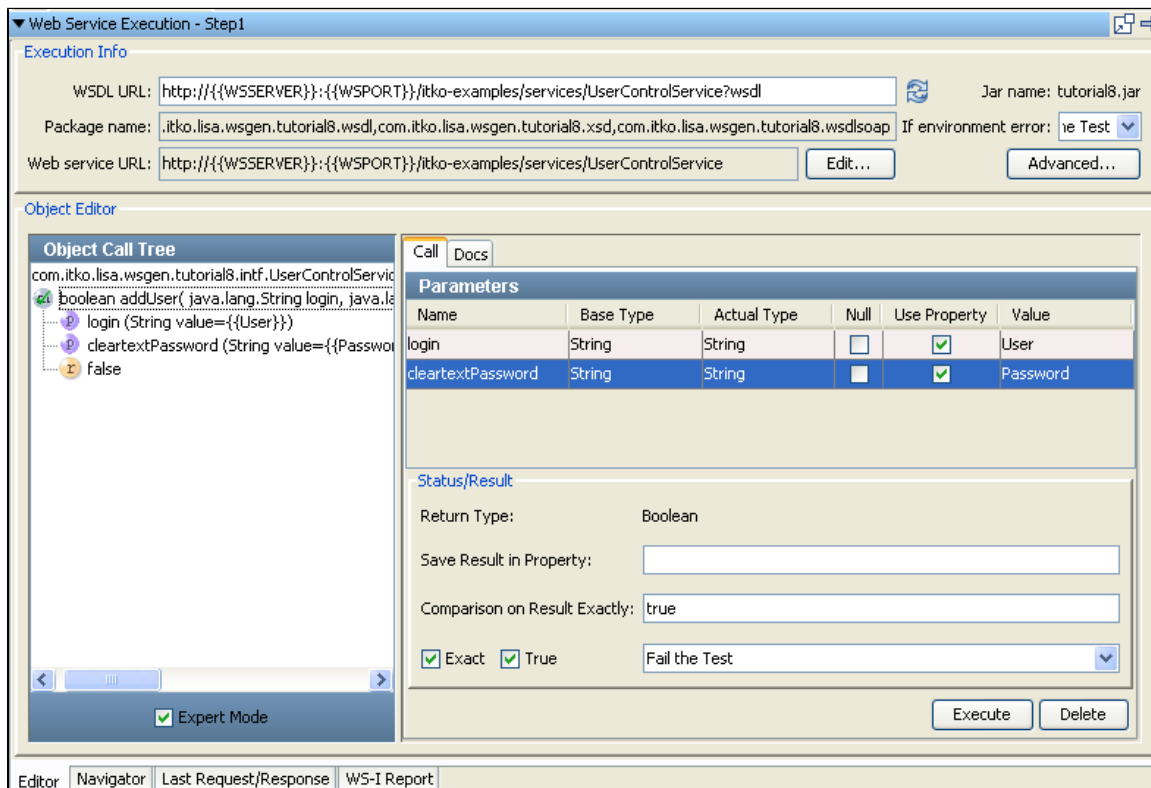
-  - the object that provided the original method call
-  - the method you wish to invoke
-  - the parameters of this method

- - the return value of this method
- - the method complete, when you have called the method



The return value of this method is **true** in the Object Call Tree.

If you execute the method again, the return changes to **false** because the user has already been added.



Step 7 - Verify the Method Execution

To verify the method executed properly,

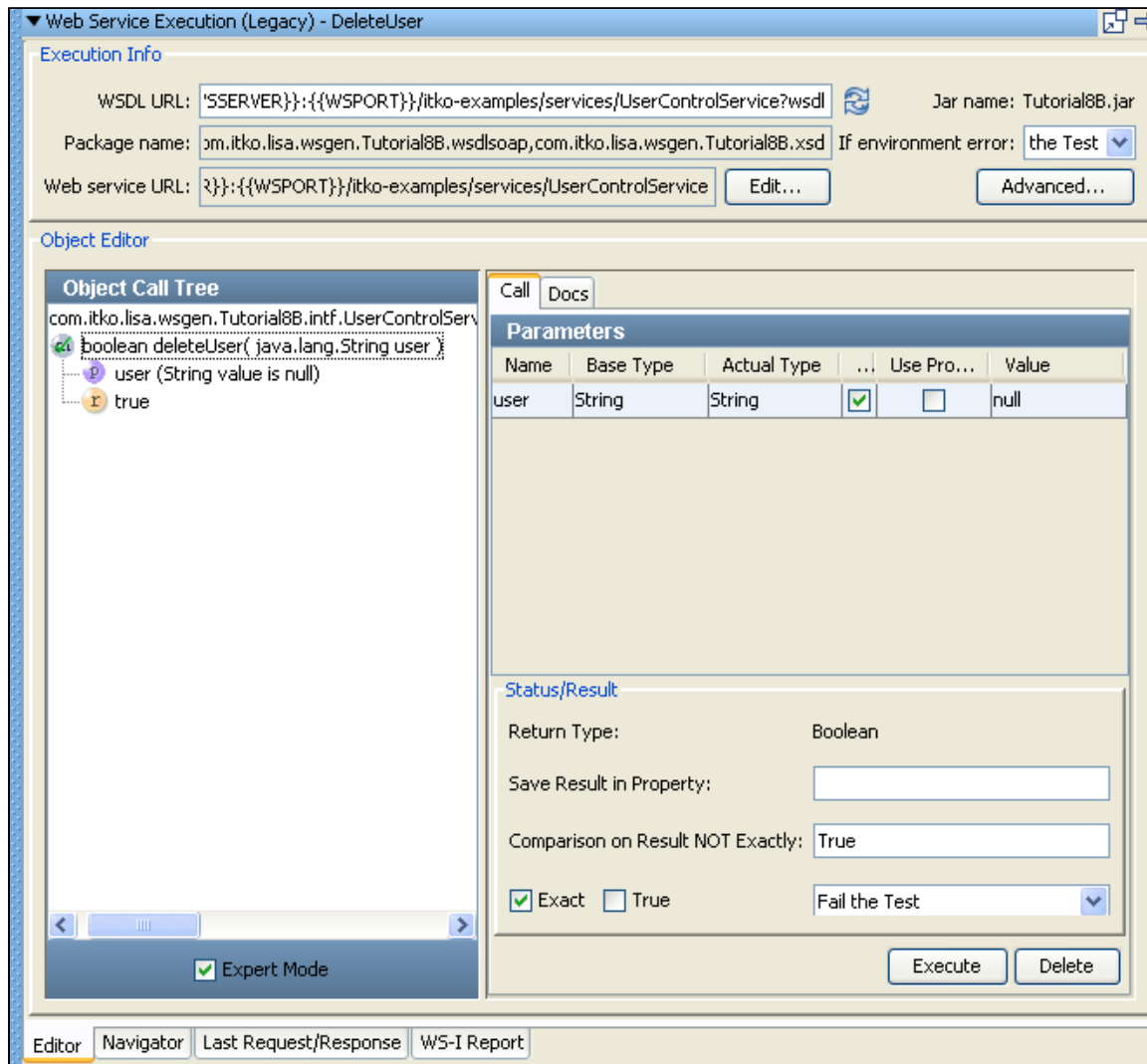
1. Go to the LISAFinancial Online site.
2. Login as user **admin** with the password **admin**.

3. Scroll through the View Users list to see if **Lisa8** is listed.

Step 8 - Add another Test Step "DeleteUser"

To add another step,



Repeat the tutorial beginning with Step 3 to add a Web Service Delete User step named **DeleteUser**. Use the method parameter property **User**.



Note - The return  is **true**, indicating the user has been deleted.

Step 9 - Save the Test Case Step 10 - Run the Test Case

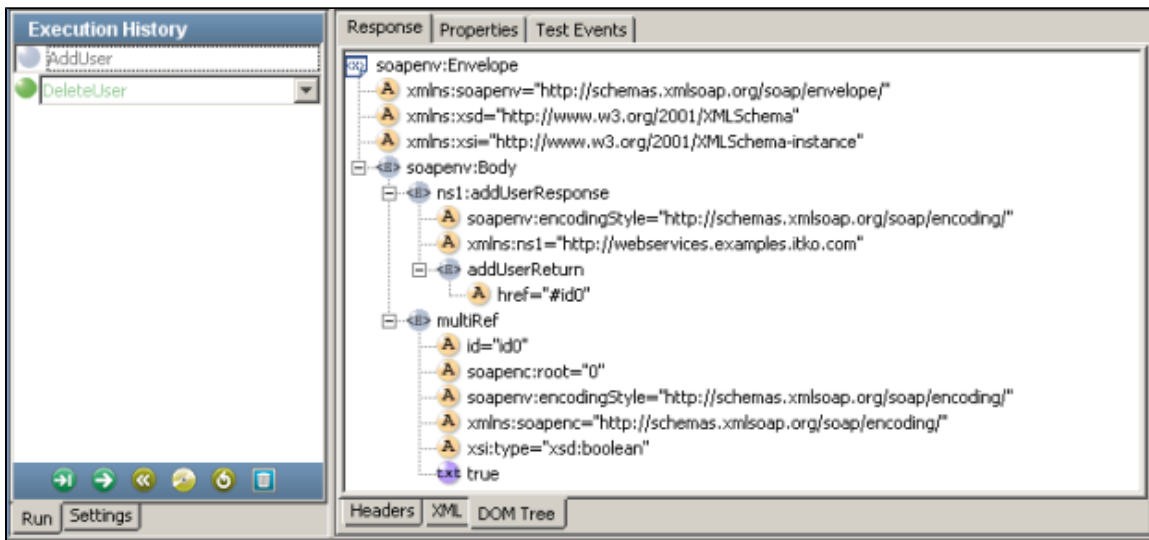
To run the test case in the ITR,

1. From the toolbar, click **Start ITR** .
2. Click **Execute Next Step** . The ITR executes and the AddUser step executes and ends normally.

The XML response is shown in the Response tab:



3. Click **DOM Tree** in the Response tab.



4. Continue the test in the ITR.

5. Click **OK** when the test is complete.

Review

In this tutorial, you did the following:

- Created a test case of two Web Service test steps. The Web Service WSDL was available as part of the users example application on the Demo Server.
- Used the Complex Object Editor to manipulate Web Service objects.
- Added inline filters and assertions to objects.

4.9 Tutorial 9 - Examining and Testing a Database

4.9 Tutorial 9 – Examining and Testing a Database

In this tutorial, you will examine and test a Database table that is part of the Web application in Tutorial 5.

Use the **SQL Database Execution (JDBC)** step to interact with a database within a test case and test the response with an assertion. Examine the **Users** table from a Derby database that is part of the application.

LISA Concepts Discussed

In this tutorial, do the following:

- Use the SQL Database Execution (JDBC) step.
- Use a configuration to store application properties.

- Edit assertions manually.
- Add filters manually.

Prerequisites

Complete [Tutorial 5](#).

Open LISA Workstation if not already open.

Make sure you have access to the demo server (either the local Demo Server, or the iTKO demo server).

Steps

Step 1 - Create a New Test Case

Create a new test case called **tutorial9**.

Step 2 - Store Properties in Configuration

Store the properties that are needed to connect to the database, in the configuration.

Note: This is a standard LISA practice that increases the portability of test cases.

1. Add the following "properties" and "values" in the project.config file.

Property Key	Value
DBDriver	org.apache.derby.jdbc.ClientDriver
DBConnect	jdbc:derby://localhost:1529/lisa-demo-server.db
DBPwd	sa
DBUserID	sa

Properties Editor		
Key	Value	Encrypt
DBDriver	org.apache.derby.jdbc.ClientDriver	<input type="checkbox"/>
DBConnect	jdbc:derby://localhost:1529/lisa-demo-server.db	<input type="checkbox"/>
DBPwd	sa	<input type="checkbox"/>
DBUserID	sa	<input type="checkbox"/>

For more information on creating a configuration, refer to [Tutorial 2](#).

Step 3 - Add a JDBC Test Step

To add a JDBC test step,



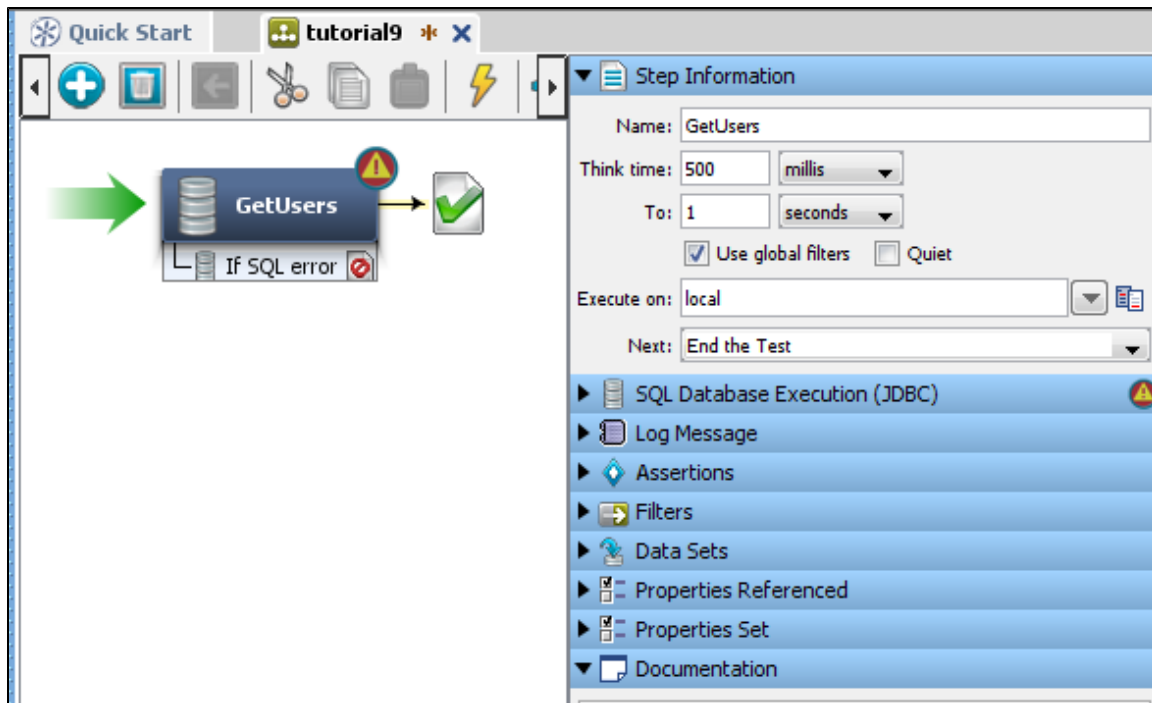
1. Right click in the model editor area or click **Add Steps** from the test case toolbar.

The sub menu of **Add Steps** is displayed.

2. Click **Other Transactions**, and select **SQL Database Execution (JDBC)** step type.

Step1 has been added to the model editor.

3. Select Step1 and re-name it as **GetUsers** in the Step Information area.



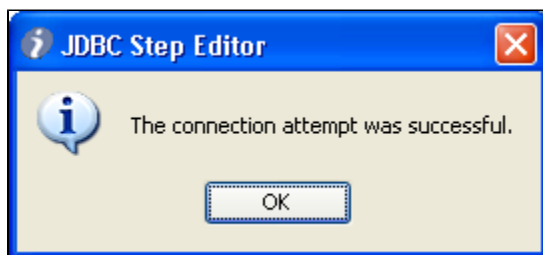
4. Double click the **GetUsers** step to open its Editor window:

▼ SQL Database Execution (JDBC) - GetUsers

<p>Connection Info</p> <p>JDBC Driver: {{DBDriver}}</p> <p>Connect String: {{DBConnect}}</p> <p>Max Rows to Fetch: -1</p>	<p>Execution Info</p> <p>User ID: {{DBUserID}}</p> <p>Password: ●●●●●●●●</p> <p><input type="checkbox"/> Keep Connection Open</p> <p><input checked="" type="checkbox"/> Use Connection Pool</p> <p><input checked="" type="checkbox"/> Returns Result Set</p> <p>If SQL error: Abort the Test</p>
--	---

2. Click **Test Connection** button at the bottom, to connect to the database.

A success message confirms a valid connection.



2. Click **OK** to close the window.

Step 5 - Execute the Query

To execute a query,

1. In the SQL Statement pane, enter the following SQL Query:

```
SELECT LNAME, LOGIN FROM Users
```

SQL Statement

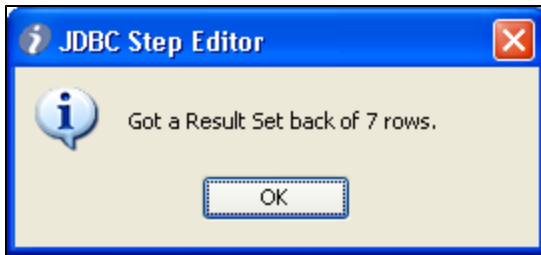
```
SELECT LNAME, LOGIN FROM Users
```

Test Connection Test/Execute SQL

Base Result Set

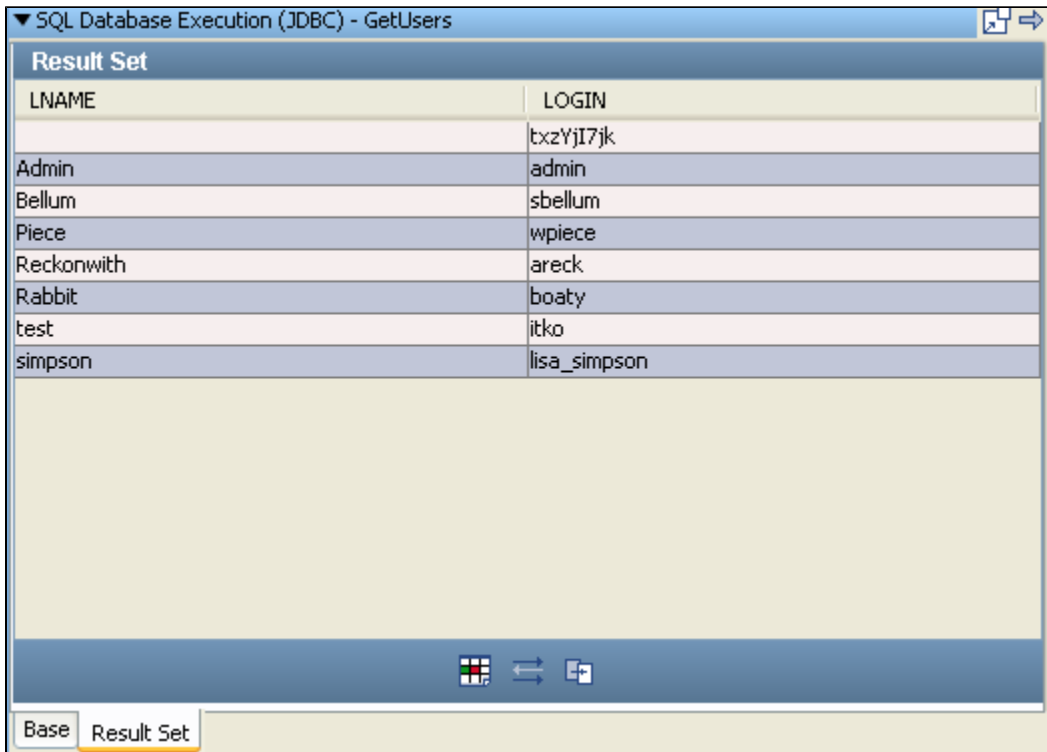
2. Click **Test/Execute SQL** button, to execute and test the query.

A success message confirms a valid query and displays the number of rows returned.



3. Click **OK**.

LISA displays the **Result Set** pane now.



You can use this Result Set to add Filters and Assertions.

Step 6 - Add an Assertion

We add an assertion to test for the presence of a last name in the **LNAME** field in the Result Set pane.

To add an Assertion,

1. In the **Result Set** tab, select a cell in the **LNAME** column. (We have selected **Bellum**).

2. Click **Generate assertion for Cell's Value**  icon.

▼ SQL Database Execution (JDBC) - JDBC

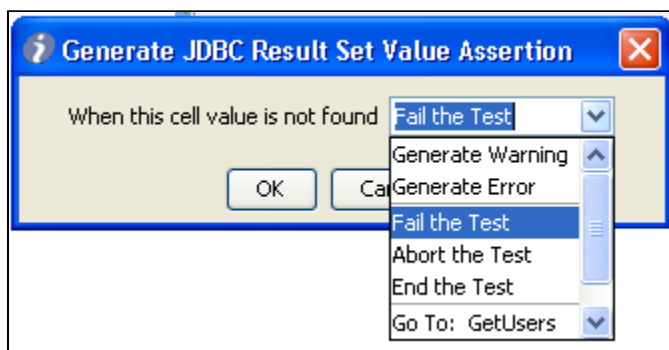
Result Set

LNAME	LOGIN
Admin	admin
Bellum	sbellum
Piece	wpiece
Reckonwith	areck
Rabbit	boaty
test	itko
simpson	lisa_simpson
	Lisa8

Base Result Set Generate Assertion For Cell's Value

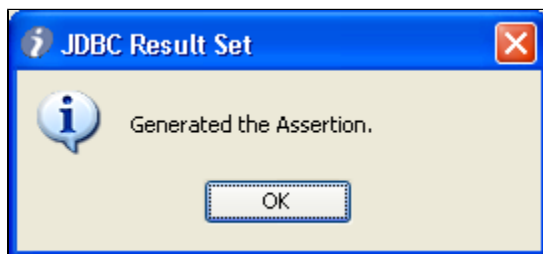
This opens the **Generate JDBC Result Set Value Assertion** dialog box.

3. From the drop down list, select the step to go to if the Assertion fires. Select **fail** for this step.



4. Click **OK**.


A message pops to show that the Assertion was generated.



Step 7 - Save the Test Case

Step 8 - Run the Test Case

To run the test case in the ITR,

1. From the toolbar, click **Start ITR**.
2. Click **Execute Next Step** . The result set is shown in the **Response** tab.

▼ Interactive Test Run - 1

Execution History

- GetUsers
- End the Test
- GetUsers

Response Properties Test Events

Result Set

LNAME	LOGIN
Admin	admin
Bellum	sbellum
Piece	wpiece
Reckonwith	areck
Rabbit	boaty
test	itko
simpson	lisa_simpson

Step 9 - Change the Assertion

We will change the Assertion to cause it to fail.

To edit the Assertion manually:

1. In the model editor, click the **JDBC** step.
2. Click the **Assertions** tab in the Element Tree.

Project Quick Start project.config tutorial9

GetUsers

If SQL error Assert86

Name: GetUsers

Think time: 500 millis

To: 1 seconds

☒ Use global filters ☐ Quiet

Execute on: local

Next: End the Test

SQL Database Execution (JDBC)

Log Message

Assertions

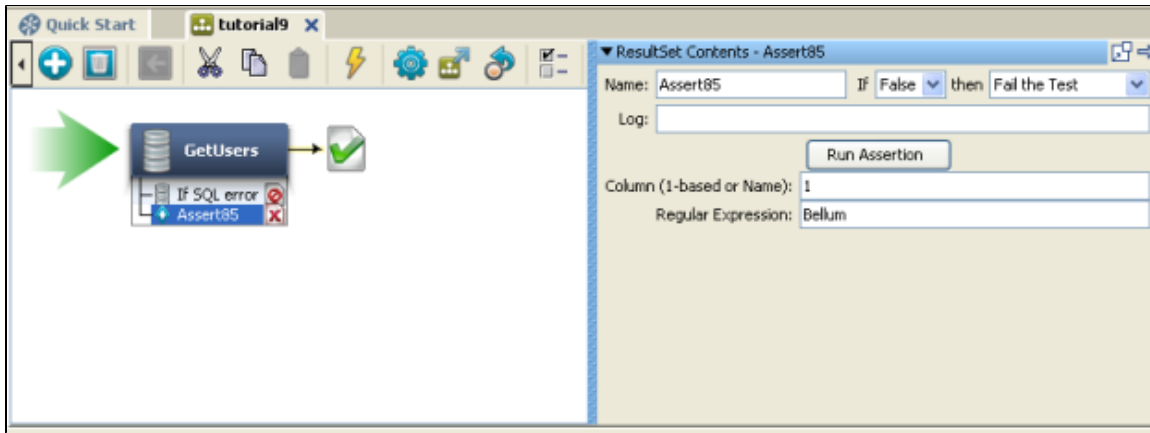
Assert86

Filters

Data Sets

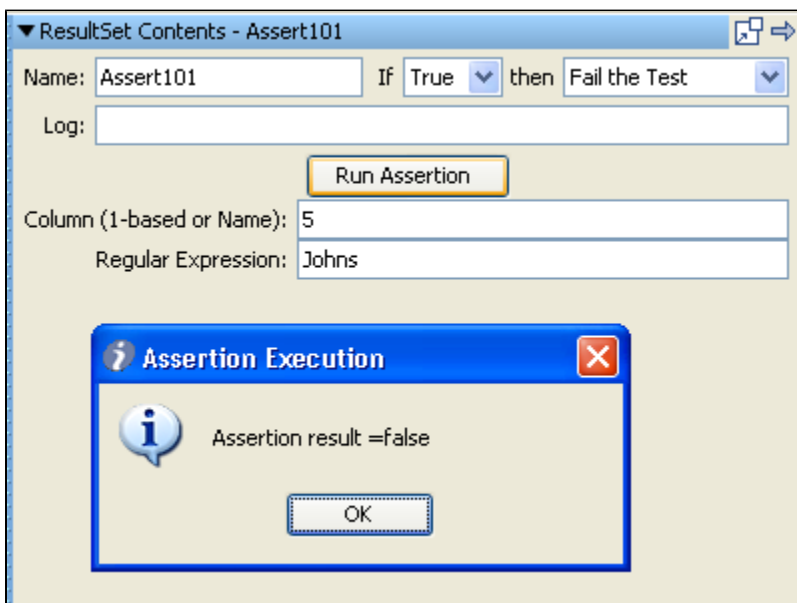
Properties Referenced

3. Click the Assertion you have created earlier. The Assertion editor is opened.



The lower section indicates that it looks for a value in column **1** of the result set that matches the regular expression **Bellum**.

4. Change **Bellum** to **Johns**.



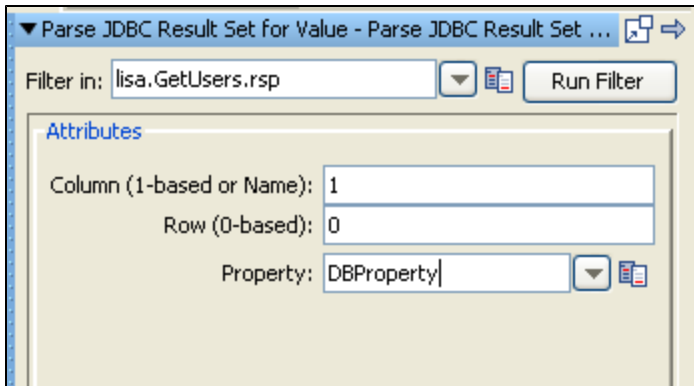
5. Run the Assertion. It will Fail.

Step 10 - Add a Filter

Add a Filter to capture the value in the first column and fourth row of the result set.

To add a Filter manually,

1. Select **JDBC** test step in the model editor, and click **Filters** in the Step Element tree.
2. Click **Add** button to add a filter.
3. Select the **Database Filters > Extract Value from JDBC Result Set Filter**.
4. In the **Column** field, enter **1**, default. (You could use **LNAME** as the column 1 heading.)
5. In the **Row** field, enter **3**. This field is zero-based.
6. In the **Property** field, enter **DBProperty** as the property to use.



Step 11 - Test the Filter and Assertion

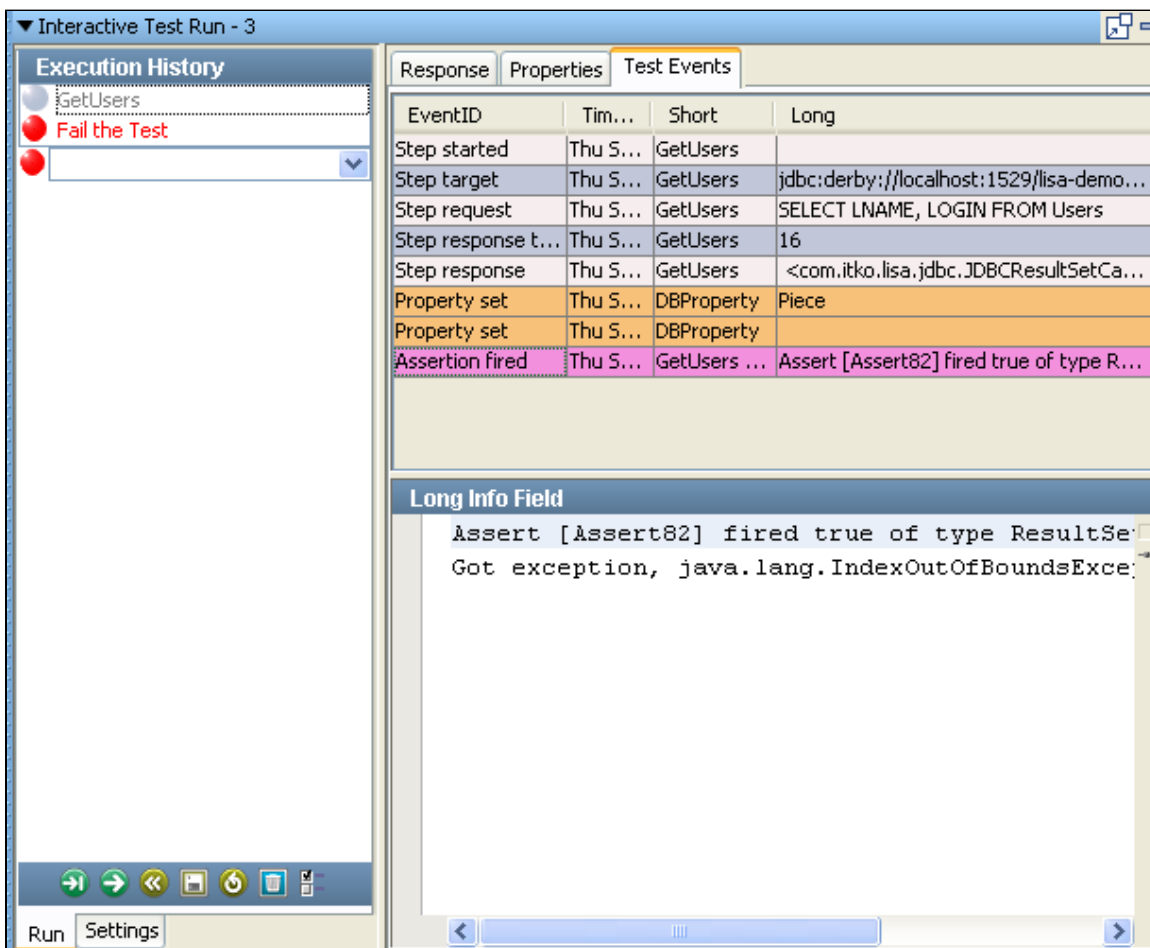
To test the Filter and Assertion,

1. Restart the test in the ITR.
The test fails because **Jones** was not found in the result set.

2. Select the **Test Events** tab to view the details.

Notice the **Property Set Event (EVENT_SETPROP)** emitted for **DBProperty**.

3. Click **Assertion fired Event (EVENT_ASSERT)** to view the log information for the failed assertion in the Long Info Filed below:



4. Click the **Properties** tab to see the **DBProperty** value listed (Bellum).

▼ Interactive Test Run - 10

Execution History

- GetUsers
- Fail the Test

Response Properties Test Events

Initial property values may be changed prior to executing the step. They are read-only after execution. Create a new property by editing an existing key.

Key ▼	Value	Previous Value
DBConnect	jdbc:derby://localh...	jdbc:derby://localh...
DBDriver	org.apache.derby.j...	org.apache.derby.j...
DBProperty	Bellum	
DBPwd	sa	sa
DBUserID	sa	sa
EJBPORT	1099	1099
EJBSERVER	localhost	localhost
ENDPOINT	http://localhost:808...	http://localhost:80...
ENDPOINT1	http://192.168.40....	http://192.168.40....
LASTRESPONSE	<list> <list> <nu...	
LISA_DOC_PATH	E:\Lisa.Workstation...	E:\Lisa.Workstation...
LISA_DOC_URL	file:/E:/Lisa.Workst...	file:/E:/Lisa.Workst...
LISA_HOST	algiers	algiers
LISA_LAST_STEP	GetUsers	
LISA_PROJ_NAME	My Tutorials	My Tutorials
LISA_PROJ_ROOT	E:/Lisa.Workstation...	E:/Lisa.Workstation...
LISA_TC_PATH	E:\Lisa.Workstation...	E:\Lisa.Workstation...
LISA_TC_URL	file:/E:/Lisa.Workst...	file:/E:/Lisa.Workst...
LISA_USER	gourik	gourik
PORT	8080	8080
Password	Pass7	Pass7
SERVER	192.168.40.171	192.168.40.171
User	Lisa7	Lisa7
WSPORT	8080	8080
WSSERVER	localhost	localhost
WSSERVER1	192.168.40.171	192.168.40.171
config_prop	42	42
instance	0	0
lisa.GetUsers.rsp	<list> <list> <nu...	

Run Settings

Step 12 - Save the Test Case

Review

In this tutorial, you created a test step to query a database. You used the **Users** table from the Derby database that accompanies the applications on the Demo Server. You learned how to:

- Connect to the database of interest.
- Execute an SQL query against the database.
- Manually edit assertions.
- Manually add filters.

4.10 Tutorial 10 - Staging a Quick Test

4.10 Tutorial 10 – Staging a Quick Test

In this tutorial, you will use the quick staging option (Quick Test), to learn how to stage tests and read subsequent reports.

Run a Quick Test on the **multi-tier-combo** example that accompanies LISA. This is the simplest way to stage a test in LISA.

LISA Concepts Discussed

In this tutorial, do following:

- Use the multi-tier-combo test case.
- Use the Quick Test facility.
- Select and format reports.

Prerequisites

Complete Tutorials 5-9.

Open LISA Workstation if not already open and start the demo server (You can use either the local Demo Server, or the iTKO demo server).

Steps

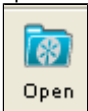
Step 1 - Open a Test Case

We will open a example test case from LISA project.

To open the test case,

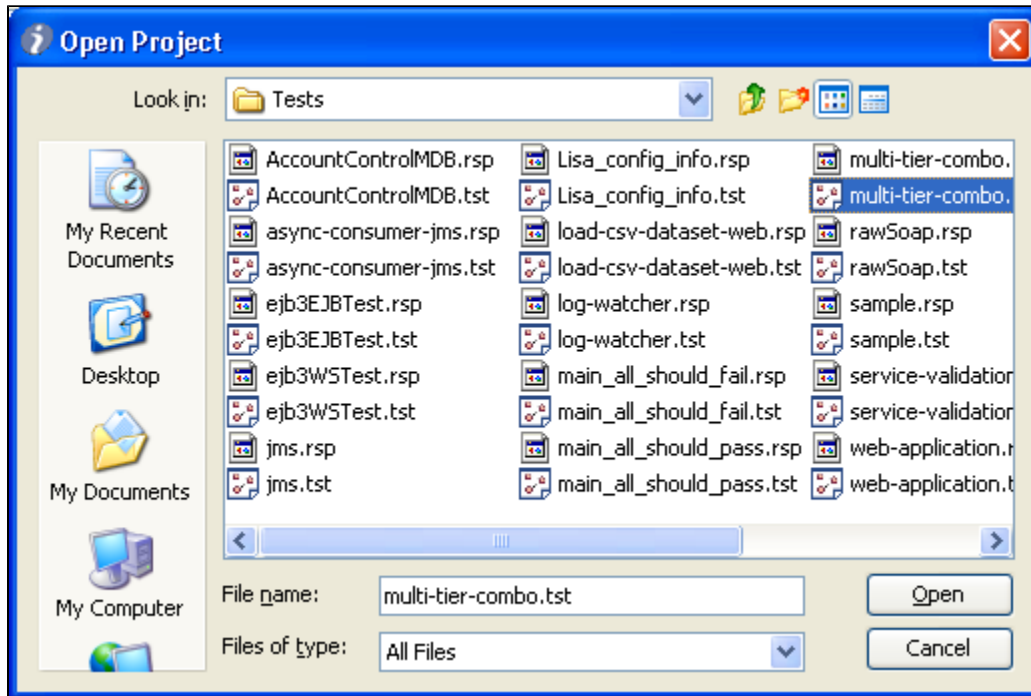
1. From the main menu, select **File > Open**.

This will open a Project directory, from where you can choose the respective test case.



Or Click **Open > Project > ..and navigate to the %LISA_HOME%/examples/Tests** folder.

2. Within the **Tests** folder, select **multi-tier-combo.tst** and click **Open**.



The multi-tier-combo test case will open in the model editor.

Step 2 - Activating a Configuration

We will make the **localhost** configuration as Active. This configuration allows the local Demo Server to be used in the tutorial.

To Make the Configuration Active,

1. Select the Configuration and right click
2. Select **Make Active**

This will make the selected Configuration Active.

Step 3 - Observing the Test Case

Now as we have opened the multi-tier-combo test case,

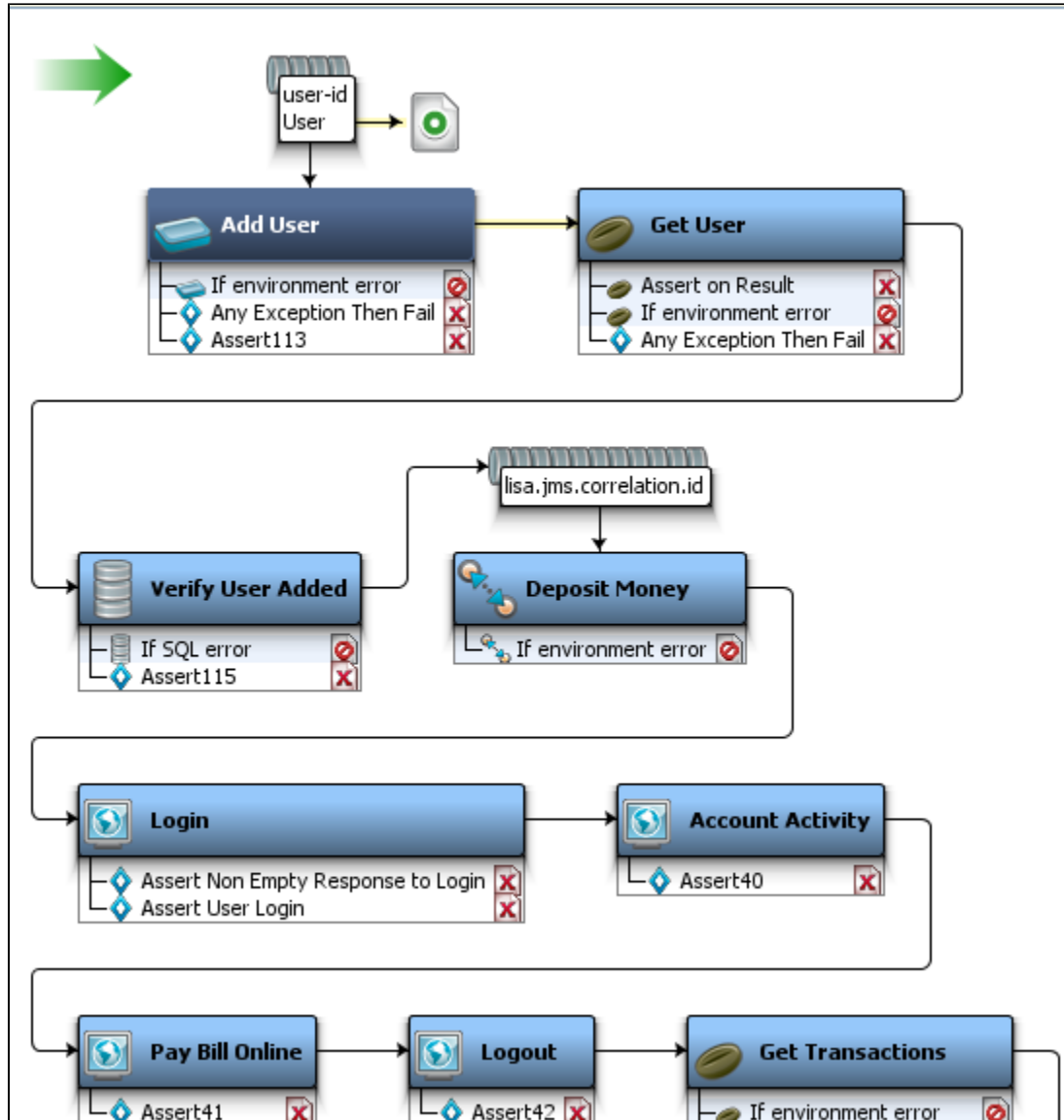
Take a few minutes to look at the multiple test step types used in this test case. All kinds of test steps have been used in this example. To name a few -

Add User - Web Execution step

Get User - Enterprise JavaBean Execution step

Verify User Added - SQL Database Execution step

Deposit Money - JMS Messaging step



You have used each of these steps individually in tutorials 6 through 9.

In this tutorial, we use all the test steps to build a more realistic test case involving several layers of the application.

[Part A Running the Quick Test](#)

[Part B Viewing the Generated Reports](#)

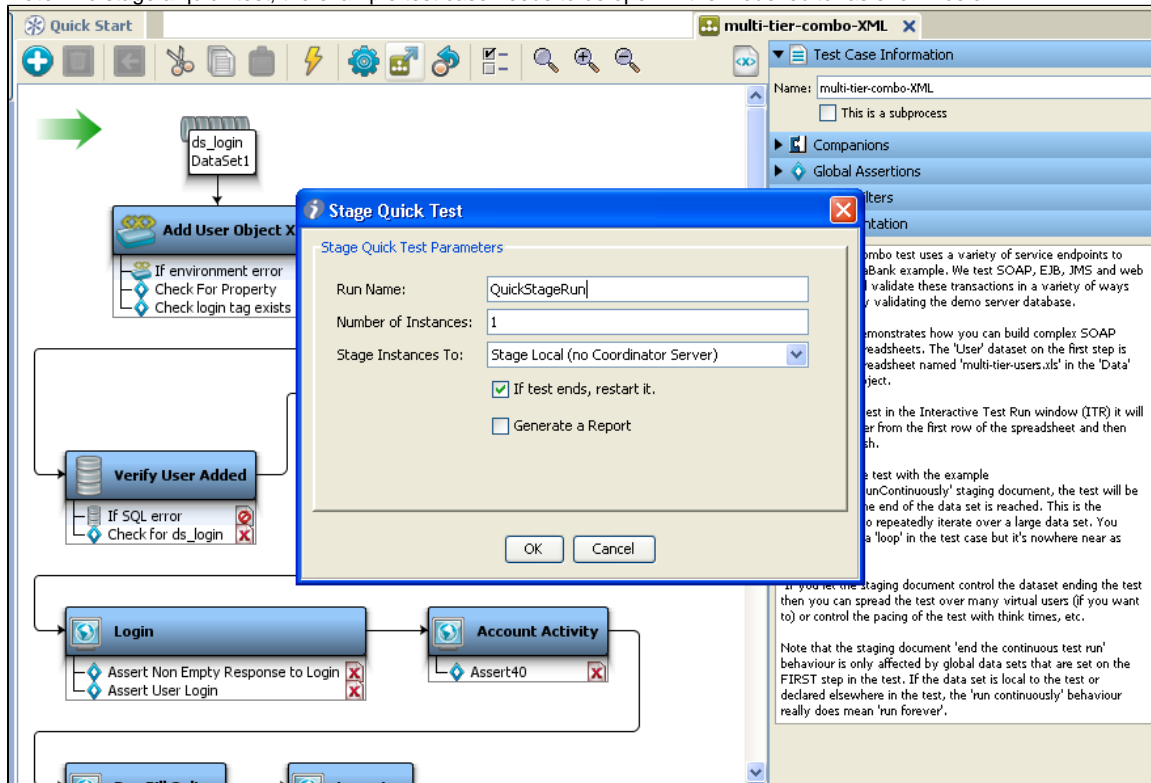
Part A Running a Quick Test

To run or stage a Quick Test,

1. From the menu bar, select **Actions > Stage a Quick Test**.
Or click the **Stage a quick test** icon on the test case toolbar.



Note - To stage a quick test, the example test case needs to be open in the model editor as shown below:



2. In the Stage Quick Test window, complete all the required information as follows:

Run Name: Enter a unique name (**tutorial10QuickTest**).

Number of Instances: Enter a number of users that you want to run the test concurrently, (**4**).

Stage Instances To: Select the name of the Coordinator Server or stage it locally.

If test ends, restart it - Check this option, to restart the test.

Generate a Report - Check this option, to generate a report.

Stage Quick Test

Stage Quick Test Parameters

Run Name:

Number of Instances:

Stage Instances To:

☒ If test ends, restart it.

☒ Generate a Report

OK Cancel

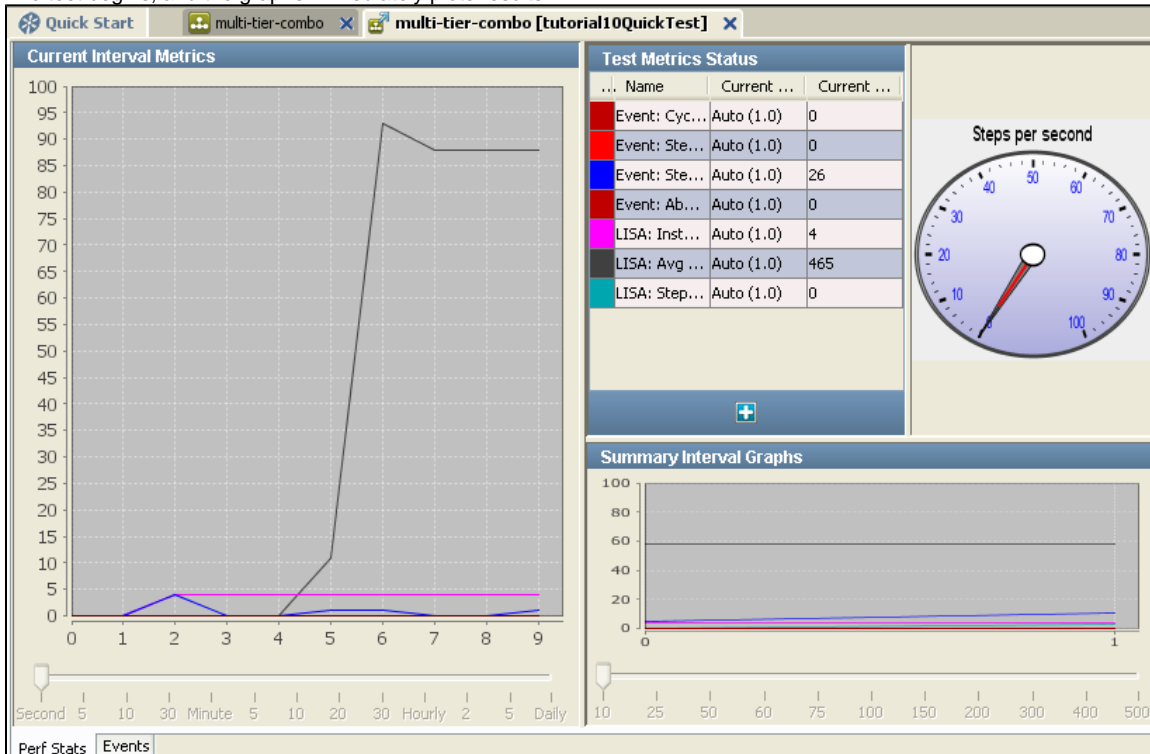
3. Click **OK**.

4. In the Running Test Dashboard window, choose whether to show the message again, and click **OK**. The Test Monitor opens, but the test has not started yet.

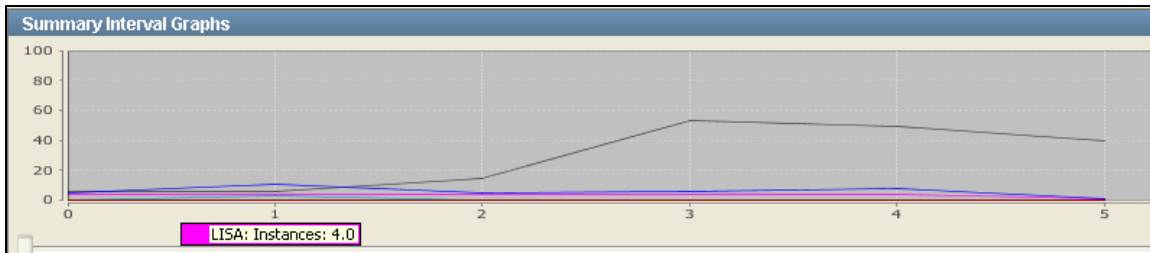


5. From the main toolbar, click the **Play** button to start the test running.

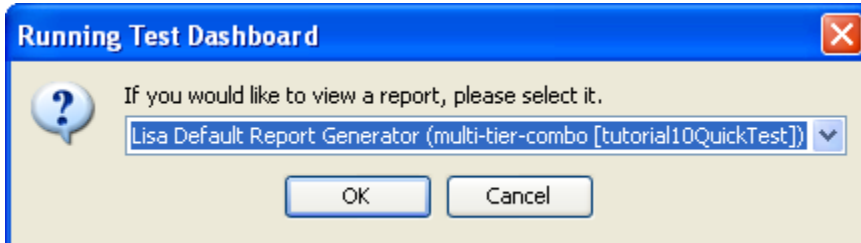
The test begins, and the graphs immediately plots results.



6. You can roll over the graph lines to view descriptions.



When the test is finished, you are asked whether to generate a report.



7. Click **OK** to generate a report.

Part B Viewing the Generated Reports

LISA provides a Report viewer to view reports.

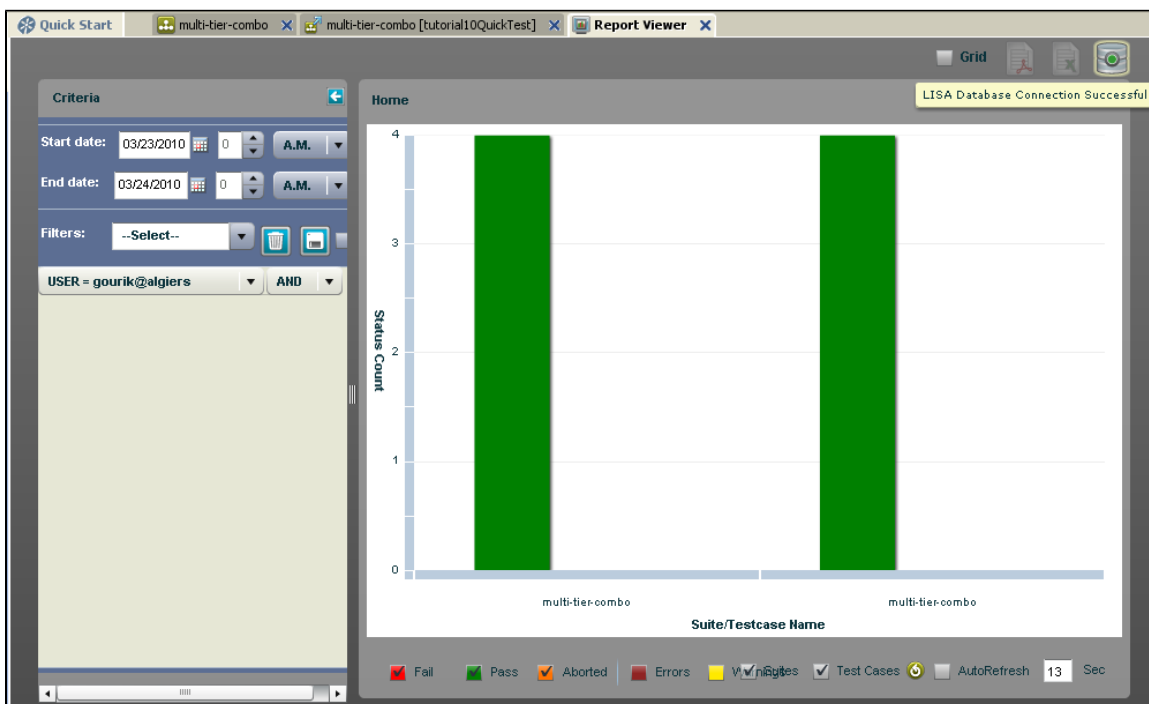
To view a report,

1. From the main menu, click **View > Reporting Console**



or click the **Reports** icon on the toolbar.

The Report Viewer opens -



Set the dates criteria to open the graphs plotted within those dates.

You can select the criteria for test cases to be plotted also.

Here, the graph shows that all the tests in this test case have passed.

You can right click the graph for different menus.

More information on the same is obtained in the Reports section of the LISA User Guide.

Review

In this tutorial, you tested the multi-tier-combo example using a Quick Test. You learned how to:

- Look at a test case containing several different test step types.
- Configure and run a test in the Quick Test utility.
- Create and chose an output format for a Performance Summary report.
- Examine a report generated from the test run.

PART 2 - Building Test Cases

PART 2 - Building Test Cases

Part 2 is an important part of LISA documentation, as it focuses on detailed information around building test cases in LISA.

To build LISA test cases, we need to know about setting Properties within LISA, Using Project Configurations and applying them to your project, Applying Filters, Adding Assertions, Companions and Adding Data Sets if required.

This section will also give an introduction to the LISA Complex Object Editor, which is used while working with complex objects.

In this section, the following topics are covered:

- 5. Using Properties
- 6. Using Configurations
- 7. Adding Filters
- 8. Adding Assertions
- 9. Using Data Sets
- 10. Using Companions
- 11. Using Complex Object Editor (COE)

5. Using Properties

5. Using Properties

LISA Test Properties are name – value pairs, alternatively known as key – value pairs.

The key to making your LISA Test Cases data independent, reusable, and portable is the ability to abstract specific data values out of the Test Case and replacing them with variables. In LISA, these variables are called "properties". Some properties are predefined and guide how LISA operates, others are created by you while you are building your tests.

Properties are both ubiquitous, and indispensable in LISA Test Cases, and a sound understanding of properties is vital to the creation of Test Cases in LISA. In the context of a Test Case, anytime there is something that can change, it is appropriate to use a property. This includes **values in test steps**, and **values in configurations** for example.

Properties can be defined in several ways, and once defined they are available to any subsequent steps, Assertions and Filters in the Test Case (they are global to the Test Case). Properties can, with a very few exceptions, be overridden in a Test Case.

Whenever a property value is set, LISA emits an EVENT_SETPROP event containing the property name and value. Property values are NOT limited to string values.

A property can hold strings, numbers, XML fragments, serialized Java objects, or the complete response from a test step! LISA creates many properties during a test run that are available to the subsequent test steps. For instance, the property 'lisa.stepname.rsp' contains the response for the step called 'stepname'.

Note: In all future release of LISA, "." (periods), will not be allowed in property keys, so you should avoid doing so.

In this section, the following topics are covered:

- 5.1 Specifying a Property
- 5.2 Property Expressions
- 5.3 String Patterns
- 5.4 LISA Property Sources
- 5.5 LISA Property File (lisa.property)
- 5.6 Common LISA Properties and Environment Variables

5.1 Specifying a Property

5.1 Specifying a Property

The syntax for a property in LISA is `property_name`.

When a property is identified, and about to be used, `property_name` is replaced by its current value. In LISA, there are times when a property is expected, and is the only choice. Other times, you are asked for the property name explicitly. In these cases, you would enter the property name without the braces. When properties are embedded in a text string, you must use the brace notation. There is an additional syntax that allows property expressions to be used: `= expression` or `{{property_name=expression}}`.

Property expressions are described in the next section.

A property name is very flexible, and can contain spaces. However; the use of spaces is not encouraged. The characters that define the property syntax (`{}`, `}`, and `=`) cannot be used.

Note: iTKO reserves all the property names that start with 'lisa.' for internal use. LISA may hide or delete properties that start with 'lisa'.

If you reference a property that does not exist, LISA will leave the property in the braces to indicate that the property was not found, or it is invalid.

5.2 Property Expressions

5.2 Property Expressions

LISA properties can store many different types of data. They can also evaluate and store expressions. These expressions can contain any valid Java or JavaScript expressions that can be evaluated by BeanShell. BeanShell is a Java interpreter environment. Further, these expressions could be string patterns, which give real-looking fake strings, appropriate for most given purposes.

For more information on BeanShell, see [Beanshell in LISA Advanced Features](#) or go to: www.beanshell.org.

To use a property expression, you use the syntax `=expression` or `key=expression`. In the first case, `expression`, LISA will use BeanShell to evaluate the expression and replace `expression` by the result of the evaluation. For example, `=Math.random()` will evaluate the static Java method and replace the `}}` construct with the random number that was returned. In the latter case, using `{{rand=Math.random ()` will set a property `rand` equal to the random number that was returned, in addition to replacing the `{{}}` construct with the random number.

Existing properties can be used in a property expression. They are referenced by name only, i.e. without the braces, since LISA knows they must be properties. If the property is not found, LISA will return the property expression within braces, to indicate that there is a problem in the expression.

5.3 String Patterns

5.3 String Patterns

String patterns are special types of property expressions which have a syntax `[:patternname:]`. Thus, to get a real looking first name, the string pattern property could be `[:First Name:]`. It would evaluate to a fake first name which looks like a real life name.

It is much better than the possibility of dealing with random strings which do not look like a real life name. Of course, the string patterns functionality supports a lot of patterns apart from just the first names: last names, dates, SSN, credit card numbers, credit card expiry date and many more. This fake data comes with LISA, in TESTDATA table in its **reportdb** database.

Basically the idea is that if you need a **FirstName** in your test case, you would use `[:First Name:]` in a data set. The `"{["` part is a signal to use the String pattern and it has a bunch of things 'registered' that it knows about.

As an example, if you put the following information in a log step:

```
=[:First Name:] =[:Middle Initial:] =[:Last Name:]
```

```
=[:Street Address:]  
=[:City:],=[:State Code:]  
=[:Zip Code:] =[:Country:]  
SSN: =[:SSN: DDD-DD-DDDD]  
Card: =[:Credit Card:] Expires =[:CC Expiry:]  
Phone: =[:Telephone:]  
Email: =[:Email:]
```

You get the Response as shown below:

Marilyn Mcguire
3071 Bailey Drive
Oelwein, IA
50662 US
SSN: 483-16-8190
Card: 4716-2361-6304-6128 Expires 3/2009
Phone: 319-283-0064
Email: Marilyn.C.Mcguire@spambob.com

If you run the step again you will get a different set of data. Basically it will keep track of the number of rows in the test data db and randomly select a row between 1 and N.

- Select **Help > Property Window > Patterns** to open the following screen, which shows all existing string patterns and shows the documentation:

Property Window

Properties

Patterns

These patterns generate Strings based on the following definitions:

Pattern-based

D - digit (0-9)

H - hex digit (0-9, A-F)

h - hex digit (0-9, a-f)

X - hex digit (0-9, A-F, a-f)

L - letters (A-Z)

l - letters (a-z)

A - alphanumeric

P - punctuation

. - (dot) anything printable

[1,2,3] - randomly choose among the options shown

{0,9,8} - do not allow these on the previous spec

**** - take the following char as a literal

^(N[-M]) - repeat the pattern N times, or randomly N-M times if M is present

Anything that is not a pattern char IS a literal, for example Jo'h'nDDD would be "John123" or the like.

Built-in String Generator Patterns

Name	Pattern	Category
SSN		TestData
Credit Card		TestData
CC Expiry		TestData
CC Verification Code		TestData
First Name		TestData
Middle Initial		TestData
Last Name		TestData
Street Address		TestData
City		TestData
State Code		TestData
Zip Code		TestData
Country		TestData
Email		TestData
Telephone		TestData
Birthday		TestData

Find:

Custom String Generator Patterns

Find:

Sample:

Exp:

Add

Close

Implementation Information

The data is stored by default in the reports db in the TESTDATA table.

When LISA Workstation is started, it checks to see if there is any data there and if there is no data, then com/itko/lisa/test/data/TestData.csv inside lisa-core.jar is read to load up the database. If reports.db gets deleted for some reason, the test data will be recreated.

Note - It's a one-time cost and takes about 15 seconds on your laptop.

Creating your own String Pattern

If you want to put your own data in, the only thing to be **careful** about is to assign the **ID** correctly, start with 1 and increment with no gaps until you get to your number of rows.

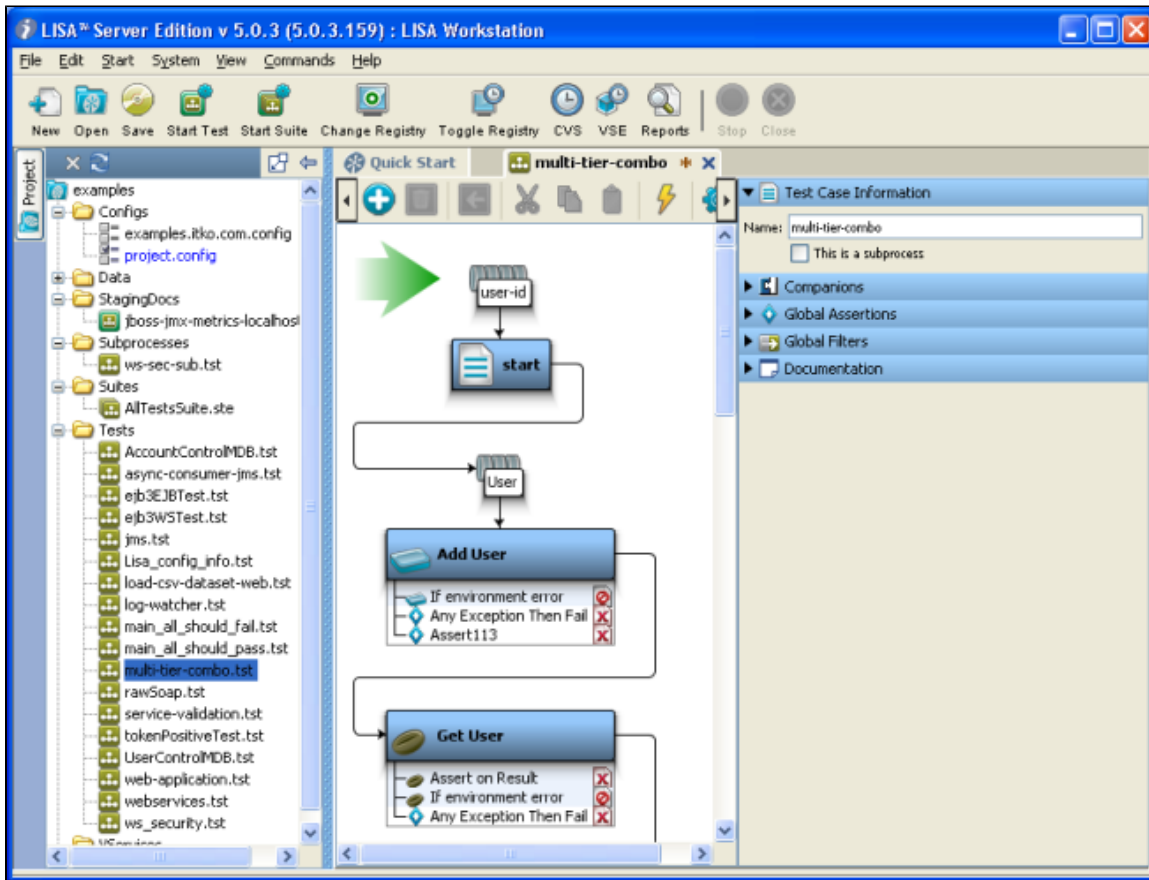
The String Generator code essentially does 'select * from testdata where id = 'n' after getting **n** from a Random object. So, ID must be the primary key of the table to ensure efficient lookups.

Example

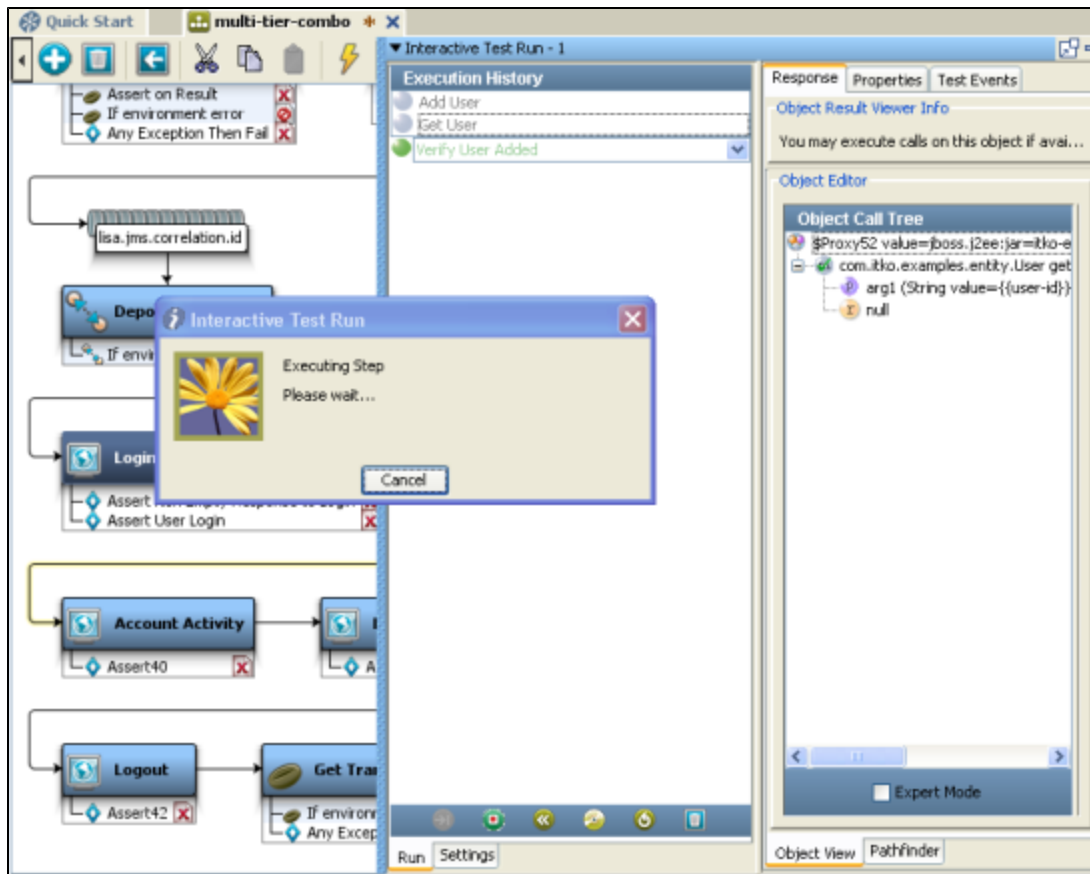
A good way to get some practice with property expressions is to build a simple Test Case that has a single step; an **Output Log Message**. Since this step just writes to a log, and displays its response in the **Interactive Test Run (ITR) Response Panel**, you can experiment using property expressions.

For the purpose of illustration, we will look at the **multi-tier-combo** Test Case in the examples directory (multi-tier-combo.tst).

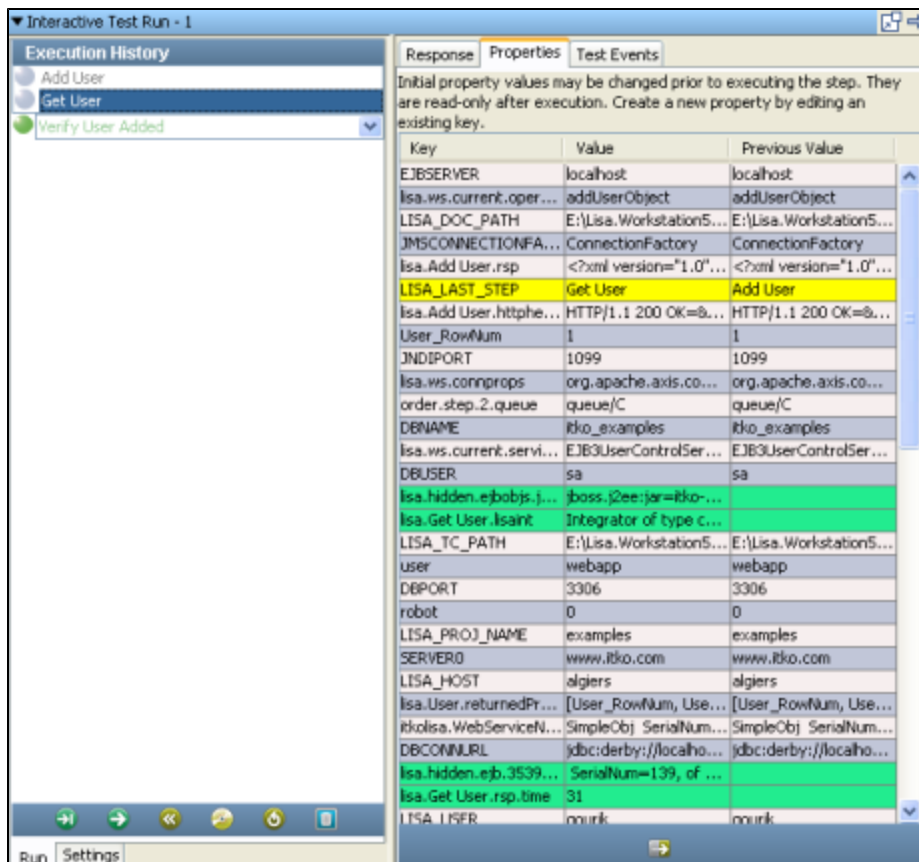
The figure below shows our example in the LISA Model Editor:



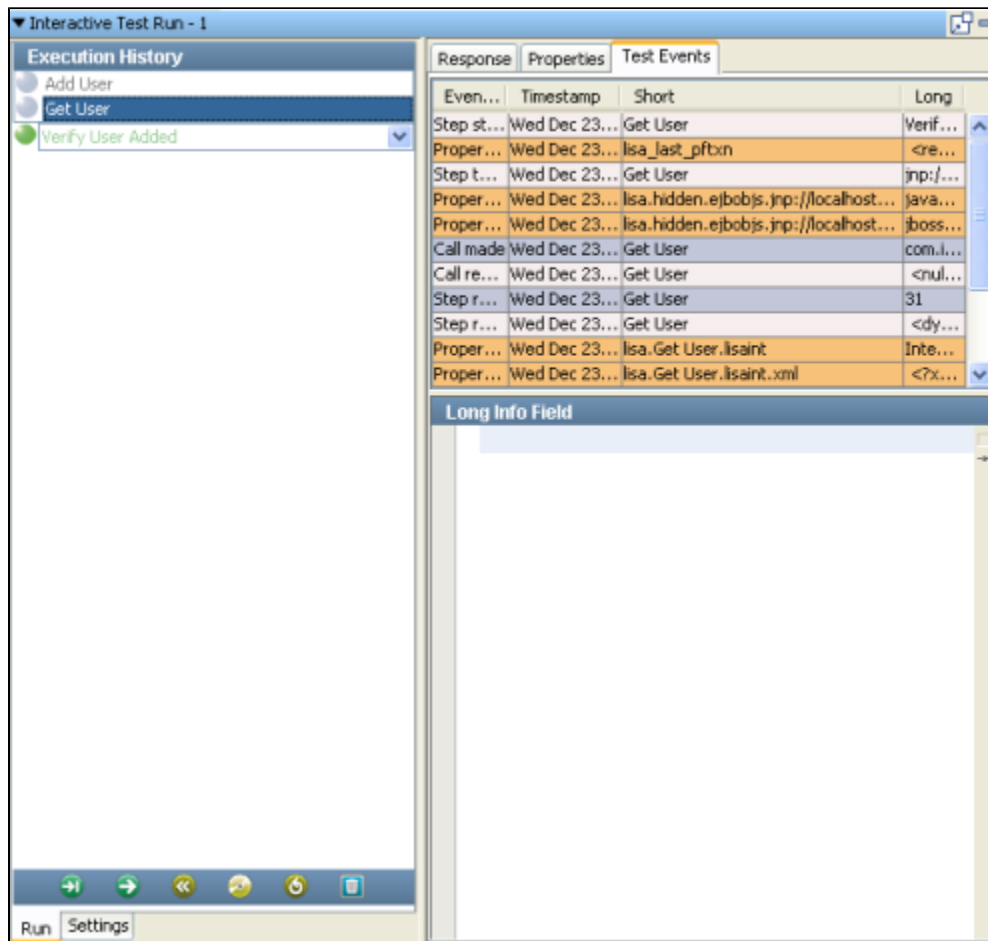
The figure below shows our example in the LISA ITR utility:



The figures below show the **Properties** tab in the ITR. This is for the step "Get user".



The figure below shows the **Test Events** tabs in the ITR:



Look for the Event - **Properties set** in the Test Events tab.

Java developers can also take advantage of the 'BeanShell' environment in the Java Script step to test property expression.

5.4 LISA Property Sources

5.4 LISA Property Sources

Properties can originate from several sources which include:

- LISA itself
- Environmental variables
- Command line variables on startup
- Configurations
- Companions
- Test Steps
- Filters
- Data Sets
- String Patterns

Since **properties can be overridden**, it is important to understand the property hierarchy, i.e. the order in which properties are read in a Test Case.

The following hierarchy is used:

- Properties loaded during the set up of a test.
- Operating system environment variables (like java.version or os.user etc.)
- LISA property files (see next section)
- Command line attributes
- The Default Configuration
- Any alternative configuration properties (from Active Configuration or runtime Configuration file)
- Properties set during a test run

- Properties in Companions (such as External Property Reader Companion)
- Properties set during test execution (i.e. Data Sets, Filters and Steps etc.). Remember that properties set here override values set earlier.

5.5.1 LISA Property File (lisa.properties)

5.5.1 LISA Property File (lisa.properties)

Lisa uses a property file, "**lisa.properties**" to store initialization and configuration information.

Property files are stored in the LISA install directory **LISA_HOME**.

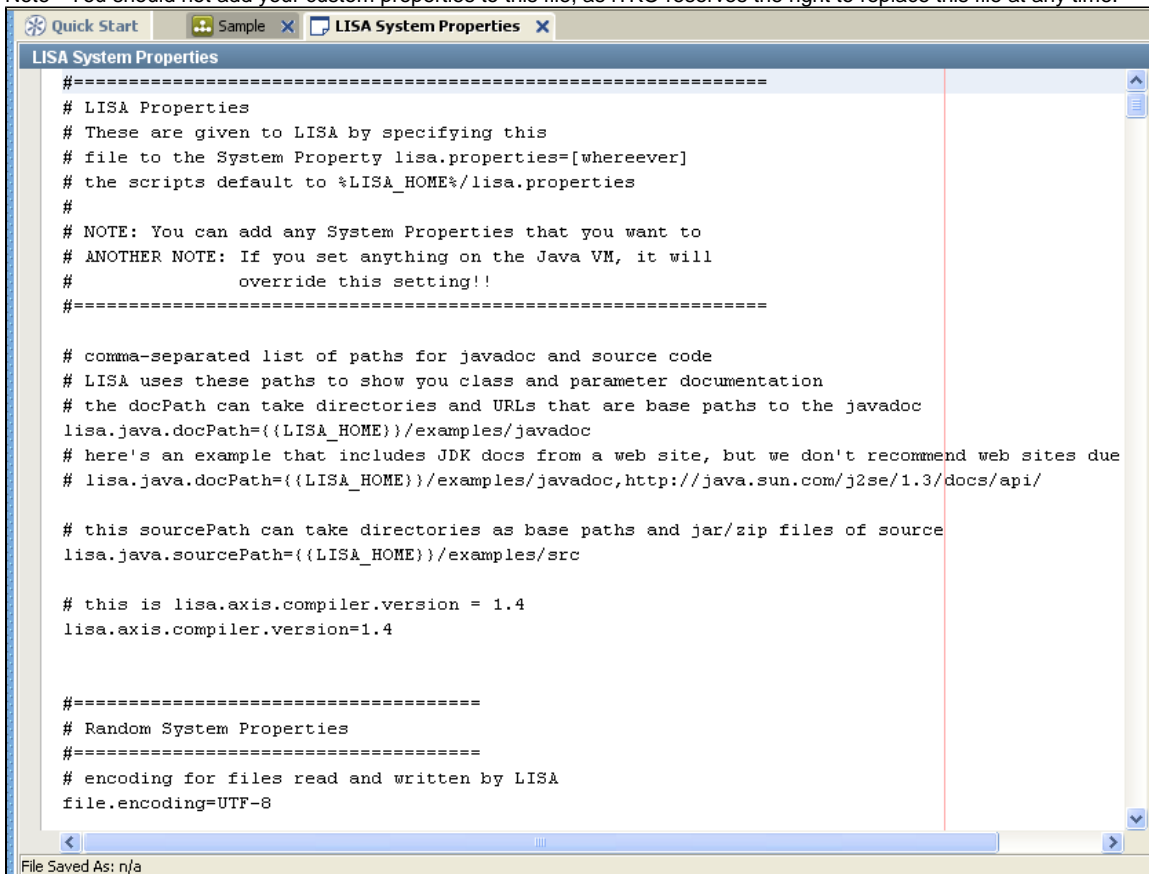
The contents of this file can be displayed within LISA workstation.

To open **LISA properties file**,

- Select **System -> Edit LISA Properties** from the main menu.

The **lisa.properties** file will open up within LISA Workstation as shown below:

Note - You should not add your custom properties to this file, as iTKO reserves the right to replace this file at any time.



```

=====
# LISA Properties
# These are given to LISA by specifying this
# file to the System Property lisa.properties=[whereever]
# the scripts default to %LISA_HOME%/lisa.properties
#
# NOTE: You can add any System Properties that you want to
# ANOTHER NOTE: If you set anything on the Java VM, it will
#           override this setting!!
=====

# comma-separated list of paths for javadoc and source code
# LISA uses these paths to show you class and parameter documentation
# the docPath can take directories and URLs that are base paths to the javadoc
lisa.java.docPath={{LISA_HOME}}/examples/javadoc
# here's an example that includes JDK docs from a web site, but we don't recommend web sites due
# lisa.java.docPath={{LISA_HOME}}/examples/javadoc,http://java.sun.com/j2se/1.3/docs/api/

# this sourcePath can take directories as base paths and jar/zip files of source
lisa.java.sourcePath={{LISA_HOME}}/examples/src

# this is lisa.axis.compiler.version = 1.4
lisa.axis.compiler.version=1.4

=====
# Random System Properties
=====
# encoding for files read and written by LISA
file.encoding=UTF-8

```

5.5.2 Custom Property Files (local.properties, site.properties)

5.5.2 Custom Property Files (local.properties, site.properties)

LISA provides two other property files that you can use for your custom properties:

- **local.properties**
- **site.properties**

To use these files,

- Go to the LISA root directory and locate **_site.properties**, **_local.properties**
- Remove the underscore from the template filenames **_site.properties**, **_local.properties**.

Properties files are loaded in the following order:

1. lisa.properties
2. site.properties
3. local.properties

Site properties can be stored at the test Server, which automatically pushes the **site.properties** file to all workstations that connect to it.

5.5 Important Property Files

5.5 Important Property Files

LISA has the following main property files.

- lisa.properties file-
- local.properties file-
- site.properties file-

5.6 Common LISA Properties and Environment Variables

5.6 Common LISA Properties and Environment Variables

LISA_HOME: This property points to the LISA install directory, and is automatically set by LISA.

Note: This value includes a final slash. So to reference a directory such as "examples" you would specify:

```
{{LISA_HOME}}examples
```

No slash is needed before the directory name.

LISA_JAVA_HOME: This property defines the JAVA VM that LISA will use.

Use this only if you do not want to use the built-in VM in LISA. If there is no Java installed, LISA uses the bundled JRE it comes with. You will also have to rename the current "**jre**" directory, in the LISA install directory, to something like "**jre_notinuse**".

HOT_DEPLOY: This property points to a project specific **hotDeploy** directory.

LISA_MORE_VM_PROPS: This property is used to provide extra arguments to the JAVA VM.

For example, to increase memory in LISA use `LISA_MORE_VM_PROPS=-Xmx=512m`. To reduce the length of the argument list, and to make it easier to modify, you can define several variables and put those as the value of `_LISA_MORE_VM_PROPS`, i.e. `_MEMORY_OPT="-Xmx512m" OTHER_OPT=...whatever... LISA_MORE_VM_PROPS="%MEMORY_OPT% %OTHER_OPT%"`

LISA_PRE_CLASSPATH: This property is used to add information before the LISA classpath.

LISA does not use the OS environment CLASSPATH variable. To add your own jars before LISA classpath, use `LISA_PRE_CLASSPATH`.

LISA_POST_CLASSPATH: This property is used to add information after the LISA classpath.

LISA does not use the OS environment CLASSPATH variable. To add your own jars after the LISA classpath, use `LISA_POST_CLASSPATH`.

LASTRESPONSE: This property shows the response to the last executed step.

LISA_TC_PATH: This property is the path to the test that you are running.

LISA_USER: This property is the user that loaded the test.

LISA_HOST: This property shows the system name on which the testing environment is running.

LISA_PROJ_ROOT: This property depicts Full path to the project that the current document belongs to. Refers to the path of LISA project root directory.

LISA_PROJ_NAME: This property depicts Name of the project that the current document belongs to.

6. Using Configurations

6. Using Configurations

A **Configuration** is a named collection of properties that usually specify environment-specific values for the '**system under test**'.

By removing hard-coded environment data from the Test Case, the same test can run against different environments by simply using a different configuration. Configurations are used everywhere, i.e in a Test Case document, Test Suite document, Staging document, Test case execution and Test Suite execution.

A configuration has to be defined at the LISA Project level and you can specify the values of these properties at the beginning of a Test Case.

Project.config is the default project configuration within any test case. You can create additional new configurations within a Project and "Make it Active" for a particular Test Case/Suite.

Even if you create a new Config file, you will **not** be able to add any new keys within them. To add keys within the new Config, you necessarily have to add them in the Project.config. You can select the newly defined keys from the drop down in the new Config file.

LISA will add properties to your configuration automatically as you develop your test.

For example, in a web service test when you enter the name of the WSDL, LISA will replace the Server name and the port that you entered with properties such as **WSSERVER** and **WSPORT**. The values of these properties are automatically added to your default project configuration. Now you can change the location of the web service merely by editing the configuration rather than looking for hard coded values in several test steps.

In another example, when working with Enterprise Java Beans (EJBs) or Java objects, you may want to switch hot deploy directories, or add extra jar files to your class path, in order to use different versions of your Java code. LISA provides standard properties for these locations, **HOT_DEPLOY** and **MORE_JARS**. These can be set in your configuration.

For details on other standard LISA properties see [Using Properties](#).

Configurations are **for storing properties** related to the system under test. Avoid using them for storage of "test-like" parameters and global parameters. These can be stored in a Companion (External Property Reader).

Note - Backslashes "\" are not preserved in configuration files. If you edit the config file manually and put something that has backslash, then LISA would overwrite the file without the backslashes.

The following topics are available in this Chapter.

- [13.1 Project Configuration](#)
- [13.2 Adding a Configuration](#)
- [13.3 Creating a Configuration File](#)
- [13.4 Default Configuration](#)
- [13.5 Active Configuration](#)
- [13.6 Editing a Configuration](#)
- [13.7 Copy Pasting a Configuration](#)
- [13.8 Deleting a Configuration](#)
- [13.9 Renaming a Configuratio](#)
- [13.10 Applying Config when Staging a Document](#)

6.1 Project Configuration

6.1 Project Configuration

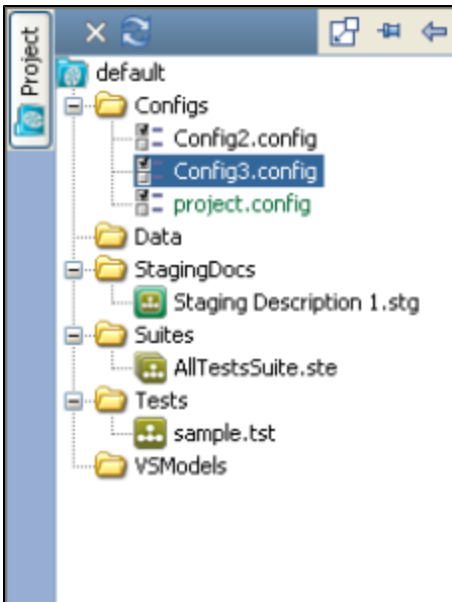
From LISA 5.0, the Configurations are defined at the Project level.

From 5.0, the green arrow in the Model Editor would not show any Configurations listed. This is just a graphic that marks the beginning of the test case.

Note - But if we open a test case that is outside of a project, the green arrow will show a drop-down of configs defined internally to this test case. The older Test Cases and examples prior to 5.0 would still be able to manage their multiple Configs from the green arrow.

Configurations can be applied to a Test Case by making them "Active".

The Configuration at the Project level can be seen in the **Configs** folder in the Project pane seen as below:

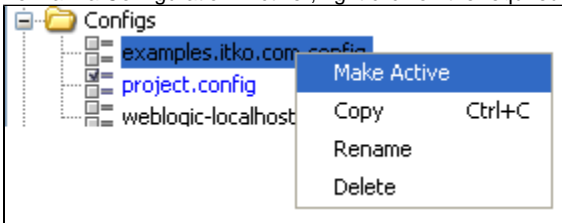


By default the Project.config is the Active config -marked in blue.



Note - You cannot Rename or Delete the Default config.

To mark a Configuration "Active", right click on the required Config and click "Make Active" to select it for test case.

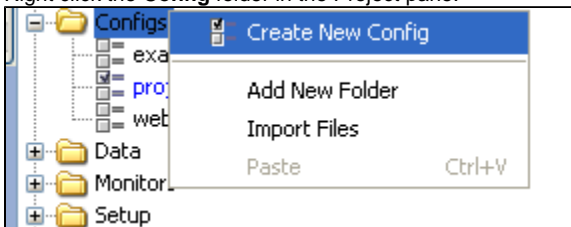


6.2 Adding a Configuration

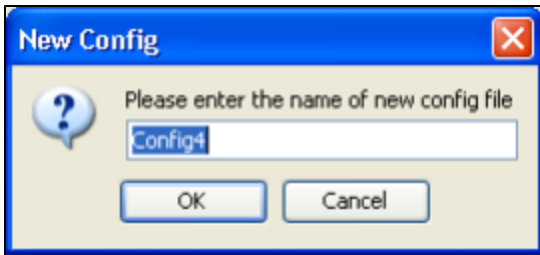
6.2 Adding a Configuration

To add/create a new configuration,

- Right click the **Config** folder in the Project pane.

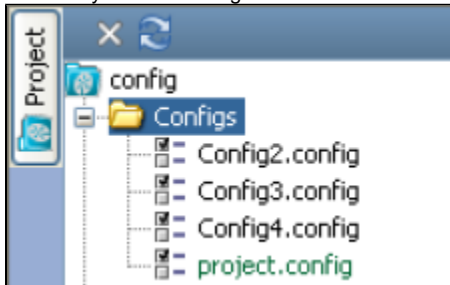


- Click **Create New Config** and enter the name of the new Configuration:



- Click OK to create the new Configuration.

The newly created configuration is now seen in the Config list under Config folder as shown below:



Note: Since 5.0, all the Configurations will be external to a Test Case and are the Project level.

The **project.config** file is the default Configuration for the LISA Projects. It has the superset of all the keys defined in every other Configuration.

Adding Key-Value pairs

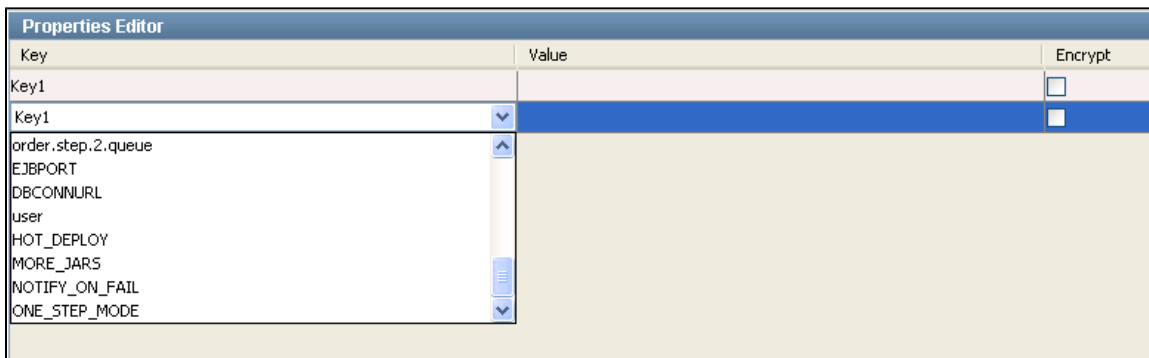
To add keys within the new Config, you necessarily have to add them in the **Project.config**.

You can select the newly defined keys from the drop down in the new Config file.

Note - In a new Config file, you will **not** be able to add any new keys.

This means that when you create a new config, the only keys that you can add are the ones that are already defined in **project.config**, in addition to the standard config keys provided by LISA.

For example, if the project.config has "Key1" defined in it, the choices available in any of the alternate configurations are as shown below:



Adding a Config at the Project Root

To add a config into the Project directory,

- Right click the Project node in the Project Panel
- Click on **Add a Config**
- Enter name of the new Config
- Click OK

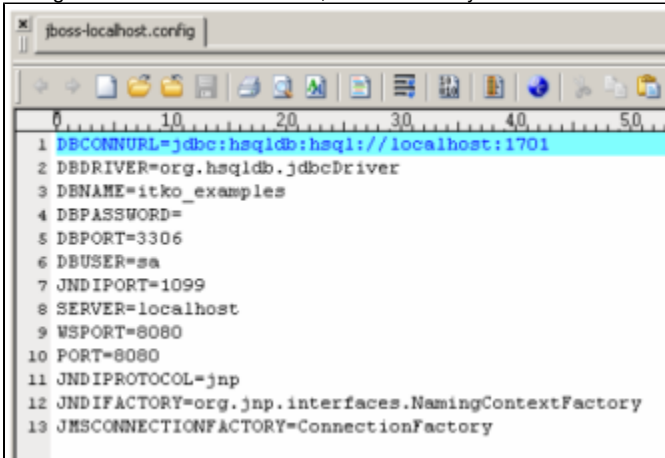
The configuration is added. But the Configuration added in this manner will not be seen in the LISA Workstation. It is added in the Project directory and can be seen from the explorer window.

6.3 Creating a Configuration File

6.3 Creating a Configuration File

A Configuration file is a text file with a 'config' extension that contains the properties as key-value pairs, such as the example below. These configuration files are an integral part of any test run.

Configuration files can be created, or edited in any text editor and can be saved with a '**.config**' extension as shown below:



When starting a test run, you will have the opportunity to choose a configuration file of your choice. There is an example of this later in this chapter.

You should establish a naming convention for configuration files, making identification of alternative configurations easier.

6.4 Default Configuration

6.4 Default Configuration

LISA has one default configuration - project.config.

Once you open a Project, the Default Configuration is shown with a different color in the Configs folder as shown below:



If at the start nothing is chosen as active, default config would be in green, but system will use it as active.

Default config will become blue when we physically right-click and mark it as active.

By default "**project.config**" is the **Default Configuration** of any LISA Project. By default it is also the Active configuration.

You can change the default config to make it **Active** or **Copy** the same, by right clicking the project.config file.

Note - You cannot delete or rename the default configuration file.

Default config will become blue when we physically right-click and mark it as active. If, initially nothing is chosen as active, default config would be in green, but system will use it as active.

Double clicking on the **project.config** will open the configuration in the editor window as shown below:

Properties Editor		
Key	Value	Encrypt
EJBSERVER	localhost	<input type="checkbox"/>
password	example-pwd	<input type="checkbox"/>
lisa.jms.correlation.id	itko-jms-example001	<input type="checkbox"/>
SERVER	localhost	<input type="checkbox"/>
DBPORT	3306	<input type="checkbox"/>
JNDIPROTOCOL	jnp	<input type="checkbox"/>
Key1	Value	<input type="checkbox"/>
WSSERVER	localhost	<input type="checkbox"/>
JNDIFACTORY	org.jnp.interfaces.NamingContextFactory	<input type="checkbox"/>
JMSCONNECTIONFACTORY	ConnectionFactory	<input type="checkbox"/>
WSPORT	8080	<input type="checkbox"/>
user_prefix	csv_usertest	<input type="checkbox"/>
DBDRIVER	org.apache.derby.jdbc.ClientDriver	<input type="checkbox"/>
DBPASSWORD	sa	<input type="checkbox"/>
DBNAME	itko_examples	<input type="checkbox"/>
DBUSER	sa	<input type="checkbox"/>
PORT	8080	<input type="checkbox"/>
JNDIPOINT	1099	<input type="checkbox"/>
order.step.2.queue	queue/C	<input type="checkbox"/>
EJBPORT	1099	<input type="checkbox"/>
DBCONNURL	jdbc:derby://localhost:1529/lisa-demo-server.db	<input type="checkbox"/>
user	webapp	<input type="checkbox"/>

These are standard LISA config parameters that are available in all configurations.

To add parameters in other configurations, you need to first add them here and then select from the drop down in the required configuration.

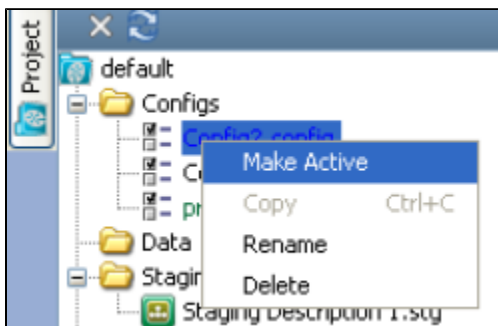
6.5 Active Configuration

6.5 Active Configuration

The configuration that is made "**Active**" for a project or test case is known as ACTIVE.

To make a configuration Active,

- Click and select the Configuration to make Active
- Right click to open a menu as shown below:



- Click **Make Active** to make the Configuration Active.

A configuration can be both Default as well as Active.

The Active configuration applies to the whole project, not a single test case.

Note - Each test case within a single project, shares the "Active configuration" applied to the project. You cannot assign a separate configuration for each test case within a project.

The Active Configuration takes precedence in the Interactive Test Run facility (ITR), but the Default Configuration is used in a Stage Run if no Configuration was specified.

6.6 Alternate Configuration

6.6 Alternate Configuration

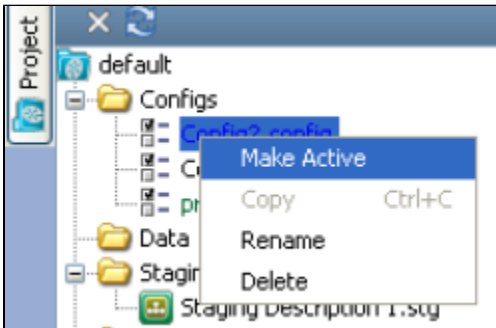
A project can have many configurations within it.

We can choose which configuration is to be applied to a project.

A project can have many Alternate Configurations, but it can have only one default configuration.

Any alternate configuration or the default configuration can be made active.

Alternate configs can only override default properties.



- Click **Make Active** to make the Configuration Active.

An Alternate configuration can be both Default as well as Active.

The Active configuration applies to the whole project, not a single test case.

Note - Each test case within a single project, shares the "Active configuration" applied to the project. You cannot assign a separate configuration for each test case within a project.

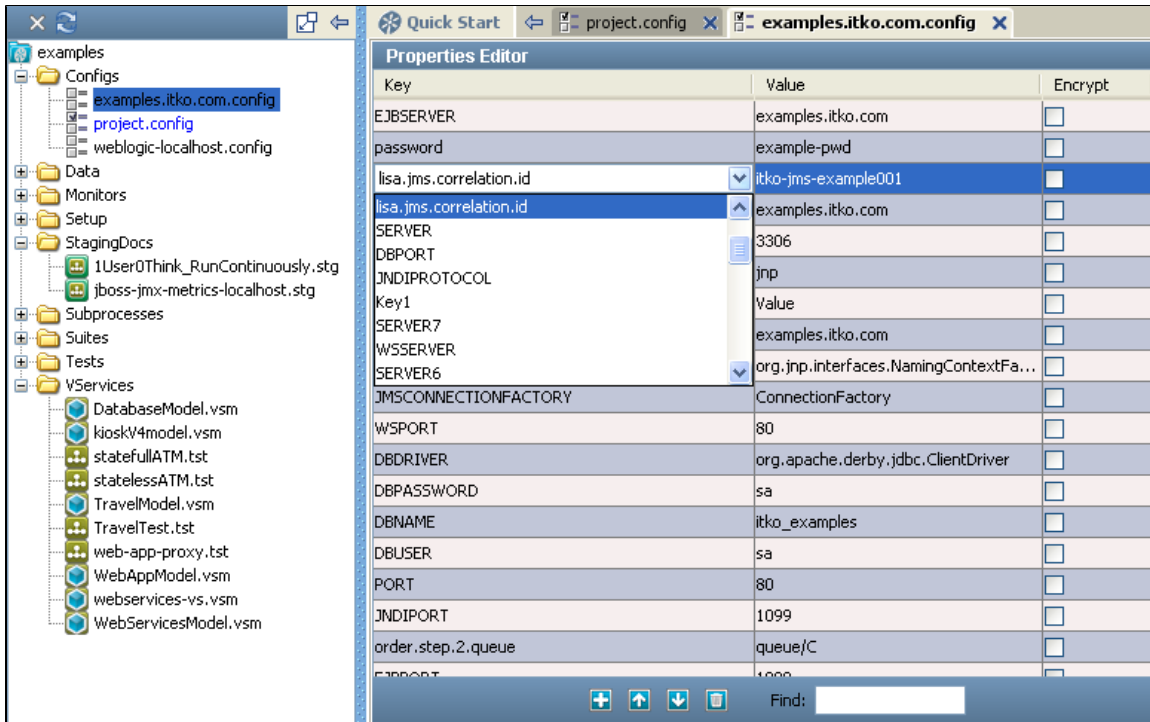
6.7 Editing a Configuration

6.7 Editing a Configuration

To edit an existing configuration,

Double click the configuration name in the **Config** folder.

The "**Properties Editor**" for this configuration opens up on the right side.



You can add/edit existing properties only if they exist in the **Project.config** file.

You can add a new property using the **Add** icon at the bottom of the Configuration editor. Adding a new property will produce a new line in the property list. Click on the drop down to select the desired key from the available list.

Common LISA property names, like HOT_DEPLOY and MORE_JARS, will show up in a pull-down list in the key column, as shown below.

A good practice is to use LISA properties in the path names stored in the configuration. Properties like LISA_HOME or LISA_PROJ_ROOT will allow for portability of Test Cases.

6.8 Copying a Configuration

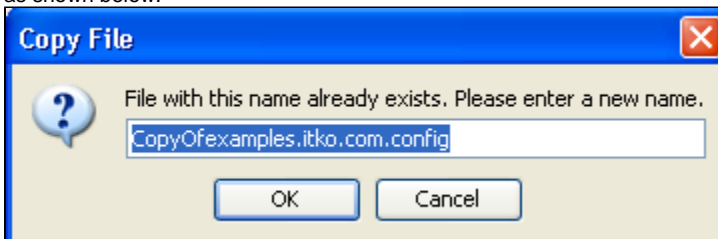
6.8 Copying a Configuration

To copy a Config:

- Select the Config to be copied, right click and select **Copy** to copy the selected configuration.

To Paste a Config:

- Click on the Config folder and Click Paste. It will ask you to rename the pasted Config as it appears to be the copy of the original config as shown below:



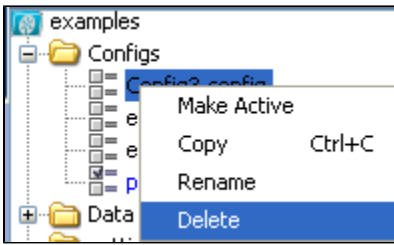
- Rename the config and click OK to add the new configuration in the Config folder.

6.9 Deleting a Configuration

6.9 Deleting a Configuration

To delete a Config:

- Select the Configuration to be deleted and Right-click to open the menu as shown below:



- Click **Delete** from the menu.

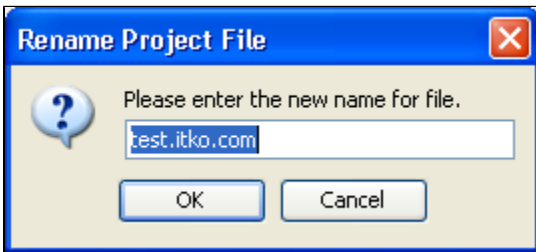
Note - You **cannot Delete** the Default configuration (project.config) within LISA.

6.10 Renaming New Configuration

6.10 Renaming New Configuration

To rename a Config:

- Select the Configuration and right click to open a menu.
- Click **Rename** to rename the configuration.



- Enter the new name and click OK to rename the file.

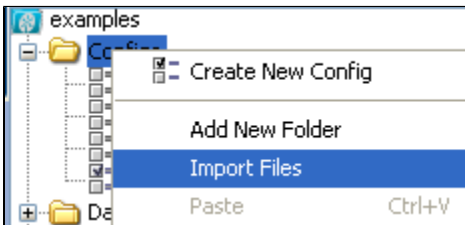
Note - You cannot **rename** the Default configuration (project.config) within LISA.

6.11 Importing a Configuration File

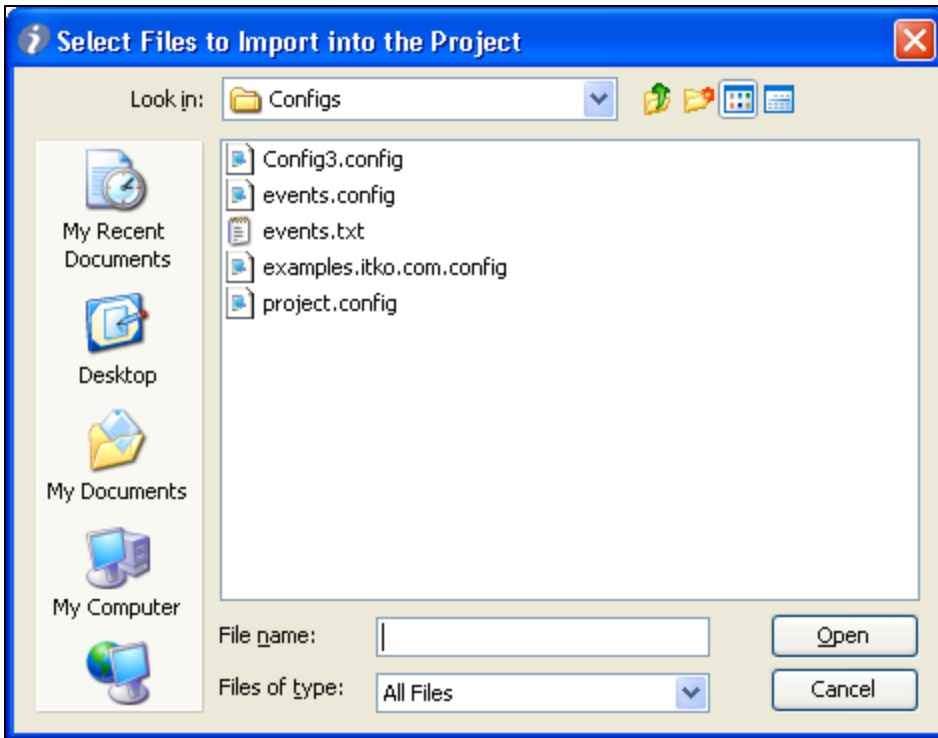
6.11 Importing a Configuration File

To Import a Config:

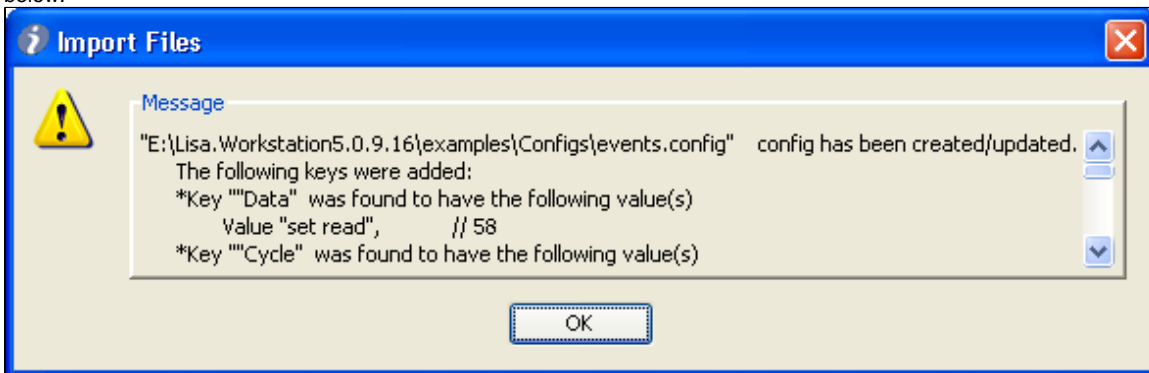
- Select the Config folder and right click to open a menu.
- Click **Import** to import a configuration file.



- Enter the name of the config file to be imported and click Open to import the file.

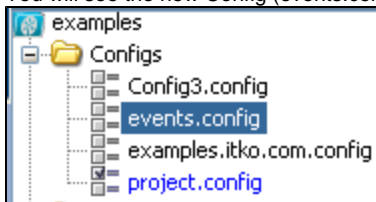


- We are importing events.config file. LISA will give a successful imported message and a list of imported parameters of the config file as below:



- Click OK.

You will see the new Config (events.config) imported in the list of configurations.



6.12 Applying Config when running Test Case

6.12 Applying Config when Running a Test Case

Running a Test Case is one of the several places where configurations are used.

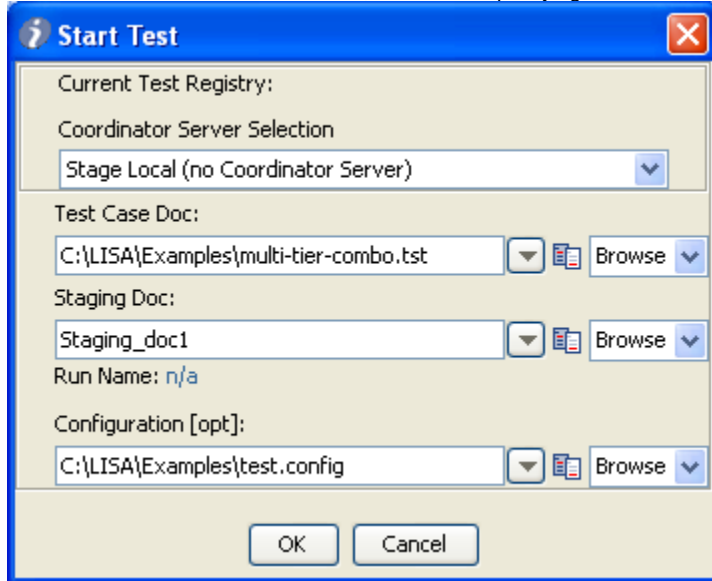
To Start a test case execution, you need to give the test case details along with the Configuration to be applied.

To Stage/Start a Test Case,

- Click on **Start a Test** icon on the Test Case toolbar.
- Or click **Start > Test Case Execution** from the main menu.

The Start test window opens up as shown below:

Here we are about to run a Test Case, and we are specifying the documents for that test run.



- Select the Coordinator server or if not there, you can select Stage Local.
- Select the **Test case document** which you need to execute.
- Select the **Staging document** to refer to while executing the test case.

When LISA knows which test you wish to run, the **Configuration [opt]** pull-down will include all the configurations defined in the Project or click **Browse** to select the config file.

Note - Assigning a configuration file here is optional. If omitted, LISA will use the default configuration in the Test Case document.

When the Test Case has not yet been determined, the configuration can be typed in.

Note: If the requested configuration cannot be found in the Test Case, an error will be generated when the test is staged.

7. Adding Filters

7. Adding Filters

A Filter is a LISA code element that runs before and after a test step, giving you the opportunity to massage the data in the result, or store values in properties.

Most Filters execute after the step has run. A Filter can be used to extract a value from a web page, XML and DOM responses, a Java object, a text document, and many other test step responses.

Once the data has been filtered, the data can be used in an Assertion, or in any subsequent Test Step. Filters usually operate on the response of the system under test. For example, Filters are used to parse values from an HTML page, or to perform conversions on the response. Filters can also be useful in other places; for instance to save a property value to a file, or convert a property to be the 'last response'.

There are two basic ways to apply a Filter, as a **Global Filter** or as a **Step Filter**. The available Filter types are the same, but how the Filters are applied differs.

Global Filter - A Filter defined on the **Test Case** is a global Filter, and executes before/after every test step that is not set to ignore global Filters (you can set a step to ignore global Filters in the "Step Information Element" of the step).

Step Filter - A Filter defined on a Test Step is a Step Filter and executes before/after each execution of that step only.

You can add as many Global and Step Filters as you need. They are executed in the order that they appear in the test case.

Filters are mainly used as property setters.

The following topics are available in this chapter.

7.1 Adding Filters
7.2 Deleting a Filter
7.3 Reordering a Filter
7.4 Drag and Drop Filter
7.5 Filter Toolbar
7.6 Types of Filters

7.1 Adding Filters

7.1 Adding Filters

LISA provides several ways to add Filters:

- Adding a test Filter manually
- Adding a Filter from an HTTP response
- Adding a Filter from a JDBC result set
- Adding a Filter from a returned Java object

All the other methods imply using Filters that are available for selection in the specific test step editor, as explained in the additional topics available.

7.1.1 Adding a Filter Manually
7.1.2 Adding a Filter from an HTTP Response
7.1.3 Adding a Filter from a JDBC Result Set
7.1.4 Adding a Filter from a Returned Java Object

LISA has many built-in Filters. For details on each of these, see PART 3 - Filters in the LISA Reference Guide.

7.1.1 Adding a Filter Manually

7.1.1 Adding a Filter Manually

To add a Filter 'manually', you need to select the Filter type from a list and enter the parameters for the Filter.

LISA provides two ways of adding a Filter manually: **Global Filter and Step Filter**

The first method allows you to add a Filter at the **Test Case level** (global Filter). Such a Filter will be applicable to all the steps in the Test Case and is automatically run for every step in the Test Case, unless a given node is instructed otherwise.

The second method allows you to add a Filter at the **test step level**. A Filter created using this method will be applicable only to that step and will execute for that step only (step Filter).

Add a Global Filter

To Add a Global Filter,

Open a Test Case and click anywhere in the editor area to open the Test Case Elements tree.

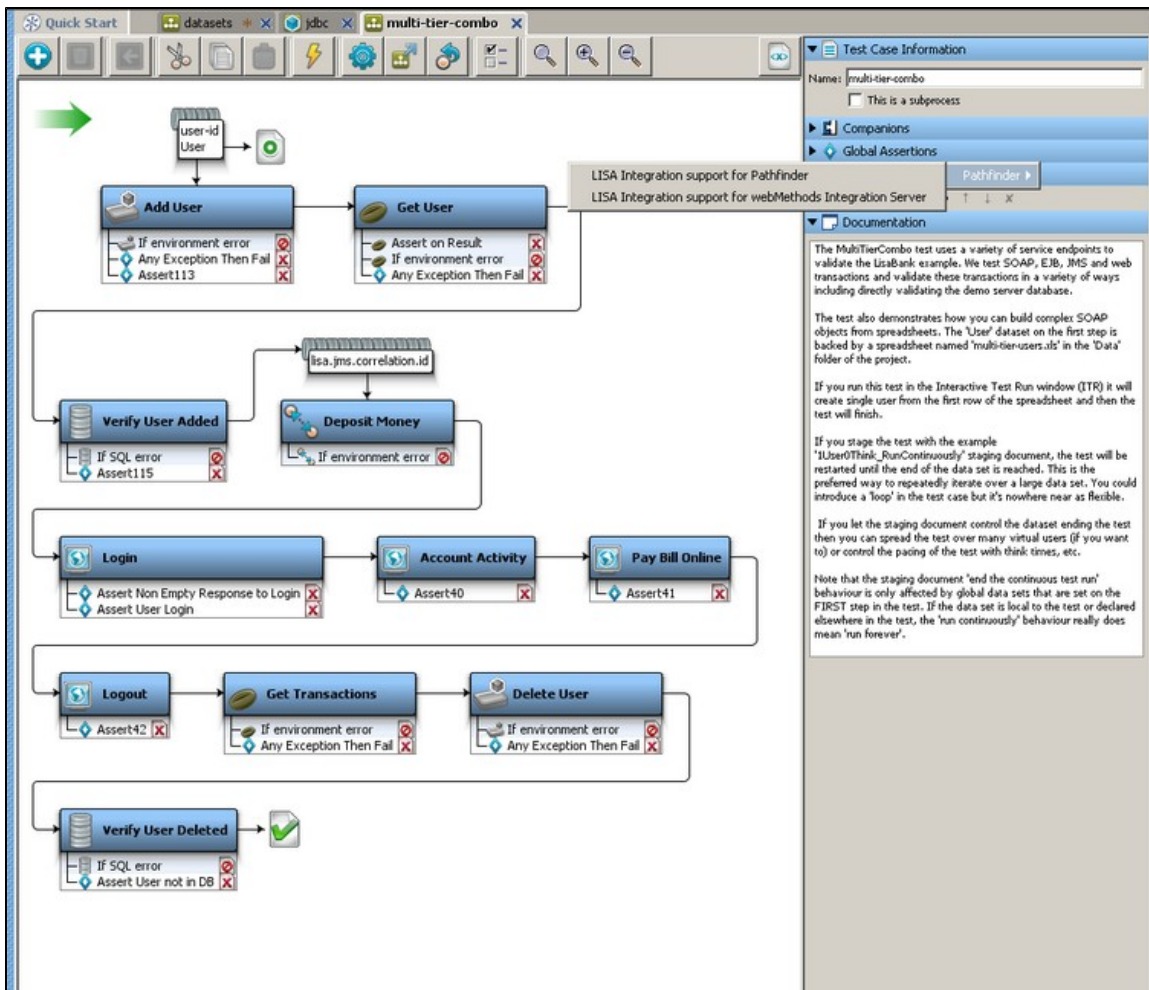
Click the **Global Filters** element.

You can apply following types of Global Filter:

PathFinder –

- LISA Integration Support for Pathfinder.
- LISA Integration Support for webmethods Integration Server.

For the purposes of illustration, we have shown the global Filters applied to the **multi-tier-combo** Test Case.



In the right panel, click the **Step Information** tab. The Step information Editor will open.

By default the **"Use Global Filter"** in all the steps is checked. Meaning the Global filter will be applied to this step.

In case you do not want the Global Filter to be applied to this particular step, uncheck the **"Use Global Filters"** box below:

The 'Step Information' dialog box is shown with the following details:

- Name:** Step2
- Think time:** 500
- To:** 1
- Use global filters:** ☒ (checked)
- Execute on:** local
- Next:** end
- Elements:** Dynamic Java Execution, Log Message, Assertions, Filters, Data Sets, Properties Referenced, Properties Set, Documentation.

Add a Step Filter

To add a Step Filter,

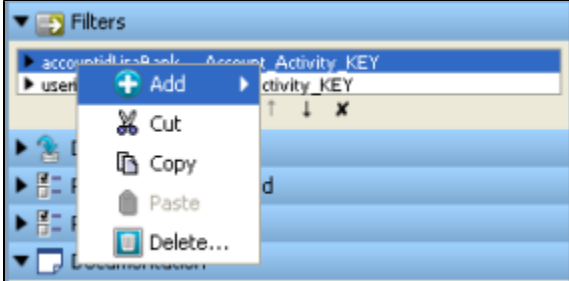
Select the step for which you want to apply the Filter and in the right panel click the **Filter** Element as shown below:



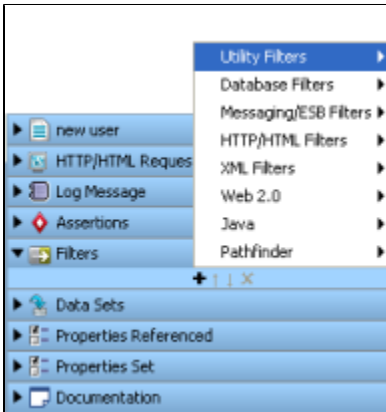
Click on the **Add (+ sign)*** icon of the Filter Element.

Or right click the step and select "Add Filter" and select the appropriate filter for this step.

Or you can also right click an existing Filter in the Elements tree to open a menu and click **Add** as shown below:



This will open the Filter menu listing the common Filters in LISA as shown below:



Note: Each Filter will have its own editor and applicable parameters which need to be set.

To add a Filter at the test step level, expand the step element in the Elements tree, then click the Filters element for that step.

Click **Add (+ sign)*** below the Filter element in the Elements tree, to add the Filter. This will open the Filter panel listing the common Filters for that particular step type. Click Next to open the appropriate Filter editor.

For the purposes of illustration, we will use the – **Override Last Response Property/Convert Property Value into Last Response** Filter in this section. This will convert a property value into the 'last response'.

Each Filter will have its own parameters and editor.

To configure the Filter, enter the following parameters:

- **Filter in (*Property to Convert):*** The name of the property you want considered as the step's last response. The property should be in the pull down menu. You may type the property name if you do not find it there. The Filter will give an error if it is not an existing property.
- **Convert to XML:** Check this if you want the response to be converted to valid XML.
- **Run Filter:** Click to run the Filter.

All the Filters available in LISA are described in depth in Chapter 3 Filters in the [LISA Reference Guide](#).

7.1.2 Adding a Filter from an HTTP Response

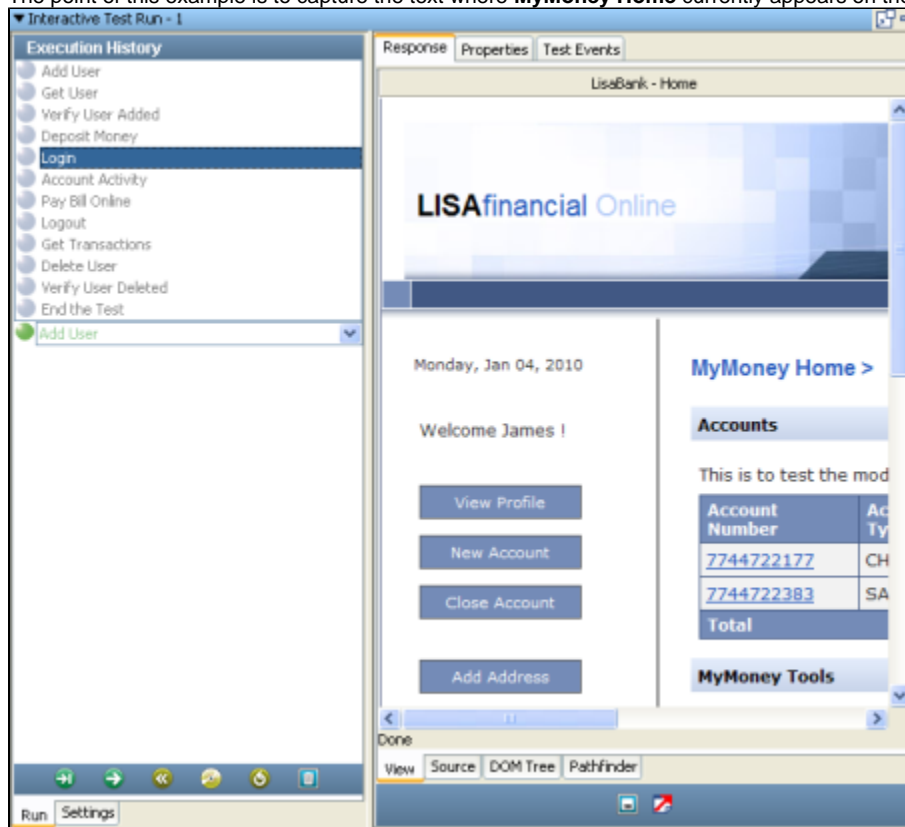
7.1.2 Adding a Filter from an HTTP Response

When you have access to the response from an HTTP-based step, you can use the response to add a Filter directly. The following is an example of how to add a Filter this way.

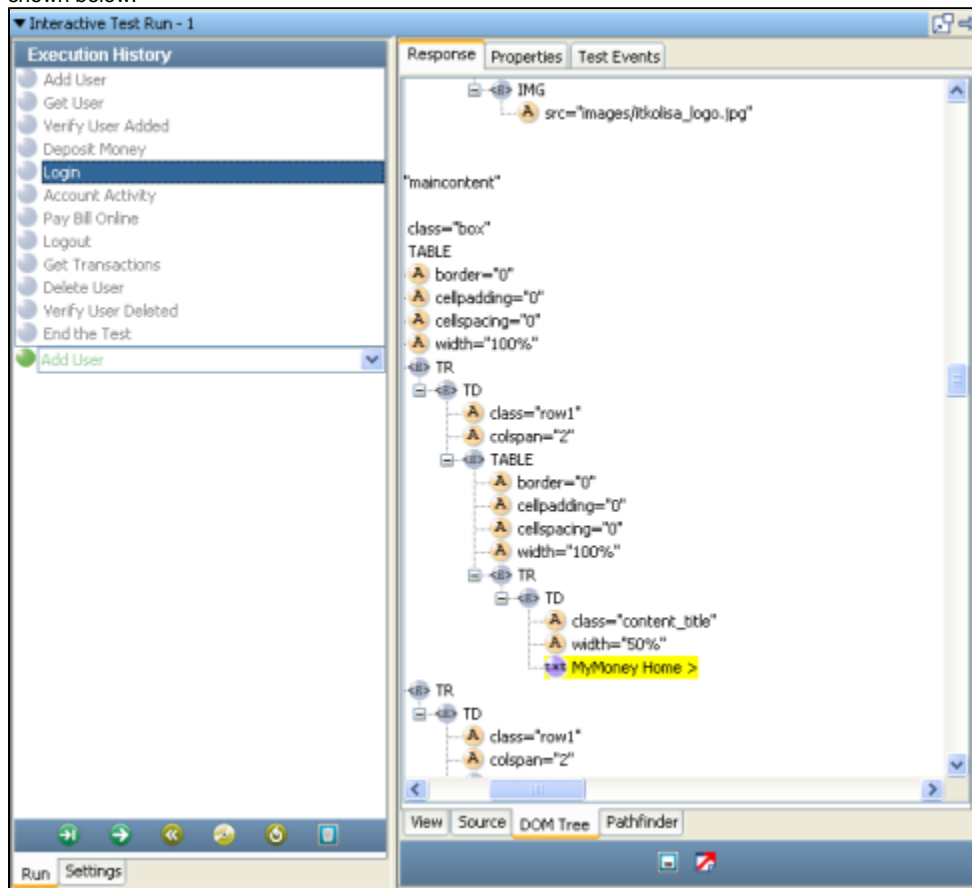
Here is an example of the HTTP/HTML response:

For the purpose of illustration, we will look at the response of the **login** step in **multi-tier-combo** Test Case in the examples directory (multi-tier-combo.tst).

The point of this example is to capture the text where **MyMoney Home** currently appears on the screen. (It will not always be the same text).



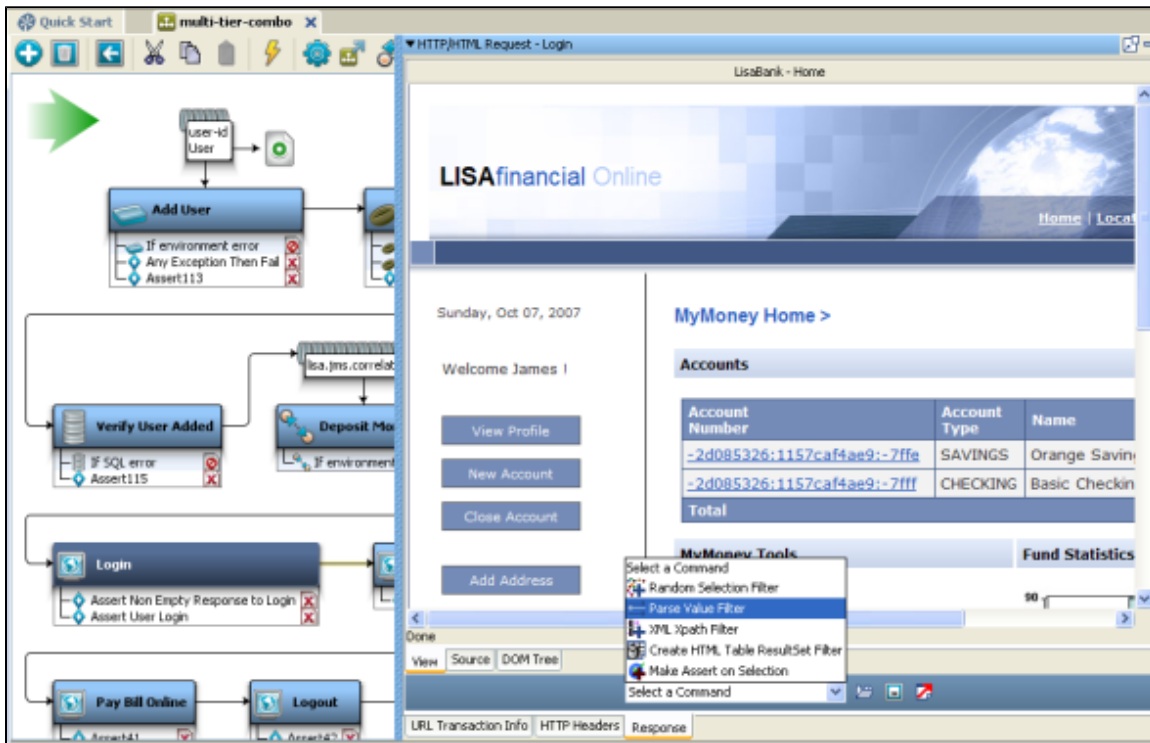
Highlight the text **"MyMoney Home"** in the **View** Tab and click the **DOM Tree** tab to make sure that this text is highlighted in the tree view as shown below:



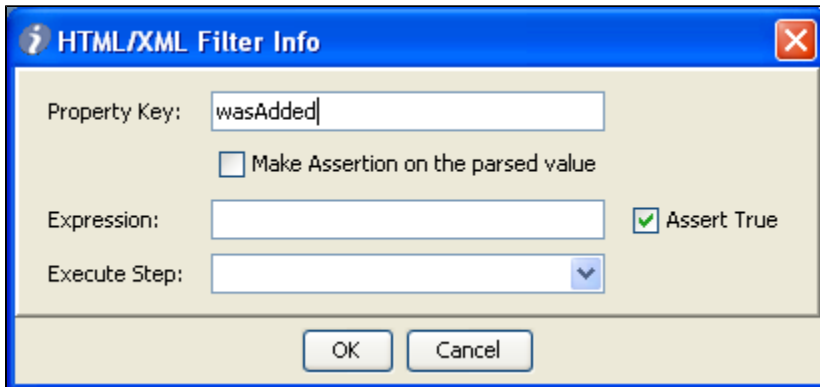
We will now apply an **Inline Filter** here,

To apply an inline filter,

- Double click the **login** step in the Model Editor to open the HTTP/HTML Step Editor.
- Click the **Response** tab as shown below:



- Select **Parse Value Filter** from the drop down menu.
- In the pop-up window that is displayed, enter the name for the Property Key "was Added":

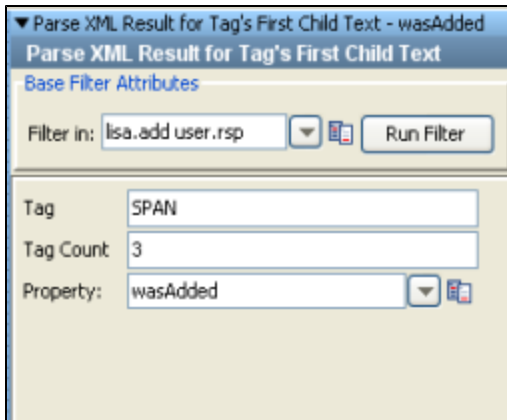


- Click **OK**.

Note: We could also add an Assertion here if we wished. For example we would probably want to test the value of the property 'wasAdded', to see if it is in fact equal to Added user. We will explore this further in [Adding Assertions](#).

The Filter that was generated can be seen as a Filter in the **login** test step.

The Filter editor is as shown below:



Tip: The same Filtering capabilities are available when an HTML response is displayed in the Step Editor.

7.1.3 Adding a Filter from a JDBC Result Set

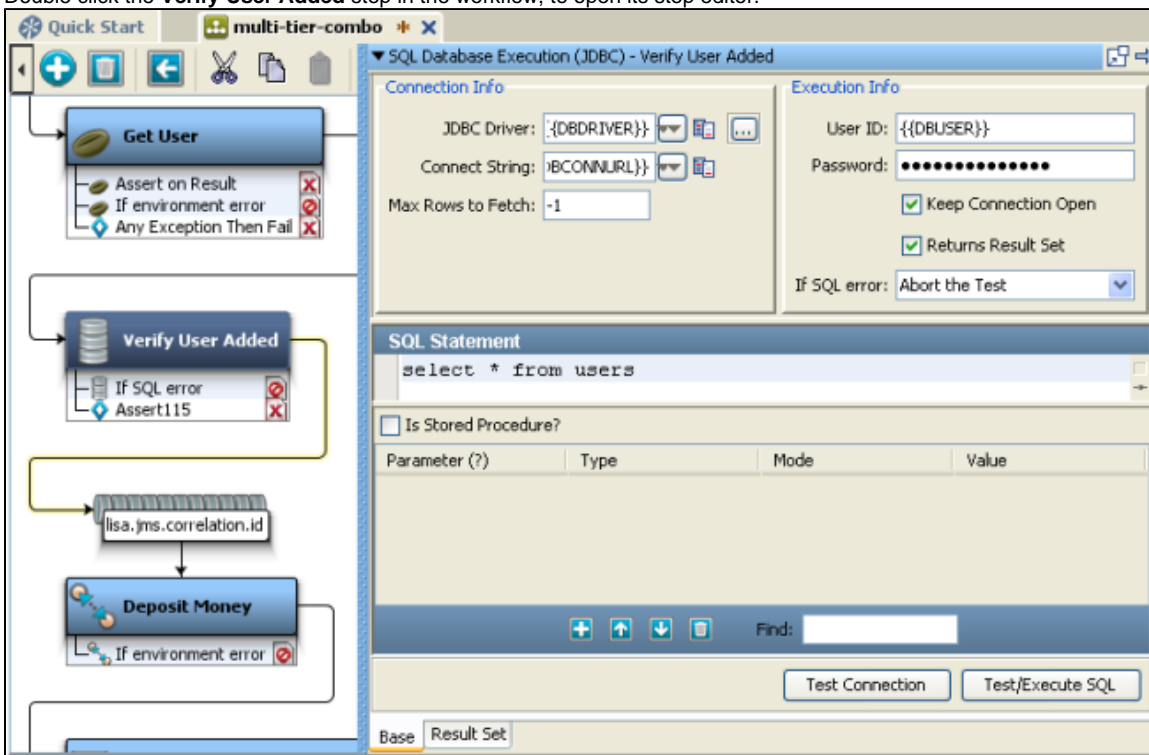
7.1.3 Adding a Filter from a JDBC Result Set

When you have access to the result set response from a JDBC step, you can use the response to add a Filter directly. The following is an example of how to create a Filter this way.

Here is an example of the JDBC Result Set response:

For the purpose of illustration, we will look at the response of the **Verify User Added** step in **multi-tier-combo** Test Case in the examples directory (multi-tier-combo.tst).

- Double click the **Verify User Added** step in the workflow, to open its step editor.




- Click the **Result Set** tab as shown below and click the **Test/Execute SQL** button to get values in the Result Set. The Result set is as shown below:

The screenshot shows a workflow diagram on the left and a SQL result set on the right. The workflow includes steps: 'Add User' (with a green arrow pointing to it), 'Get User', and 'Verify User Added'. The 'Verify User Added' step has a filter 'Assert115' with the condition 'If SQL error'. The SQL result set is titled 'SQL Database Execution (JDBC) - Verify User Added' and contains a table with columns: LOGIN, PWD, NEWF..., FNAME, LNAME, EMAIL, PHONE, ROL..., and SSN. The table lists several users, including 'admin', 'sbellum', 'wpiece', 'areck', 'boaty', 'itko', 'lisa_simpson', and several users with IDs like 'user-1179473389' through 'user-745327623'.

LOGIN	PWD	NEWF...	FNAME	LNAME	EMAIL	PHONE	ROL...	SSN
admin	0DPKUNlrrYmD...	1	ITKO	Admin	lisaban...	123-4...	2	434-4...
sbellum	26yJsXNpIb5AK...	1	Sara	Bellum	sbellum...	232-4...	1	614-4...
wpiece	/UuJ0MeQFuRN...	1	Warren	Piece	wpiece...	455-3...	1	546-7...
areck	AHdRRjD4AdIU...	1	Amanda	Reckonwith	areck...	555-2...	1	350-0...
boaty	RQIILdpTRdN...	1	Boaty	Rabbit	boaty...	333-4...	1	616-5...
itko	qUqP5cyxm6Yc...	1	itko	test	itko.te...	650-2...	1	140-7...
lisa_simpson	60fAFoq+WOR...	1	lisa	simpson	lisa.sim...	123-4...	1	295-2...
user-1179473389	N6DKWIKSp02...	1	James	Kirk	James...	234-4...	1	
user-1862116447	N6DKWIKSp02...	1	James	Kirk	James...	234-4...	1	
user-1058417265	N6DKWIKSp02...	1	James	Kirk	James...	234-4...	1	
user-1031881219	N6DKWIKSp02...	1	James	Kirk	James...	234-4...	1	
user-1688389507	N6DKWIKSp02...	1	James	Kirk	James...	234-4...	1	
user-2049285897	N6DKWIKSp02...	1	James	Kirk	James...	234-4...	1	
user-745327623	N6DKWIKSp02...	1	James	Kirk	James...	234-4...	1	

- Click on the cell in the Result Set tab that represents the location of the information you want to capture (**sbellum**).

- Click the **Generate Filter for Current Col/Row Value** icon  located below.
- In the dialog box that opens, enter the Property Key "theLogin":

A dialog box titled "Please enter the property key for t..." with a question mark icon. The text input field contains "theLogin". There are "OK" and "Cancel" buttons at the bottom.

- Click the **OK** button.

LISA will add a Filter called **Parse Result Set for Value** in the list user step and you will get the confirmation for the same.



A dialog box titled "JDBC Result Set" with an information icon. The text inside says "Added Parse Result Set For Value Filter." There is an "OK" button at the bottom.

- Click on the Filter Editor to take a look.

▼ Parse Result Set for Value - theLogin



Parse Result Set for Value

Base Filter Attributes

Filter in:   Run Filter

Column (1-based or Name):

Row (0-based):

Property:  

In the example, the value in the cell in the 1st column, and 2nd row, **sbellum**, will be stored in the property **theLogin**.

Applying a Second Filter




There is a second Filter that can be applied here. You can look for a value in one column of the result set, and then capture a value from another column in the same row.

From within the result set select the two values in two different columns from the same row, using the control key.

▼ SQL Database Execution (JDBC) - list users

Result Set

LOGIN	PWD	FNAME	LNAME	EMAIL	PHONE	ROLEKEY
admin	0DPIKuNirr...	ITKO	Admin	lisabank-ad...	123-4567	2
sbellum	26yJsXNpI...	Sara	Bellum	sbellum@m...	232-4345	1
wpiece	/UuJOMeQ...	Warren	Piece	wpiece@m...	455-3232	1
areck	AHRRRjD4...	Amanda	Reckonwith	areck@my...	555-2244	1
boaty	RQIiOLdpT...	Boaty	Rabbit	boaty@rab...	333-4521	1
itko	qUqP5cyx...	itko	test	itko.test@i...	650-234-1...	1
lisa_simpson	60fAFoq+...	lisa	simpson	lisa.simpso...	123-456-7...	1
multitier-30...	AL2slstNsa...					



Base Result Set

- Select **Filter for a value and then get another column value** Filter using the  icon as shown above.

Note: You need to select two cells in the same row in order to create this filter.

One will be the search column and the other will be the column who's value you want to extract.

- In the dialog box that will open, check or reassign the columns for the search and the value, then enter the Property Key theEmail:

 **Generate Value for Value Filter** 

Search Column (1-based or Name):

Value Column (1-based or Name):

Property Key to save value into:

OK Cancel

- Click the **OK** button.

LISA will add a Filter called **"Get Value For Another Value in a ResultSet Row"** in the Verify User Added step.

▼ Get Value For Another Value in a ResultSet Row - theEmail
Get Value For Another Value in a ResultSet Row

Base Filter Attributes

Filter in:

Search Column (1-based or Name):

Value Column (1-based or Name):

Search Text (Regular Expression):

Property:

We are looking for **sbellum** in the LOGIN column, and if found we want the value in the EMAIL column in the same row to be stored in a property called **theEmail**.

Note: The same Filtering capabilities are available when a JDBC result set is displayed in the Step Editor.

7.1.4 Adding a Filter from a Returned Java Object

7.1.4 Adding a Filter from a Returned Java Object

When the result of your test step is a Java object, you can use the **inline Filter** panel in the Complex Object Editor to filter the returned value from the method call directly. The following is an example of how to add a Filter this way.

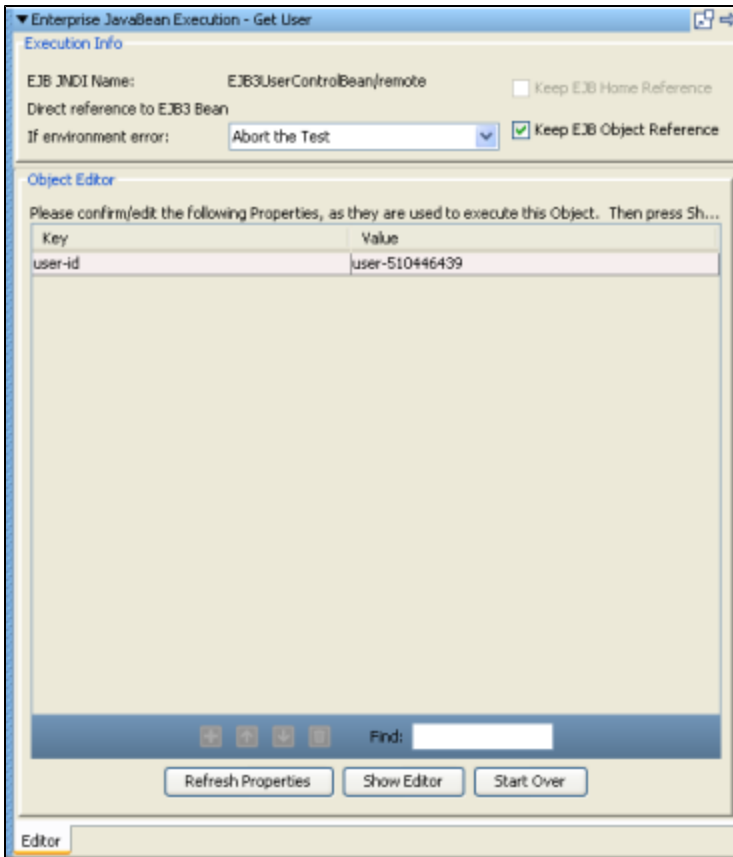
Here is an example of an object in the complex object editor:

For the purpose of illustration, we will look at the **get user** step (EJB step) in **multi-tier-combo** Test Case in the examples directory (multi-tier-combo.tst).

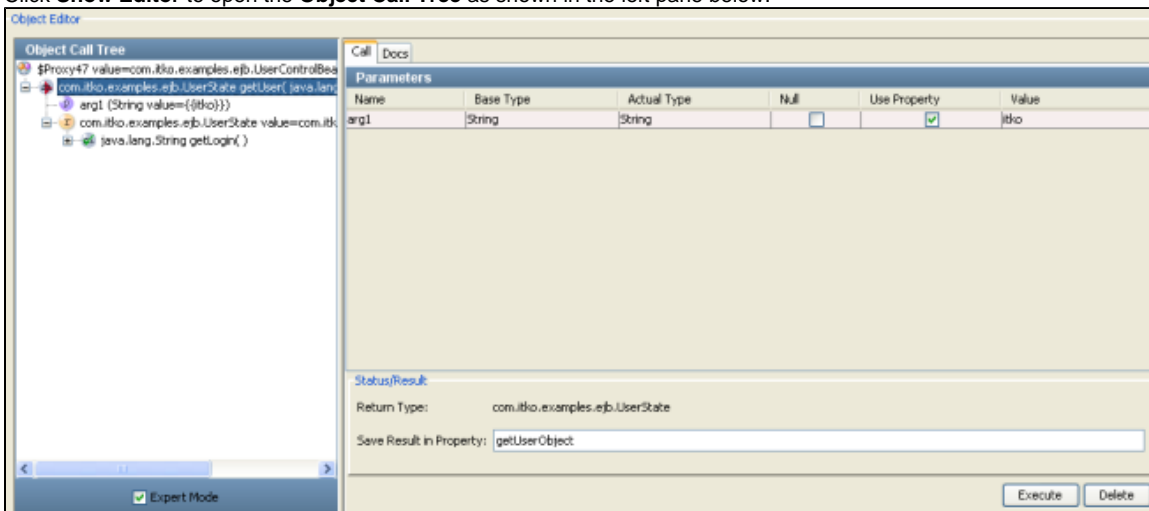
- Double click the **get user** step in the workflow, to open its step editor as shown below:

The screenshot shows the 'multi-tier-combo' test case workflow on the left and the 'Enterprise JavaBean Execution - Get User' step editor on the right. The workflow includes steps: 'User-Id User', 'Add User', 'Get User', and 'Verify User Added'. The 'Get User' step is highlighted with a green arrow. The step editor on the right shows the 'Required EJB Setup' panel with 'Connection Info' fields: JNDI Naming Factory (jndi.NamingContextFactory), JNDI Server URL ({EJB_SERVER}/{JNDI_PORT}), User ID, and Password. Navigation buttons (First, Prev, Next, Finish) are at the bottom.

- Click **Next**



- Click **Show Editor** to open the **Object Call Tree** as shown in the left pane below:

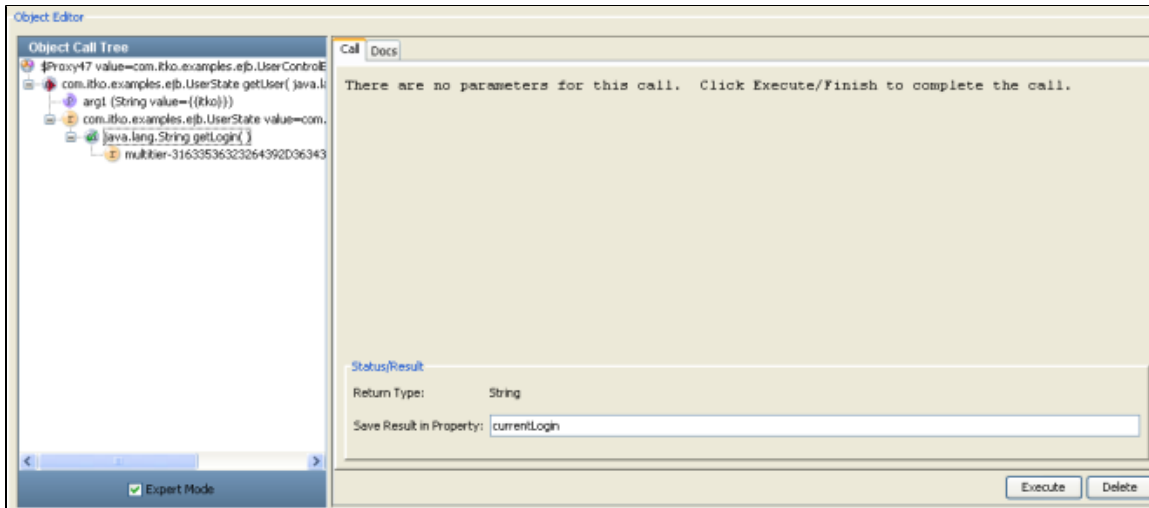


- Enter an input parameter "itko***" in the value field.
- Click **Execute** to execute this method.

The returned value upon executing the **getLogin** method will be stored in the property **getUserObject**. Notice that in this case the returned **value** is an **object** (of type **UserState**).

Note: We could also add an Assertion here if we wished. We will explore this further in [Adding Assertions](#).

In this example, we could also call a method on the returned object to get the actual login value for this user, and save the login in another property:



The returned value upon executing the **getUser** method will be stored in the property "currentLogin". Notice that in this case the returned value is a **String**.

Note: Inline Filters (and Assertions) do not result in a Filter being added to the test step in the element tree. Inline Filter management is always done in the complex object editor.

For more details on the complex object editor see [Using Complex Object Editor](#)

7.2 Deleting a Filter

7.2 Deleting a Filter

To delete a Filter:

- Select the Filter in the Filter Elements tab and right click to open a menu. Click Delete to delete the Filter.
- Select the Filter in the Filter Elements tab and click the Delete icon (X) on the toolbar as shown above.

Note: You cannot delete a test step which has Filters attached to it.

7.3 Reordering a Filter

7.3 Reordering a Filter

To reorder a Filter:

- Select the Filter in the Filter Elements tab
- click the **Move up and/or Move down** icons on the toolbar.

7.4 Drag and Drop Filter

7.4 Drag & Drop Filter

You can drag and drop Filters in the Model Editor from one Test Step to the other.

- Click on the Filter attached to a Test Step --say Step 1.
- Select, Drag and Drop that Filter to other Test step – say Step 2, in the Model Editor.
- The dragged Filter will then be applied to Step2.

Filter Right click Menu

When you right click a filter at a test step level in the Elements tab,

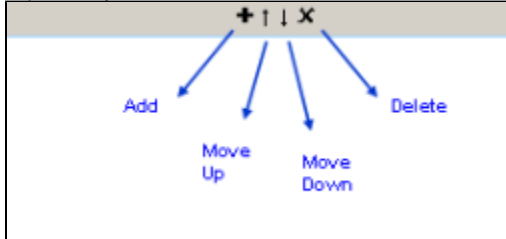
the following options will be available to you...

Add, Delete, Cut, Copy or Paste.

7.5 Filter Toolbar

7.5 Filter Toolbar

At the bottom of every element in the Elements tree, there is a toolbar which has icons to Add/Delete/Reorder as shown below and are self explanatory.



7.6 Types of Filters

7.6 Types of Filters

LISA provides the following Filters:

Utility Filters	▶
Database Filters	▶
Messaging/ESB Filters	▶
HTTP/HTML Filters	▶
XML Filters	▶
Web 2.0	▶
Java	▶
VSE	▶
Pathfinder	▶

Utility Filters

- Create property based on surrounding values
- Store Step Response
- Override "Last Response" Property
- Save Property Value to File
- Parse Property Value as Argument String
- Save Property From one key to another
- Time Stamp Filter

Database Filters

- Extract Value from JDBC Result Set
- Simple Result Set Filter
- Size of JDBC Result Set
- Set Size of a Result Set to a Property
- Get Value For Another Value in a Result Set Row

Messaging Filters

- Extract Payload and Properties from Messages
- Convert a MQ Message to a VSE request
- Convert a JMS Message to a VSE request

HTTP/HTML Filters

- Create Resultset from HTML Table Rows
- Parse Web Page for Properties
- Parse HTML/XML Result by Searching Tag/Attribute Values
- Parse HTML Result for Specific Tag/Attribute's Value and Parse It
- Parse HTML Result for Tag's Child Text
- Parse HTML Result for HTTP Header Value
- Parse HTML Result for HTTP Attribute Value
- Parse HTML Result for LISA Tags
- Parse HTML Result and Select Random Attribute Value
- Parse HTML Result into List of Attributes
- Parse HTTP Header Cookies

- Dynamic Form Filter
- Parse HTML Result for Searching Tag/Attribute's Values

XML Filters

- Parse text from XML
- Read Attributes from XML Tag
- Parse XML Result for LISA Tags
- Choose Random XML Attribute
- XML XPath Filter

Web 2.0

- Web 2.0 Element Filter
- Web 2.0 Text Filter
- Web 2.0 Attribute Filter
- Web 2.0 Java Script Filter
- Web 2.0 Function Filter
- Web 2.0 Composite Filter

Java

- Override "Last Response" Property
- Store Step Response
- Save Property Value to File

VSE

- Standard Data Protocol Filter
- Dynamic Data Protocol Filter

Pathfinder

- LISA Integration support for Pathfinder
- LISA Integration support for webMethods Integration Server

Note: All these Filter types are described in detail in Chapter 4 Filters in the [LISA Reference Guide](#).

8. Adding Assertions

8. Adding Assertions

An **Assertion** is a LISA code element that runs after a step and all its Filters have run, to verify that the results from running the step match expectations.

The result of an Assertion is **Boolean - either true or false**, (there are no other possibilities).

The outcome may determine whether the test step passes or fails, and also determines the next step to run in the Test Case. An Assertion is used to dynamically alter the Test Case workflow by introducing conditional logic (branching) into the workflow – very much like an 'if' conditional block programming.

For example, you might create an Assertion for a JDBC step that ensures that only one row in the result set contains a specific username. If the results of the JDBC step contain more than one row, the Assertion changes the next step to execute. In this way, **an Assertion provides conditional functionality**.

The Test Case flow is usually modeled with one of the following two possibilities:

- The next step defined for each step, is the next logical step in the Test Case – in which case the Assertions are pointing to failure;
- The next step is set to fail, and the Assertions all point to the next logical step.

Usually in a Test Case model, either the next step defined for each step, is the next logical step in the Test Case – in which case the Assertions are pointing to failure; or the next step is set to fail, and the Assertions all point to the next logical step. The choice will depend, for the most part, on the actual logic being employed.

You can add as many Assertions as you need, giving you the capability to build a workflow of any complexity that you desire. **Nothing except for Assertions can change the LISA workflow.**

Note: The Assertions are executed in the order that they appear, and the workflow logic will usually depend on the order that the Assertions are applied.

Once an Assertion fires, the next step can be configured and is determined by that Assertion, and the remaining Assertions are ignored. An **Event** is generated every time an Assertion is evaluated and fired.

The following topics are available in this chapter.

10.1 Adding Assertions
10.2 Assertions Toolbar
10.3 Deleting Assertions
10.4 Reordering Assertions
10.5 Renaming Assertions
10.6 Drag & Drop Assertions
10.7 Configuring Next Step of an Assertion
10.8 Types of Assertions

8.1 Adding Assertions

8.1 Adding Assertions

LISA provides several ways to Add Assertions into the Test Case:

- Adding an Assertion Manually
- Adding an Assertion from an HTTP Response
- Adding an Assertion from a JDBC Result Set
- Adding an Assertion from a Returned Java Object

All methods except the first imply using Assertions that are available for selection in the specific test step editor, as explained below.

8.1.1 Adding an Assertion Manually
8.1.2 Adding an Assertion from an HTTP Response
8.1.3 Adding an Assertion from a JDBC Result Set
8.1.4 Adding an Assertion for Returned Java Object

LISA has many built-in Assertions. For more information, see [PART 4 - Assertions](#) in the *LISA Reference Guide*.

8.1.1 Adding an Assertion Manually

8.1.1 Adding an Assertion Manually

To add an Assertion **manually**, you need to select the Assertion type from a list and enter the parameters for the Assertion.

LISA provides two ways of adding an Assertion manually (**Global** and **Step Assertion**).

The first method allows you to add an Assertion at the **Test Case level** (**Global** Assertion). Such an Assertion will be applicable to all the steps in the Test Case and is automatically run for every step in the Test Case, unless a given node is instructed otherwise.

The second method allows you to add an Assertion at the **Test Step level** (**Step** Assertion). An Assertion added using this method will be applicable only to that step and will execute for that step only.

Add a Global Assertion

To add a Global Assertion,

Open a Test Case and in the right panel click **Global Assertions** element.

You can apply following types of Global Assertions:

HTTP –

- Simple Web Assertion
- Check Links on Web Responses

XML –

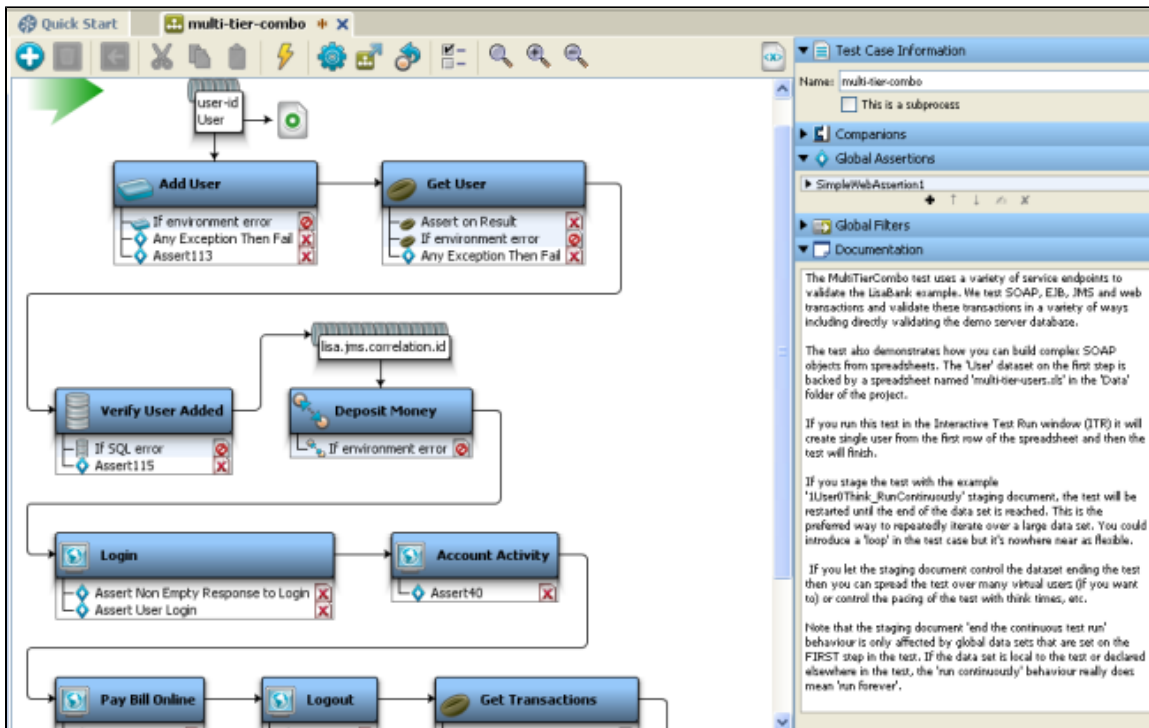
- Ensure Step Response Time

Others –

- Ensure Result Contains Expression

- Ensure Step Response Time
- Scan a File for Content

For the purposes of illustration, we have shown the presence of Global Assertion applied to the **multi-tier-combo** Test Case.

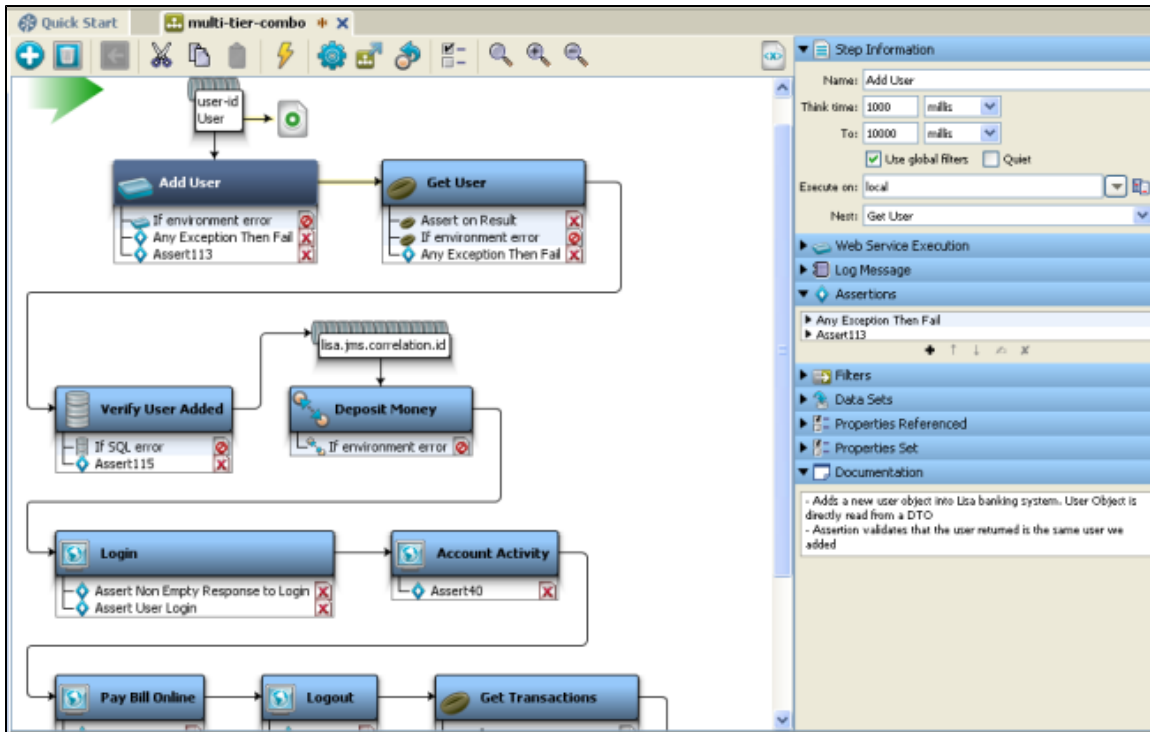


Add a Step Assertion

To add a Step Assertion,

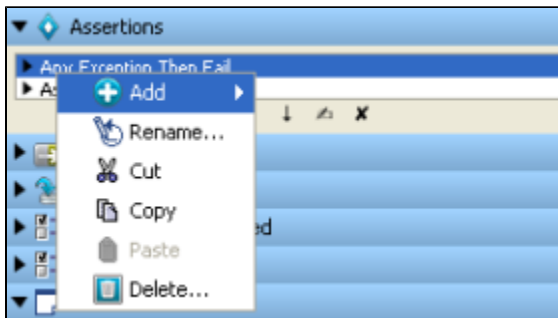
- Select the step for which you want to apply the Assertion and in the right panel click the Assertion Element.
- or right click the step and select "Add Assertion" and select the appropriate Assertion for the step.

For the purposes of illustration, existing Assertions are shown as applied to the **Add user** step in the **multi-tier-combo** Test Case.

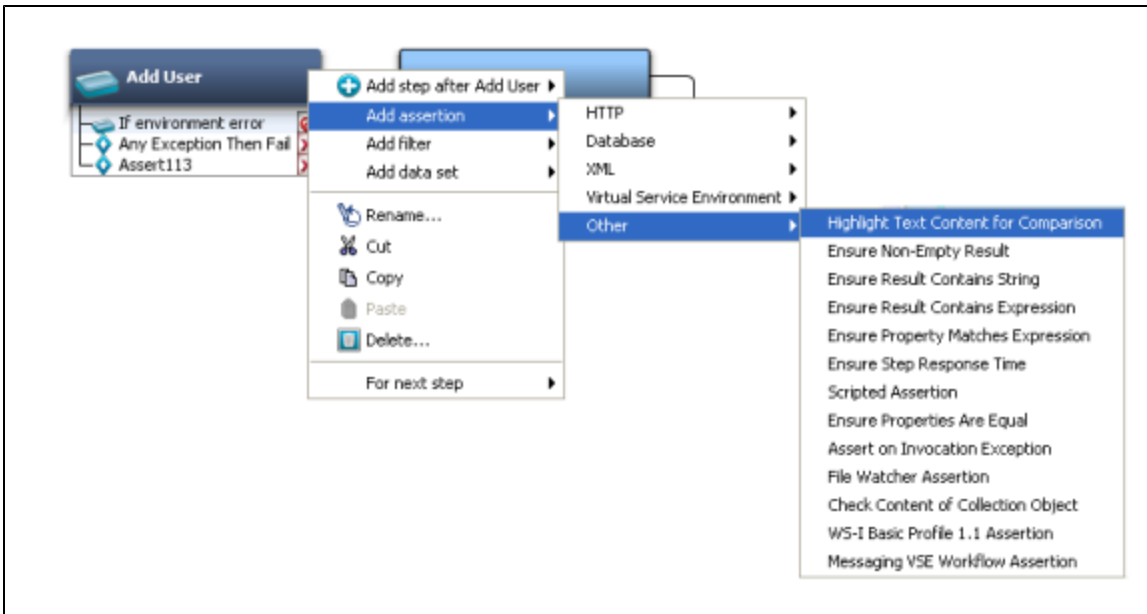


To Add an Assertion, Click on the **Add (+ sign)** icon on the Assertion toolbar.

Or you can right click an existing Assertion in the Elements tree to open a menu and click **Add** as shown below:



Or you can right click a step in the Model Editor to add an Assertion. The Assertion panel opens up and shows different kind of Assertions which can be applied to the step as below:



To Select/Edit an Assertion:

- Click the step in the Model Editor to which the Assertion is applied and/or click the Assertion related to that step in the Assertion tab.
- Double click the Assertion to open the Assertion editor. The Editor would be unique for each type of Assertion.

For the purposes of illustration, we will use the "**Assert Step Response Time Thresholds**" Assertion.

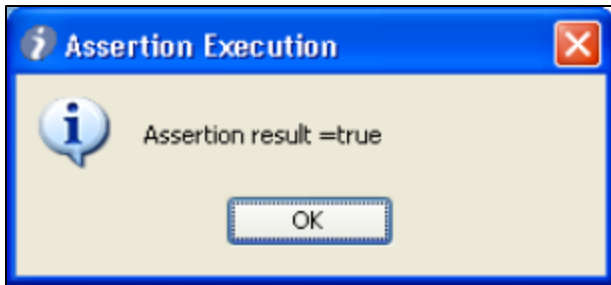
This Assertion will assert that the response time is within the defined upper and lower bounds. The Assertion editor is shown below:

While each Assertion editor will have its own custom parameters, all Assertions will include the **Base Attributes** section shown in the figure above.

To configure our Assertion, enter the following parameters:

Base Attributes:

- **Name:** The name of this Assertion. This will help you identify events for this Assertion
- **If:** Select the behavior of the Assertion using the drop down. Here you can choose the behavior of the Assertion i.e. should it fire on true, or should it fire on false.
- **Then Run:** Select the step to redirect to if the Assertion fires. All the possible steps are listed in the pull-down menu (including the End and the Fail steps)
- **Log:** The text that will be printed out as event text if the Assertion fired
- **Run Assertion:** click to Run the Assertion and get the message as below.



Assertion Specific Parameters:

- **Time must be at least (millis):** The lower bound for the response time in milliseconds. By default it is set to 0.
- **Time must not be more than (millis):** The upper bound in for the response time in milliseconds. By default it is set to -1 and is ignored if it is set to -1.

Tip: Parameters can contain properties.

In our example, if the condition that the response time is more than 5 milliseconds evaluates to false, or in other words, the response time is NOT more than 5 milliseconds, then the Assertion will fire. If the Assertion fires, then the workflow will be altered to run 'fail' step (which will fail the Test Case) instead of any other step that would have run.

All the Assertions available in LISA are described in depth in Chapter 4 Assertions in the [LISA Reference Guide](#).

8.1.2 Adding an Assertion from an HTTP Response

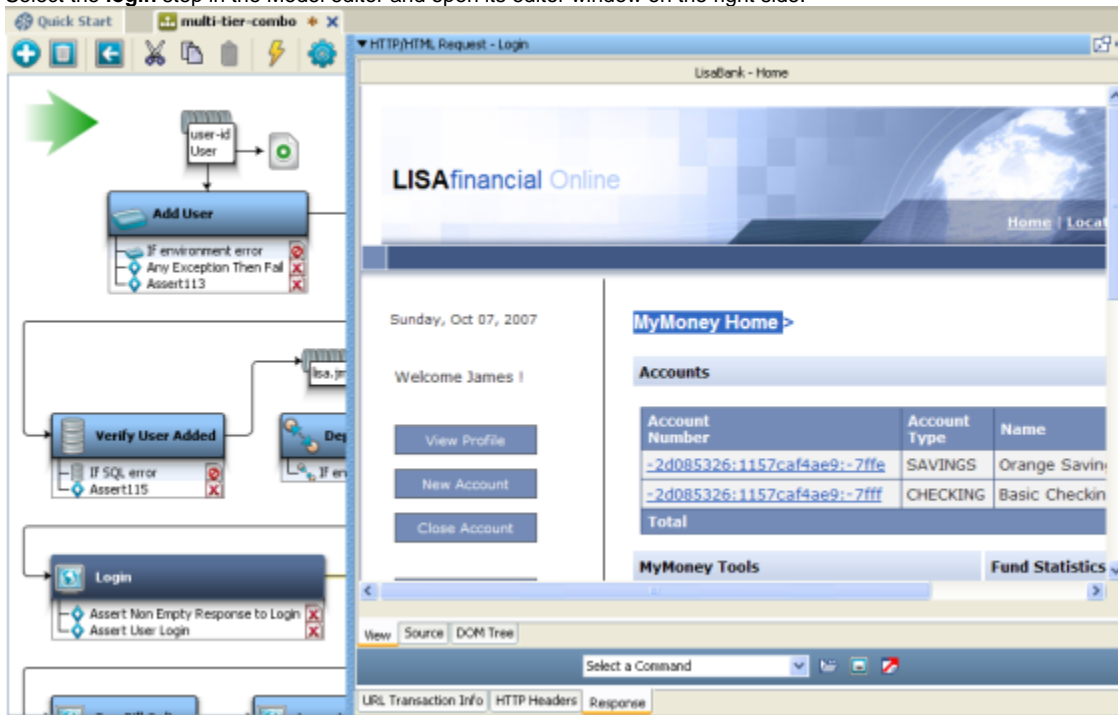
8.1.2 Adding an Assertion from an HTTP Response

When you have access to the response from an HTTP-based step, you can use the response to add an Assertion directly. The following is an example of how to add an Assertion this way.

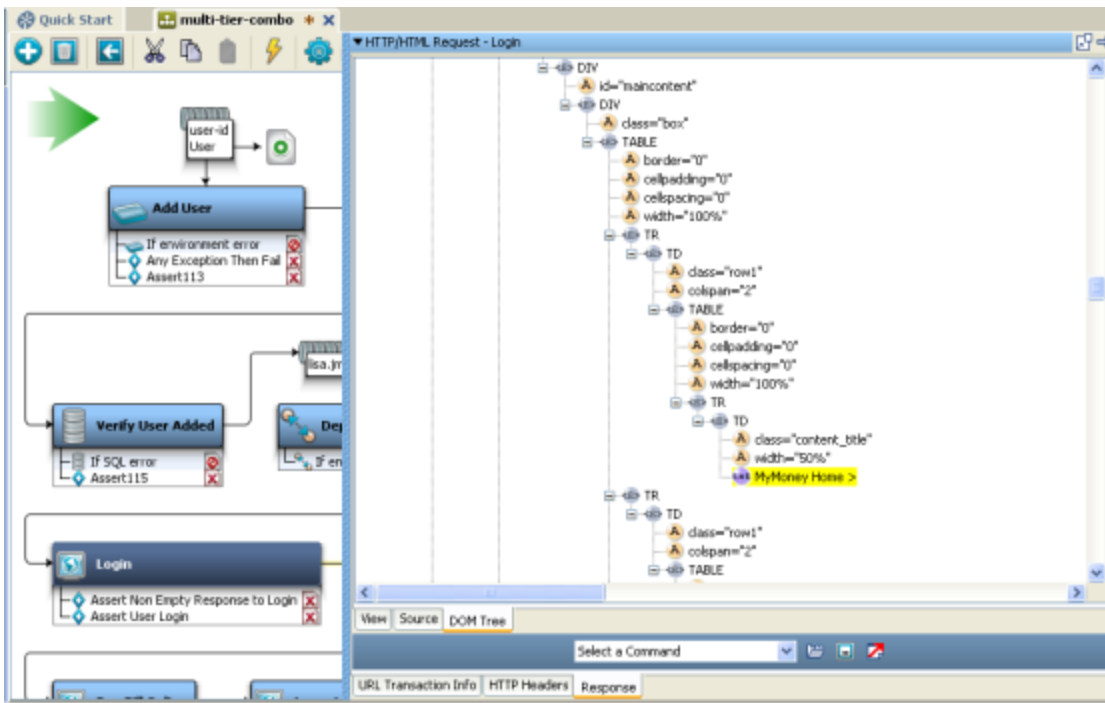
Here is an example of the HTTP/HTML response:

For the purpose of illustration, we will look at the "login" step in **multi-tier-combo** Test Case. The point of this example is to test whether the text "MyMoney Home" appears in the response.

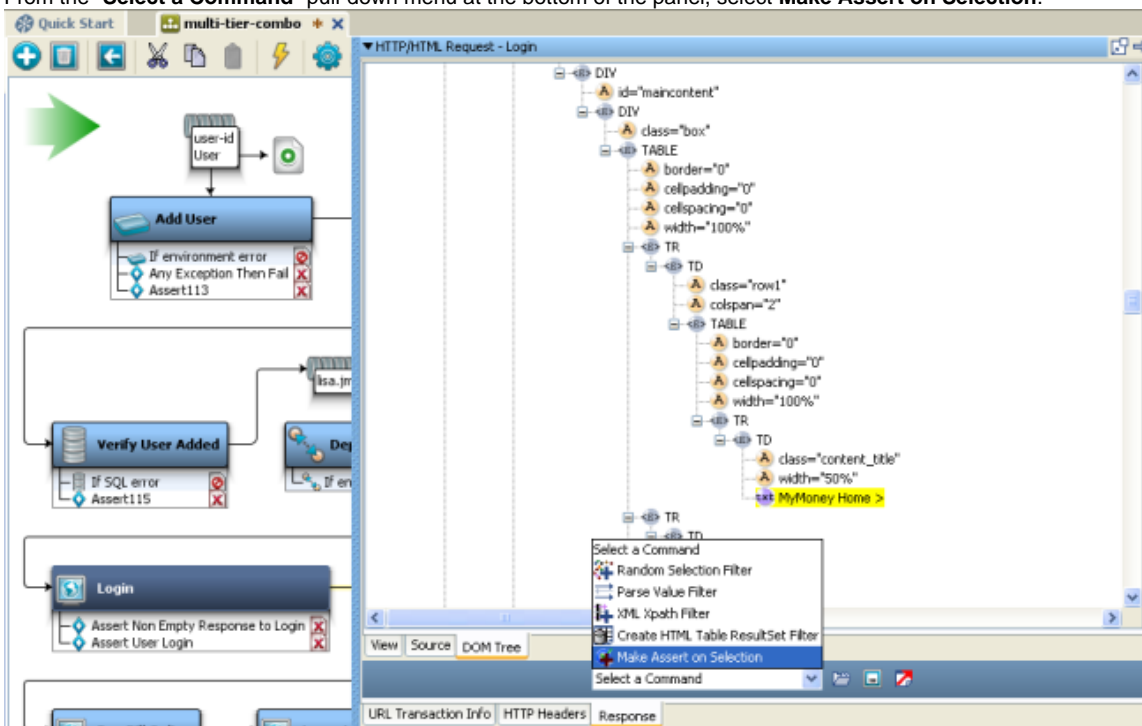
Select the **login** step in the Model editor and open its editor window on the right side.



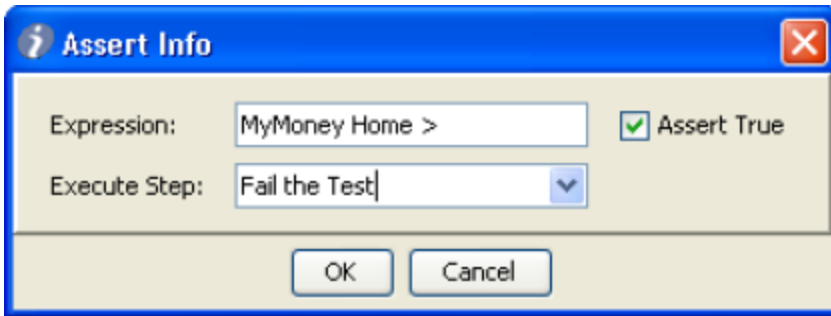
- Highlight the text "MyMoney Home" in the View tab.
- Click the DOM Tree tab to view and make sure that this text is highlighted in the tree.



- From the "Select a Command" pull-down menu at the bottom of the panel, select **Make Assert on Selection**.

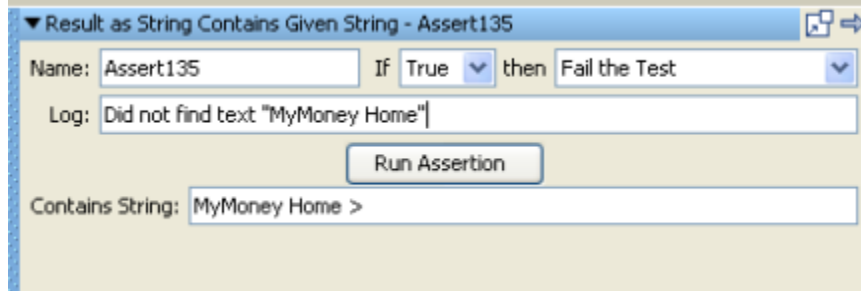


- In the pop-up window that is displayed, enter the expression that the selected text should match with, and choose the appropriate Assertion behavior.



- In our example we choose to make the Assertion fire if the text "**MyMoney Home**" is not present, and then redirect to the "**fail**" step. Click **OK** to save the Assertion.

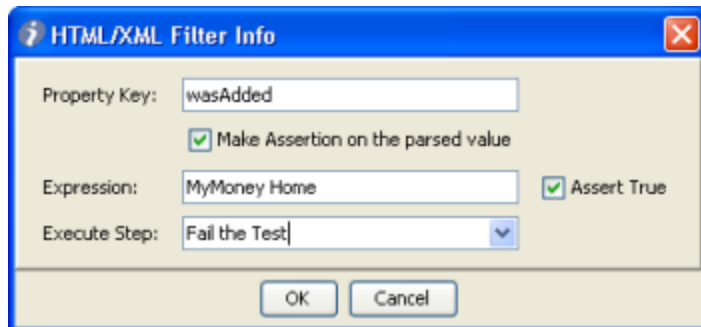
The Assertion that was generated can be seen as an Assertion in the **login** step, as shown below:



To run one Filter and one Assertion

Alternatively, if we wanted a **Filter** to capture the value "**MyMoney Home**", and then run it as an Assertion, we could use the "**Parse Value Filter**", which could do both the things as described in earlier chapter.

- The pop-up window displayed by the **Parse Value Filter** is shown below:



Where **Property Key** value is the Filter to be applied and **Expression** is the Assertion to be fired.

As a result, one Filter and one Assertion will be added to the **login** step and can be seen in the Model editor.

Tip: The same Assertion capabilities are available when an HTML response is displayed in the Step Editor.

8.1.3 Adding an Assertion from a JDBC Result Set

8.1.3 Adding an Assertion from a JDBC Result Set

When you have access to the Result Set response from a JDBC step, you can use the response to add an Assertion directly. The following is an example of how to add an Assertion this way.

Here is the example for result set response:

For the purpose of illustration, we will look at the response of **verify added user** step in **multi-tier-combo** Test Case in the examples directory (multi-tier-combo.tst).

- Select **verify added user** step in the Model Editor and double click to open its editor window.
- Select the Result Set tab and click on the cell in the result set that represents the information you want to test for (Ex: **sbellum**)

▼ SQL Database Execution (JDBC) - list users

Result Set						
LOGIN	PWD	FNAME	LNAME	EMAIL	PHONE	ROLEKEY
admin	0DPIKuNirr...	itko	Admin	lisabank-ad...	123-4567	2
sbellum	26yJsXNpI...	Sara	Bellum	sbellum@m...	232-4345	1
wpiece	/UuJ0MeQ...	Warren	Piece	wpiece@m...	455-3232	1
areck	AHdRRJD4...	Amanda	Reckonwith	areck@my...	555-2244	1
boaty	RQIil0LdpT...	Boaty	Rabbit	boaty@rab...	333-4521	1
itko	qUqF5cyx...	itko	test	itko.test@...	650-234-1...	1
lisa_simpson	60fAFoq+...	lisa	simpson	lisa.simpso...	123-456-7...	1
multitier-30...	AL2slstNsa...					

Base Result Set Generate Assertion For Cell's Value

- Click the **Generate Assertions for Cell's Value** icon in the toolbar below the Result set window as shown above.

We want to test that "sbellum" appears in a cell in the **LOGIN** column.

- In the dialog box that opens, enter the test step (**fail**) to redirect if the value is not found:

Generate JDBC Result Set Value Assertion

Please select the Step to execute when this cell value is not found.

fail

OK Cancel

LISA will create an Assertion called **Result Set Contents** in the **verify added user** step.

▼ ResultSet Contents - Assert221

Name: Test for User If False then Fail the Test

Log: User Not Found

Run Assertion

Column (1-based or Name): 1

Regular Expression: sbellum

Note: The same Assertion capabilities are available when a JDBC result set is displayed in the Step Editor.

8.1.4 Adding an Assertion for Returned Java Object

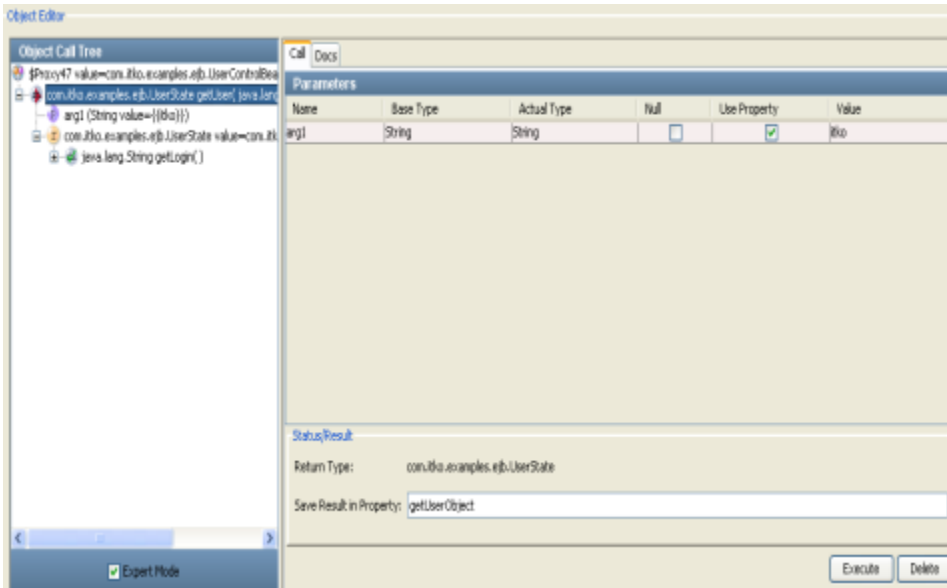
8.1.4 Adding an Assertion for Returned Java Object

When the result of your test step is a Java object, you can use the inline Assertion panel in the Complex Object Editor to add an Assertion on the returned value from the method call directly. The following is an example of how to add an Assertion this way.

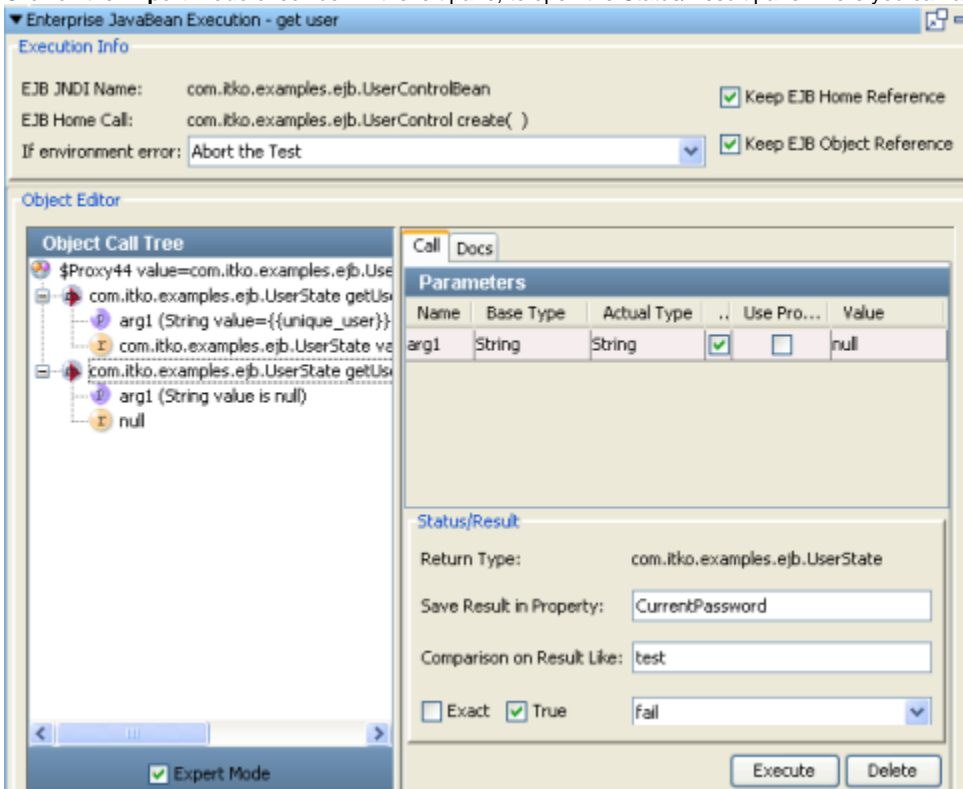
Here is an example of an object in the complex object editor:

For the purpose of illustration, we will look at the **get user** (an EJB step) step in **multi-tier-combo** Test Case in the examples directory (multi-tier-combo.tst).

We have entered an input parameter **itko**, and executed the method call **getUser**. We are now about to execute the **getPwd** call on the **UserState object** that was returned from that call.



- Click on the **Expert Mode** check box in the left pane, to open the Status/Result pane where you can add the Assertion.



- The returned value upon executing the **getPwd** method will be stored in the property **CurrentPassword**.
- We then add an Assertion that tests to see if the returned value is equal to the String **'test'**. If it is not that, then we redirect to the **Fail**

- step.
- Click **Execute** to Execute this step.

Note: In-line Assertions (and Filters) do not result in an Assertion being added to the test step. In-line Assertion management is always done in the complex object editor.

For old test cases, all inline Assertions that were set to **Fail** step, will now change to **Abort** step.

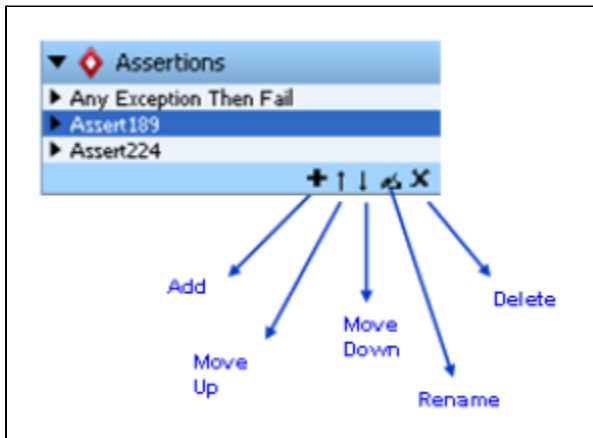
For more details on the complex object editor see [Using Complex Object Editor \(COE\)](#).

8.2 Assertions Toolbar

8.2 Assertions Toolbar

All the elements have their own toolbar to Add/Delete/Reorder at the bottom of the Element.

Assertions can also be renamed as shown below:

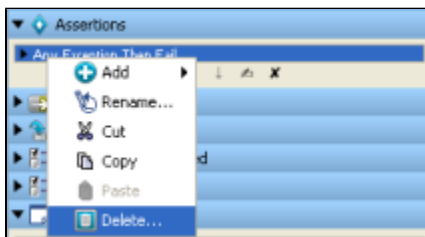


8.3 Deleting Assertions

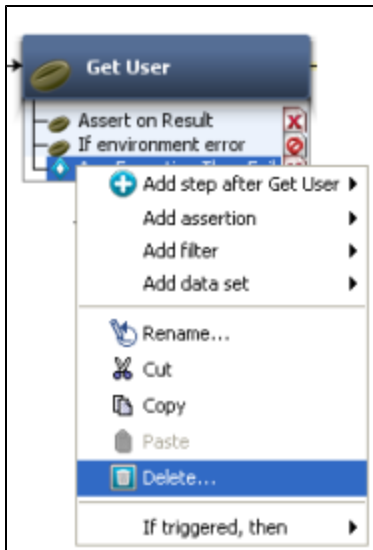
8.3 Deleting Assertions

To delete an Assertion:

- Right Click any Assertion in the Elements tab to open a menu. Click **Delete** to delete the Assertion.



- Select the Assertion in the test step and right click to open a menu. Click Delete to delete the Assertion.



- Select the Assertion in the Elements tab and click the **Delete (X)** icon on the toolbar as shown above.

8.4 Reordering Assertions

8.4 Reordering Assertions

Reordering Assertions is needed since Assertions are evaluated in the order that they appear in. Thus, changing the order of the Assertions can affect the workflow.

To reorder an Assertion:

- Select the Assertion in the Elements tab and click the **"Move Up or Move Down"** icon on the toolbar as shown above.
- Drag and drop the Assertions in the Model Editor to the desired location as explained below.

8.5 Renaming Assertions

8.5 Renaming Assertions

To rename an Assertion:

- Select the Assertion and Right click to open a menu. Click **Rename** to rename the Assertion.
- Select the Assertion and click the **Rename** icon on the toolbar as shown above.

8.6 Drag & Drop Assertions

8.6 Drag & Drop Assertions

You can drag and drop Assertions in the Model Editor from one Test Step to the other.

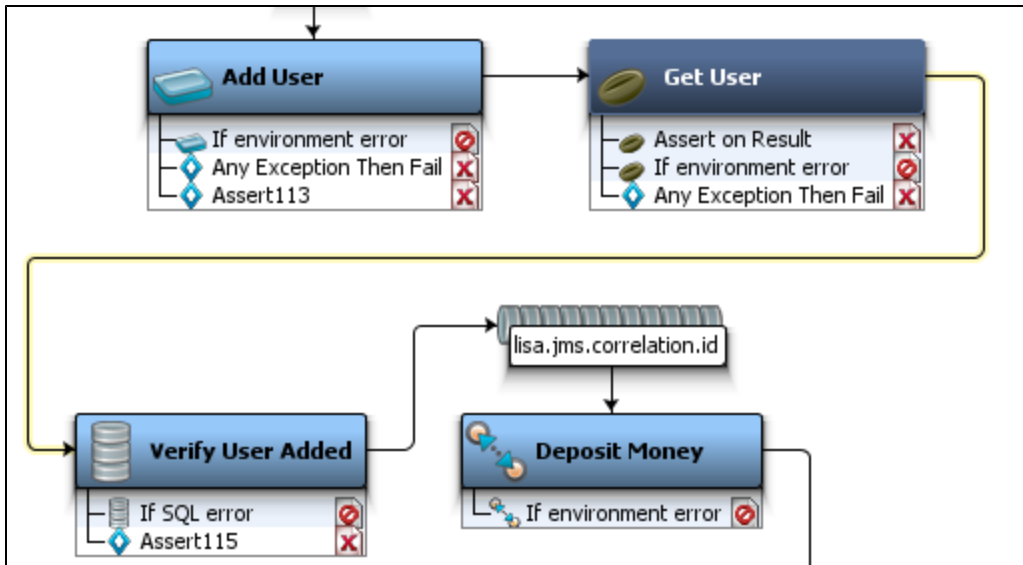
- Click on the Assertion in one test step – say Step 1.
- Select and Drag the Assertion to other Test step – say Step 2, in the Model Editor.
- The dragged Assertion will then be applied to Step2.

8.7 Configuring Next Step of an Assertion

8.7 Configuring Next Step of an Assertion

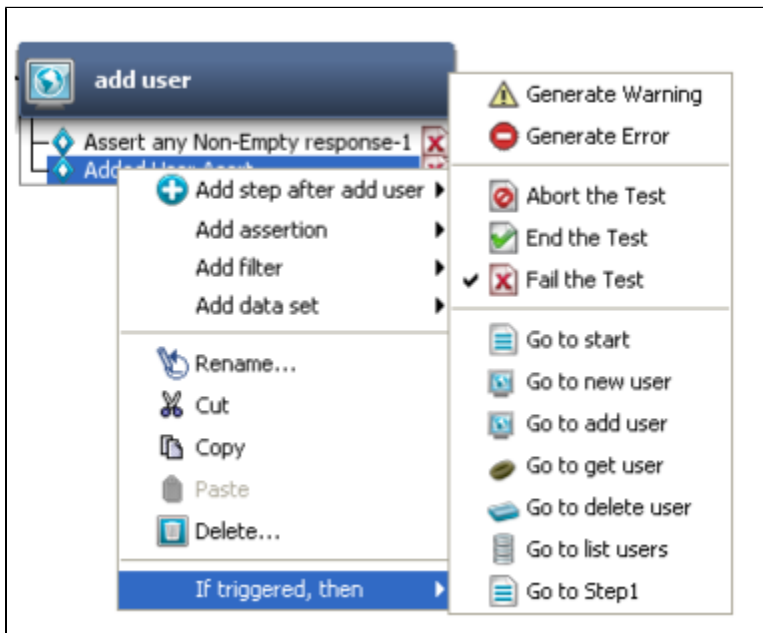
The Assertion added to a step can be seen once you select that step:

The Assertion can be seen highlighted as shown below:



Once the Assertion is added to a step, you can now select its next step to be executed, incase you want the workflow to be altered.

- Right click the Assertion in the Model Editor to open the menu as shown below:



- Click "If Triggered, then" menu and select the next step to be executed.

There are different steps in this: [Generate Error and Warning Step](#) other than Abort, End and Fail Steps.

8.8 Types of Assertions

8.8 Types of Assertions

LISA provides the following Assertions:

HTTP	▶
Database	▶
Web 2.0	▶
XML	▶
Virtual Service Environment	▶
Other	▶

http

- Highlight HTML Content for Comparison
- Check HTML for Properties in Page
- Ensure HTTP Header Contains Expression
- Check HTTP Response Code
- Simple Web Assertion
- Check Links on Web Responses

database

- Ensure Result Set size
- Ensure Result Set Contains Expression

web20

- Web 2.0 Basic Assertion
- Web 2.0 Validation Assertion
- Web 2.0 Branching Assertion

xml

- Highlight Text Content for Comparison
- Ensure Result Contains String
- Ensure Step Response Time
- Graphical XML Side-by-Side Comparison.
- XML Side-by-Side Comparison
- XML Xpath Assertion
- Ensure XML Validation

VSE

- Assert on Execution Mode.

Other

- Highlight Text Content for Comparison
- Ensure Non-Empty Result
- Ensure Result Contains String
- Ensure Result Contains Expression
- Ensure Property Matches Expression
- Ensure Step Response Time
- Scripted Assertion
- Ensure Properties are Equal
- Assert on Invocation Exception
- File Watcher Assertion
- Check Content of Collection Object
- WS-I Basic Profile 1.1 Assertion
- Messaging VSE Workflow Assertion.

All these Assertion types are described in detail in Chapter 4 Assertions in the [LISA Reference Guide](#).

9. Using Data Sets

Using Data Sets

A **Data Set** is a collection of values that can be used to set properties in a Test Case while a test is **running**. This provides a mechanism to introduce external test data to a Test Case.

Data Sets are often rows of data that can be inserted into LISA properties as **Name-Value** pairs, but this is not always the case, sometimes a Data Set will return a single property value.

Data Sets can be created internal to LISA, or externally; for instance in a file or a database table.

While a test is running, LISA will assign properties to the steps specified in the Data Set editor. When the last data value(s) are read from the Data Set, the data can be re-used starting at the top of the Data Set, or the test can be re-directed to any step in the Test Case.

[Local Data Sets](#)

The following topics are available in this Chapter.

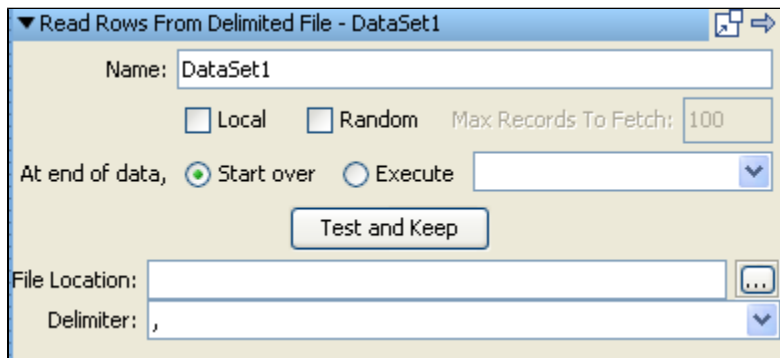
- 11.1 Adding a Data Set
- 11.2 Random Data Set
- 11.3 Example Scenarios
- 11.4 Data Set Toolbar
- 11.5 Deleting a Data Set
- 11.6 Reordering a Data Set
- 11.7 Renaming a Data Set
- 11.8 Drag and Drop Data Set
- 11.9 Data Set Next Step Selection
- 11.10 Data Set Types

9.1 Global and Local Data Set

9.1 Global and Local Data Set

Global Data Set

By default, a Data set is **Global**  (The Local check box is unchecked).

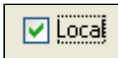


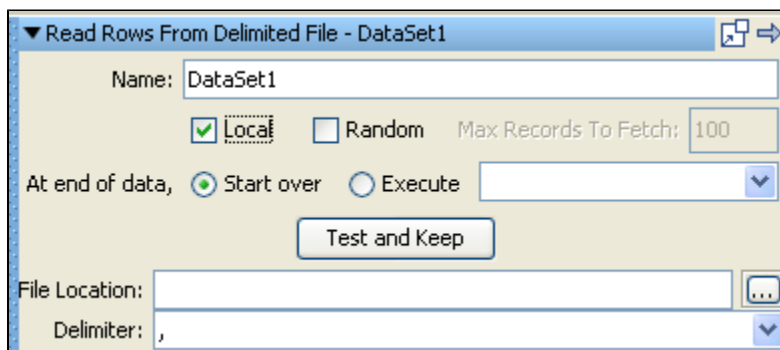
The Coordinator server is responsible to provide data to all the test steps.

Here, all the model instances share a single instance of the Data Set.

The Global Data Set is shared and applied to all instances of the model, even if they are run in different Simulators.

Local Data Set

You can make the Data Set **Local**  by checking the Local check box while building the Data Set.



If "Local" is checked, then each instance gets (essentially) its own copy of the Data Set.

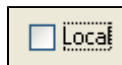
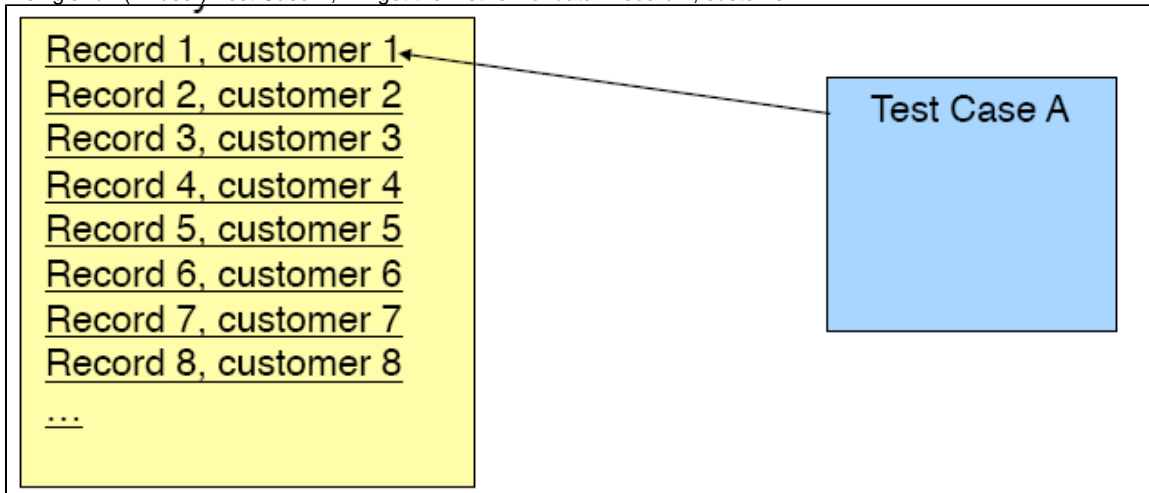
A local Data Set will provide one copy of the Data Set to each instance being run.

Example

We have 3 concurrent virtual users, a local Data Set with 100 rows of data, and a Test Case that loops over 100 rows of data and stops. Each virtual user will see all the 100 rows of data in the Data Set.

For a Local Dataset -

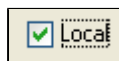
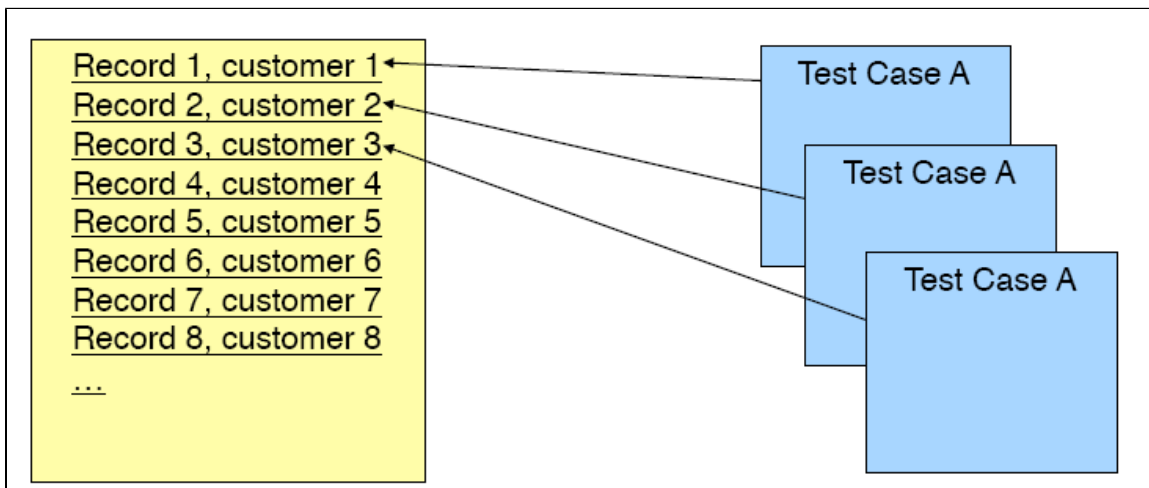
A single run (1 vuser): Test Case A, will get the first row of data: Record 1, customer 1



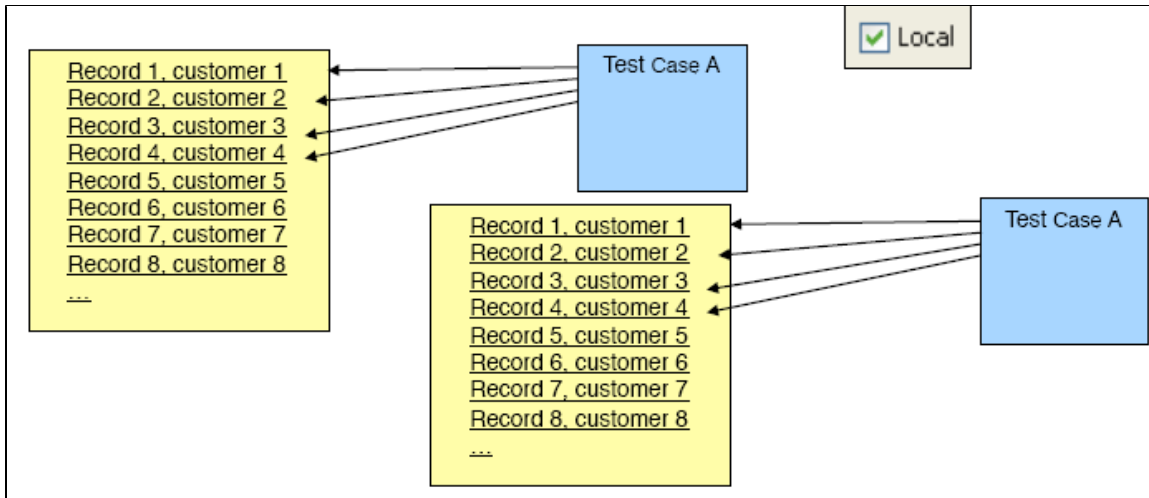
A data set that is shared across virtual users is Global

For a Global Data set:

When **Test Case A** is staged with 3 virtual users or staged to run continuously, each test case will get the next row of data. LISA will share a data set across multiple runs given the instructions specified in the staging document.



Each virtual user (instance) gets its own copy of the data set is Local



When local is marked and Test Case A has a loop, the test will read every row of data in the data set. Each run will get its own copy of the data set.

Test Case Looping

Often the data in a Data Set drives the number of times a Test Case is run.

This can be implemented several ways:

1. A test is set to finish when all the data in the Data Set has been exhausted.
2. A test is set to re-use the Data Set when the data has been exhausted.
3. A test step can call itself, or a series of steps can be configured in a loop that runs until the data in the Data Set has been exhausted.
4. A numeric counter data set can be used to cause a specific step to execute a fixed number of times, which can be set at the step-level or test case-level"

For example, consider a test where we want to test the login functionality of our application using 100 user ID/password pairs. This can be achieved very easily by having a single step call itself until the Data Set (with 100 rows of data) has been exhausted; at which point it can redirect to the next natural step in the Test Case, or perhaps the End step. Alternatively, we could use a counter Data Set in conjunction with our userID/password Data Set.

Note: If a test case contains a global Data Set on the first step and the Data Set is set to end the test when the data set is drained, then - all instances of the test will end for a staged run, overriding any other staging parameters such as steady state time. Local Data Sets will not end the staged run in this fashion nor will Data Sets on steps other than the first test.

9.2 Random Data Set

9.2 Random Data Set

A Random Set is a special type of Data Set, which can be thought as a wrapper around another Data Set.

In case of Random Data Sets, you can choose a Data Set to be randomized for specific steps, & also choose the max number of records for randomization.

☒ Random
 Max Records To Fetch:

Random data sets work as follows:

When a Random wraps a DataSet, N copies of the dataset are added to the Random's list, where N = Max Number of Rows. So when N=10, 10 copies of the rows from the dataset are made.

When a step references the Random data set, a random row is selected from the data set.

Let's say you have a Random data set named 'RAND1', and it reads one column named 'Fruit', and the maxRows is set to '10'.

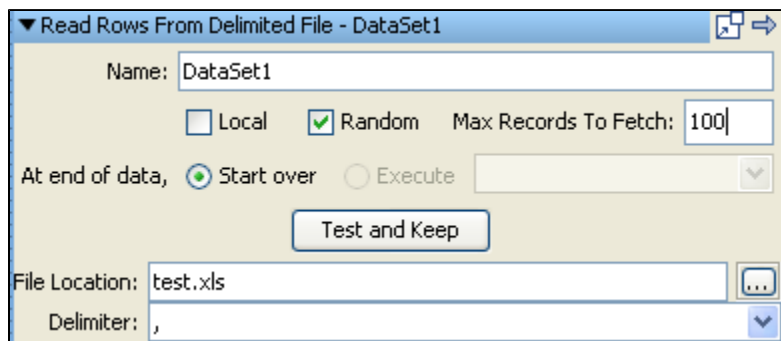
RAND1 will be the random number generated to pick a row; it'll be in the range 0 through N-1. Fruit will be the value of 'Fruit' column found in the that row.

To make a Data Set Random

Select the Data Set which you want to make as Random.

For the purpose of illustration, we again look at the "Read Rows from Delimited File" Data Set.

The Data Set editor opens as shown below:



All other fields are same as the Data Set Editor explained earlier.

- Check the **Random** check box to make this Data Set Random.
- Enter the **Max Record to Fetch** number.

Max Record to Fetch is the maximum number of records you want from the Data Set. If this is larger than the records in the Data Set, the smaller number will be used to create the Random Set.

9.3 Example Scenarios

9.3 Example Scenarios

To show the behavior of tests using Data Sets, consider the following scenarios:

1>A test has fifteen virtual users, and a Data Set has two rows of data.

Scenario 1: At the end of data -> Start over.

The first user would read the first row of data, the second user would read the second row. The third user would start over and read the first row, etc. This would continue until all fifteen users have run the test. The first row was read eight times, the row seven times.

Scenario 2: At end of data -> Execute End step.

The first user would read the first row of data, the second user would read the second row. The third user through the fifteenth user would start the test, and immediately jump to the End step.

2>A test has 100 virtual users, and the Data Set has 1500 rows of data. Tests are running concurrently.

Scenario 1: At the end of data -> Start over.

When the 100 users start, they would read the first 100 rows of data. Depending on the staging document, as cycles end the users would start new runs of the Test Case and consume more rows of the Data Set. If all rows are consumed, and the test run has not ended, it would start over with the first row again until the test run ends.

Scenario 2: At end of data -> Execute End step.

The test run would end after 1500 cycles.

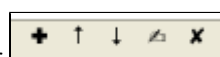
3>A test has ten virtual users, and the Data Set has 10,000 rows of data. The staging document specifies that the test should run for two minutes.

The 10 users will start and read the first 10 rows of data. They will continue consuming rows of data from the 10,000 rows, and depending on how fast they run would determine how far down the Data Set they go. At the two minute mark, the test would end.

9.4 Adding a Data Set

9.4 Adding a Data Set

Data Set Toolbar

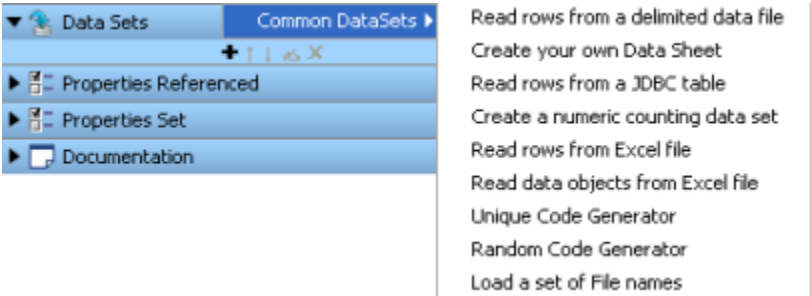


At the bottom of every element, there is a toolbar

which has icons to Add/Delete/Reorder/Rename as shown above and is self explanatory.

To Add a Data Set

- Click and expand the **Data Sets** tab in the right panel.
- Click the **Add** icon in the Data Sets tab, to open the Data Set panel listing the **Common Data Sets** in LISA as shown below:

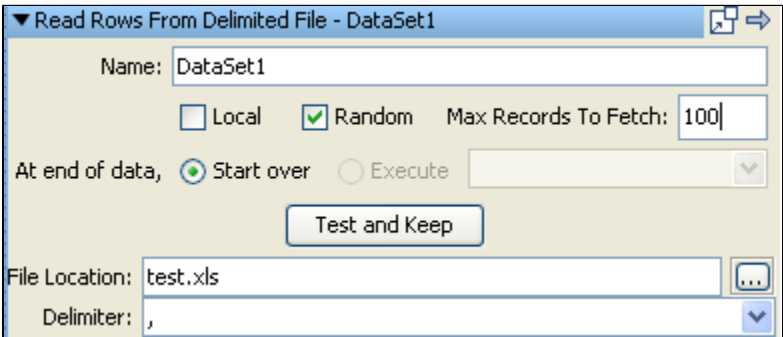


- Click the required Data Set to open the appropriate **Data Set Editor**. The editor would be specific to each Data Set.

All the Data Set types available in LISA are described in depth in Chapter 5 Data Sets in the [LISA Reference Guide](#).

For the purpose of illustration, we will use the "Read rows from a delimited data file" Data set in this section.

The editor for this Data Set is shown below:



This Data Set editor consist of:

Top panel: This panel is common to all Data Set types. It consists of:

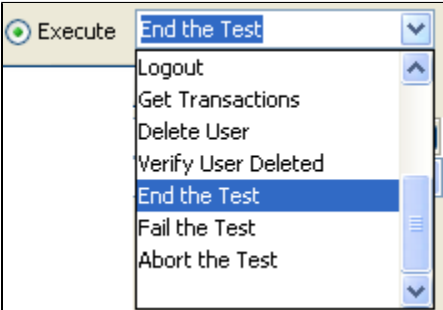
Name: The name of the Data Set.

Local: Check the textbox if you want a local Data Set. The default is global (unchecked)

Random: Click the Random check box, if you want to make this a Random Data Set and enter the Max records to fetch number.

At End of Data: Instructions for how to proceed after all the data has been read.
There are two options:

- **Start Over:** Continue reading data from the top of the Data Set
- **Execute:** Choose the step to execute after all the data has been read. The pull-down menu has been pre-populated with all the available



steps in the Test Case like:

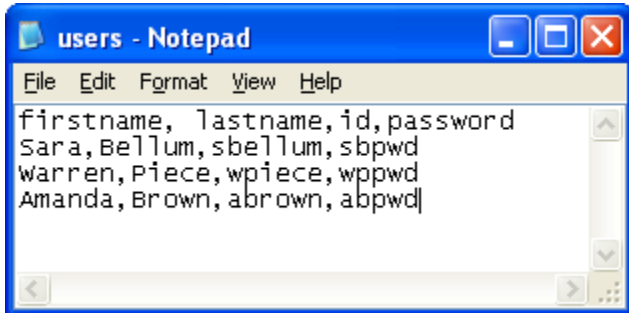
Test & Keep... : After all the parameters have been entered click this button to test the Data Set, and to load it into the appropriate steps in the

Test Case

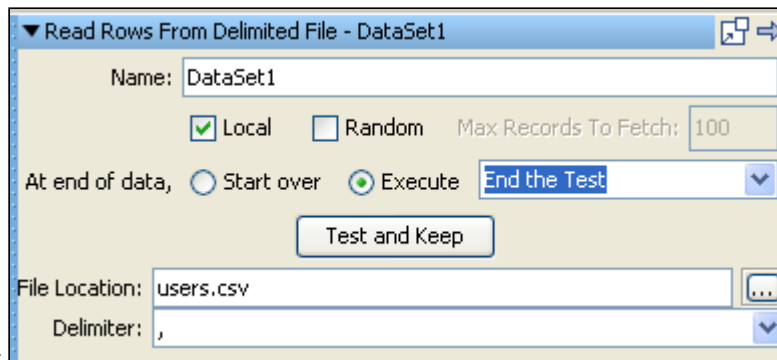
Bottom panel: This panel is specific to the Data Set being created. In the bottom left panel of the Data Set editor we enter the following:

- **File Location:** The full path name of the text file, or browse to file with the browse button. You can use a property in the path name. (for instance `LISA_HOME`)
- **Delimiter:** Enter the delimiter being used. Any value is allowed as a delimiter. LISA provides common delimiters in the pull-down menu.

For our example, we must create a comma delimited text file `users.csv` using a simple text editor:

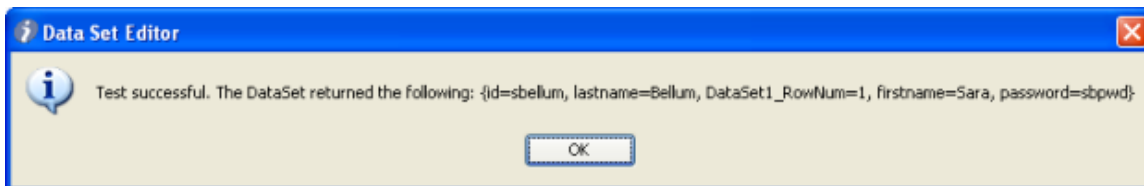


The first line in this file specifies the names of the properties: (firstName, lastName, id, password). Subsequent lines list the data values to be used for these properties.



The completed input for our example is shown below:

Our example shows a Data Set called **DataSet1** that will be used until all the rows have been used, and then the **end** step will be executed. Notice that after clicking the **Test & Keep...** button, the first row of data is loaded, and we get a popup that confirms that the Data Set can be read, and shows the first row of data as shown below:



Ending a Test Case by a DataSet

There are three conditions that must be met for a data set to end a test run:

- 1) The dataset must be Global
- 2) The dataset must be set "At end of data: Execute end"
- 3) The dataset must be incremented on the first step of the test case

Note - Be careful when you apply a Global DataSet to the first step in a test case as it will pull down the entire staged run before completing the cycles.

9.5 Deleting a Data Set

9.5 Deleting a Data Set

To delete a Data Set,

- Select the Data Set in the right panel, and click the **Delete (X)** icon on the toolbar or

- Right-click the Data Set in the workflow, and select **Delete** from the menu as shown below:



9.6 Reordering a Data Set

9.6 Reordering a Data Set

To re-order a Data Set,

- Select the Data set in right panel
- Click **Up** or **Down** arrow on the toolbar to move it up or down.

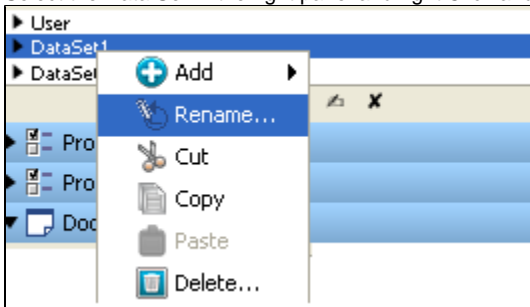



9.7 Renaming a Data Set

9.7 Renaming a Data Set

To rename a Data Set

- Select the Data Set in the workflow and right click to open a menu to rename.
- Select the Data Set in the right panel and right Click and select **Rename** to rename the Data Set.



- Select the Data Set in right panel and click the **Rename** icon  on the toolbar.

9.8 Drag and Drop Data Set

9.8 Drag and Drop Data Set

You can drag and drop Data Sets in the Model Editor from one Test Step to the other.

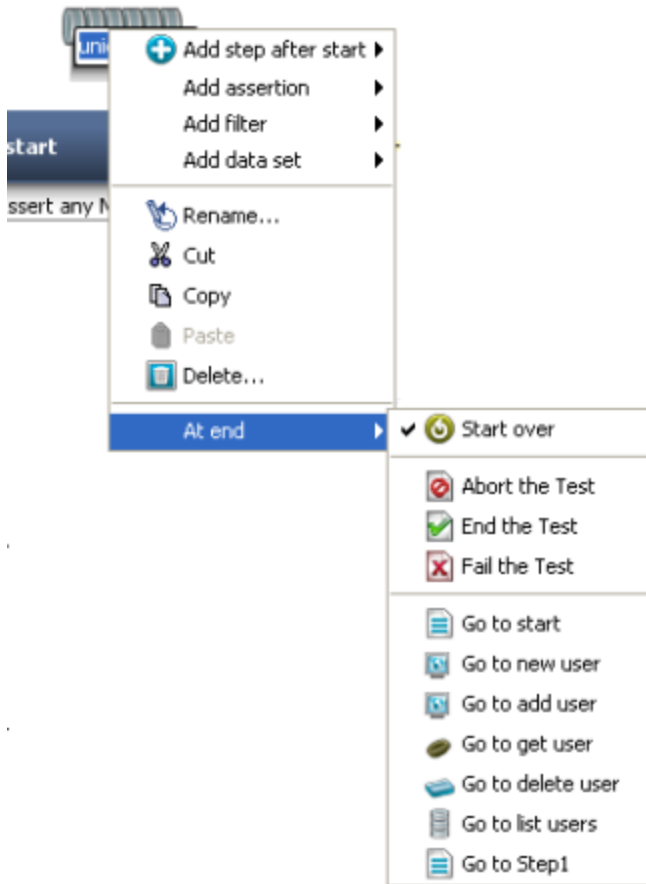
- Click on the Data Set attached to a Test Step --say Step 1.
- Select, Drag and Drop that Data Set to other Test step – say Step 2, in the Model Editor.
- The dragged Data Set will then be applied to Step2.

9.9 Data Set Next Step Selection

9.9 Data Set Next Step Selection

Once the Data Set is fired, you can select the next step to be executed.

- Right click the Data Set in the Model Editor to open the menu as shown below:



- Click the **At end** menu and select the next step to be executed.

9.10 Data Set and Properties

9.10 Data Set and Properties

Data Sets can use properties.

There are some major uses of properties:

- When specifying the location of an external Data Set, we can use a property, perhaps LISA_HOME rather than a hard coded value for the path name. For example - **LISA_HOME/myTests/myDataset.csv**

This increases the portability of the Data Set. Properties used in this way must be specified as a system property, or be defined in a Configuration, since they usually have to be available early in the test run. You can define a dummy value in your Configuration and modify it at a latter date. This will allow the file to be found in a design time. This use of properties is important when you are running your tests on LISA Server.

- Data set values can contain properties. These will be evaluated when the value is read. For example: we could set up a login value in the Data Set as student _1. Then, if the current value of **student** is "**Bart**" the resulting data value becomes "**Bart_1**".
- Local Data Sets can sue properties from the Test Case, but Global Data Sets cannot since they are shared by all virtual users.

9.11 Data Set Types

9.11 Data Set Types

LISA provides the following Data Set types:

Read rows from a delimited data file
Create your own Data Sheet
Create your own set of Large Data
Read rows from a JDBC table
Create a numeric counting data set
Read rows from Excel file
Read data objects from Excel file
Unique Code Generator
Random Code Generator
Load a set of File names
XML Data Set

- Read Rows from a Delimited Data File
- Create your own Data Sheet
- Create your own set of Large Data
- Read Rows from a JDBC table (Read Rows from JDBC result set)
- Create a numeric counting Data Set (Generate Numeric values counter)
- Read rows from Excel file
- Read Data Objects from Excel file (Read DTO's from Excel file)
- Unique Code Generator
- Random Code Generator
- Load a set of File names
- XML Data Set

All these Data Set types are described in detail in *Data Sets* in the [LISA Reference Guide](#).

10. Using Companions

10. Using Companions

A **Companion** is a LISA element that runs before and/or after every Test Case execution. Companions are used to configure behavior that is **global** to the Test Case. These behaviors include simulating browser bandwidth and browser type, setting synchronization points in load tests and reading properties from an external file. In a way, companions set some kind of a context for the Test Case execution.

Companions work as helpers for the test case - before any of the steps are executed.

The following topics are available in this Chapter.

[12.1 Adding a Companion](#)
[12.2 Companion Toolbar](#)
[12.3 Deleting a Companion](#)
[12.4 Reordering Companions](#)
[12.5 Companion Types](#)
[12.6 LISA Hooks](#)

10.1 Adding a Companion

10.1 Adding a Companion

To add a Companion,

Open a new Test case and click on the **Companion** Element in the right panel.

This will open the Companion main menu and consequently the sub menu as shown below:=



Click the required Companion to open the appropriate Editor.

All the Companions available in LISA are described in depth in the [LISA Reference Guide](#).

For the purposes of illustration, we will configure the "Create a Synchronization Point" Companion.

This companion is used in a load test scenario, when you need all the running users to execute a step at a particular time.

The Companion editor for this Companion is shown below:

Note: Each Companion will have its own parameters and editor.

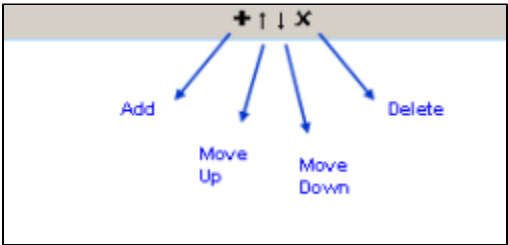
To configure the Companion enter the following parameters:

Parameter	Description
Sync point Name	The name of the sync point.
At Step	Select the Step at which you want to sync
Time out secs	Enter the time out time in seconds

10.2 Companion Toolbar

10.2 Companion Toolbar

At the bottom of every element, there is a toolbar which has icons to Add/Delete/Reorder as shown below and is self explanatory.



10.3 Deleting a Companion

10.3 Deleting a Companion

To delete a Companion:

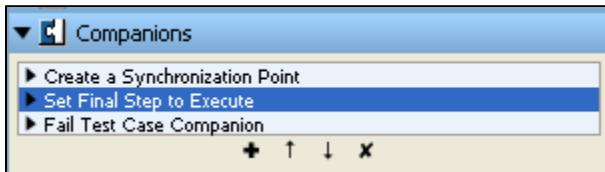
- Select the Companion from the Companion tab, and click the **Delete (X)** icon in the toolbar Or
- Right-click the Companion and click **Delete** from the menu.

10.4 Reordering Companions

10.4 Reordering Companions

To reorder a Companion:

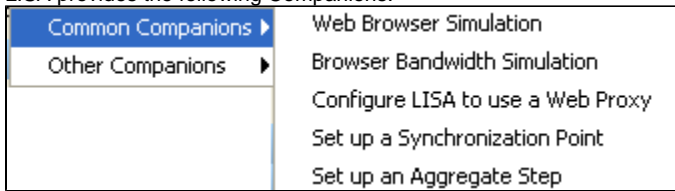
- Open the Companion tab and select the Companion to be moved.
- Click the **Move up or Down** icon in the toolbar to set its position as shown below:



10.5 Companion Types

10.5 Companion Types

LISA provides the following Companions:



Common Companions:

- Web Browser Simulation
- Browser Bandwidth Simulation (Bandwidth Simulation)
- Configure LISA to use a Web Proxy (Web Proxy Setup)
- Set up a Synchronization Point (Create a Synchronization Point)
- Set up an Aggregate Step (Aggregate Transaction)

Other Companions:

- Create a Sandbox Class Loader for Each Test
- Set Final Step to Execute
- Negative Testing Companion
- Fail Test Case Companion
- XML diff Ignored Node Comparison

LISA has many built-in Companions. For details on all these, see [_Companions](#) in the [LISA Reference Guide](#).

10.6 LISA Hooks

10.6 LISA Hooks

Hooks work similar to Companions; they run before a test starts and after a test finishes.

The difference is, hooks are defined at the **LISA application level**, and every Test Case in LISA runs the hooks. There is no option not to run a hook that is added to LISA, for any Test Case.

A hook is a mechanism in LISA that allows for the automatic inclusion of test setup and/or teardown logic for all the tests running in LISA. An alternate definition of a hook is a **system-wide Companion**.

Hooks are used to configure test environments, prevent tests from executing that are not properly configured or do not follow defined best practices, or simply to provide common operations.

Hooks are Java classes that are on the LISA Class path. They are registered in the **lisa.properties** file or better still, the **local.properties** file.

Anything that a hook can perform can be modeled as a Companion in LISA.

However, the following **differences between hooks and Companions** should be noted:

- Hooks are **global in scope**. Users do not specifically include a hook in their Test Case as is the required practice for Companions. Hooks are registered at the system level. If you need every test to include the logic and do not want users to accidentally not include it, a hook is a better mechanism.
- Hooks are deployed at the **LISA install level**, not at the Test Case level. If a test is run on 2 machines, 1 machine has a hook registered and the other does not, then the hook will only run when the test is staged on the machine where it is explicitly deployed. Companions

defined in the Test Case would execute regardless of any install-level configuration.

- Hooks are practically **invisible** to the user and therefore cannot request any custom parameters from the user. They get their parameters from properties in the Configuration or from the system. Companions can have custom parameters since they are rendered in the Model Editor.

For more details on hooks, see [LISA Advanced Features](#).

11. Using Complex Object Editor (COE)

Using Complex Object Editor (COE)

The LISA **Complex Object Editor (COE)** is an editor which allows you to interact with **Java objects** without having to write any additional Java code.

You can change the current value of an input parameter, make method calls, and examine return values with the help of the Object Editor. Within the editor you can do simple in-line filtering and/or add simple Assertions on the return value.

Many of the test steps in a Test Case involve the **manipulation of java objects**. Whether you are working directly with Java objects, such as in a Dynamic Java Execution step, or an Enterprise Java Bean (EJB) step, or indirectly - such as input parameters to a web service, or return messages from an Enterprise Service Bus (ESB), you will be manipulating Java objects and working within the complex object editor. It is important therefore to have a solid grasp of the complex object editor.

We will start by describing how to use the COE user interface. Then we will look at several usage scenarios that get progressively more complex, since this is the best way to illustrate the various ways you can manipulate the object editor.

The following topics are available in this Chapter.

- [11.1 COE User Interface](#)
- [11.2 Object Call Tree Panel](#)
- [11.3 Data Sheet & Call Sheet Panels](#)
- [11.4 Object Interaction Panels](#)
- [11.5 Using Data Sets in the COE](#)
- [11.6 Usage Scenarios - Simple Objects](#)
- [11.7 Usage Scenarios - Complex Objects](#)

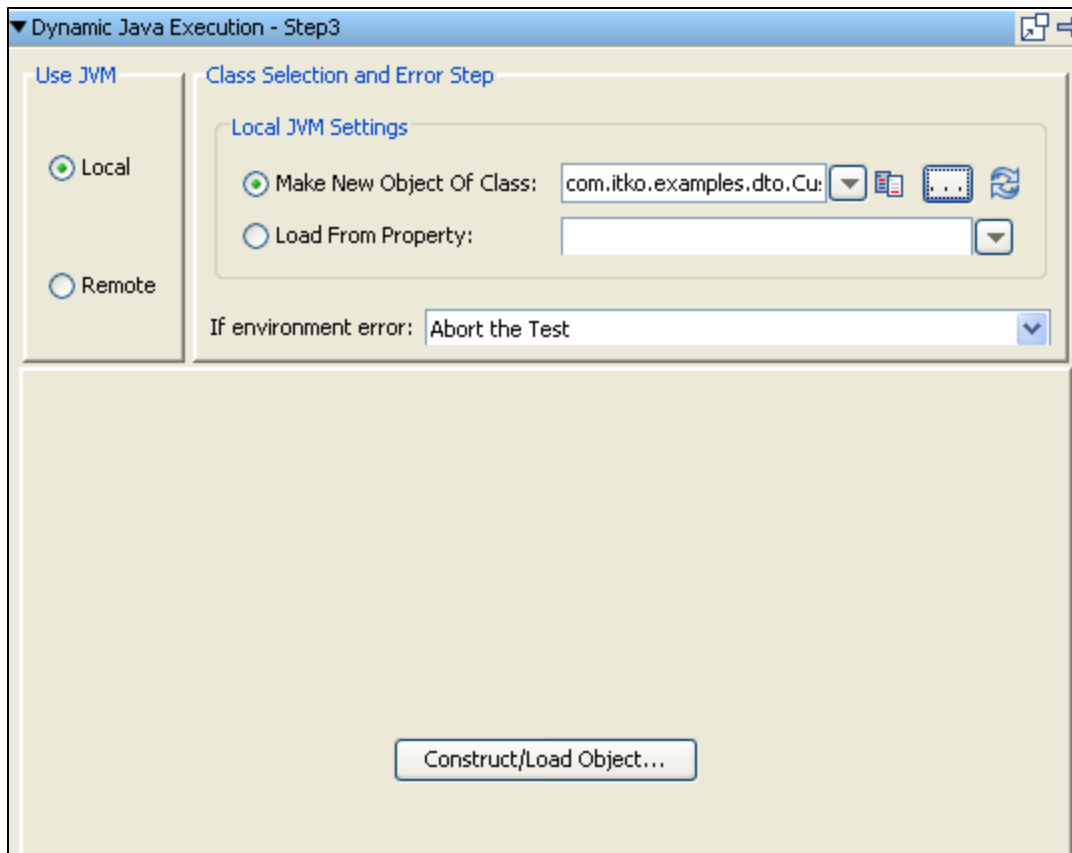
11.1 COE User Interface

11.1 COE User Interface

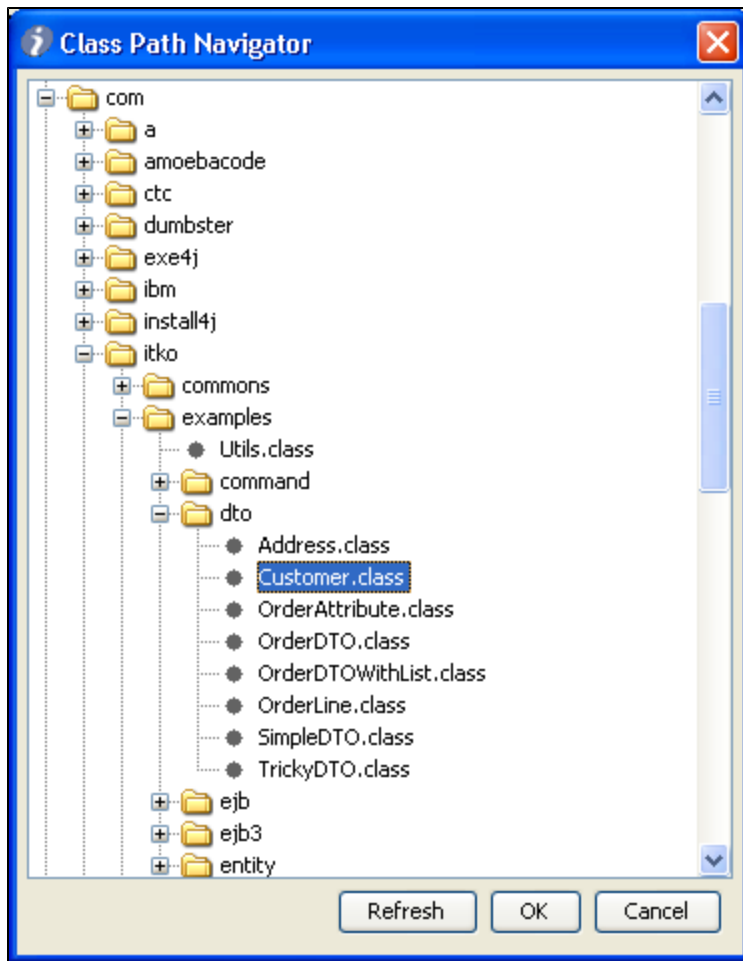
Regardless of where it appears, the Complex Object Editor (COE) has the same familiar look and feel.

For the purpose of illustration, we have shown the **Dynamic Java Execution** step, using the **Customer** class.

When you invoke this step, this is what you see:

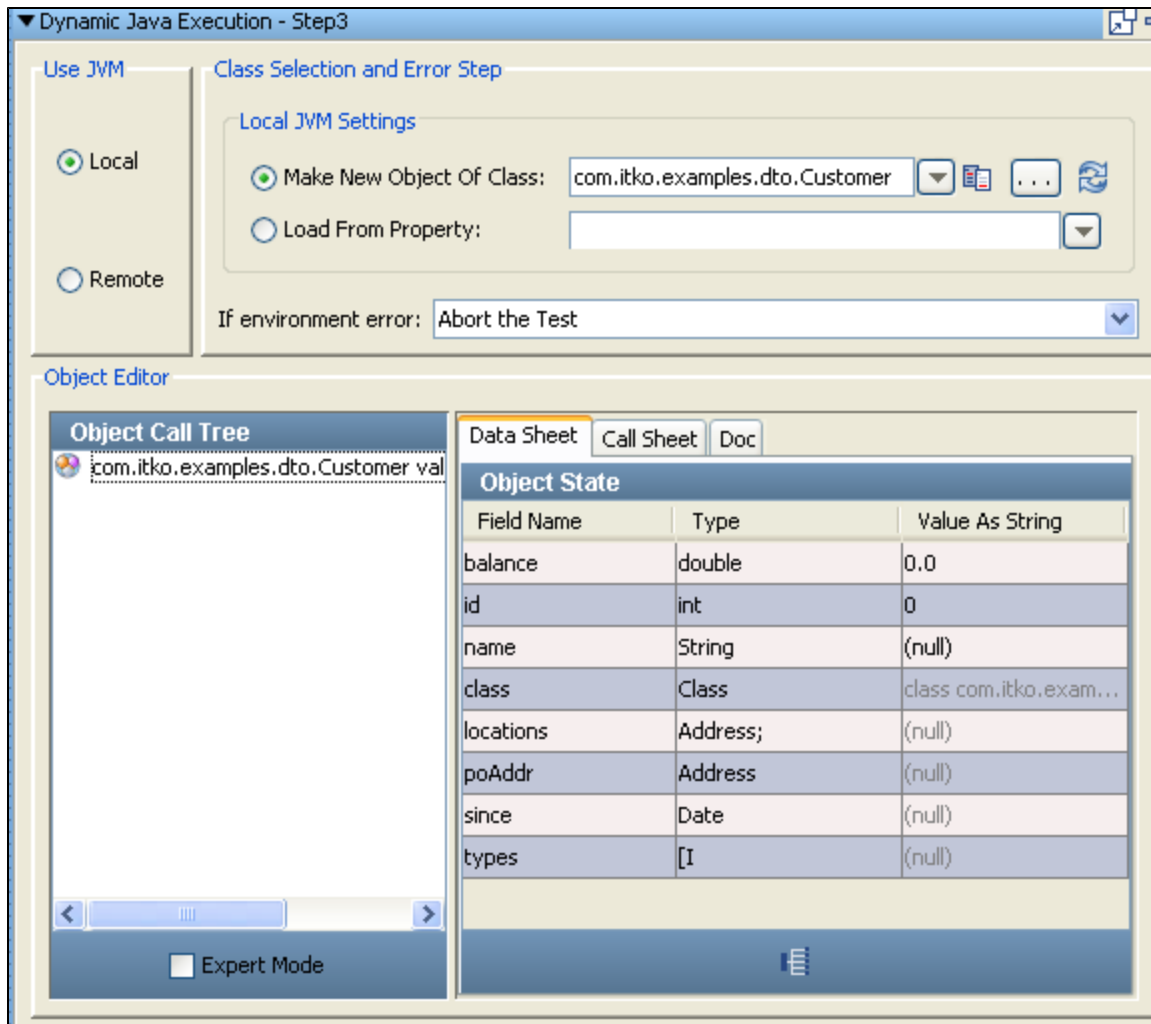


Select to use the "**Local JVM**" and browse to select the "**Customer class**" from the class path navigator as shown below:



Click **OK** to select the class and then click "**Construct/Load Object**" to load the object into the object editor.

Once the object is loaded, the object editor is as shown below:



The Object editor is divided into 2 panels:

The left panel, the **Object Call Tree**, keeps track of method invocations, and their input parameters and return values. The **Object Call Tree Panel** is described in detail in the next topic.

The right panel has the **Object State**, which has a set of dynamic tabs (**Data Sheet Panel**, **Call Sheet Panel**, **Doc Panel**) that will show you available options at any given time.

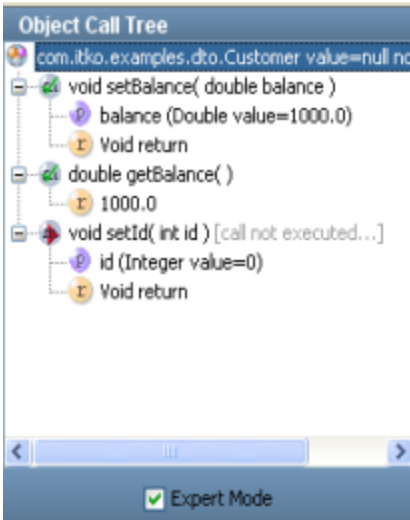
The screen above shows a Java object, of type **Customer** loaded in the editor. This particular object was loaded using the **Dynamic Java Execution** test step, but it could have been loaded as the result of any number of operations. No calls have been invoked on the object.

11.2 Object Call Tree Panel

11.2 Object Call Tree Panel

As you manipulate your Java object (Customer), the **Object Call Tree** will expand to keep track of the calls invoked and the associated parameter values.

An **Object Call Tree** after several methods have been invoked is as seen below:

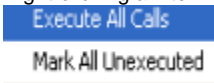


The following icons are used to identify the branches in the object call tree:

Icon	Description
	The type (class) of the object currently loaded, followed by response from calling 'toString' method of object.
	The Constructor that was called. This is shown if multiple constructors exist.
	A method call that has not been executed.
	A method call that has been executed.
	The input parameters (type and current value) for the enclosing method.
	The return value (current value if call has been executed) for the enclosing method.

Clicking an item in the **Object Call Tree** will display the appropriate set of tabs in the **Data and Call Sheet**.

Right-clicking an item in the **Object Call Tree** displays the following menu:



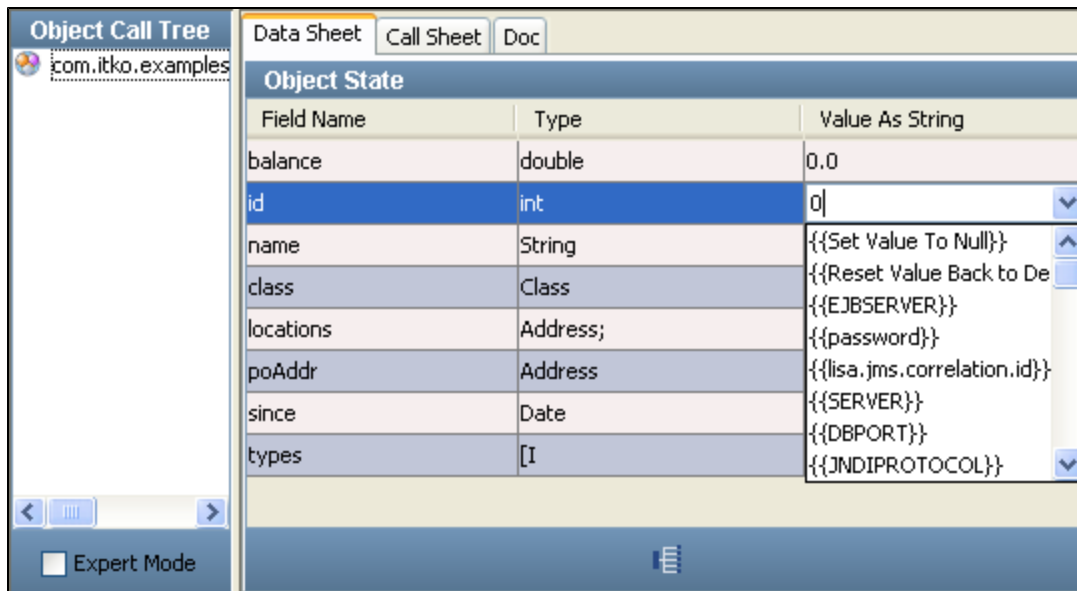
You can **Execute All Calls** or **Mark All calls as Unexecuted** in the Object Call Tree.

11.3 Data Sheet & Call Sheet Panels

11.3 Data Sheet & Call Sheet Panels

Data Sheet Panel

The right panel shows three tabs, with the **Data Sheet** tab active by default. The Data Sheet tab shows the data that LISA has been able to identify by inspecting the current state of the object is as shown below:

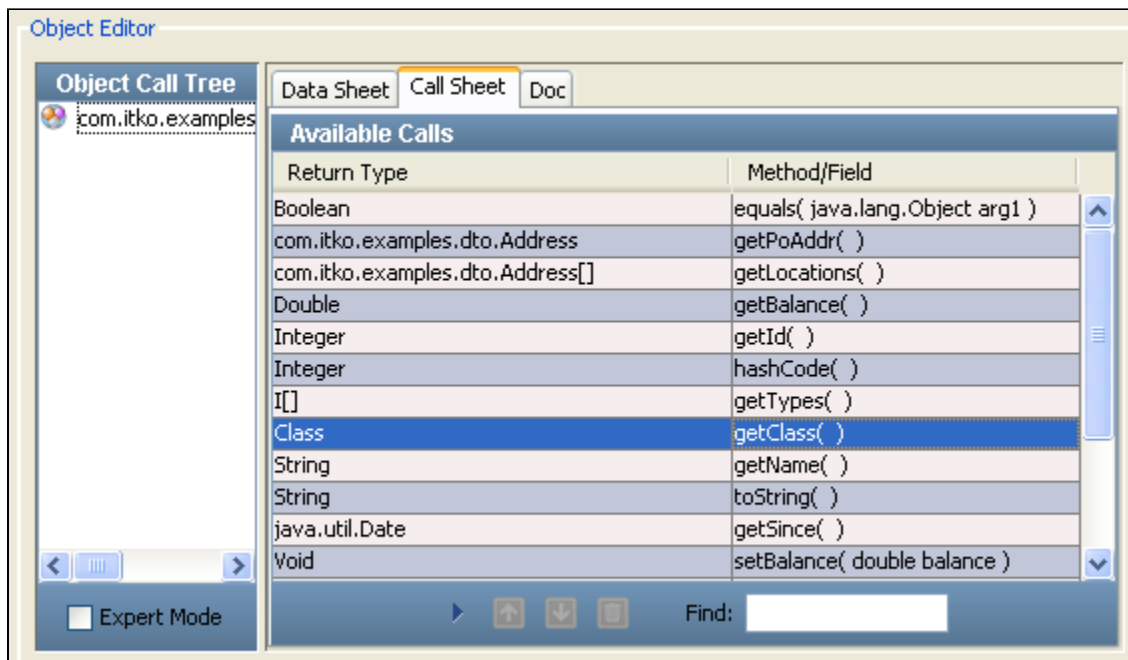


Data shown in black can be edited in this tab. The data values are edited in the **Value as String** column. These will always be primitives or strings. Values grayed out cannot be edited here, but there will be other screens where these objects can be edited. The address field for example, is an object of type **Address** that cannot be edited in this tab.

Call Sheet Panel

The figure below shows the second tab the **Call Sheet** tab showing the **Available** calls.

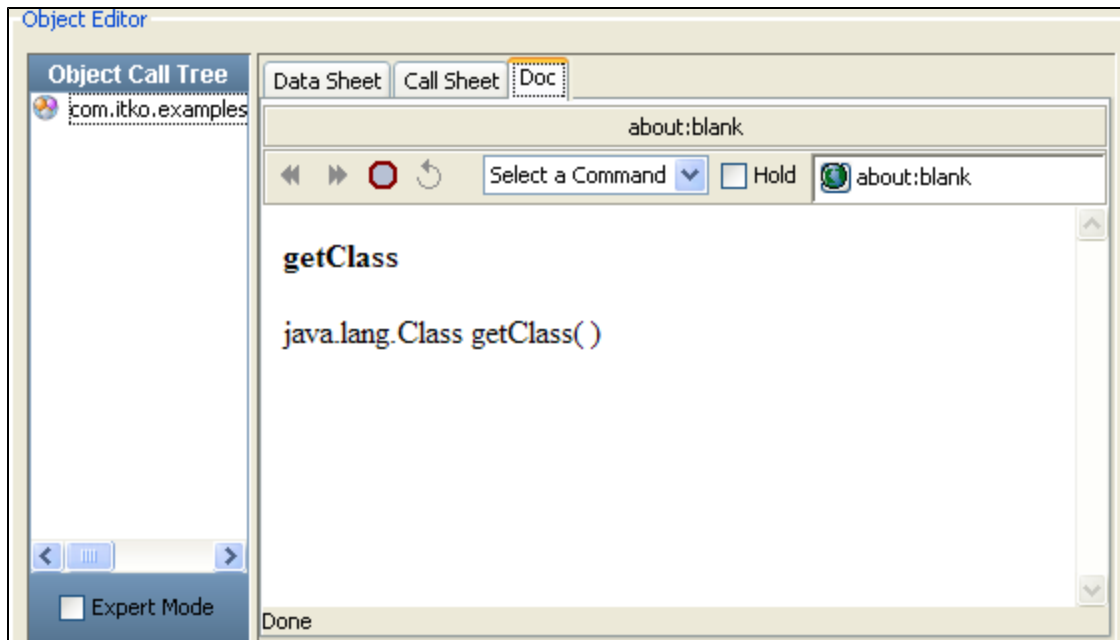
Here it shows the **Methods/Fields** calls that are available, and their **Return types**:



To invoke/run a method, highlight it in the **Methods/Fields** list and click the **Invoke Method** icon at the bottom of the tab. The selected method now appears in the **Object Call Tree** (on the left). Once in the object call tree you can provide input parameters and invoke the method.

Doc Panel

The third tab, the **Doc** tab will display any **JavaDoc** that has been made available for this class as shown below:

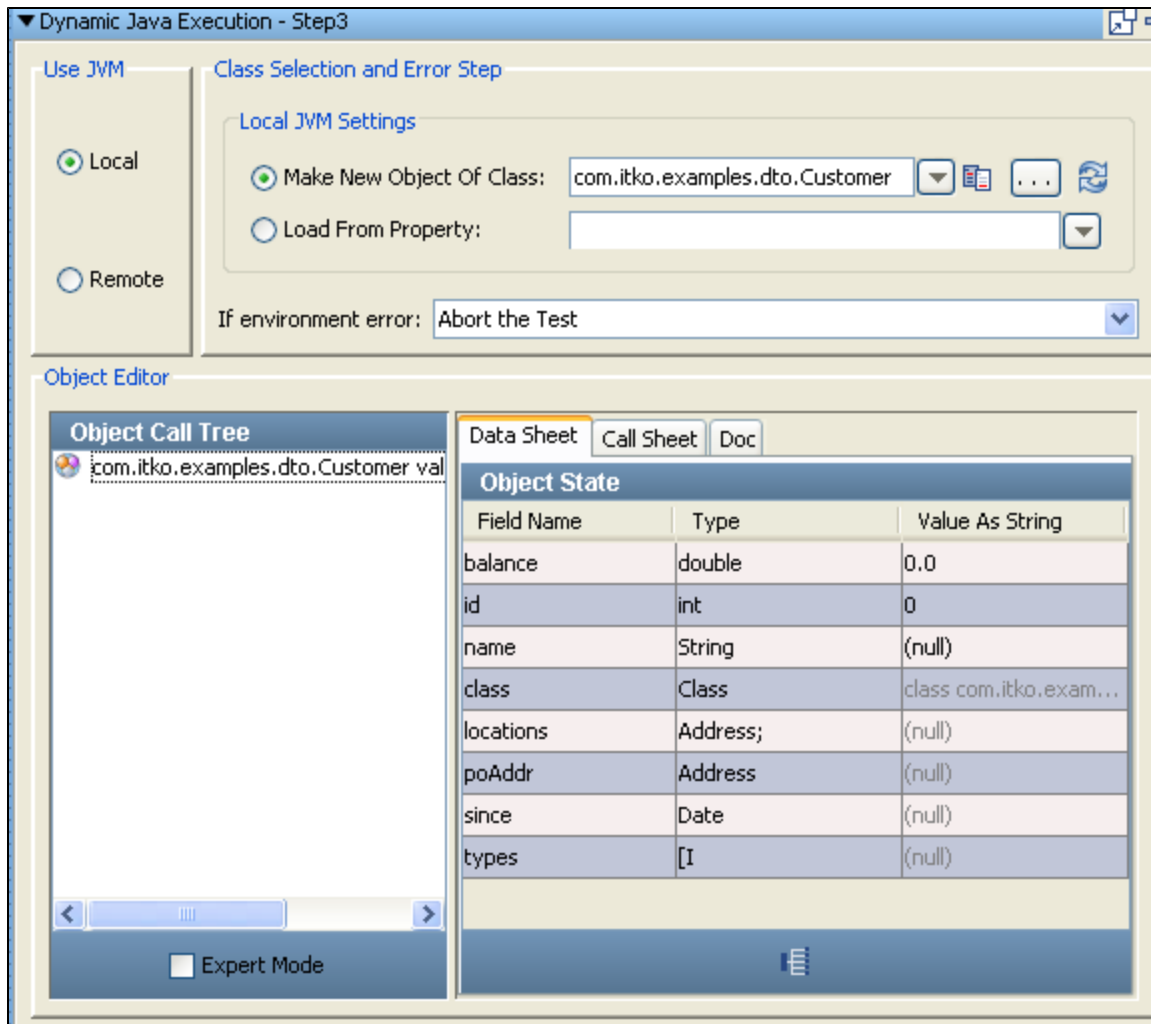


11.4 Object Interaction Panels

11.4 Object Interaction Panels

There are different tab combinations can be displayed in the **Data and Call Sheet**.

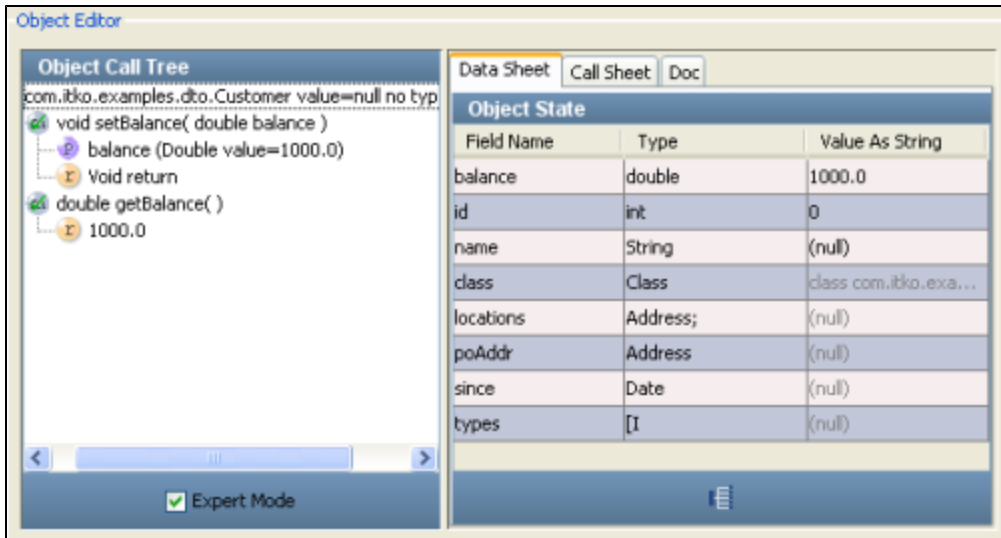
The tabs will change depending on what you have selected in the **Object Call Tree** in the left panel.



Object is selected
Method is selected
Input Parameter is selected
Return Value is selected

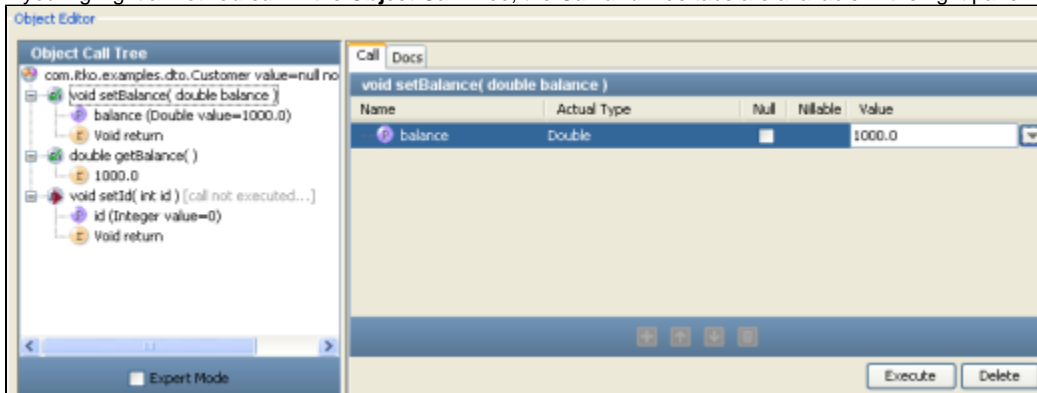
Object Panels

In the previous figures you have seen that the **Data Sheet**, **Call Sheet**, and **Doc** tabs/panels are available when an object is highlighted in the **Object Call Tree** as shown below:



Method Call Panels

If you highlight a **method call** in the **Object Call Tree**, the **Call** and **Doc** tabs are available in the right panel.



This is where you provide values for the input parameters.

LISA provides the information on each parameter so you only need to supply the value. In the example above, it is straightforward; the single parameter is of type '**double**', so we can type in a value, or a property name, in the **Value** column.

The pull-down menu maintains a list of the current properties. If you are required to enter an object as an input parameter this requires more work. We will describe several approaches to providing objects in the subsequent sections.

Notice that the **Expert Mode** at the bottom of the left panel is not checked. This denotes we are in the **Simple mode** currently.

Simple and Expert Mode

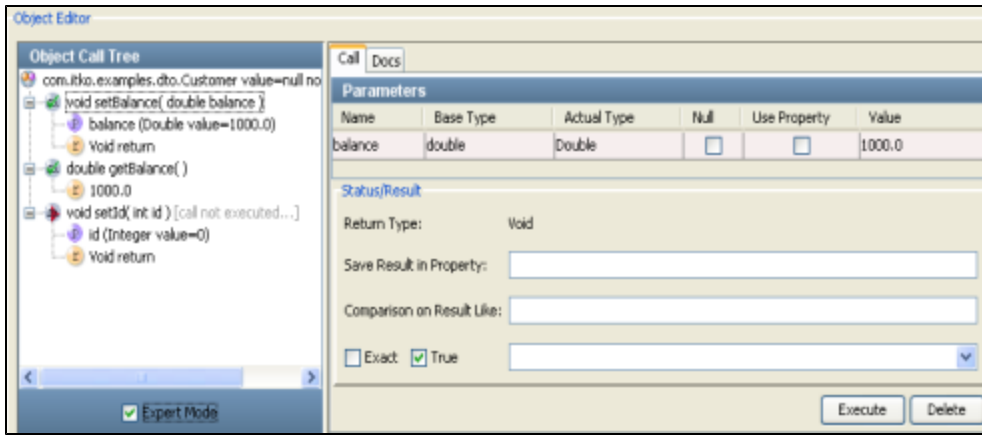
There are two editing modes available: "**Simple**" and "**Expert**".

Simple Mode is useful when your object is a **simple object**, such as a **Java Bean** with just a default constructor and several setter/getter methods. This is the classic Data Transfer Object (DTO). These are very common as inputs to web service calls. With objects of this type you can toggle back and forth between simple and expert mode. A DTO that contains a DTO as a property can be manipulated in simple mode. We shall see examples of this a little later.

Expert Mode must be used for more **complex objects** such as objects that have multiple constructors. Some composite objects that contain other complex objects cannot use the simple mode, and in fact the simple mode option is disabled if the current object requires expert mode. We shall see examples of using the expert mode a little later.

All the figures shown above have used **simple mode**.

The figure below shows the example shown in the previous figure, but with **expert mode** selected:



A new **Status/Result** panel opens up as shown.

You can now add **Inline Filters (Save Result Property In)** and **Assertions (Comparison On Result Like)** right here in the object editor.

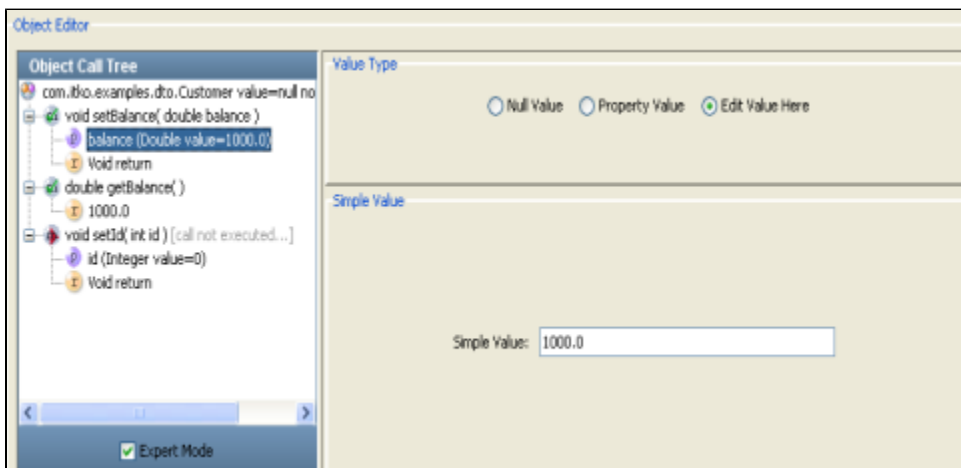
Note: The in-line Filters and Assertions that are applied are not seen in the Filters or Assertions list.

There are several other differences that will become apparent in our later examples.

Input Parameter Panels

If you highlight an input parameter in the **Object Call Tree**, the tabs available in the right panel will vary depending on the input parameter type: **Primitives/Strings or Objects**.

For input parameters which are **primitives and strings**, the following panel will be displayed:



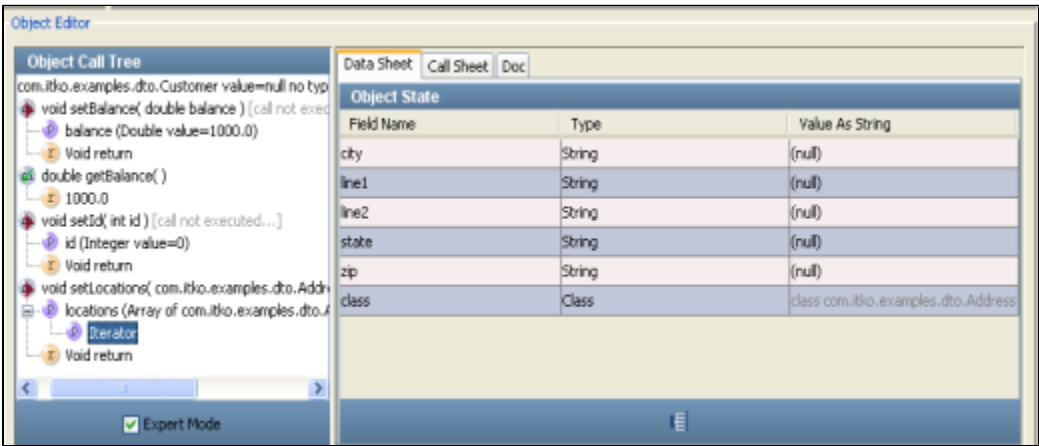
You can edit the value in this panel in **Simple Value** field.

If you want to use a property as the value, click the **Property Value** radio button to display the following panel:



You can type the property name or use the pull-down menu or click on the List icon to open the available Property keys.

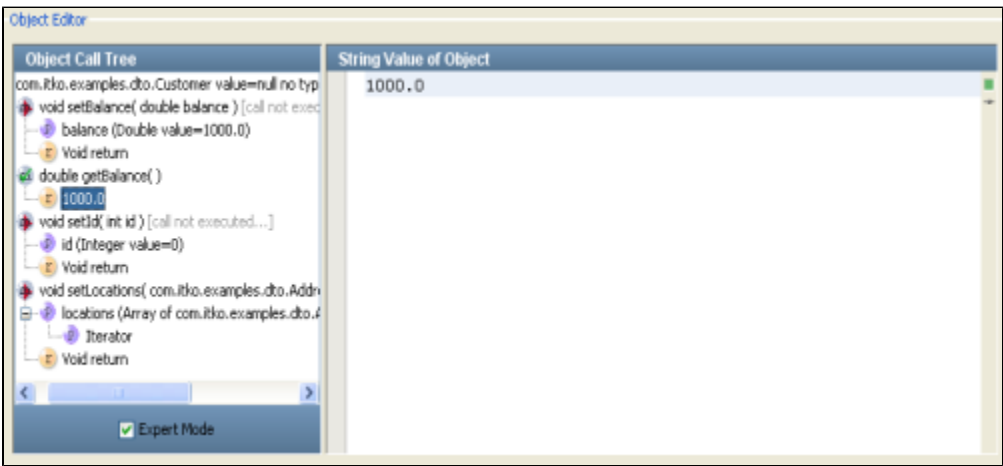
For input parameters that are **objects**, the following panel will be displayed:



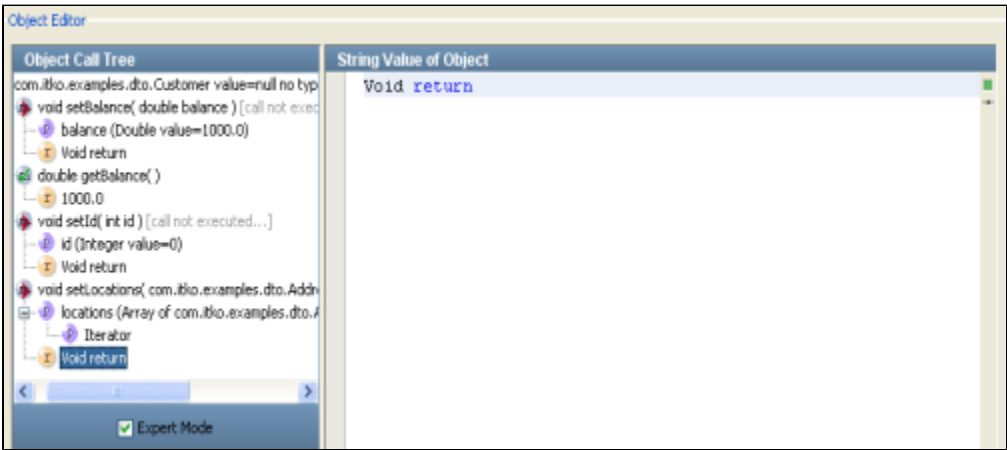
Return Value Panels

If you highlight a return value in the **Object Call Tree**, the tabs available in the right panel will vary depending on the input parameter type.

For input parameters that are **primitives and strings**, the following panel will be displayed:



For input parameters that are **objects**, following panel is displayed:



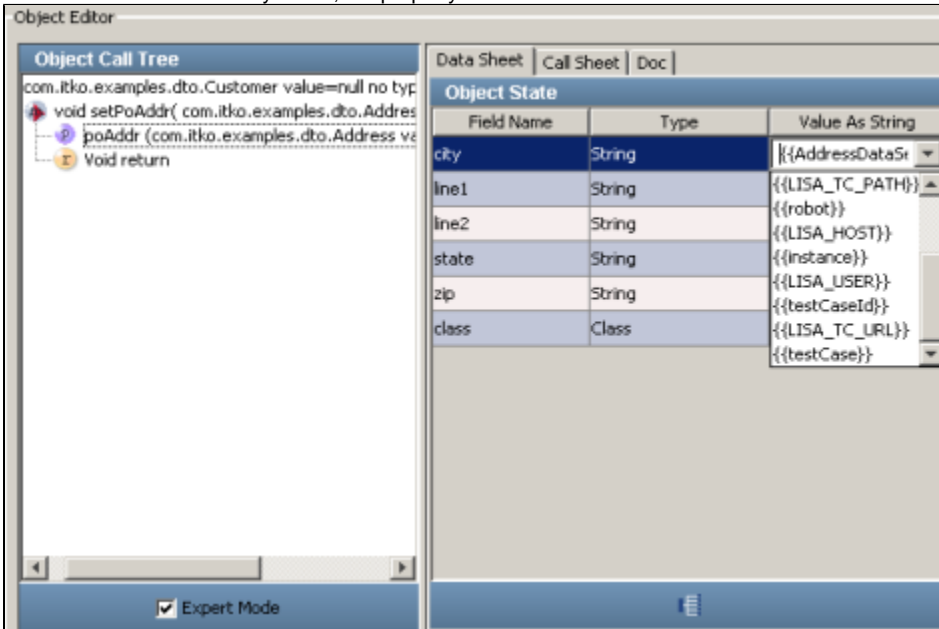
11.5 Using Data Sets in the COE

11.5 Using Data Sets in the COE

A common way to provide data for a Java DTO object is a Data Set.

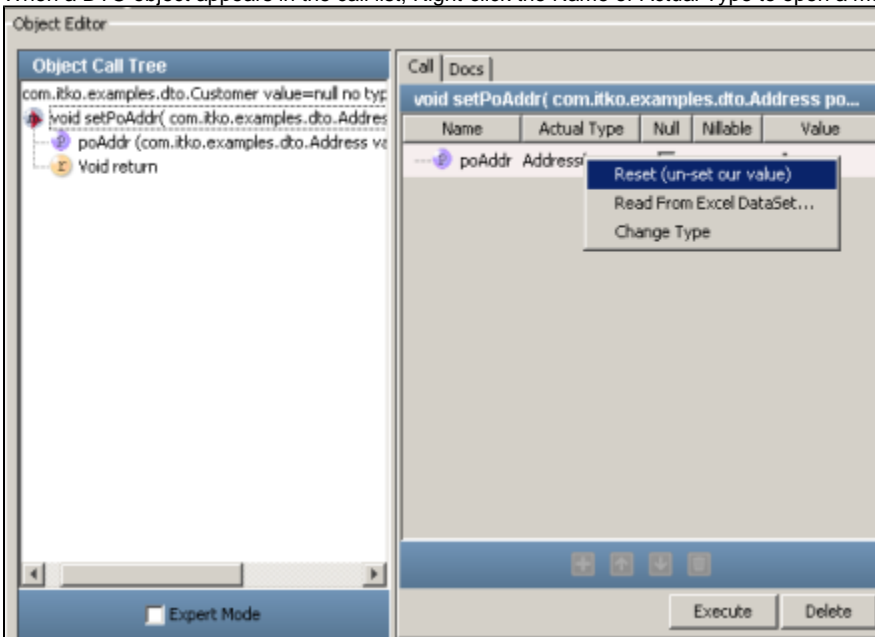
LISA provides a Data Set, **Read DTOs from Excel File**, just for this purpose.

If the Excel Data Set already exists, the property that contains the Data Set can be entered as a value for the DTO object, as shown below:

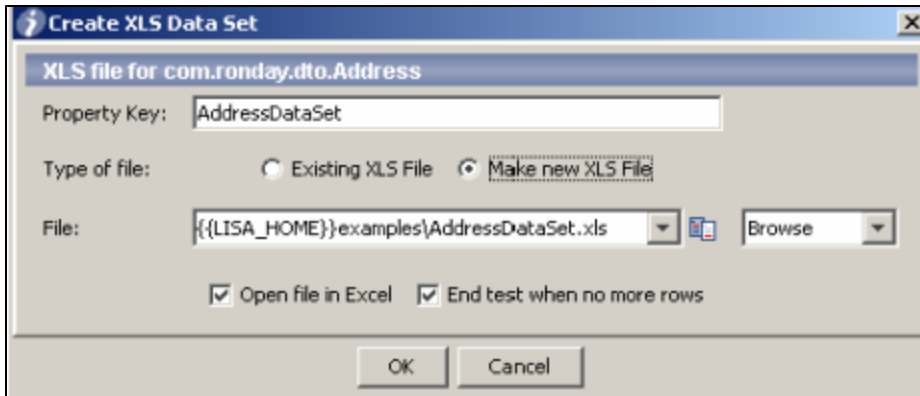


You can also initiate the creation of a **new DTO Data Set** from within the Object editor.

When a DTO object appears in the call list, Right-click the Name or Actual Type to open a menu as shown below:



Select the Data Set **Read From Excel Dataset...** to display the following screen:



The parameters on this screen are required to initiate the creation of this Data Set.

Enter the following parameters:

- **Property Key:** The property that stores the current values obtained from the Data Set.
- **Type of File:** Choose if it is an **Existing XLS file** or you need to **Make a new XLS file**. Here we choose to make a new XLS file.
- **File:** The name of the Excel file that will be the template for the DTO data
- Check the Open file in Excel box
- Check **End test when no more rows** to end the test after all rows have been read.
- Click **OK**

You are now ready to create your Excel Data Set.

For more information on using Excel to store data for Java DTO's see [LISA Reference Guide](#).

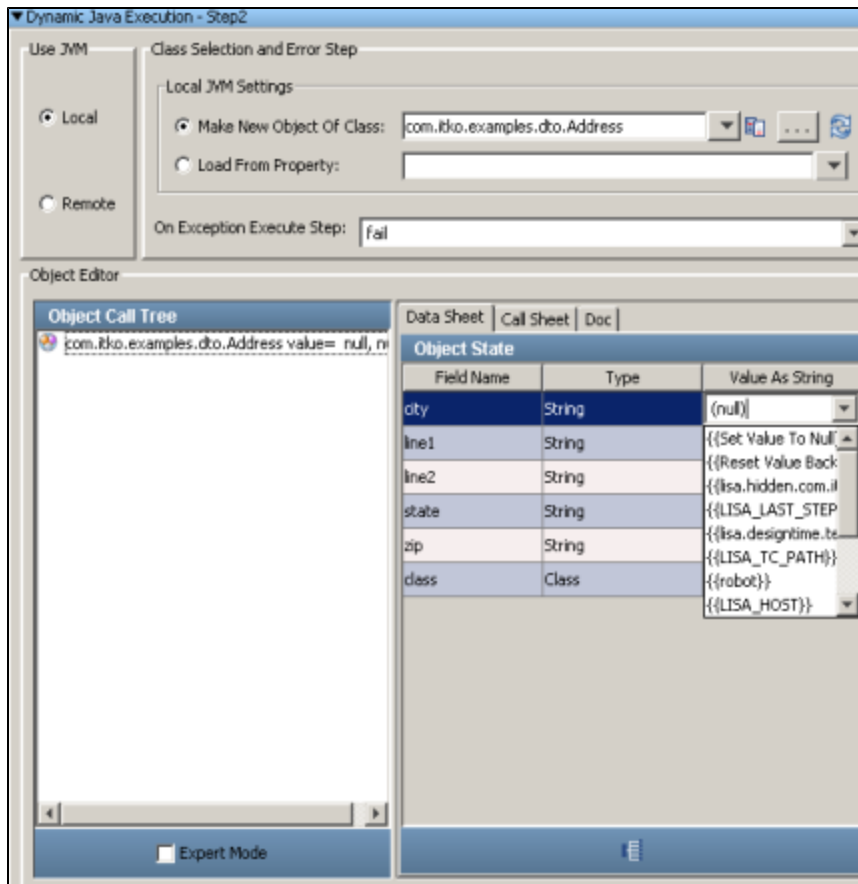
11.6 Usage Scenarios - Simple Objects

11.6 Usage Scenarios for Simple Objects

The following examples are based on standard Java classes for simple objects. We have used classes from the Demo Server included with LISA. You should have no difficulty reproducing them in your environment.

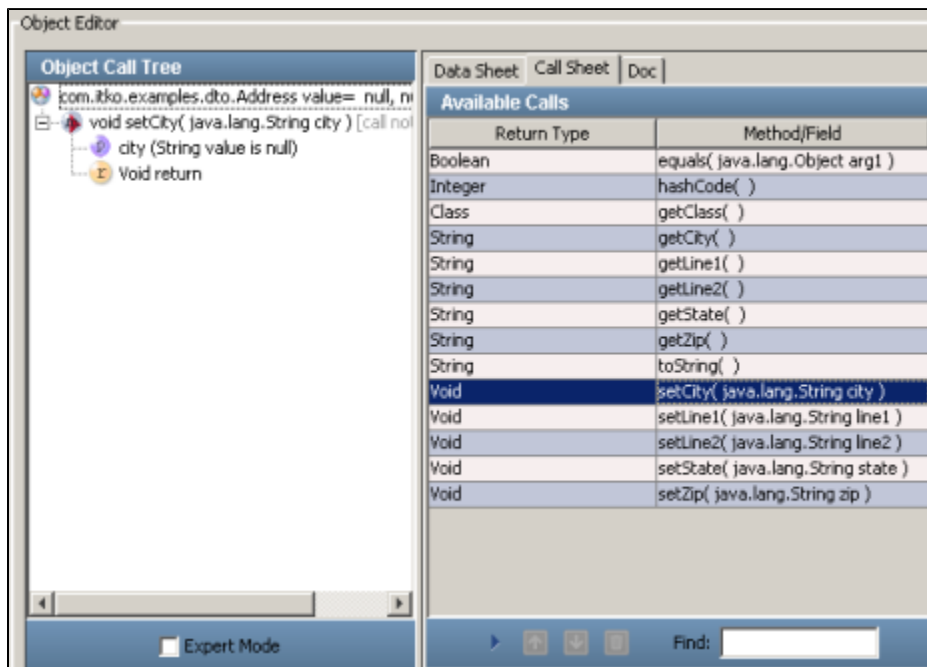
Simple DTO Object Scenario 1

A simple DTO **com.itko.examples.dto.Address**, has been loaded in the COE using a **Dynamic Java Execution** step. The Address class has simple properties only, as shown in the figure below:

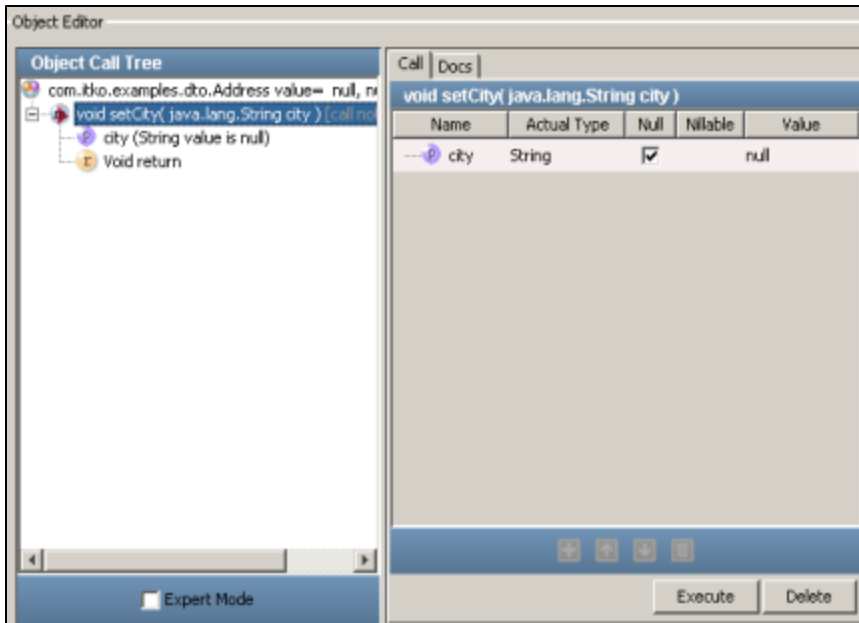


For a simple DTO, parameter values can be entered into the **Data Sheet panel** as fixed values or properties. In the example above, city could be set equal to the property `currentCity`.

Parameters can also be entered using the DTO setters in the Call sheet panel.



In the **Call Sheet** tab, select a getter, such as `setCity(java.lang.String city)` and click the **Invoke Method** icon. The method will run and COE will open in the **Call** tab as shown below:

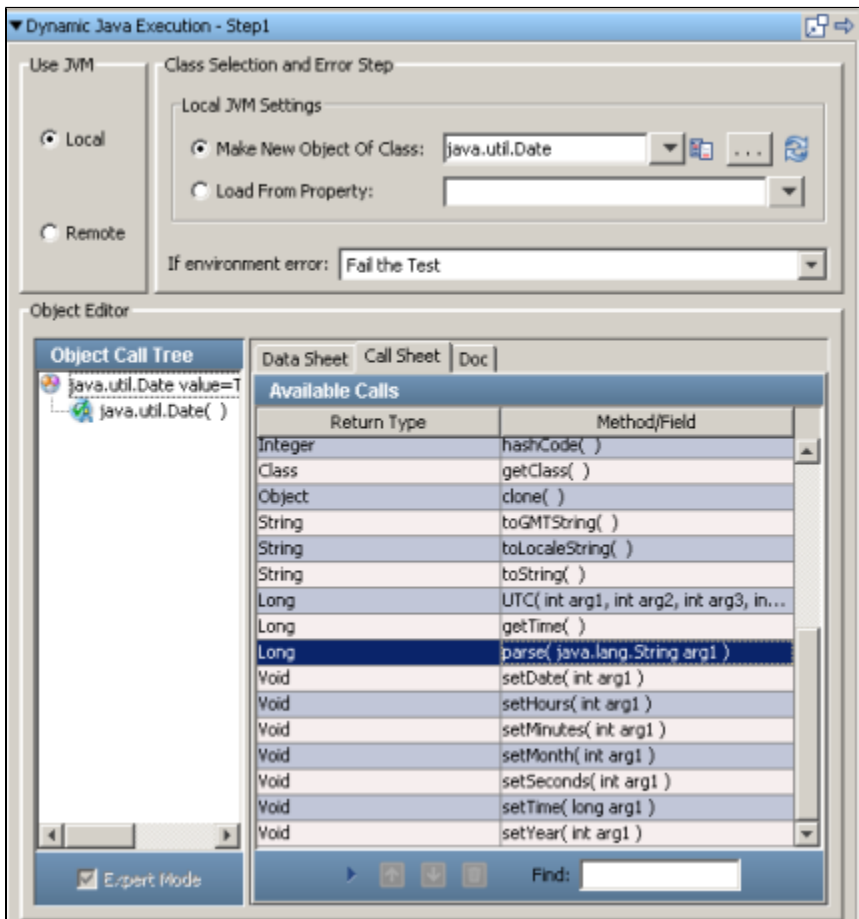


- Enter the parameter value as a fixed value or a property. You must use the LISA property syntax here, `propname`.
- Click the **Execute** button to invoke the method.

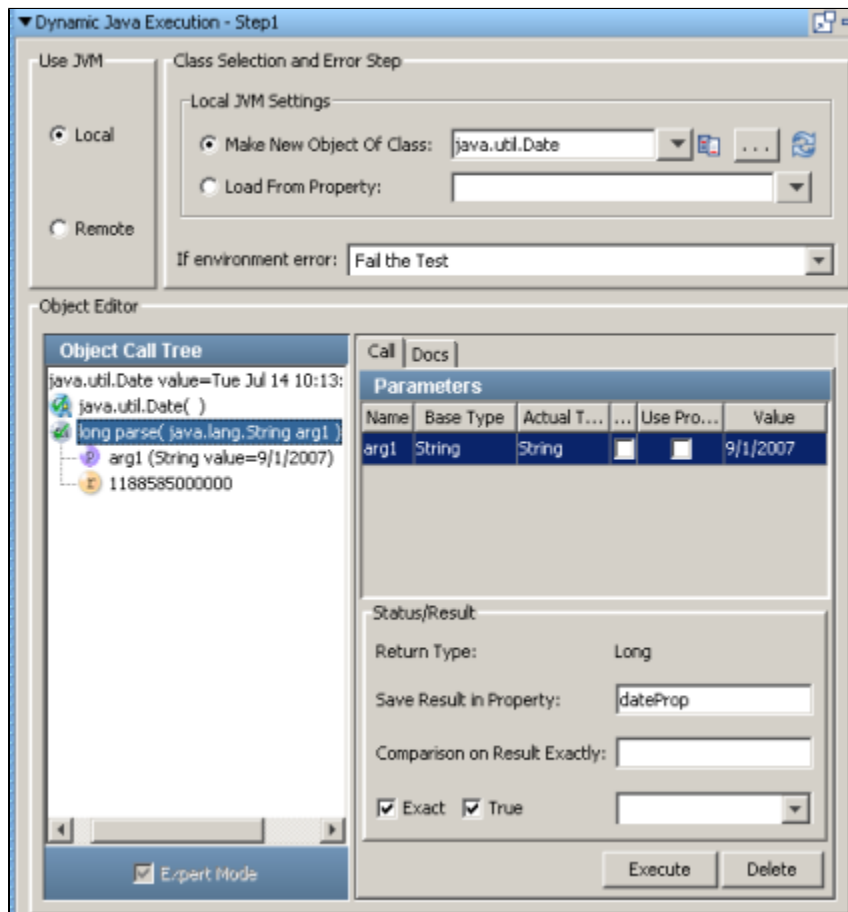
Repeat this procedure to set other DTO properties.

Simple Java Object Scenario 2

A simple Java object, `java.util.Date`, has been loaded in the COE using a **Dynamic Java Execution** step.



In this case we must stay in **Expert Mode**, since the **Date type is not a DTO**. In fact the COE forces us into Expert mode.



But we can still enter parameter values and invoke methods. In the example above we execute the parse method, which requires one input parameter. We have entered it as a string value, 9/1/2007.

Note: that in expert mode we use the **Null** or **Use Property** checkbox to denote the parameter type. If neither is checked we are going to enter a fixed value. We would not use the property syntax here. Even if we were entering a property rather than a string value, we would just enter the property name.

Because we are in **Expert Mode** we have the opportunity here to add **inline Filters** and **Assertions**.

11.7 Usage Scenarios - Complex Objects

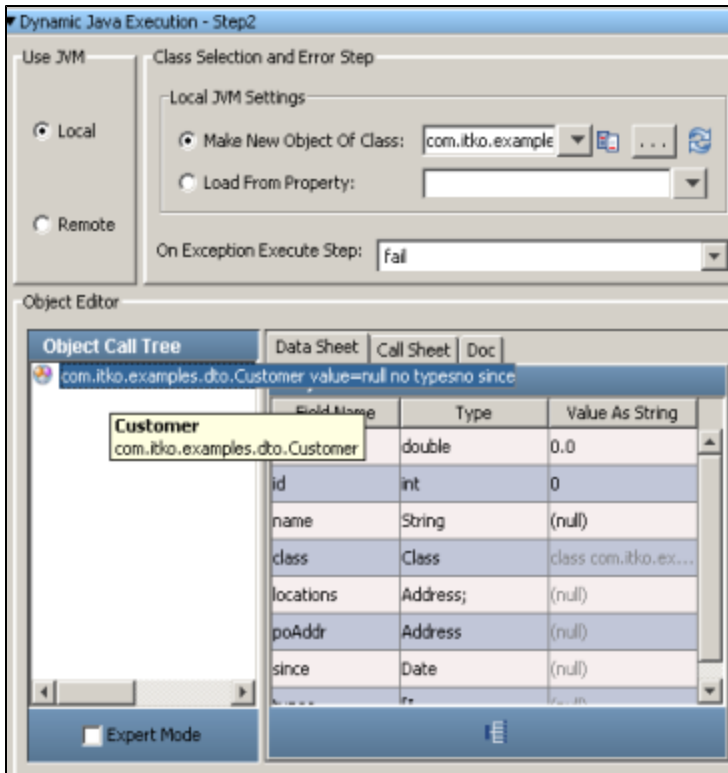
11.7 Usage Scenarios - Complex Objects

The following examples are based on standard Java classes for Complex objects. We have used classes from the Demo Server included with LISA.

You should have no difficulty reproducing them in your environment.

Complex DTO Object Scenario 1

A complex DTO Object, **com.itko.examples.dto.Customer**, has been loaded in the COE using a **Dynamic Java Execution** step:



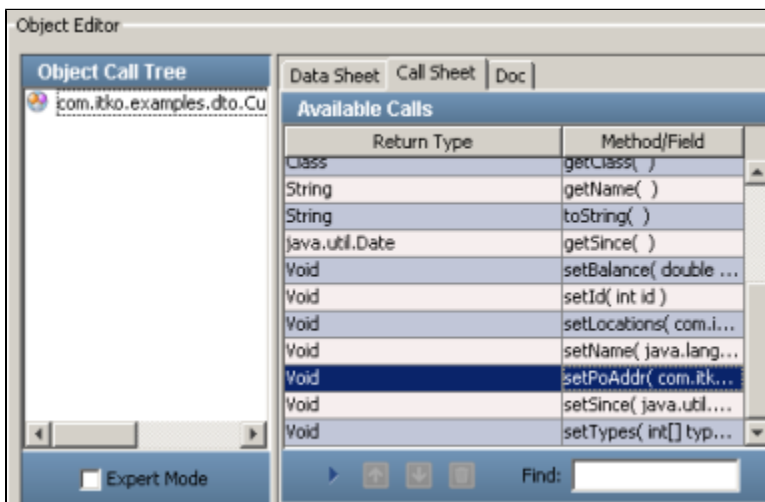
This DTO is complex because its properties are not all simple values such as primitives or Strings. However, because of its DTO structure we can still use **Simple Mode** here incase we wish to.

The following properties exist:

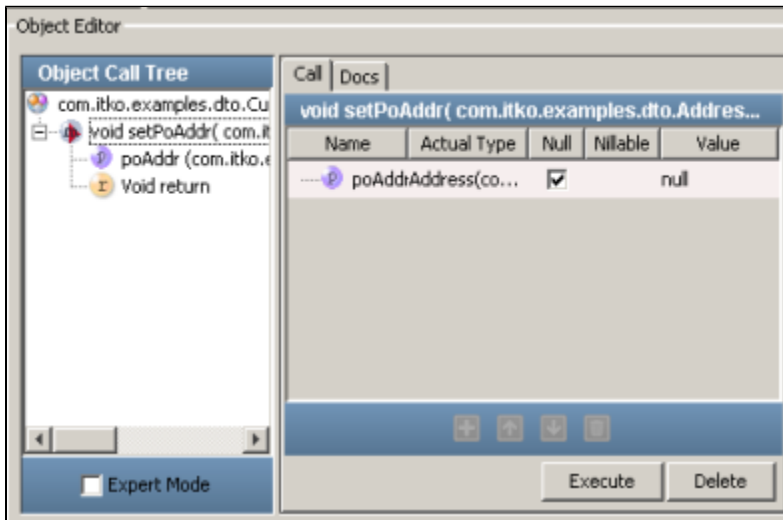
Property	Description
locations	An Array of Address objects
poAddr	An Address object
since	A Java Date object
types	An array of integers

Each of these properties must be given values before the Customer object can be used.

Starting with the **poAddr** object, we identify the **setPoAddr** method in the **Call Sheet**:

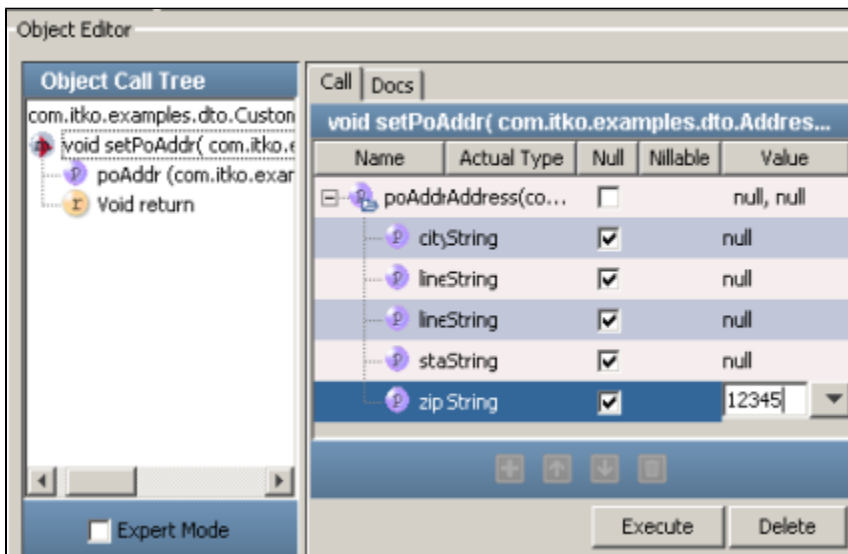


Select the **setPoAddr** method and double click or click the **Invoke Method** icon to invoke/run this method as shown below:



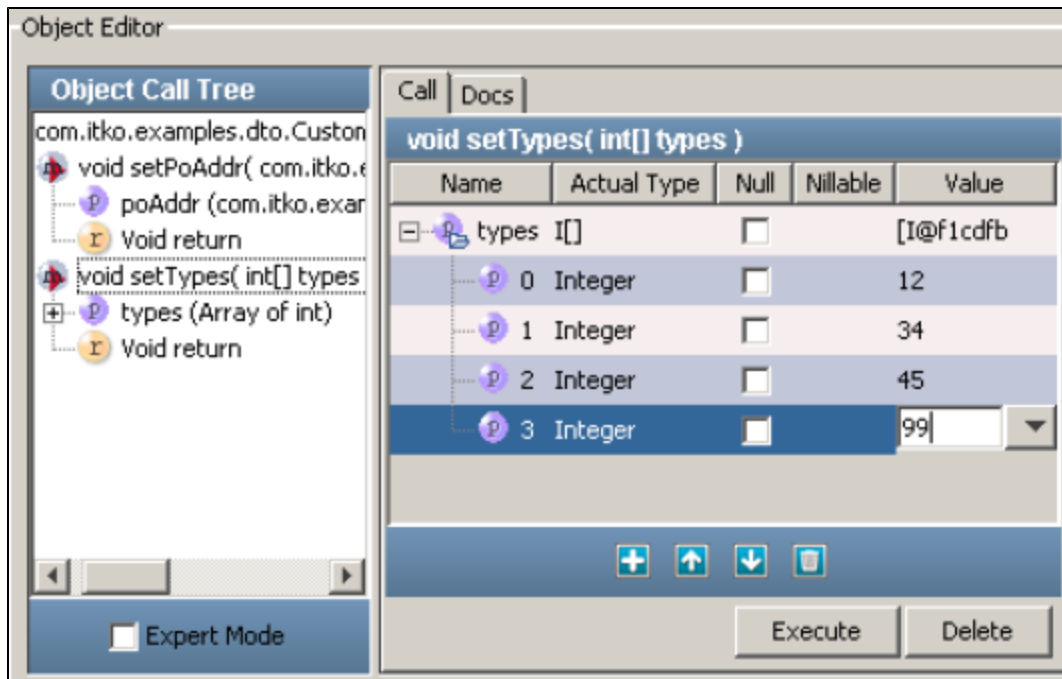
This is a DTO that allows the use of **Simple** mode, so the **Expert Mode** is left unchecked. LISA identifies the **poAddress** property to be of type Address.

In a **Simple mode**, when you uncheck the **Null** parameter*, LISA expands the Address object to expose its properties. We know, from the Simple Data Object Scenario above, that Address has simple properties, so we can enter them as values or LISA properties in the **Value** column as seen below.



Click the **Execute** button to invoke the **setPoAddr** method.

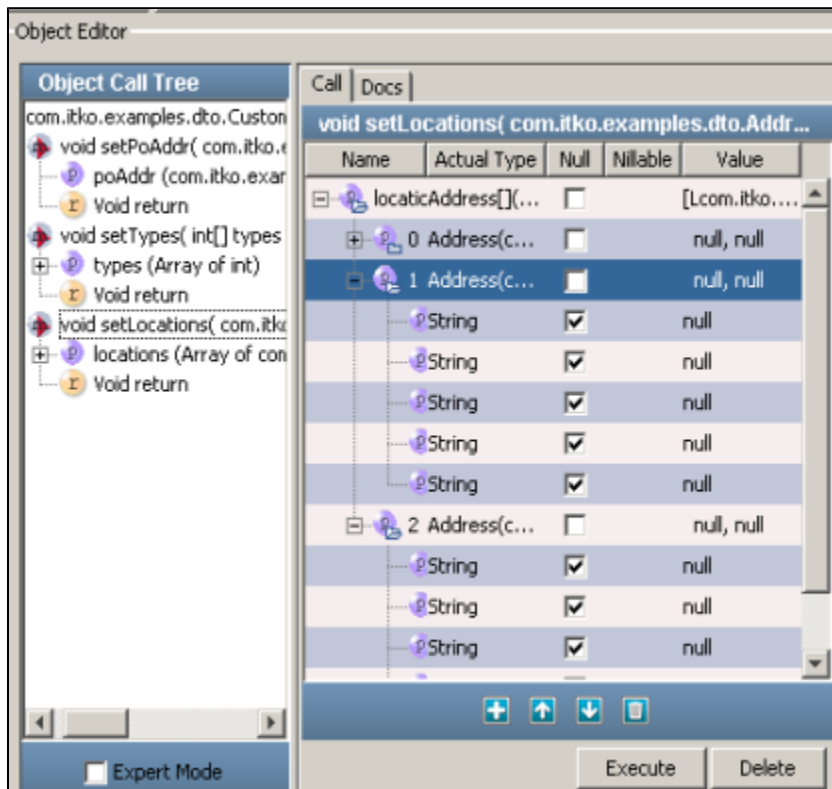
Next, select the **setTypes** method on the **Call Sheet** and click the **Invoke Method** icon:



"Types" is an array of integers, so LISA expects you to click the **Add** icon at the bottom, to add as many elements in the array as needed. In the example above we added four elements, and entered values for each.

Click the **Execute** button to invoke the **setTypes** method.

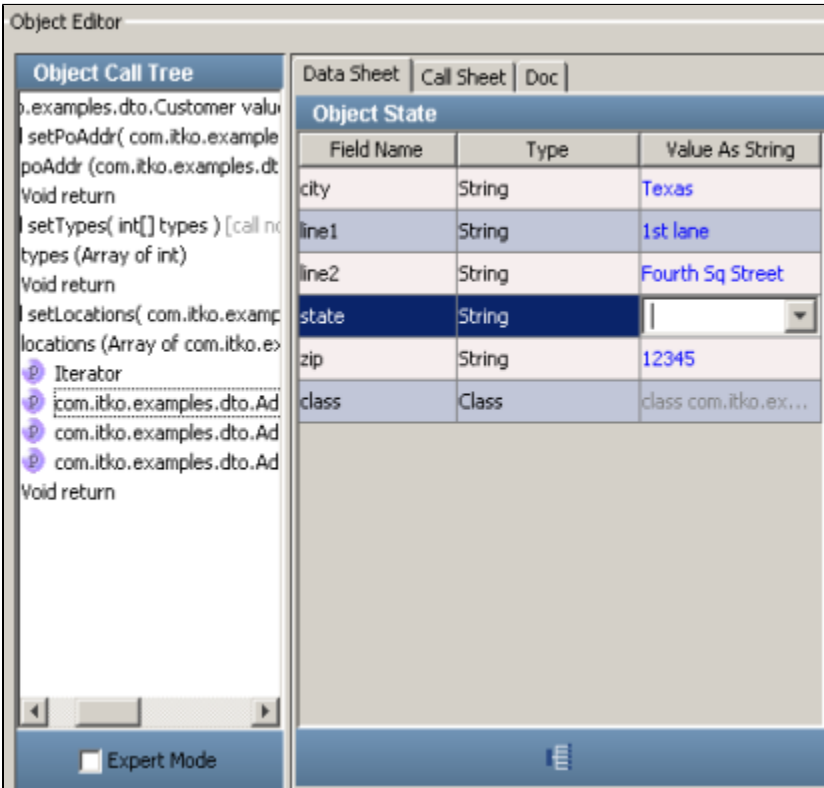
Next, select the **setLocations** method on the **Call Sheet** and click the **Invoke Method** icon:



Locations, is an array of Address objects, so LISA expects you to click the **Add** icon to add as many elements (of type Address) as needed. In the example above we added 3 Address objects. Two have been completed, we are about to expand the third Address object to enter values for the properties.

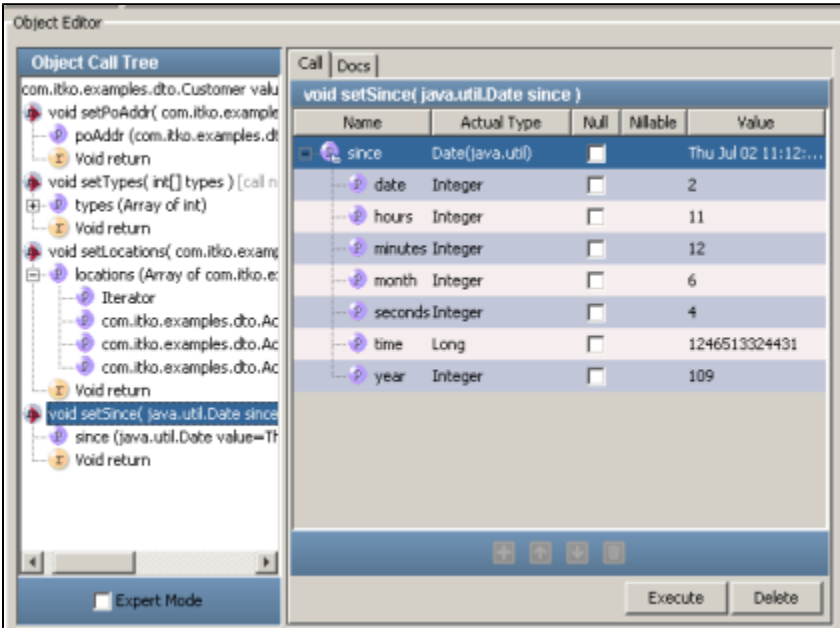
When complete, click the **Execute** button to invoke the **setLocations** method.

Notice that you could click on one of the Location elements in the **Object Call Tree** to display and edit its properties in the **Data Sheet** tab as shown below:



This holds true for all the properties listed in the **Object Call tree**.

Finally, select the **getSince** method on the **Call Sheet** and click the **Invoke Method** icon:



The input parameters for the Data object are displayed by LISA and can be given values.

Click the **Execute** button.

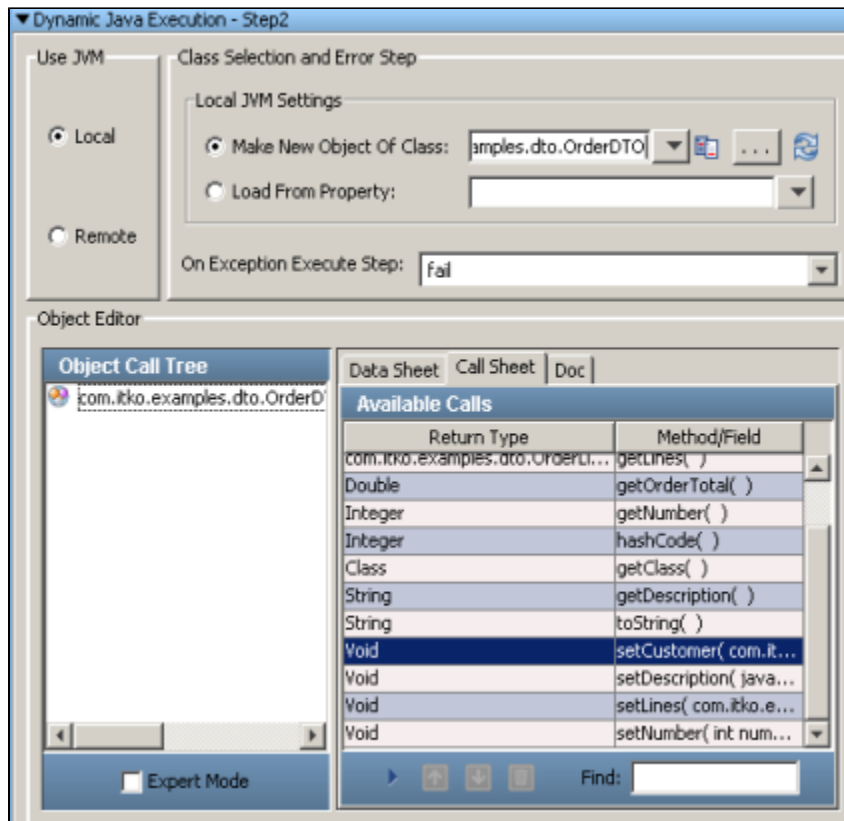
The Customer object is now fully specified and can be used in your Test Case.

Complex DTO Object Scenario 2

The last scenario shows an example that builds on the last three scenarios.

This DTO, **com.itko.examples.dto.OrderDTO**, has a Customer object as one of its properties. This scenario shows how easy it is to build a Customer object in Simple mode without calling any setter methods.

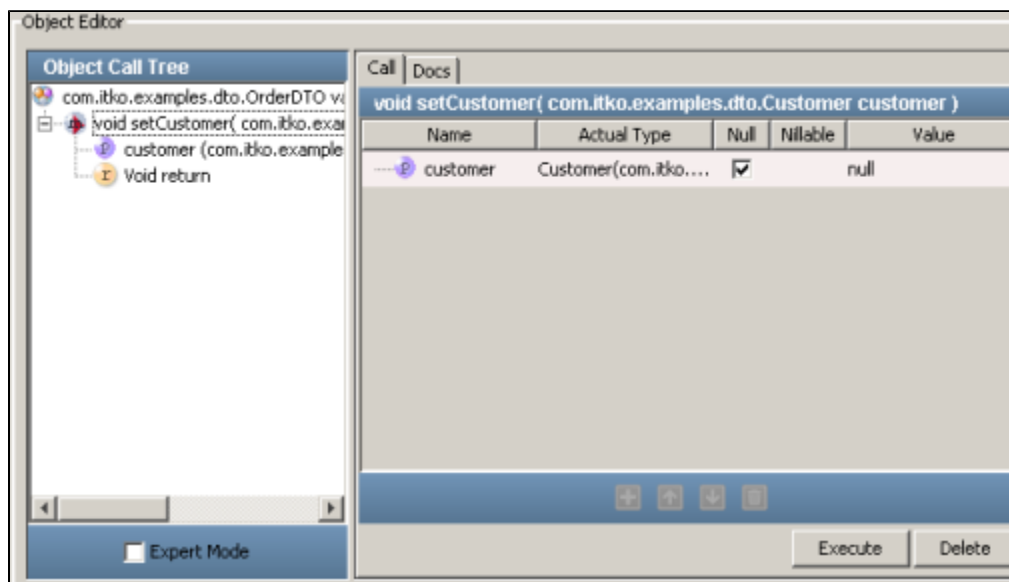
The **OrderDTO** object has been loaded in COE using a **Dynamic Java Execution** step as shown below:



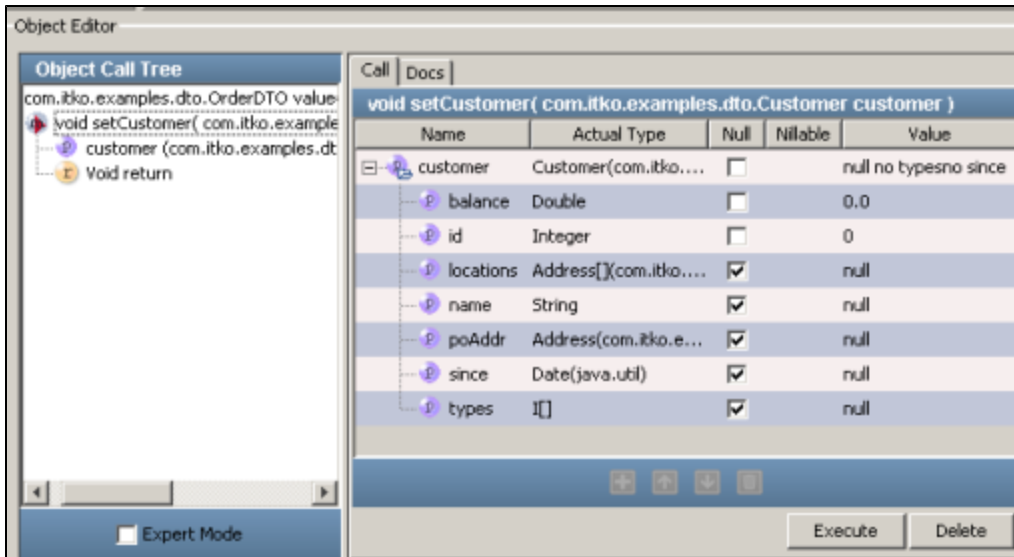
Again we can use **Simple Mode** for the **OrderDTO** object.

Select the **setCustomer** method in the **Call Sheet** and click the **Invoke Method** icon:

As expected the input parameter is a Customer object.

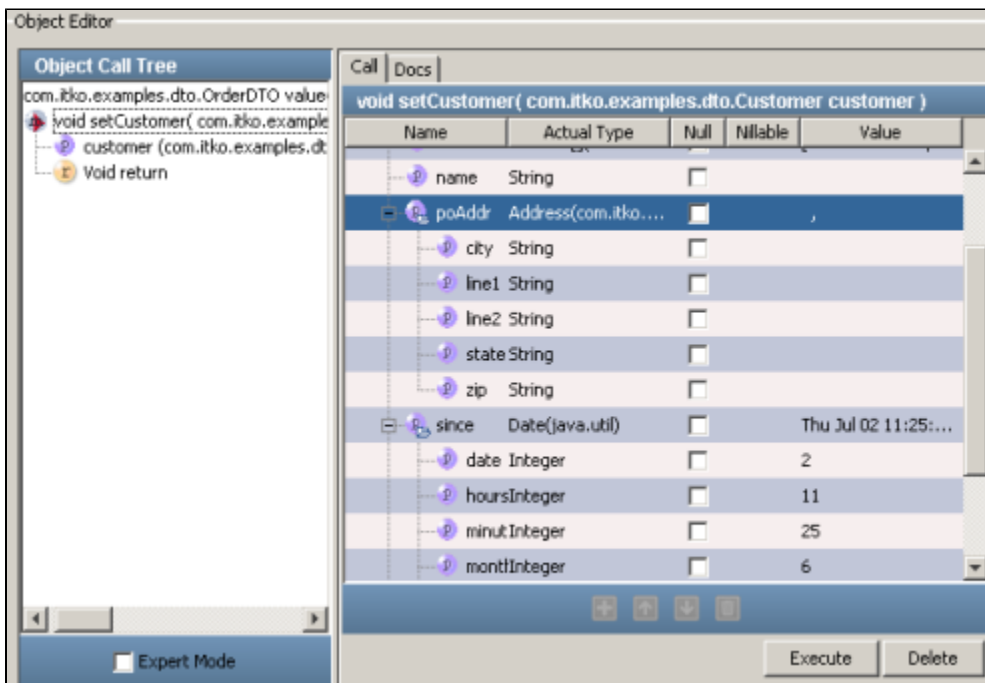


Uncheck the checkbox in the **Null** column. LISA expands the Customer row to expose its properties:



All of the properties can be edited on this screen. The Integer and String properties can be added in the Value column. The remaining properties will expand to show their properties when you uncheck the **Null** box for the property. If the property is a single object it will expand to expose its properties. If it is an array or collection you can add the appropriate number of elements using the **Add** icon.

The figure below shows a snapshot of the editing process:



The **locations** property has 2 elements; the **types** property has 3 elements.

The **poAddr** object is of type Address, and is expanded exposing simple string properties as shown above.

.

PART 3 - Building Documents

PART 3 - Building Documents

Part 3 of the Users guide gives details regarding building documents within LISA. There are many editors within LISA, that help us build effective documents.

Test Suite documents and Staging documents are important aspects of LISA.

You can apply Metrics and Add appropriate Events to your Test Cases, which are used for monitoring the test case after the test run.

In this section, the following topics are covered:

- 12. Building Test Cases
- 13. Building Staging Documents
- 14. Building Test Suites
- 15. Building Audit Documents
- 16. Applying Metrics
- 17. Understanding Events

12. Building Test Cases

The following topics are available in this chapter..

12.1 Creating a LISA Project

12.1 Creating a LISA Project

From 5.0 version, a new concept of a "Project" is introduced in LISA.

A Project is actually a "folder" in the file system that holds a file called **lisa.project**, and a mandatory folder called Configs, that should hold at least one config file **project.config**.

The other LISA documents are organized into standard folders like Tests, Suites, StagingDocs etc. The existing LISA documents can be imported into a new project.

NOTE: You can still open and work with your old test case outside of the project, but we recommend using a project from this release (5.0) onwards.

Hence now it is mandatory to create a **LISA Project** before you create any Test cases, VServices, Staging document etc.

Ideally while opening a test case, you should open it through a valid LISA project.

This section provides detailed information on projects on how to create and work with them in LISA.

The following topics are available.

- 12.1.1 LISA Project
- 12.1.2 LISA Sample Project
- 12.1.3 Create a New Project
- 12.1.4 Open a Project
- 12.1.5 Creating a new Document in Project
- 12.1.6 Importing Files into a Project
- 12.1.7 Creating Files in Project Root

12.1.1 LISA Project

12.1.1 LISA Project

From 5.0, a new concept of a LISA "Project" is introduced within LISA Workstation.

Since 5.0, now all the LISA Test cases, Virtual Services, Staging documents, Suites, Data Sets etc need to be under a valid LISA Project. In short a LISA Project is now a home of all LISA Documents.

It is mandatory to open a **LISA Project** before you start creating any Test cases, Virtual Services, Staging document etc.

A LISA Project holds a file called **lisa.project**, and a mandatory folder called Configs, that should hold at least one default configuration file **project.config** and a suite document **Alltestsuite.ste**, amongst other folders like Staging documents, Data sets, Monitors, Test Cases etc.

Initially when you install LISA, there will not be any Project directory in the %LISA_HOME%. But once you create a Project, you will see the Project directory existing in the %LISA_HOME%.

Note: At a time, only one Project can be opened in LISA workstation.

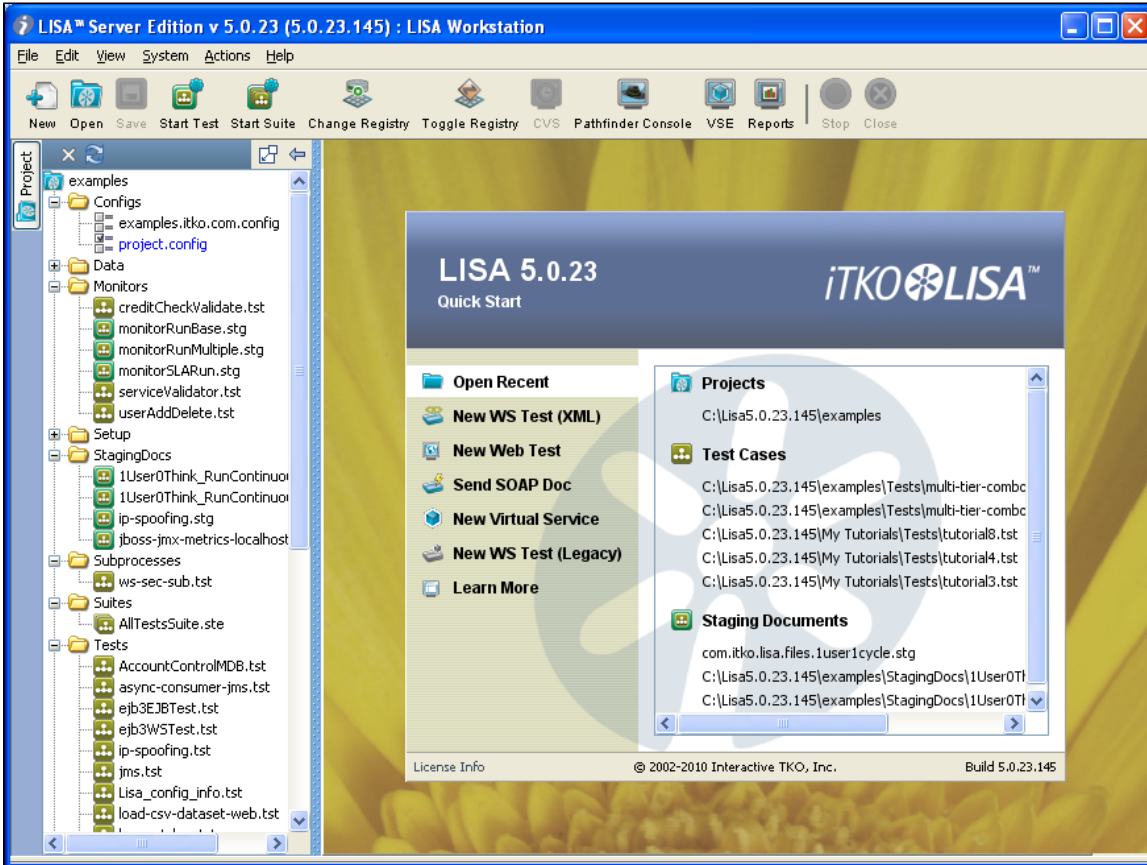
12.1.2 LISA Sample Project

12.1.2 LISA Sample Project

Once you start LISA, by default it opens a Project named "**examples**".

This is a default LISA project and is automatically created by LISA. It contains the entire example Test Cases that are shipped with LISA.

The "examples" project contains samples of Configuration files, Data Sets, Monitors, Staging Documents, Sub process documents, Test cases, Test Suites etc a shown below:



By default, the "examples" project contains the following configuration files:

- Project.config
- Examples.itko.com.config

Out of which the "**project.config**" is shown in "blue" color indicating that it is the **Active** configuration.

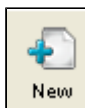
You can invoke any of the above documents by double clicking the same to open in the editor window

12.1.3 Create a New Project

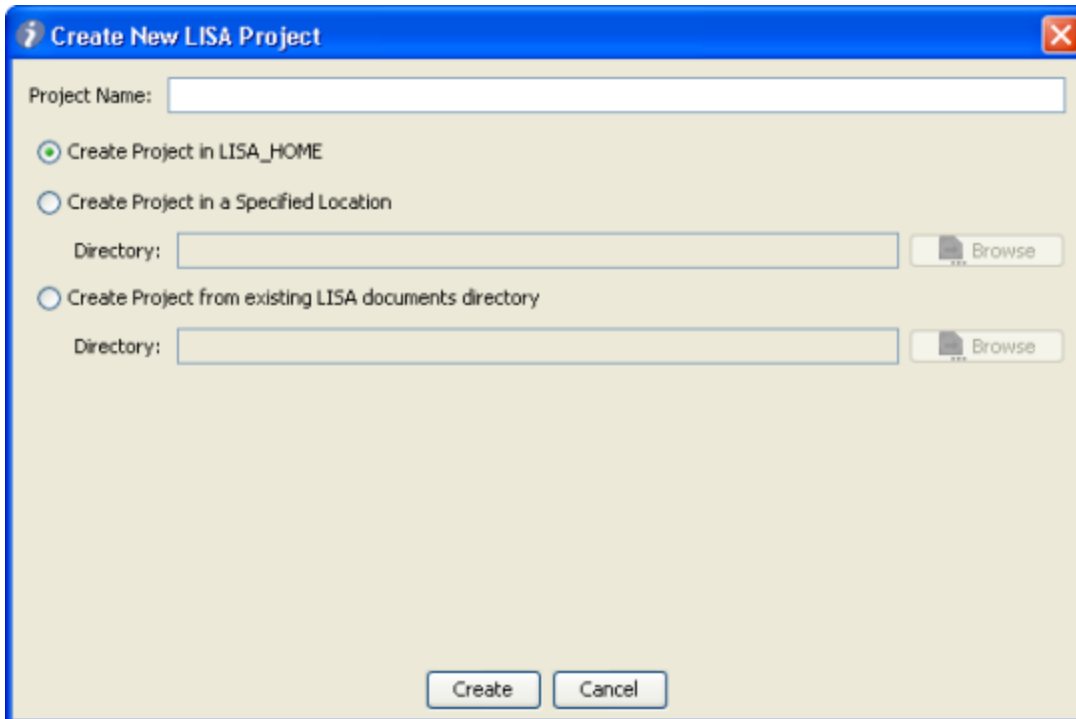
12.1.3 Create a New Project

To create a new Project,

Click **File > New > Project** in the main menu



Or click on the **New** icon on the main toolbar, to open the screen as shown below:

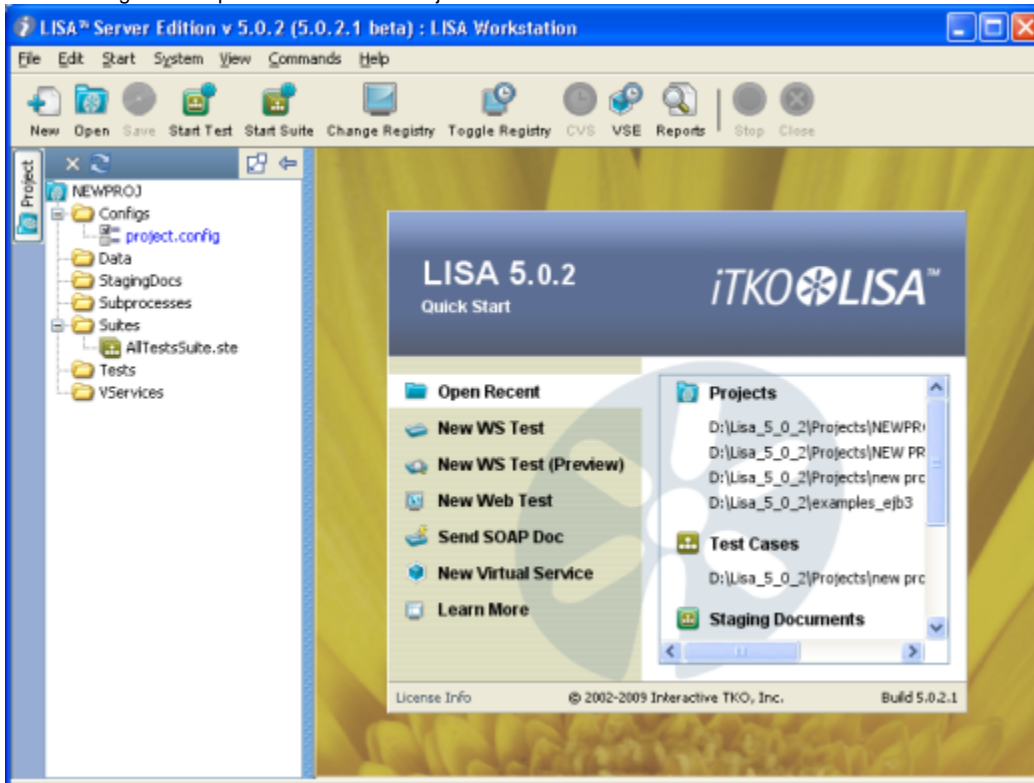


- **Project Name** - Enter the name of the Project.

Choose the place of the Project, from the available options:

- **Create Project in LISA_HOME** – Click to create a Project in the project sub folder of LISA home directory.
- **Create Project in Specified Location** – Click to specify the location of the Project. Click **Browse** to browse to the directory.
- **Create Project from existing LISA documents directory** – Click to create a new Project from a existing projects file. Used for importing a projects file from another directory.
- Click **Create** to Create the Project.

The following screen opens once the new Project is created:



As you see below, a **Project tree** is seen on the left hand side in a **Tray panel** - this is a docable panel and can be opened or closed by clicking on the Project button.



For a new project, created from scratch, the following folders will be created automatically in the Project panel: Configs, Data, StagingDocs, Subprocesses, Suites, Tests, VServices

On the top of the Project tree, you will see the new Project created. Also there are its sub-folders – Configs, Data, Staging Docs, Suites, Tests and VServices created.



The Project toolbar also has buttons to **Close or Refresh** the project respectively.

There's also a **".settings"** directory which is not seen in the Projects pane. This directory is used for saving some settings internally and can be seen in the file system in the Project directory.

The **Quick start** panel on the right will also change to have other menu items like New WS Test, New Web Test etc as shown below:

You will notice a Configuration – **project.config** file and a Test Suite **AllTestsSuite.ste** file being automatically added to this project by default.

Project.config is the default configuration for the entire project. This default config can be overridden by another configuration.

AllTestsSuite.ste is a suite file created for the "Tests" directory of this project and uses the basic default staging document.

Note:

1>Once you create a new project, that project can now be seen in the Projects directory in the file system.

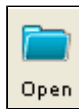
2> All the folders created in the Project pane are similar to the ones created in the File system. Check your LISA_Home to see the same folder structure and files in there.

12.1.4 Open a Project

12.1.4 Open a Project

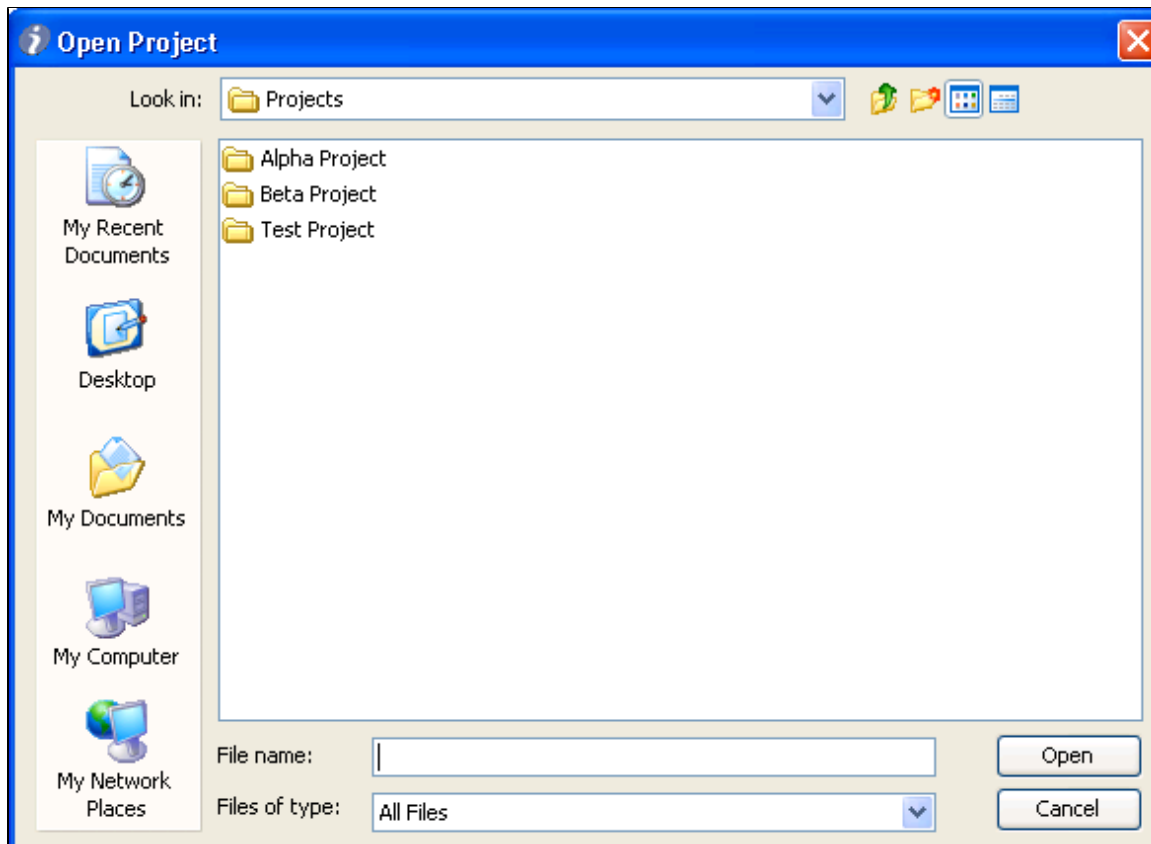
You can open the LISA project by either choosing the project directory or the lisa.project file.

To open a Project,



- Click the Open Project icon on the main toolbar.

This will open the Open Project dialog box, which will have a list of earlier created Projects as shown below:

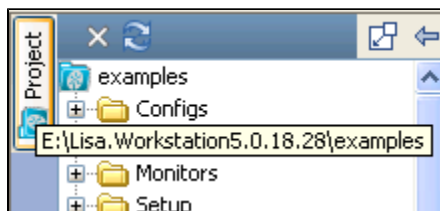


- Select the required project and click Open to open the Project.

Note: If you have been using LISA, then when you re-open, it will open the most recently used Project by default.

When you open a LISA project, it will have a list of folders and sub folders listed in the left panel. Only those folders will open/expand in the project node, which have atleast one of LISA files like .tst, .vsm, .config, .ste, .stg existing within them.

Once the project is opened, clicking on the **Project button** will enable a tooltip which shows the complete project path as shown below:

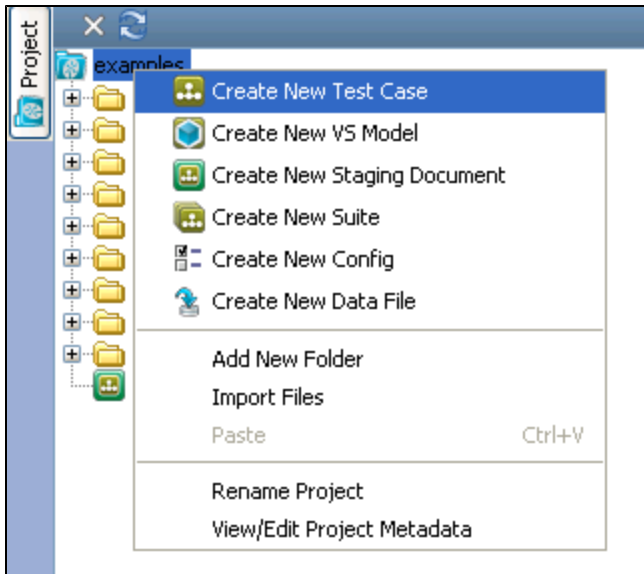


12.1.5 Creating a new Document in Project

12.1.5 Creating a New Document within a Project

You can create various documents within a Project from the project panel.

When you right click on the "**project**" node, you will see the following menu:

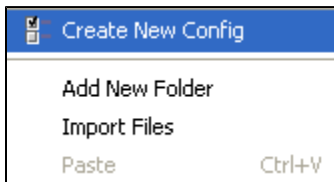


- Create new Test Case
- Create new VS Model
- Create new Staging document
- Create new Test Suite
- Create new Config
- Create new Data File
- Add New Folder
- Rename Project
- Import Files

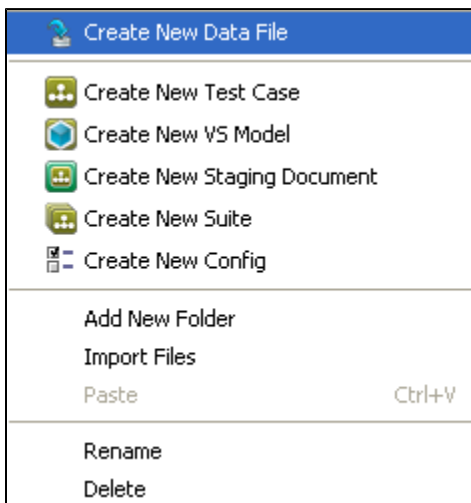
Note - You are not limited to keeping a certain kind of file (like .tst), under the recommended folder (like Tests). It can stay anywhere within the project. There's an exception for .config files & data resources though, that they need to stay under Configs & Data folders respectively.

When you right click any **"folder"** within the project (Ex Config, Data, StageDoc, Tests), you will see the menu as per the selected folder as shown below:

For example, If you right click the **"Configs"** folder, here is what you see:



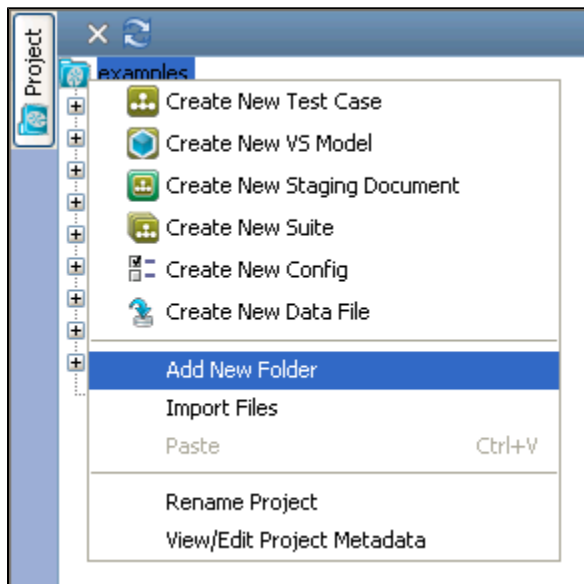
Where as if you right click the **"Data"** folder, here is what you see:



Adding a New Folder

To add a new folder in the Project folder,

- Right click the Project and click **Add New folder**.

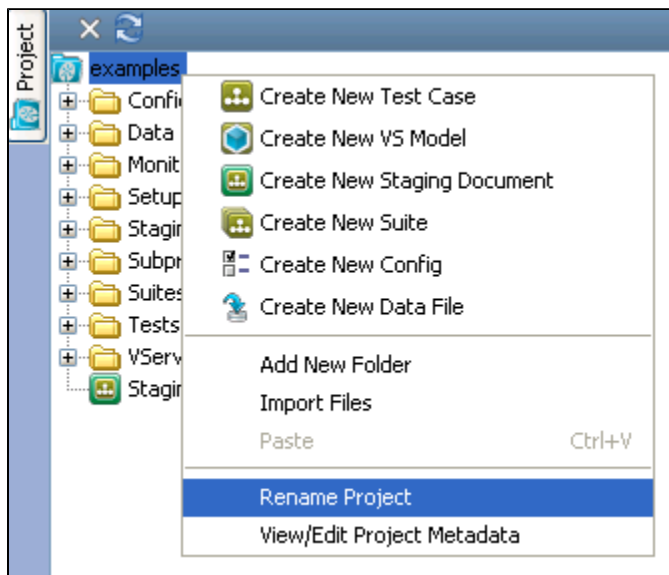


- Enter the name of the new folder and click OK to add that folder in the Project directory.

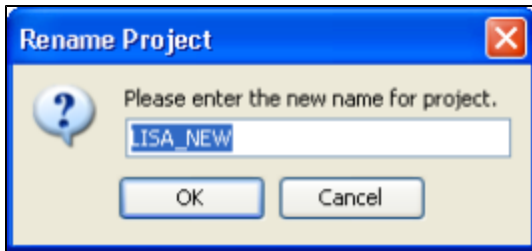
Renaming a Project

To rename a project,

- Right click the Project name to open a menu.



- Click **Rename Project** to open a rename Project dialog box as below:



Enter the **New name** of the project and click **OK** to change the project name.

12.1.6 Importing Files into a Project

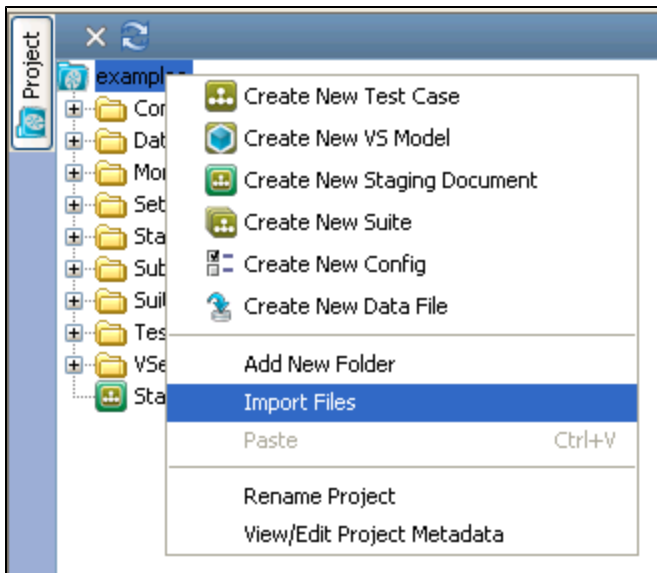
12.1.6 Importing Files into a Project

You can import files into the existing Project.

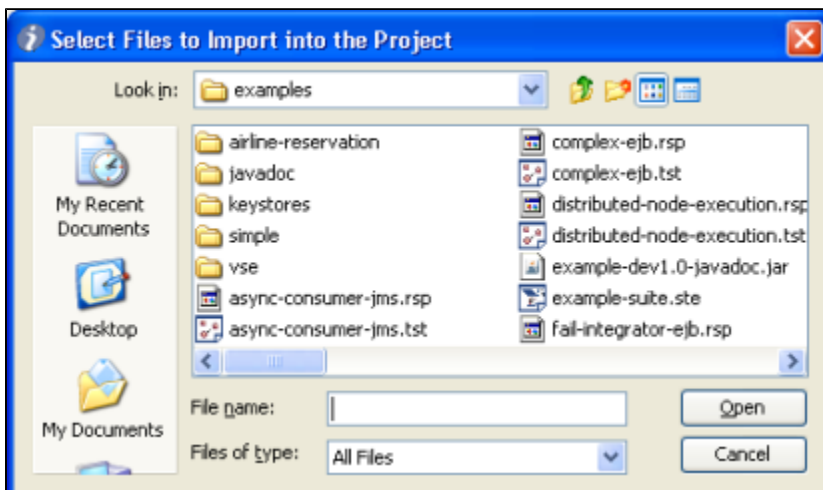
This is specially useful if you want to import old versioned files into 5.0 version.

To import a file,

- Right click the Project name and select **Import files**.



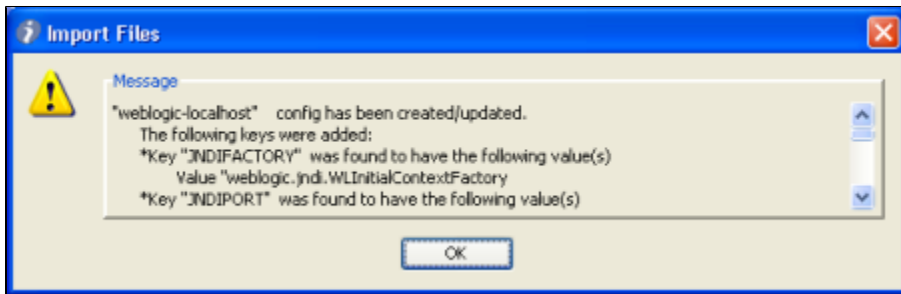
- The Import files dialog box will open as follows:



- Select the file to be imported

- Click open to import the selected file into this Project.

Once the file is imported, LISA pops up a message regarding the imported files as below:

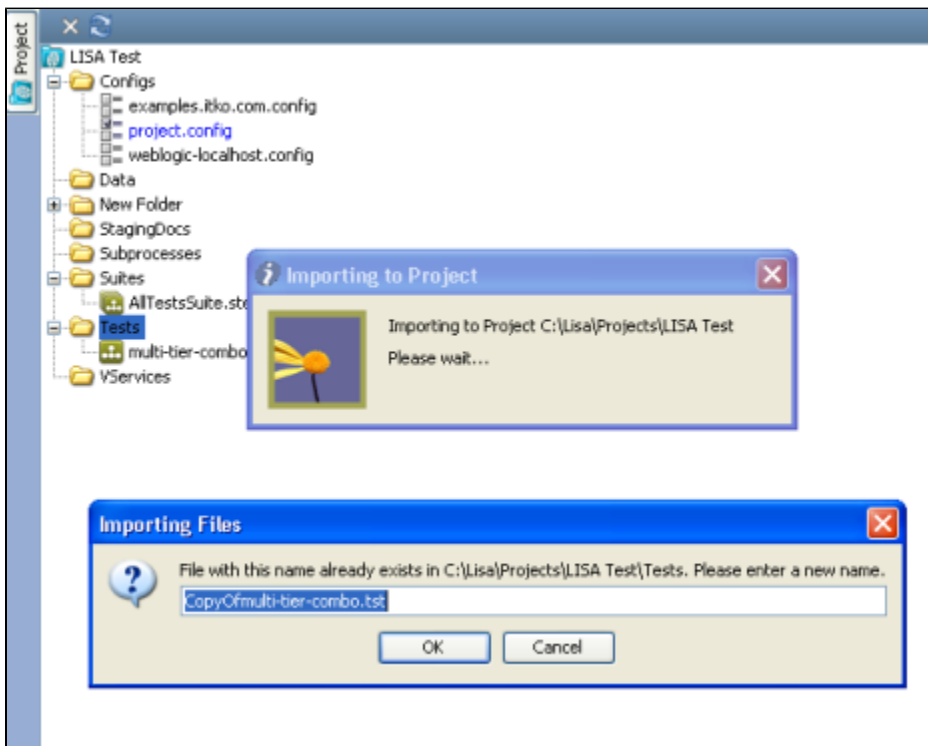


Similarly you can right click the Config folder, Data folder, Staging docs folder, Tests folder and others in the Project directory and import files into these folders.

Importing Files with same name

Incase you import files with same name, LISA will detect that.

To avoid the issues with already existing file names, LISA allows you to rename or make a copy of the newly imported file as shown below:

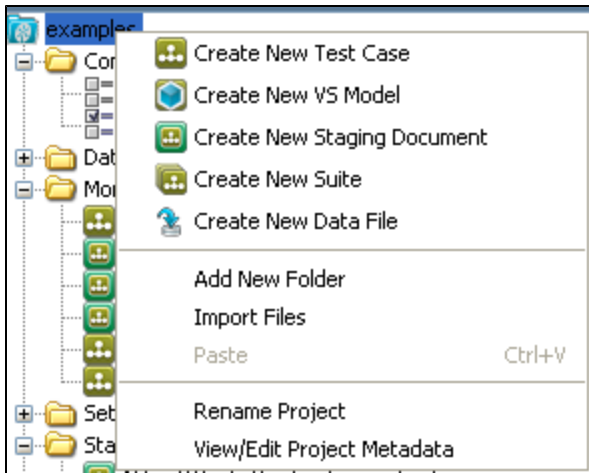


- Enter the new name of the file and click **OK** to add this file to the Project.

12.1.7 Creating Files in Project Root

12.1.7 Creating new files in Project Root

You can also create documents i.e.; Configs, Staging docs, Test cases, Test Suites etc from the Project root directory.



For example, let's create a staging document from the Project node.

- Right click the Project root folder "examples"
- Click on "Create New Staging Document"
- Enter the name of the new staging document.
- Click OK to create the new staging document.

The document is created, but cannot be seen in LISA Workstation.

To view this document, open the explorer window and under Project root directory, you will see this new document created.

You can copy paste this document in PROJECT_ROOT/StageDoc directory and restart LISA Workstation to view the document.

12.2 Building Test Steps

12.2 Building Test Steps

A Test Step is an element in the LISA Test Case workflow that represents a single test action that is to be performed. There are two major **categories of Test Steps**.

Most test steps perform an action on the **'system under test'**, and evaluate the response. Some common examples are testing an enterprise Java bean (EJB) method, a web service, or a message through messaging service provider.

A second category of test steps perform **"utility functions"**, such as data conversion, data manipulation (such as encoding), logging, and writing information to files etc.

Both categories of steps go into the building of a Test Case.

The following topics are available in this chapter.

- 12.2.1 Adding Test Step
- 12.2.2 Configuring Test Steps
- 12.2.3 Adding Filter to a Step
- 12.2.4 Adding Assertion to a Step
- 12.2.5 Adding Data Set to a Step
- 12.2.6 Drag and Drop a Test Step
- 12.2.7 Configuring the Next Step
- 12.2.8 Triggering Next Step
- 12.2.9 Common Test Step Actions
- 12.2.10 Types of Steps

12.2.1 Adding Test Step

12.2.1 Adding Test Step

To add a new test step,



Click on the **Add Step button** on the Test Case toolbar Or from the main menu select **Commands > Create a New Step**

You can also add a step in a specific place in the workflow by **right clicking the step** in the workflow to open a menu. Click **Add Step After** and select the required test step.

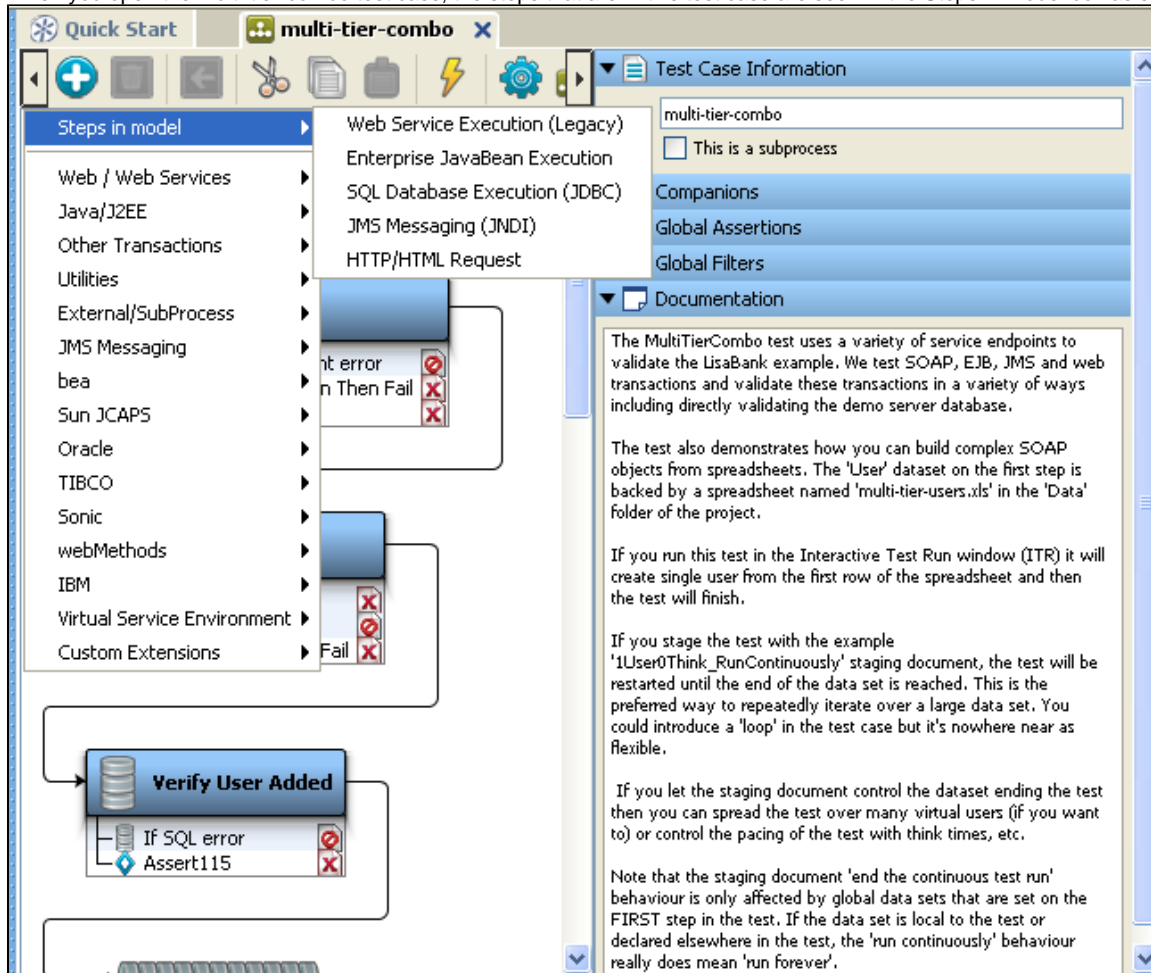
Example

For the purpose of illustration, we will add a new **Dynamic Java Execution** step to the **multi-tier-combo** Test Case in the examples directory (multi-tier-combo.tst). We will add a new **Dynamic Java Execution** step to the **multi-tier-combo** Test Case, after the **get user** step.

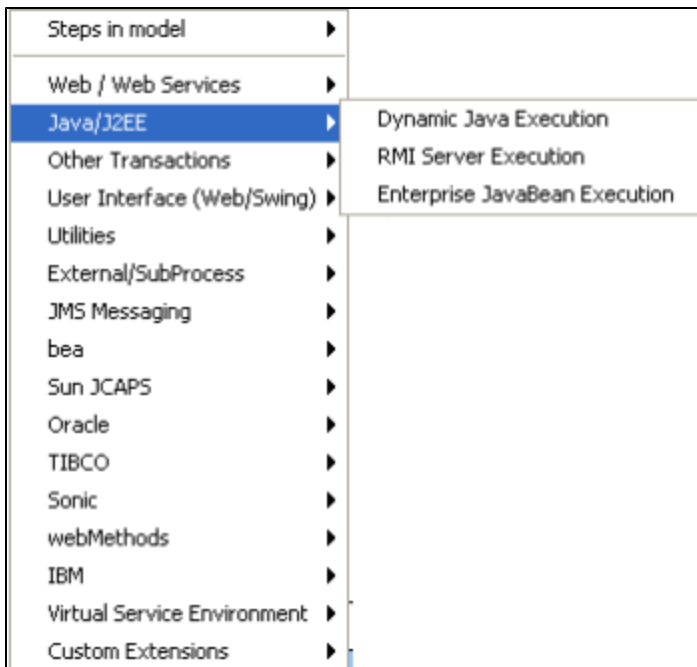
Clicking on the **Add Step** button will open a panel which has the listing of the common **Test Steps**.

Steps in Model: If steps have been already created in the Test Case, as in this Test Case, the panel will also show **Steps in Model** on top, which will list all the steps present in the Test Case. For a new test case, this field is empty.

When you open the multi-tier-combo test case, the steps that are in this test case are seen in the Steps in Model box as shown below:

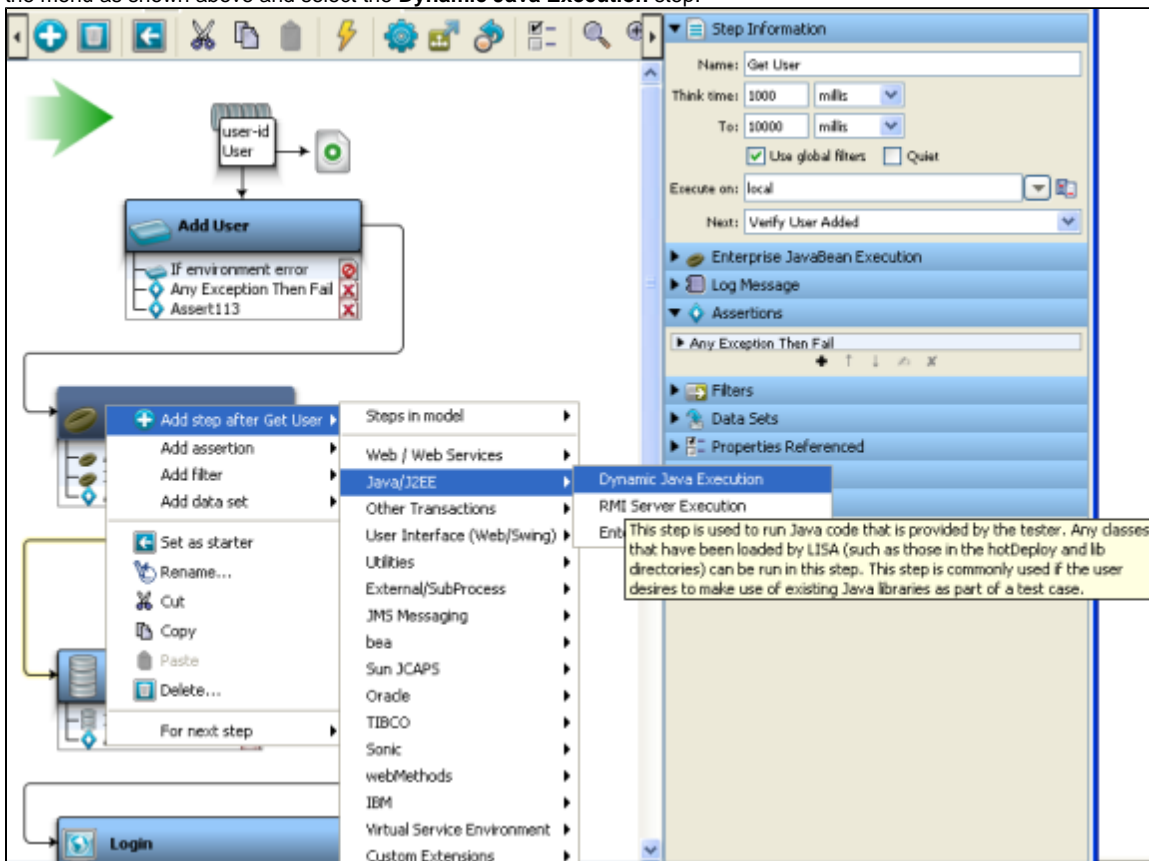


Select the **main category** of the step to be configured (Ex: Web/Web Services, Java/J2EE, Utilities etc), which will open the **sub category** as shown below:

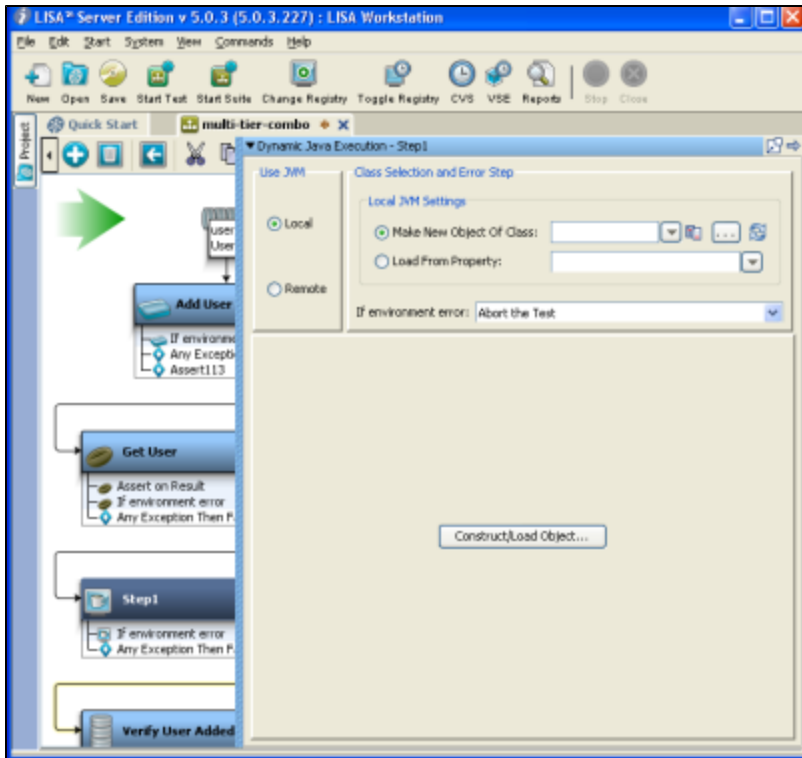


Click on the **Step**, to add it to the Test Case. This will open its step editor.
Note: Each step has a different step editor

As we will be adding the new **Dynamic Java Execution** step after the **get user** step, select **get user** step. Right click the **get user** step to open the menu as shown above and select the **Dynamic Java Execution** step.

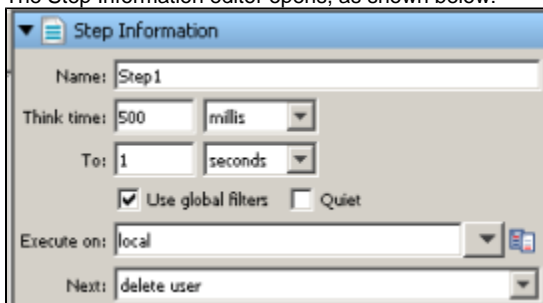


The step is added and the **Step editor** for the **Dynamic Java Execution** step opens up as shown below:



Adding Step Information

To add the basic step information, open and expand the **Step Information** tab in the Elements tree in the right panel. The Step Information editor opens, as shown below:



Enter the following parameters:

- **Name:** The name of the step. By default it will appear as **Step X** (X being a number). You can rename the step in this text box.
- **Think Time:** The amount of time the Test Case should wait before executing this step. This provides the ability to simulate the amount of time it takes a user to decide what to do before taking action. To specify how the time is calculated, select the time unit by clicking the appropriate radio button (Millis, Seconds, Minutes), then enter a value in the form "from-to", where "from" is the minimum amount of time to wait, and "to" is the maximum amount of time to wait. LISA will pick a random think time within this range. For example, to simulate a user think time for a random amount of time between three and five seconds, click Seconds, and enter a value of 3-5. To enter a precise think time, just enter a single number.
- **Use Global Filters:** Check this box if the step should be instructed to use global Filters. For more information on Filters, see [Adding Filters](#).
- **Quiet:** Check this box if you want LISA to ignore this step for response time events, and performance calculations.
- **Execute On:** Specifies the Simulator that the step is to be run on. Specific Simulators should be specified for steps that must run on a specific machine. For example, when reading a log file the step should be run on the machine where the log resides.
- **Next:** The next step to execute in the test. If the step in question ends the Test Case execution, you will not specify a next step. An Assertion that fires in this step would override this value.

12.2.2 Configuring Test Steps

12.2.2 Configuring Test Steps

To configure a particular test step,

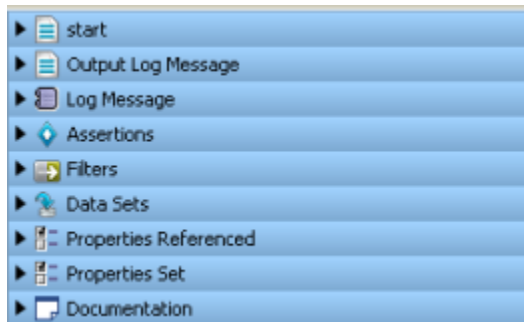
- Add that step in the LISA Model Editor.

Once the step is added, a different test step panel appears in the Element tree.

You can configure the test step by setting the parameters in the configuration elements (Assertions, Filters, Data Sets etc)

Details of the each Test Case/step element can be seen by clicking the element arrow

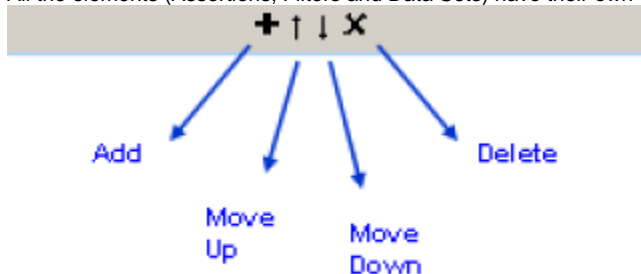
▶ next to the configuration elements to expand it as shown:



- **Step Information (Start):** This is the first element in the element tree and carries the name of the step as its label (**Step1** in the figure above). After expanding it, one may be able to change the name of the test step, and set the next step to be executed after the current step is completed. The other options are explained in [Adding Test Step](#) section in this chapter.
- **Main Step (Output log message):** This appears next and has the title of the selected Step (Output Log Message in figure above). Each step is different and has its own specific configuration requirement, and hence has a custom editor to provide the information needed to run the step and test it (and possibly to get a response as well). This custom editor opens up when the element is expanded.
- **Log Message:** This appears after any step is added in LISA and allows you to set the log message that appears after the step execution is complete.
- **Assertion Element:** The addition and configuration of one or more Assertions. In an Assertion configuration, one also needs to set the next step to be executed when the Assertion fires.
- **Filter Element:** The addition and configuration of Filters. Filters are added under the Filter element of each test step.
- **Data Set Element:** The addition and configuration of one or more Data Sets that apply to the test step. Data Sets are fired before executing a test step. This means that any property that the Data set sets, is available to the test step.
- **Properties Referenced:** A read-only list of properties that the step references (reads)
- **Properties Set:** A read-only list of properties the step sets (assigns a value)
- **Documentation:** Notes accompanying the test step.

Step Element Toolbar

All the elements (Assertions, Filters and Data Sets) have their own toolbar at the bottom of the Element tab as shown below:



You can add/delete an element to a step, by clicking the icons at the bottom of the individual elements.

Note: LISA has many built-in steps. For details on each of these, refer to the [Reference Guide](#).

For detailed information on adding Filters see [Adding Filters](#).

For detailed information on adding Assertions see [Adding Assertions](#).

12.2.3 Adding Filter to a Step

12.2.3 Adding Filter to a Step

Filters are added under the Filter element of each step in the right panel.



For detailed information on adding and configuring Filters see [Adding Filters](#).

12.2.4 Adding Assertion to a Step

12.2.4 Adding Assertion to a Step

Assertions are added under the Assertion element of each step in the right panel.

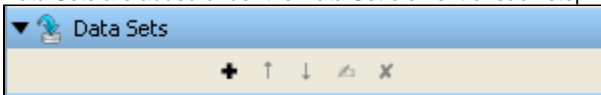


For detailed information on adding and configuring Assertions see [Adding Assertions](#).

12.2.5 Adding Data Set to a Step

12.2.5 Adding Data Set to a Step

Data Sets are added under the Data Set element of each step in the right panel.



For detailed information on adding and applying Data Sets see [Adding a Data Set](#).

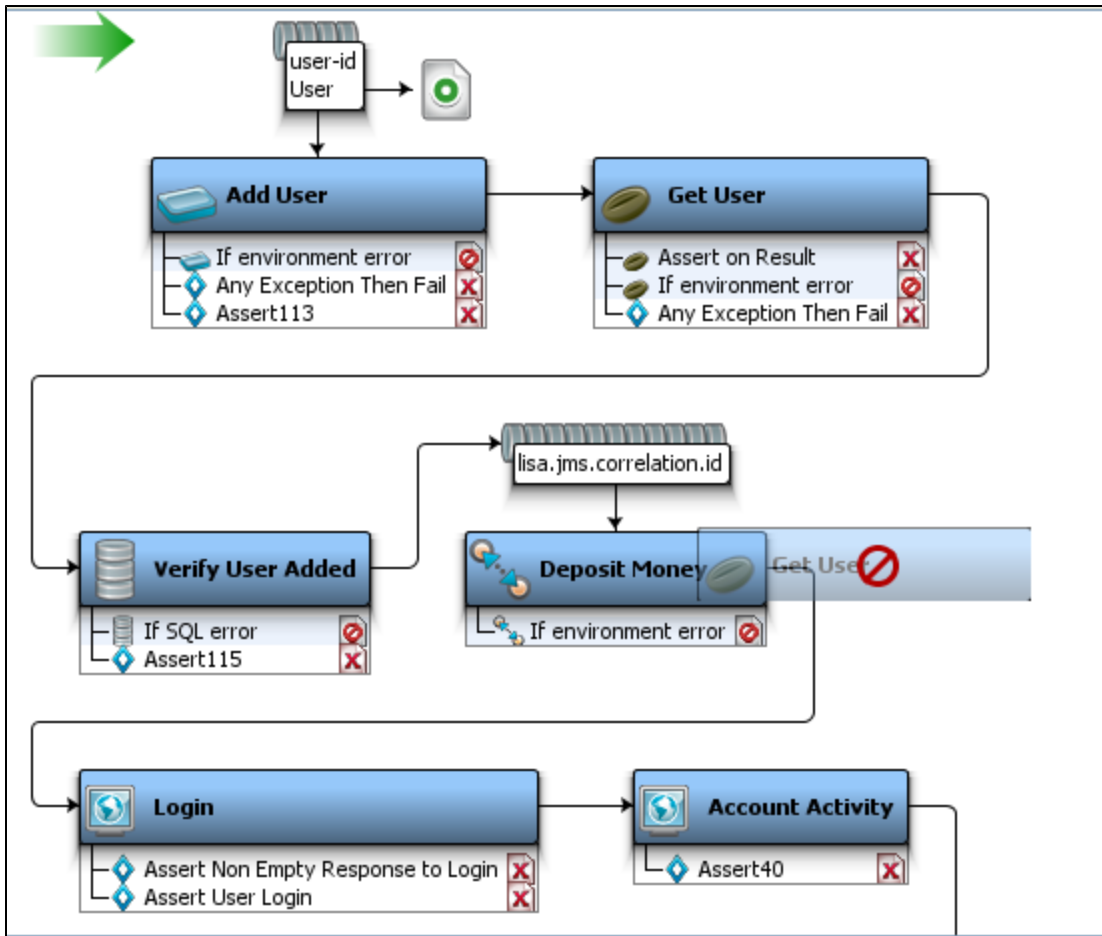
12.2.6 Drag and Drop a Test Step

12.2.6 Drag and Drop a Step

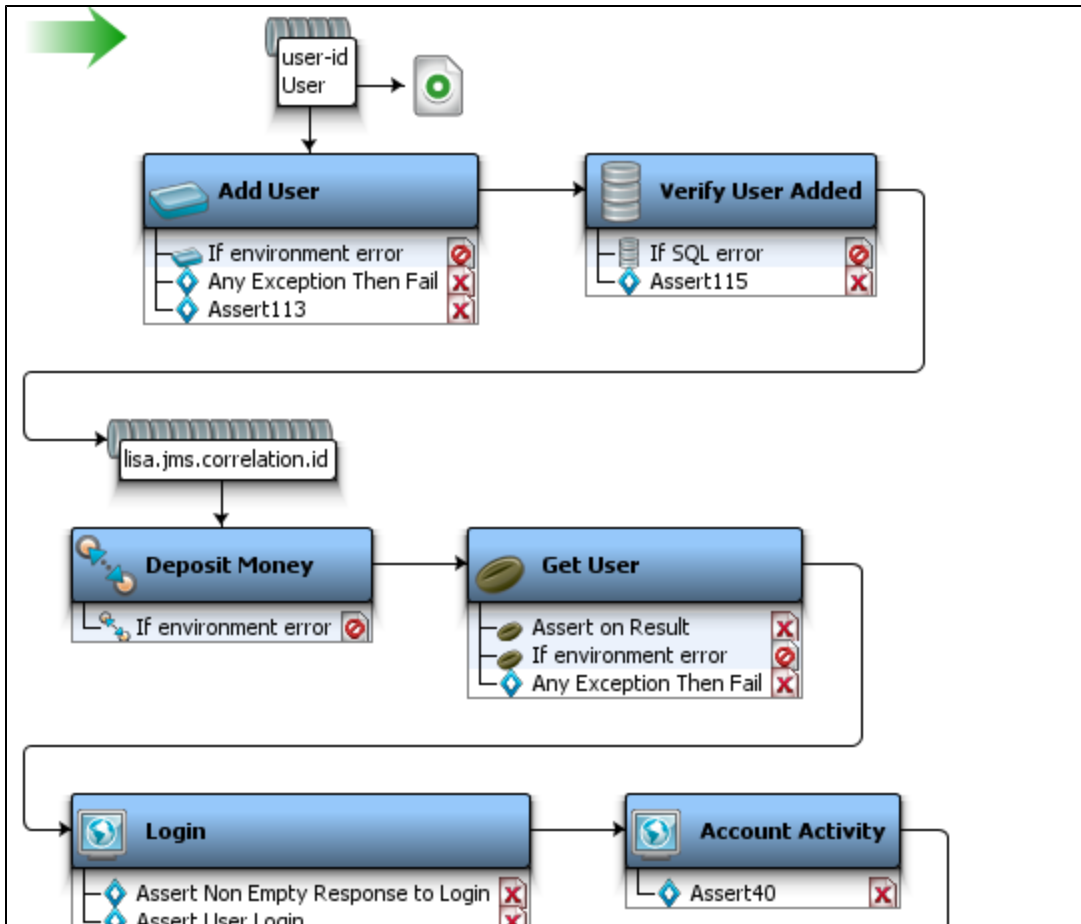
Within the LISA Workstation, you can **drag and drop** a test step to a different location.

- Select the step to be dragged
- Hold the mouse and drag the step to the desired location.

For example, we are dragging the **Get User** step to the **Deposit Money** Step as shown below:



Once you leave the mouse in the desired location, it will drop the test step there as shown below:

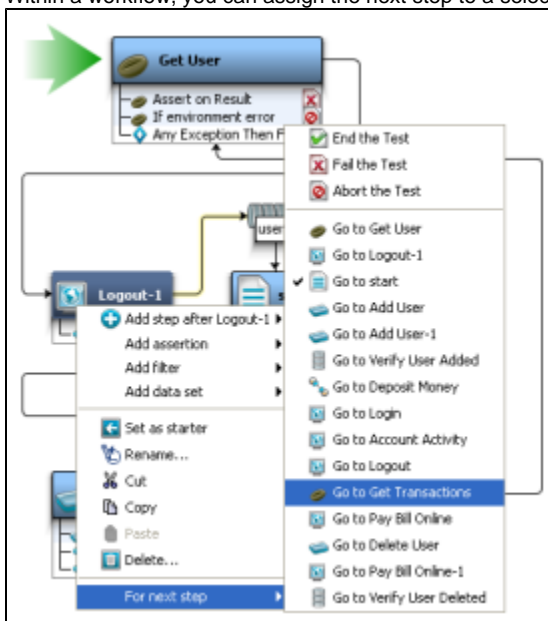


You will now see the Get User step next to the Deposit Money step.

12.2.7 Configuring the Next Step

12.2.7 Configuring the Next Step

Within a workflow, you can assign the next step to a selected test step.



- Click on the Step for which you want to decide the next step. (logout1)

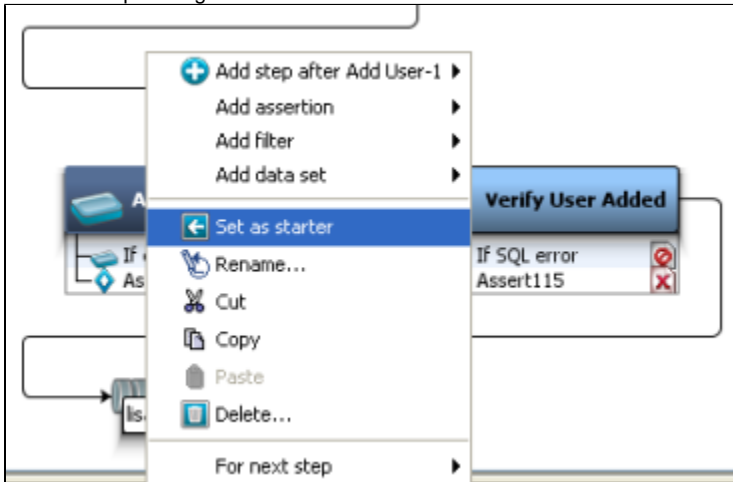
- Right click and select "For next Step" and click on the desired Next Step (Get Transactions)

Accordingly the workflow in the Model Editor will change. This will also change the information in the "Next" field in the Step Editor.

Setting a Step as Starter Step

You can set a step to be a starter step within the workflow.

- Click the Step and right click to select "Set as Starter" as shown below:



This will set the selected step as the first step in the workflow.

Note – This option is not available to the first step in the workflow.

12.2.8 Triggering Next Step

12.2.8 Triggering Next Step of an Assertion

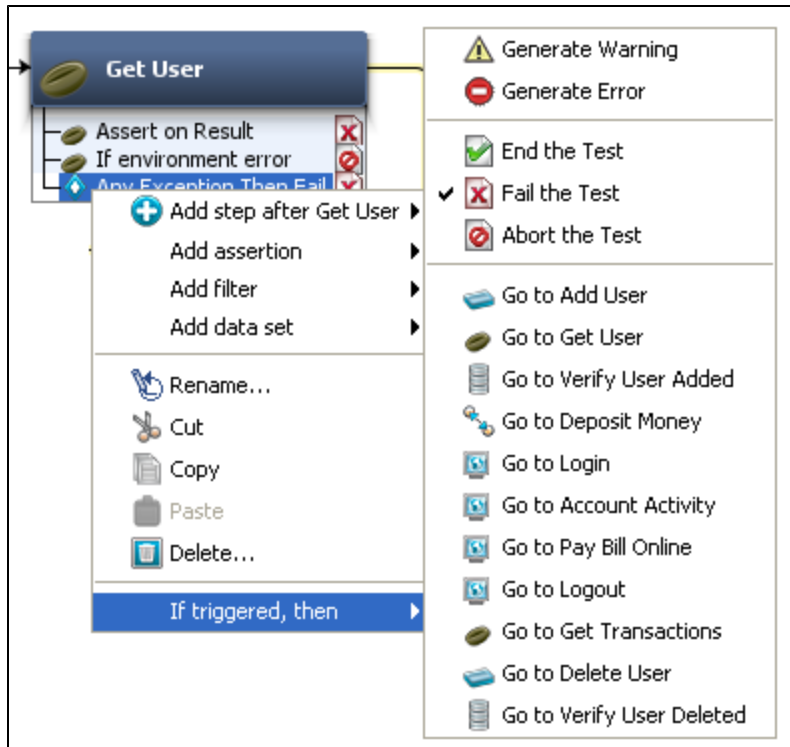
If you are using an Assertion, to configure the **Next** step – there are following options:

- Generate Warning
- Generate Error
- End the Test
- Fail the Test
- Abort the Test

For example, we have selected the Assertion in the **Get User** step as shown below:

- Select a Step and right click on the **Assertion** of a step to open a menu as shown below:

Note - You can select the next step for this Assertion, once it is triggered.



Generate Error Step

Test steps can either pass or fail. When they fail they don't actually fail the test. To fail the test, the test step sets the test case workflow to execute the "fail" step; this implicitly makes the step consider failing.

If they explicitly fail they generate an error and raise a NODEFAILED event and continue the test step.

Note: This is the "Continue" step in all previous versions of LISA.

Generate Warning Step

When the test step fails, there is also an "Ignore" type of a step logic that will not raise an alarm or event. Hence this will not change the test case workflow. This is the Generate Warning Step.

Note: This is the "Continue Quite" step in all previous versions of LISA.

End Step

The **End step** is to bring an end to a workflow, and is run when a workflow completes successfully. The entire Test Case is deemed to be successful if the execution reaches this step.

Fail Step

The **Fail step** is the end of a workflow, and is run when a workflow fails due to an error event. The entire Test Case is deemed to have failed if the execution reaches this step. The fail step is the default for many exceptions internal to LISA (for example, an exception in an EJB), but it can be set as the next step by Assertions to fail a Test Case.

Abort Step

The **Abort step** is also the end of a workflow, and is run when a workflow is abruptly aborted. The Test Case is deemed to be aborted (without completion) if this step is reached.

12.2.9 Common Test Step Actions

12.2.9 Other Step Actions

- Editing a Step
- Deleting a Step
- Reordering a Step
- Renaming a Step

- Copying a Step
- Cutting a Step
- Pasting a Step
- Assigning the Next Step
- Starter Step

Editing/Modifying a Step

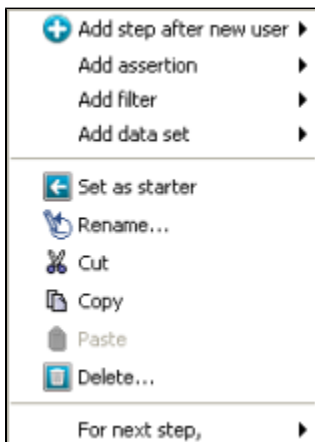
To modify a step,

Click on any of the Step elements associated with that step, modify that step and save the test.

Deleting a Step

To delete a Step:

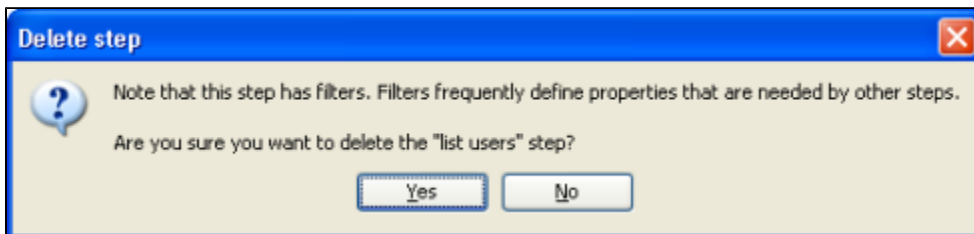
- Select the step in the workflow and right click to open a menu.
Click **Delete** to delete the step.



Note - Be Careful when you want to delete a test step which has a filter associated to it.

Ideally you **should not** delete a test step that has Filters associated to it. Filters define properties that could be needed by other test steps, and deleting a step also means deleting the filter.

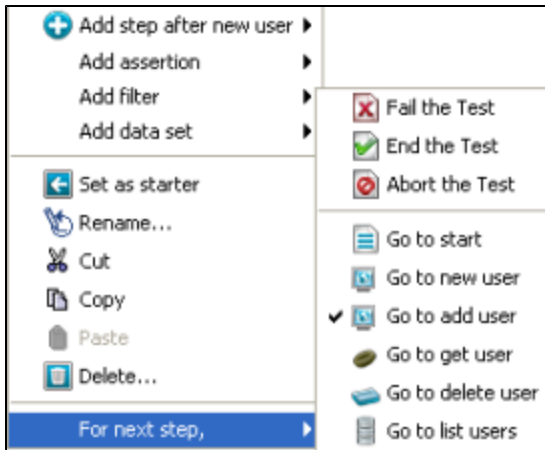
Incase you attempt to delete such a test step, you get the following message.



Reordering a Step

To reorder a step:

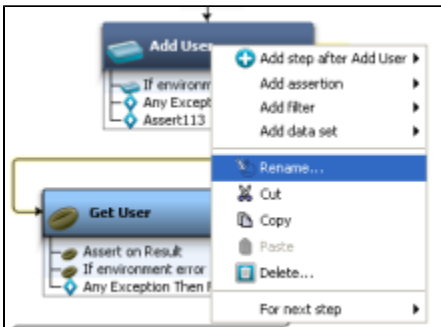
- You can use the drag and drop facility in the Model Editor. Click on the Step and drag it to the new location to rearrange the workflow.
- Or select the step in the workflow and right click to open a menu. Click **For Next Step** and select the step to reorder.



Renaming a Step

To rename a step:

- Right click a step to open a menu and click Rename as shown below:



- Open the Step Information panel and rename the step in the "Name" field.

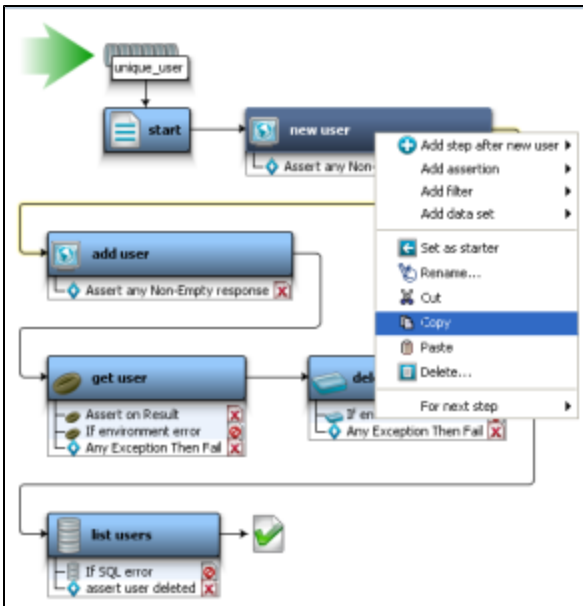
Once you rename the step in the workflow, the same will be reflected in the "Step Information" tab in the right panel or vice versa. Any step information pointing to this step will be updated.

The next step is updated for linear and non-linear workflows in this case.

Copying a Step

You can copy a step and paste it anywhere within the model editor.

- Select the step to be Copy. (New User)
- Right click the step and click **Copy**.

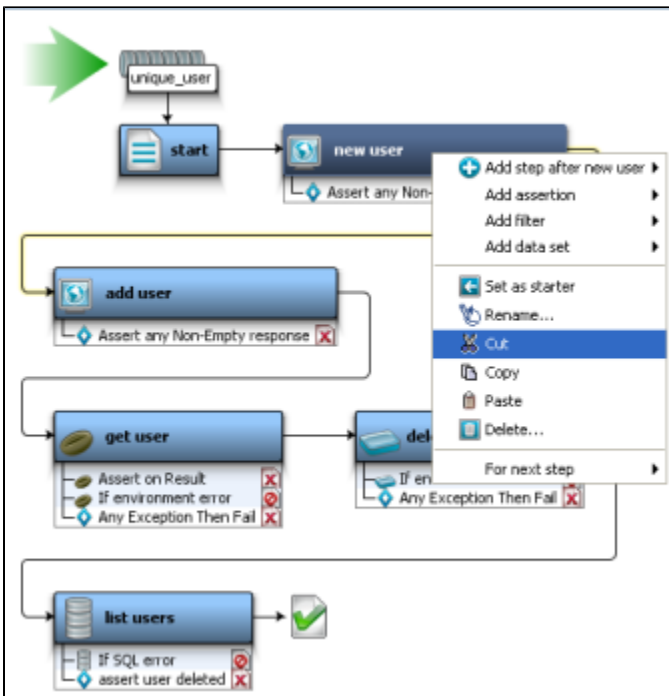


This will copy the selected test step, in this case "new user".

Cutting a Step

You can cut a step and paste it anywhere within the model editor.

- Select the step to be Cut. (New User)
- Right click the step and click **Cut**.

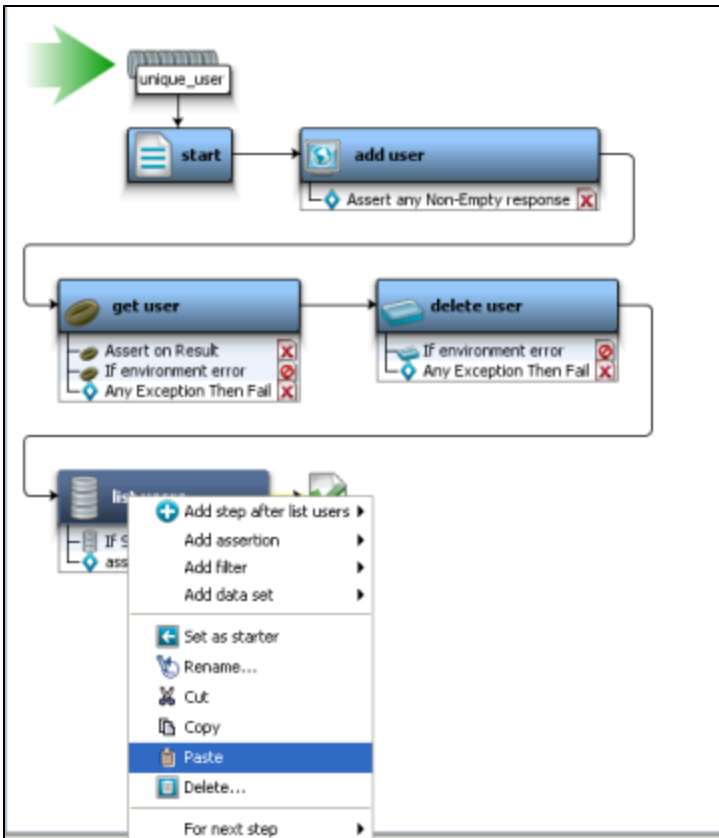


This will Cut the step (New User) from the Model Editor.

Pasting a Step

You can paste the step on any desired step in the workflow. Once you perform the Paste operation, the step will be added after the selected step within the workflow.

- Select the **Step** after which you need to paste the Cut step.
- **Right click** and select **Paste**.

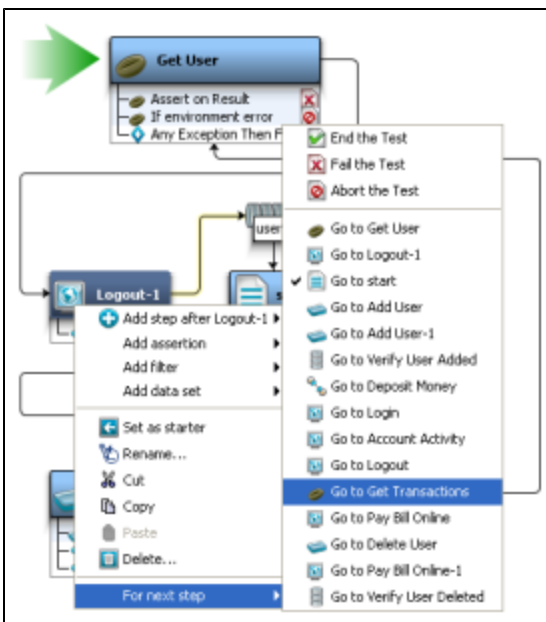


This will paste the step (New User) after the selected step (List user).

Note – you need to right click and paste the step on a selected step within the model editor.

Assigning the Next Step

Within a workflow, you can assign the next step to a selected test step.



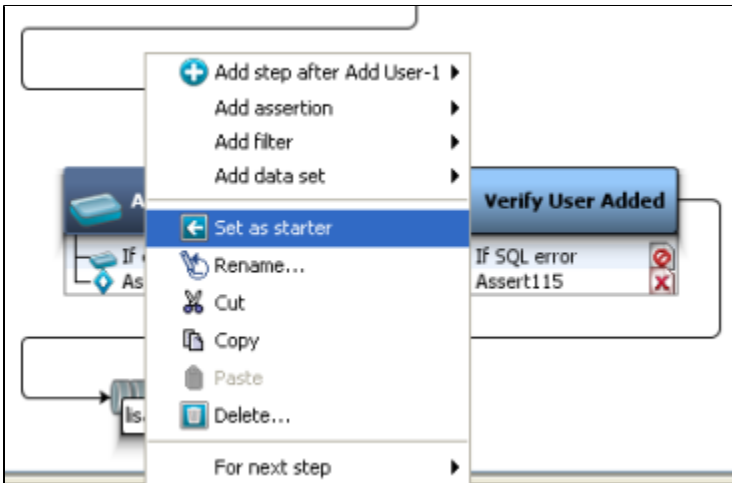
- Click on the Step for which you want to decide the next step. (logout1)
- Right click and select "For next Step" and click on the desired Next Step (Get Transactions)

Accordingly the workflow in the Model Editor will change. This will also change the information in the "Next" field in the Step Editor.

Setting a Step as Starter Step

You can set a step to be a starter step within the workflow.

- Click the Step and right click to select "Set as Starter" as shown below:



This will set the selected step as the first step in the workflow.

Note – This option is not available to the first step in the workflow.

12.2.10 Types of Steps

12.2.10 Types of Steps

LISA 5.0 provides the following test steps:

Web/Web Services

- HTTP/HTML Request
- REST Step
- Web Service Execution (XML)
- WSDL Validation
- Raw SOAP Request
- Base64 Encoder Step
- Multipart Mime Step
- SAML Assertion Query
- Web Service Execution (Legacy)
- Start or Stop Web Server (Legacy)

Java/J2EE

- Dynamic Java Execution
- RMI Server Execution
- Enterprise Java Bean Execution

Other Transactions

- SQL Database Execution (JDBC)
- Corba Execution

Utilities

- Save Property as Last Response
- Output Log Message
- Write Properties to File
- Read Properties to File
- Do-Nothing Step
- Parse Text as Response
- Audit Step
- Base64 Encoder Step
- Checksum Step
- Convert XML to Element Object
- Compare Strings for Response Lookup

- Compare Strings for Next Step Lookup

External/Sub Process

- Execute External Command
- File System Snapshot
- Execute Sub Process
- Execute JUnit Test Case/Suite
- Read a File (Disk, URL or Class Path)
- FTP Step

JMS Messaging

- JMS Messaging (JNDI)
- Message Consumer

bea

- Weblogic JMS (JNDI)
- Message Consumer
- Read a File (Disk, URL, or Classpath)
- Web Service Execution (XML)
- Raw SOAP Request
- FTP Step
- Web Service Execution (Legacy)

Sun JCAPS

- JCaps Messaging (Native)
- JCaps Messaging (JNDI)
- Message Consumer
- Read a File (Disk, URL, or Classpath)
- Web Service Execution (XML)
- Raw SOAP Request
- SQL Database Execution (JDBC)
- FTP Step
- Web Service Execution (Legacy)

Oracle

- Oracle OC4J (JNDI)
- Oracle AQ (JMS)
- Oracle AQ (JPub)
- Message Consumer
- Read a File (Disk. URL, or Classpath)
- Web Service Execution (XML)
- Raw SOAP Request
- SQL Database Execution (JDBC)
- FTP Step
- Web Service Execution (Legacy)

TIBCO

- TIBCO Rendezvous Messaging
- TIBCO EMS Messaging
- TIBCO Direct JMS
- Message Consumer
- Read a File (Disk. URL, or Classpath)
- Web Service Execution (XML)
- Raw SOAP Request
- SQL Database Execution (JDBC)
- FTP Step
- Web Service Execution (Legacy)

Sonic

- SonicMQ Messaging (Native)
- SonicMQ Messaging (JNDI)
- Message Consumer
- Read a File (Disk, URL, or Classpath)
- Web Service Execution (XML)
- Raw SOAP Request
- SQL Database Execution (JDBC)
- FTP Step
- Web Service Execution (Legacy)

WebMethods

- webMethods Broker
- webMethods Integration Server Services
- Message Consumer
- Read a File (Disk, URL or Classpath)
- Web Service Execution (XML)
- Raw SOAP Request
- SQL Database Execution (JDBC)
- HTTP/HTML Request
- REST Step
- FTP Step
- Web Service Execution (Legacy)

IBM

- IBM Websphere MQ
- Message Consumer

Virtual Service Environment

- Virtual Service Router
- Virtual Service Tracker
- Virtual Conversational/Stateless Response Selector
- Virtual HTTP/S Listener
- Virtual HTTP/S Live Invocation
- Virtual HTTP/S Responder
- Virtual JDBC Listener
- Virtual JDBC Responder
- Socket Server Emulator
- Messaging Virtualization Marker
- Compare Strings for Response Lookup
- Compare Strings for Next Step Lookup

Custom Extensions

- Custom Test Step Execution
- Java Script Step
- Pathfinder Agent Script step
- Swing Test Step
- Create a Virtual Web service
- Java Protocol Request Listener
- Java Protocol Responder
- Java Pass Through

Note: These step types are described in detail in the [Reference Guide - Part 2](#)

12.3 Creating Test Cases

12.3 Creating Test Cases

A Test Case is a complete specification of how to test a business component in the 'system under test', or in some cases the complete 'system under test'.

A Test Case is persisted as an XML document, and contains all the information needed to test the given component or system. Test cases are created and maintained in Workstation.

To create a Test Case, you will first need to create a Project. Within this project, you can create single or multiple Test Cases.

The following topics are available in this chapter.

- 12.3.2 Opening a Test Case**
- 12.3.1 Creating a Test Case**
- 12.3.3 Saving a Test Case**
- 12.3.4 Test Cases in Model Editor**
- 12.3.5 Branching & Looping in Test Case**
- 12.3.6 Response (.rsp) Documents**

12.3.1 Creating a Test Case

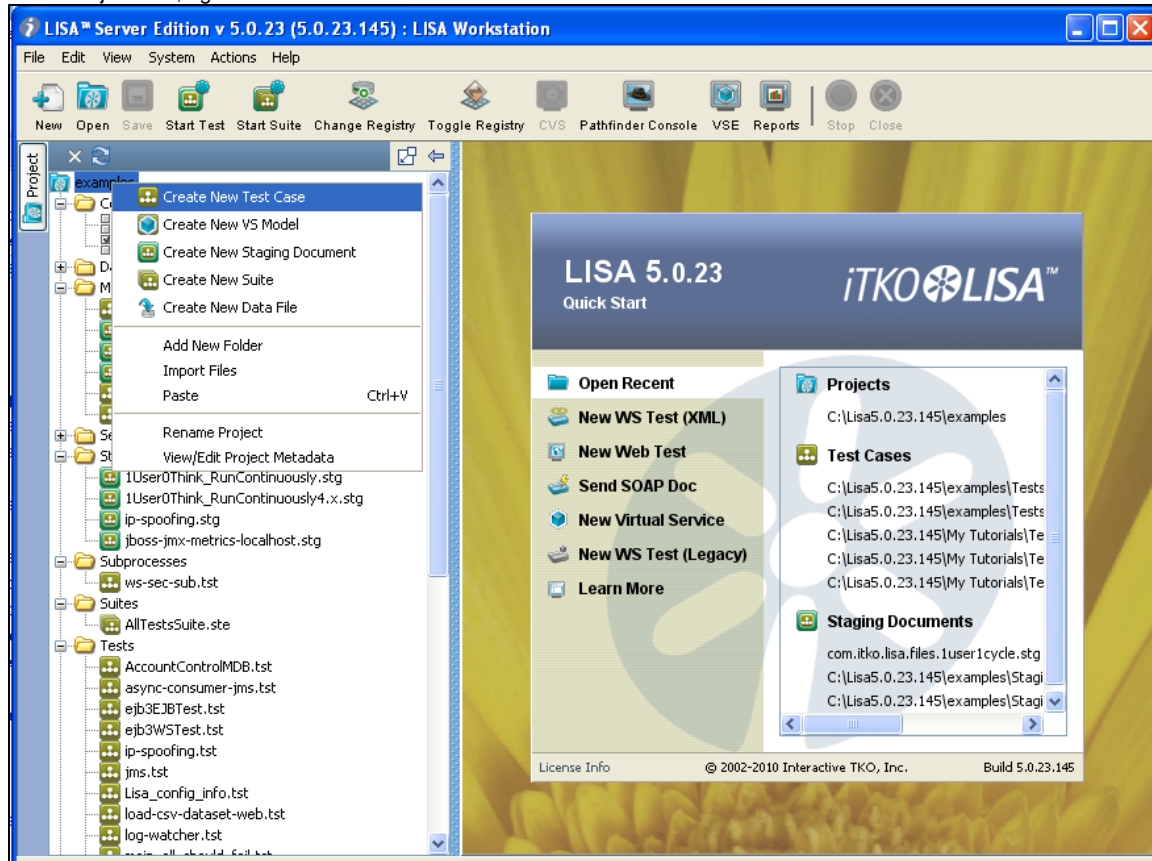
12.3.1 Creating a Test Case

You can create a test case, by opening a Project or creating a new Project for the same.

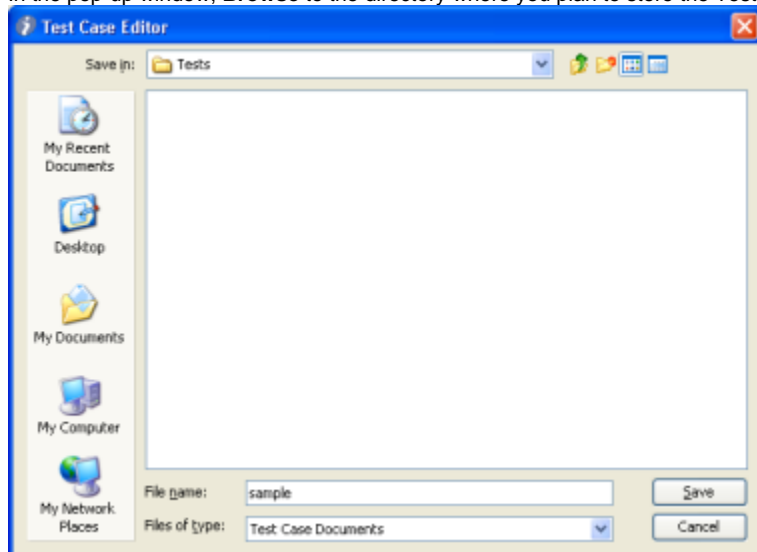
If you open the default Project "examples" within LISA, it opens with a tree structure which has folders for Configs, Data, Staging Doc, Suites, Tests etc.

We can create a new test case, under the Tests folder:

In the Project tree, right click the **Tests** folder and Click **Create New Test Case** as shown below:



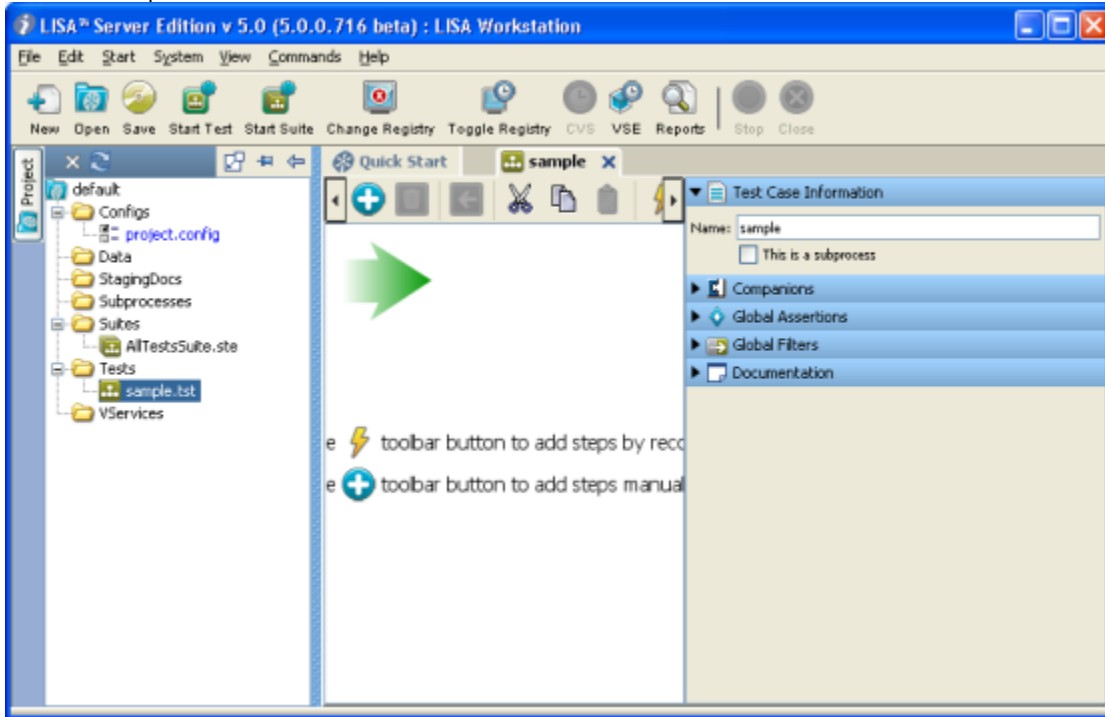
In the pop-up window, **Browse** to the directory where you plan to store the Test Case and enter the name of the new Test Case:



Or we can create a new Project called "**default**" and create a new Test case named "**Sample**" as shown below:

Note - You do not have to type the .tst suffix, LISA will add it.

Click **Save** to open the new Test Case in the Model Editor as shown below:



Refer Test Cases in [Test Cases in Model Editor](#) for more information.

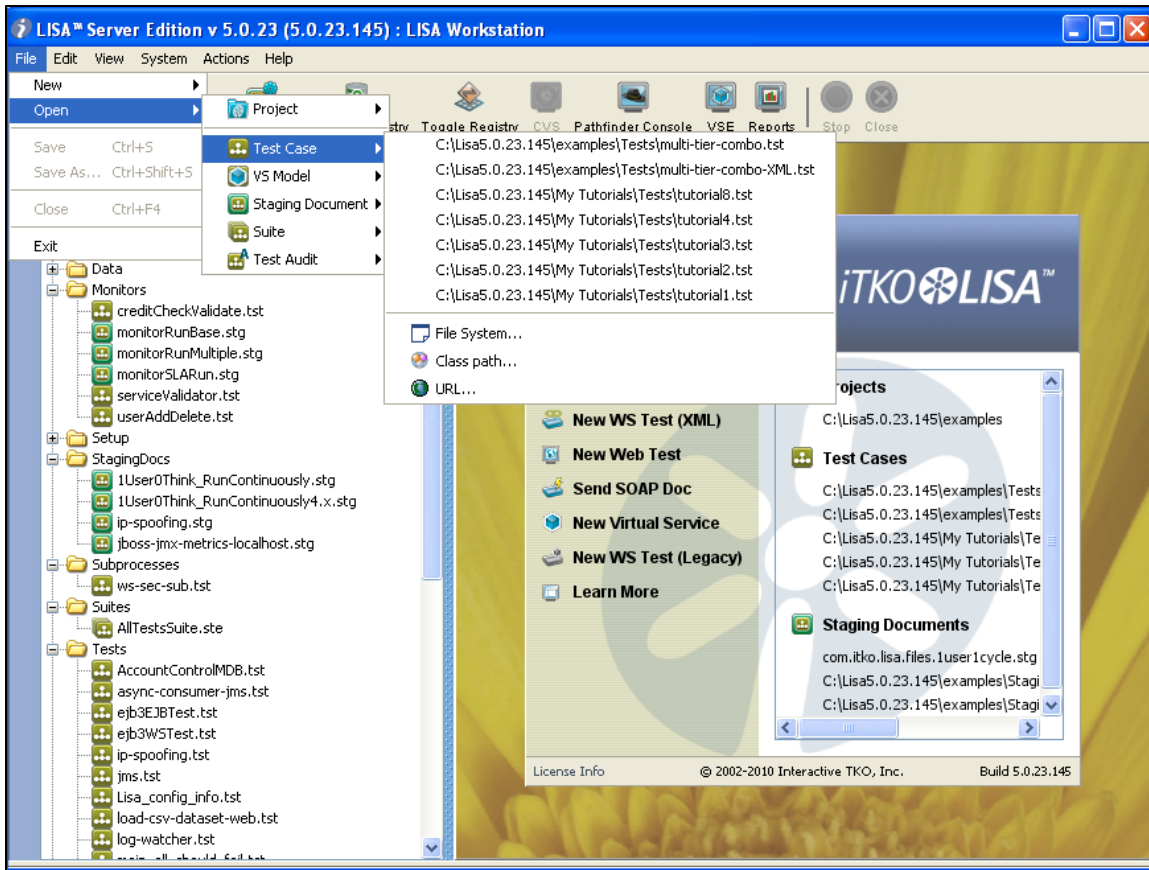
12.3.2 Opening a Test Case

12.3.2 Opening a Test Case

To open/view an existing Test Case,

From the main menu, select **File -> Open -> Test Case**

LISA displays the recently opened Test Cases for you. If the desired Test Case is in the list, select it. If it is not, then **browse** to it by clicking File System, Class path or URL.



Select the file you wish to open, and click **Open**.

The selected Test Case will be opened, and displayed in the Model Editor. You are now ready to add new elements, or modify the existing elements in the Test Case.

12.3.3 Saving a Test Case

12.3.3 Saving a Test Case

To save a new Test Case,



- Click **Save** icon on the toolbar.
- Or from the main menu, select **File -> Save Sample** (LISA displays the original Test Case name for you)

To save a Test Case with a different name or location,

From the main menu, select **File -> Save As...**

In the pop-up window, browse to the directory where you plan to store the Test Case, type in the name of the Test Case (you do not have to type the .tst suffix, LISA will add it), and click **Save**.

When you save a Test Case, LISA also saves the results of each step in the Test Case to a Response document, with a suffix of ".rsp" in the same directory.

For more information, refer to Response(.rsp) Documents.

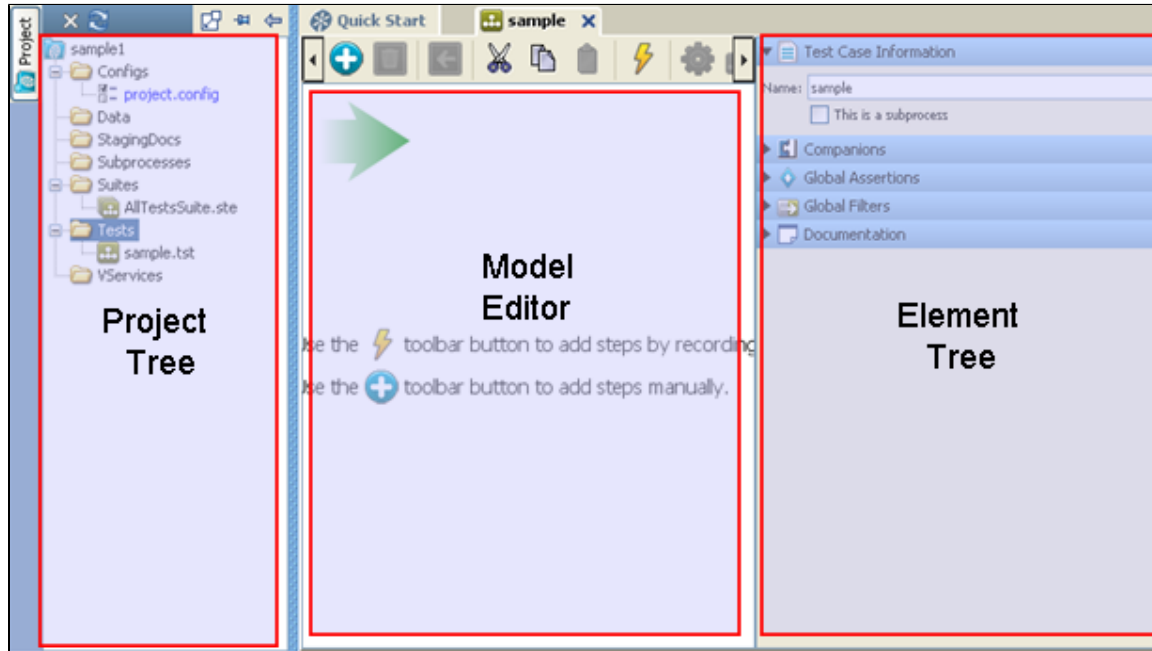
12.3.4 Test Cases in Model Editor

12.3.4 Test Cases in Model Editor

The Model Editor is LISA's graphical user interface which is used to create and manage Test Cases.

When you open an already existing Test Case, the view in the Model Editor provides a graphical view of a Test Case.

The **LISA Workstation** screen is as shown below:



For more information, refer to:

LISA Project - A place where you create a project and under which you will create, view, edit test cases or other LISA documents.

Model Editor - A place where you can add, edit, delete the test steps created for a test case.

Element Tree - A place where you can apply Filters, Assertions, Data Sets, Companions to a test case or test step.

Adding Test Steps to a Test Case in the Model Editor

Steps can be added in several ways:

- From the main menu, click **Commands > Create a New Step**
- From the Test Case toolbar, Click the **Add Step**  icon
- Select a step and right click anywhere in the Model Editor to **Add a Step**.
- In the workflow, right-click a step and click **Add Step After**. This adds a new step right after/below that step.
- Recorders and Test Generators can generate workflows comprising of several steps.

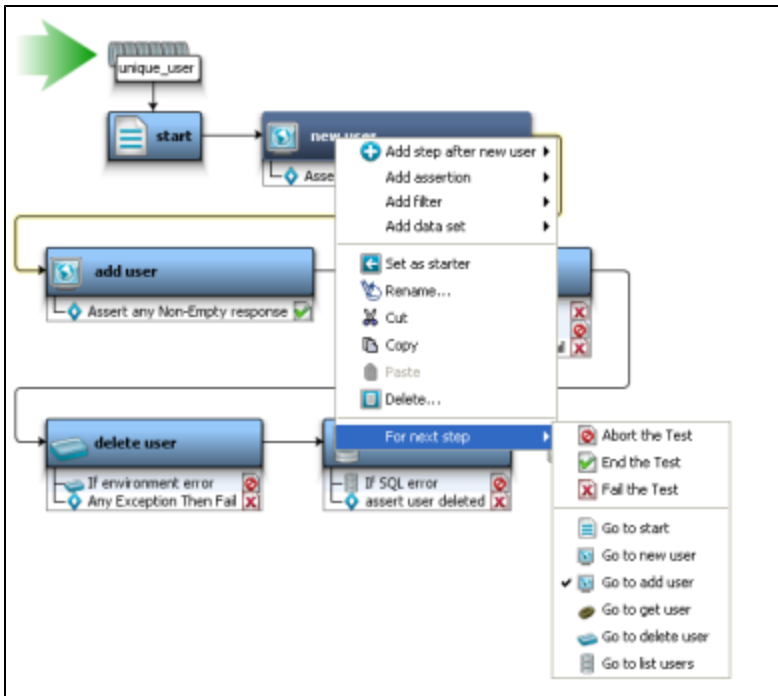
When a step is inserted into an existing workflow, and workflow is linear, the steps on either side of the inserted step will be updated automatically, keeping the workflow linear.

If the workflow contains branches or loops, the next step will not be set automatically.

Configuring the Next Step in the Test Case

For a particular step, you can configure the **Next step** within the Model Editor.

- Select the desired Test step and right click to open a menu. Here we have clicked on the New User step.
- Click "For Next Step" menu and select the desired next step as shown below:

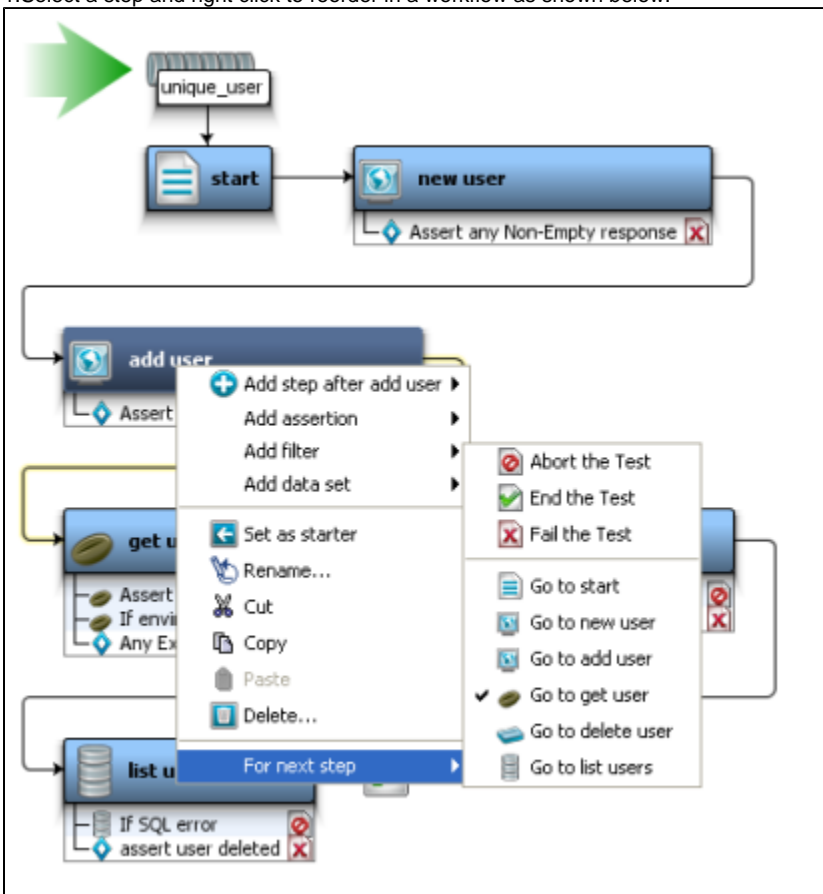


The steps are explained below.

Reordering Test Steps Within a Test Case

You can reorder the steps within the Model Editor.

1. Select a step and right click to reorder in a workflow as shown below:



For example as shown above: Select the **add user** step and right click to open a new menu as shown.

Click **For Next Step**, and select the step you need to set as the next step.

2. You can also drag and drop in the steps within the Model Editor

All the steps will be updated automatically if the workflow is **linear**.

When the workflow is **non-linear**, which contains branches and loops, the next steps will not be updated. In non-linear workflows the next step can be updated in the "**Step Information**" tab of that step.

12.3.5 Branching & Looping in Test Case

12.3.5 Branching & Looping in a Test Case

Branching in a Test Case

Branching in a Test Case is done **via Assertions**.

Any number of Assertions can be applied to a test step. Every Assertion has a condition which evaluates to true or false. The first Assertion for which the Assertion condition is satisfied gets fired, and that changes the path of the workflow. In many steps, a default error condition is automatically created as an Assertion to failure. When creating Assertions, it is best to be consistent, and always branch positive, or always negative.

Assertions are described in detail in [Adding Filters](#).

Looping in a Test Case

Loops are created when a test step down the flow from a given test step again sends control to the test step which started the flow as in a loop. What is important is the ability to come out of the loops. As you may imagine, conditionally breaking out of a loop (equivalent to a while loop in programming) is achieved by setting **Assertions** on a test step participating in the loop.

The other kind of loops which execute a given number of times or till a particular data is exhausted (equivalent to for or foreach loop in programming), are achieved with **Data Sets**. A Data Set is used to assign value(s) to one or more properties a finite number of times. The next step that needs to be executed once the Data Set is exhausted is specified with the Data Set definition, and this can be used to break the loop.

For example; if a Data Set contains twenty rows of users that need to be logged into a system, a loop can be created to run the login step for each row in the Data Set. Alternatively, a numeric counting Data Set can be used to cause a specific step to execute a fixed number of times.

A single step can call itself, and loop over a Data Set. Several steps can run in a loop, and use the data from a Data Set. This is accomplished when the final step in the group of steps points to the first step in the group.

Data sets are described in detail in [Using Data Sets](#).

12.3.6 Response (.rsp) Documents

12.3.6 Response (.rsp) Documents

When you record from a website or interact with a Server, **LISA saves the responses** into a Response Document so that it can refer to the information later.

LISA creates and maintains response documents automatically, with no effort on your part, and saves them as files with the same name as the Test Case file with an **.rsp** extension. Like the Test Case files, the response documents are also XML files.

A response document maintains the HTTP response for each of the HTTP-based steps in a Test Case, the response information from web service calls, as well as JDBC results from a database query.

For example, if you have executed the HTTP-based steps via the Interactive Test Runner (ITR), the saved response document contains the entire DOM tree for the result of each HTTP-based step in the test. You can use this response information to validate data, create simple Filters or create simple Assertions.

For more information on using HTTP responses to create Filters, see [9. Adding Filters](#).

For more information on using the HTTP response, to create Assertions, see [10. Adding Assertions](#).

Not all steps have results that are amenable to storage in a response document.

If you copy a Test Case file to another location, you should consider copying the associated response document as well, so that you do not lose the saved responses.

Response documents are optional, in that they are not necessary for LISA to run tests. They exist solely to give you the ability to view results, view DOM Tree, view JDBC table, etc. When you use the replay function in LISA, information is read from the response document.

LISA Version and Build number

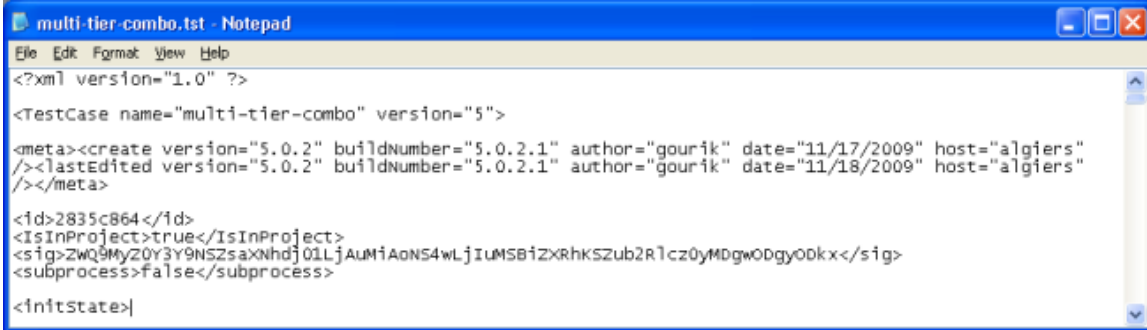
In LISA, you can view the LISA version and build number by clicking the Help menu.

You can also view the version and build number in all the documents created by LISA.

In LISA, you can view all the documents as XML document.

For example - A test case, Staging document, Test Suite, VS Model and/or Audit document – all these can be view as XML documents.

The documents will contain all the information about the document like – LISA Version no, Build no, Author, Created and Edited Date and host name as follows:

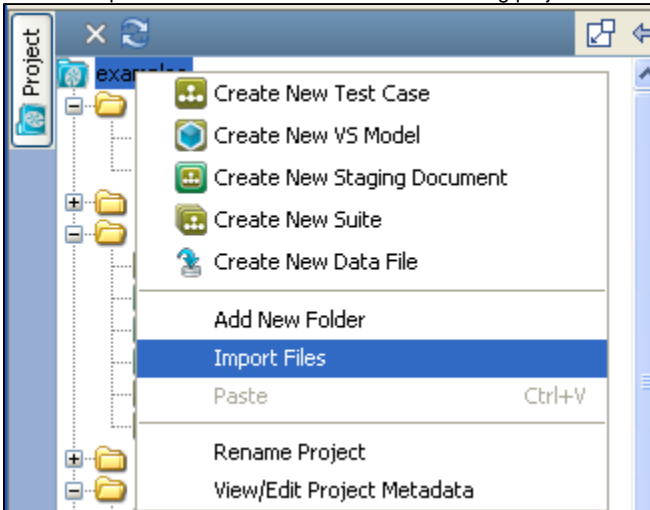


The XML document also contains all the information regarding the steps added in the test case or suite document.

12.3.7 Importing Test cases

12.3.7 Importing Test Cases

You can import old test cases into the current working project directory.



To import test cases,

- Right click the "Tests" folder in the Project tree
- Select the "Import Files" option.
- Choose the Files to be imported into that folder.
- Click OK.

The process of Importing starts and copies all the files (including the files within sub-directories) to the selected folder. It would also merge all the configs from within the test cases & VS models into the project configs.

For Importing older versioned files -

- Create a Project from existing LISA directory.
- Choose a directory that has test cases with older version.
(version less than 5.0 or anything created prior to LISA 4.7)

For example - Copy the lisa-examples/ejb3 directory from 4.6 root to a temp location, and make all files read/writable, and used this dir.

- Click "Create".

This will create the project with all old versioned files converted to new 5.0 version.

NOTE: If a Project with the same name exists, it will ask for another name for the new project.

Along with other project creation messages, you'll also see the auto-convert message for all the test cases and VS models that were transformed to meet version 5 requirements.

While completion, you are able to see the following messages:

- Migration of the project
- Configuration change details
- Test case change details

All the imported files are highlighted.

12.4 Building Sub Processes

12.4 Building Sub Processes

A **Sub Process** is a LISA Test Case that is designed to be called from another Test Case.

This "sub process" Test Case is called by another Test Case, rather than run as a stand-alone Test Case within LISA.

Sub processes can be used as modules in other Test Cases, hence increasing their re-usability. You can build a library of sub processes that can be shared across many Test Cases.

In computer programming a sub process would be identifiable as a function or a subroutine.

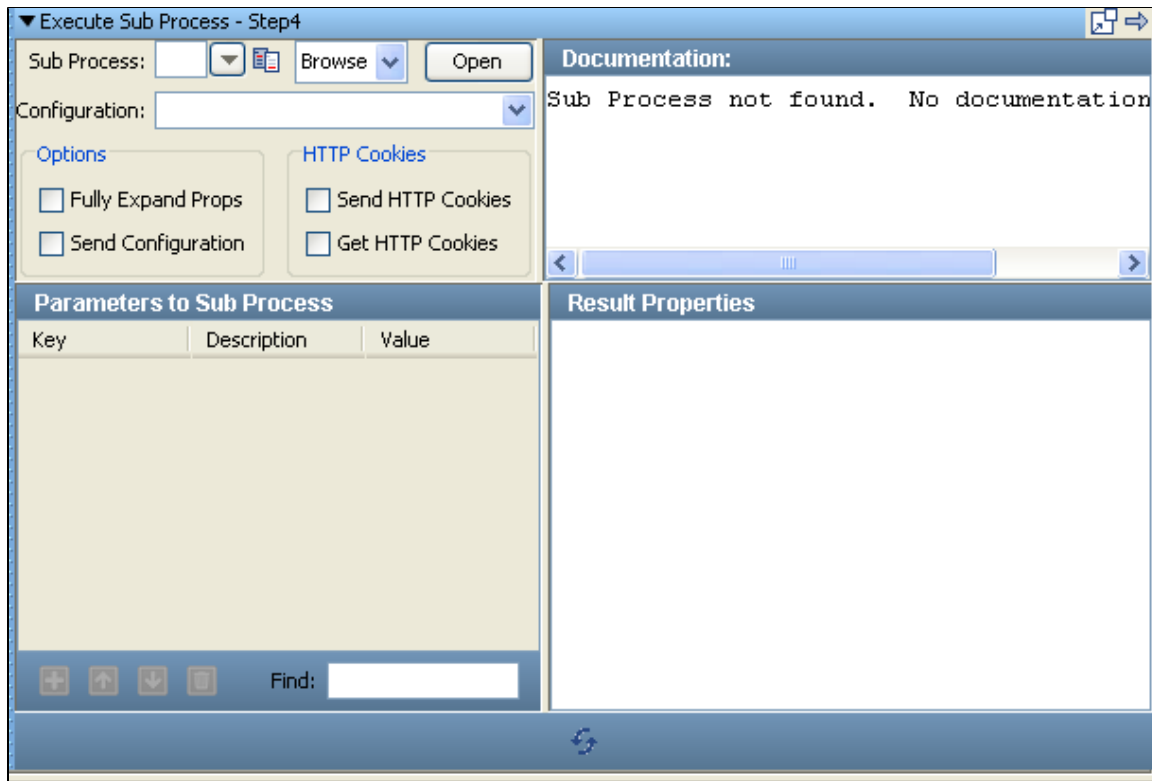
A Test Case must be self-contained, that is, the value for all the properties used in the Test Case must come from within the Test Case. A sub process expects some property values to be provided by the test step that invokes it (input properties), and when sub process completes, it makes property values available to the calling step, (return properties).

You can create the steps in the sub process in the same way you would for a regular Test Case, with the following differences:

- Mark the Test Case as a sub process in the **Test Case Information** tab of the Test Case (as explained in next section).
- Do not add Data Sets like you would in a Test Case. Instead the Data Set should be part of the calling case, and the current values passed to the sub process when it is invoked. The exception here is when a Data Set is an integral part of the sub process logic itself. In that scenario, a local Data Set should be utilized.
- Do not use a configuration file or a Companion inside the sub process to initialize any parameters that you expect to be passed from the calling step. For testing purposes, we add these values elsewhere, as shown below. When the sub process is called, the calling step will pass these values.

You can build sub processes from scratch, or you can convert an existing Test Case into sub process very easily.

LISA provides a test step **Execute Sub Process** that makes it easy to call a sub process test case.



The Execute Sub Process step is described further in Test Steps in the *LISA Reference Guide*.

The following topics are available in this Chapter.

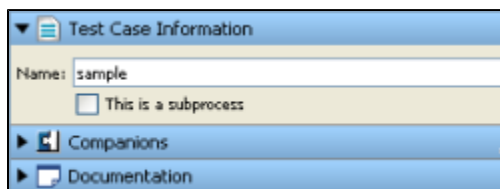
- 12.4.1 Creating a Sub Process Test Case**
- 12.4.2 Converting an Existing Test Case into Sub Process**
- 12.4.3 Sub Process Example**

12.4.1 Creating a Sub Process Test Case

12.4.1 Creating a Sub Process Test Case

To create a **sub process test case**:

- Create a new Test Case or open an existing one.
- Open the **Test Case Information** tab of a Test Case. To open the **Test Case Information** tab, click anywhere in the empty space in the Model Editor (no steps should be selected). The **Test Case Information** tab will open in the right panel as shown below:



- Check "This is a Sub Process" box to make this Test Case a sub process. By default, a Test Case is not designated to be a sub process and hence **This is a Sub Process** box is not checked.
- New tabs for **Subprocess Input** and **Subprocess Output Parameters** are added as shown below:

Test Case Information

Name: sample

☒ This is a subprocess

Subprocess Input Parameters

Subprocess Output Properties

Companions

Documentation

- In the **Documentation** tab, provide detailed documentation of the sub process. This text is visible in any test step that calls the sub process, so it is the information that a user of the sub process will read when he is about to use it in his Test Case.

When you have finished adding the sub process steps, you configure the input and output properties for the sub process as follows:

Subprocess Input Parameters

Click the **Subprocess Input Parameters** tab:

Test Case Information

Name: sample

☒ This is a subprocess

Subprocess Input Parameters

Key	Description	Default Value
Input Parameters		
Prospective Input Parameters		
+ □ X		

Subprocess Output Properties

Key	Description
Key	Set in default configuration
WSPORT	Set in default configuration
WSSERVER	Set in default configuration
lisa.Step1.rsp	Set 1st in Step1
lisa.Step1.rsp.time	Set 1st in Step1
+ X	

Companions

Documentation

In the **Subprocess Input Parameter** tab, you can define the Input parameters and the Prospective Input parameters as shown below:

Subprocess Input Parameters

Key	Description	Default Value
Input Parameters		
Key	<< description >>	<< default value >>
Key1	<< description >>	<< default value >>
Key2	<< description >>	<< default value >>
Prospective Input Parameters		
+ □ X		

Subprocess Input Parameters: A list of the parameters needed by the sub process. LISA will add these, but LISA cannot always identify all the properties in use in the sub process, so you may have to add some yourself.

Use the **Add** icon at the bottom of this panel to add a new property. Enter values for **Key** (property name), **Description**, and **Default Value**. LISA uses the default value that is provided here when you run the sub process in the Interactive Test Run facility (ITR). This allows you to test the sub process as though it was a regular Test Case. These default values are ignored when the sub process is invoked by another test step. Remove properties using the **Delete** icon.

Prospective Input Parameters (May be required parameters): If LISA finds a property in the sub process that may be an input property, but there is some doubt, the property is listed here. If on inspection it is deemed to be a valid input property, use the **Add** icon at the bottom of this panel to promote this property to the **Subprocess Input Parameters** list.

Subprocess Output Properties

Subprocess Output (Result) Properties: A list of all the properties set by the sub process. These properties will be made available to the calling Test Case. LISA will add this value, but LISA cannot always identify all the properties in use in the sub process, so you may have to add some yourself.

Subprocess Output Properties	
Key	Description
Key	Set in default configuration
WSPORT	Set in default configuration
WSSERVER	Set in default configuration
lisa.Step1.rsp	Set 1st in Step1
lisa.Step1.rsp.time	Set 1st in Step1
+ X	

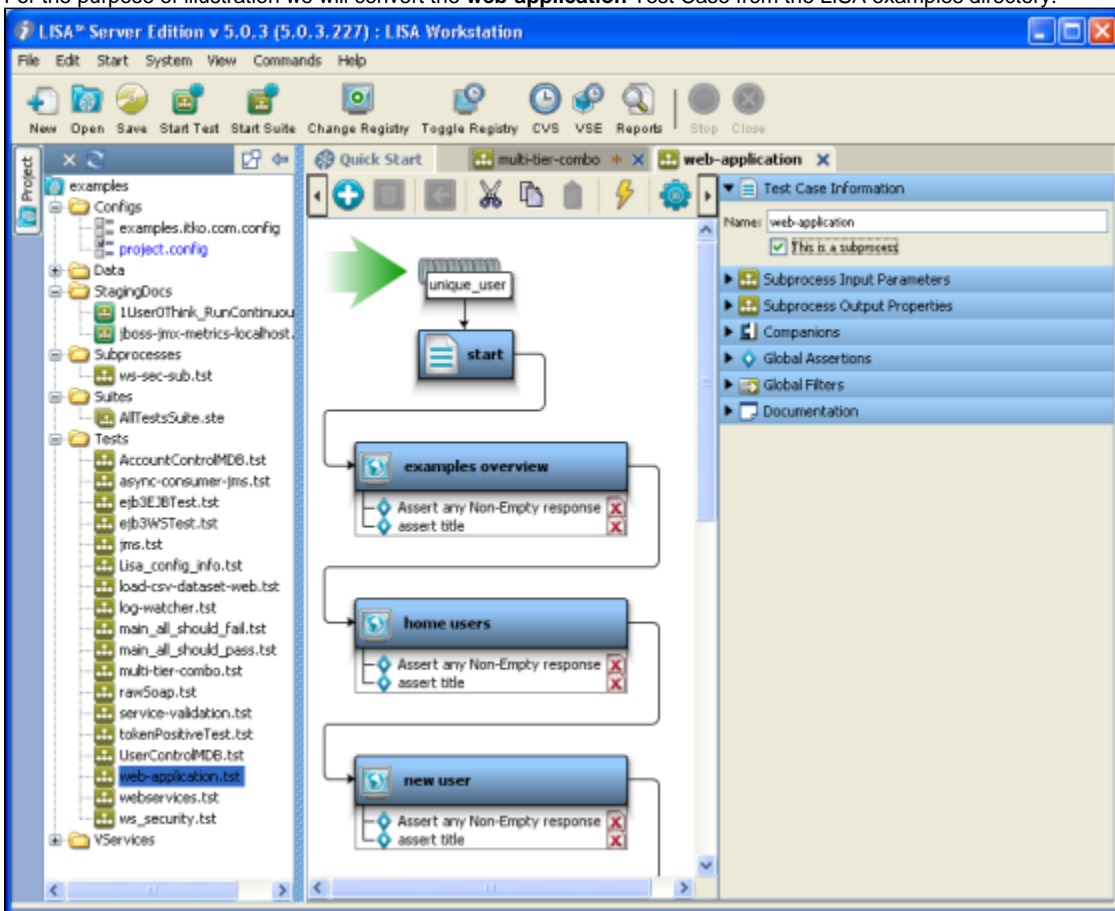
Use the **Add** icon at the bottom of this panel to add a new property. If there are properties here that you do not want to be made available to the calling step, remove them using the **Delete** icon.

Once the input and return properties have been checked, and default values have been given to all the input parameters, the sub process can be run in the ITR.

12.4.2 Converting an Existing Test Case into Sub Process

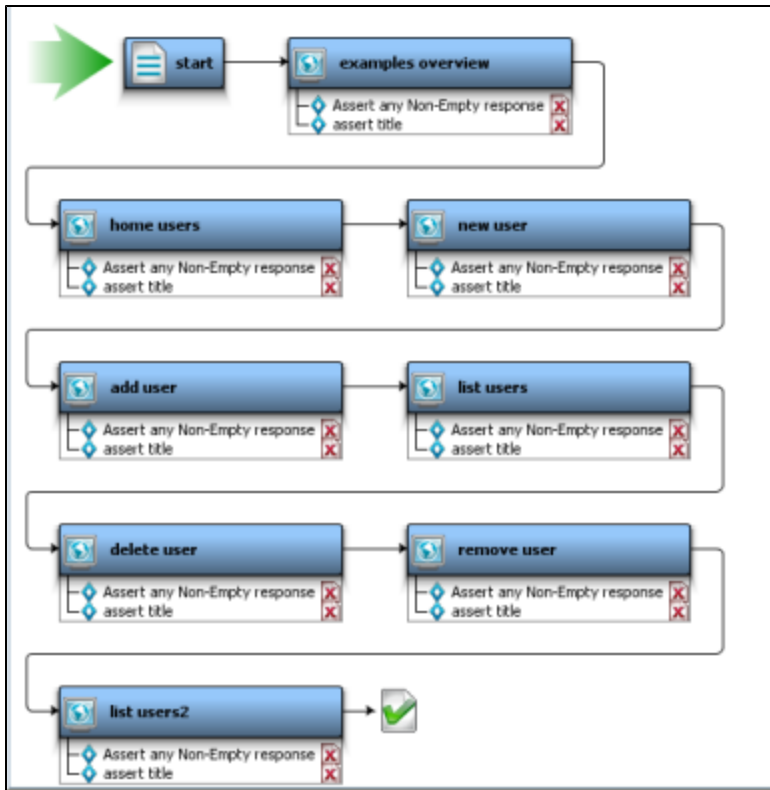
12.4.2 Converting an Existing Test Case into Sub Process

- To convert an existing Test Case into a sub process, open the Test Case and rename it appropriately.
- For the purpose of illustration we will convert the **web-application** Test Case from the LISA examples directory.



- Mark the Test Case as a sub process in the **Test Case Information** tab of the Test Case.
- Remove any Data Sets that provide the values of properties that are to be input properties of the new sub process. The exception here is when a Data Set is an integral part of the sub process logic itself.

In our example we removed the **unique_user** Data Set.



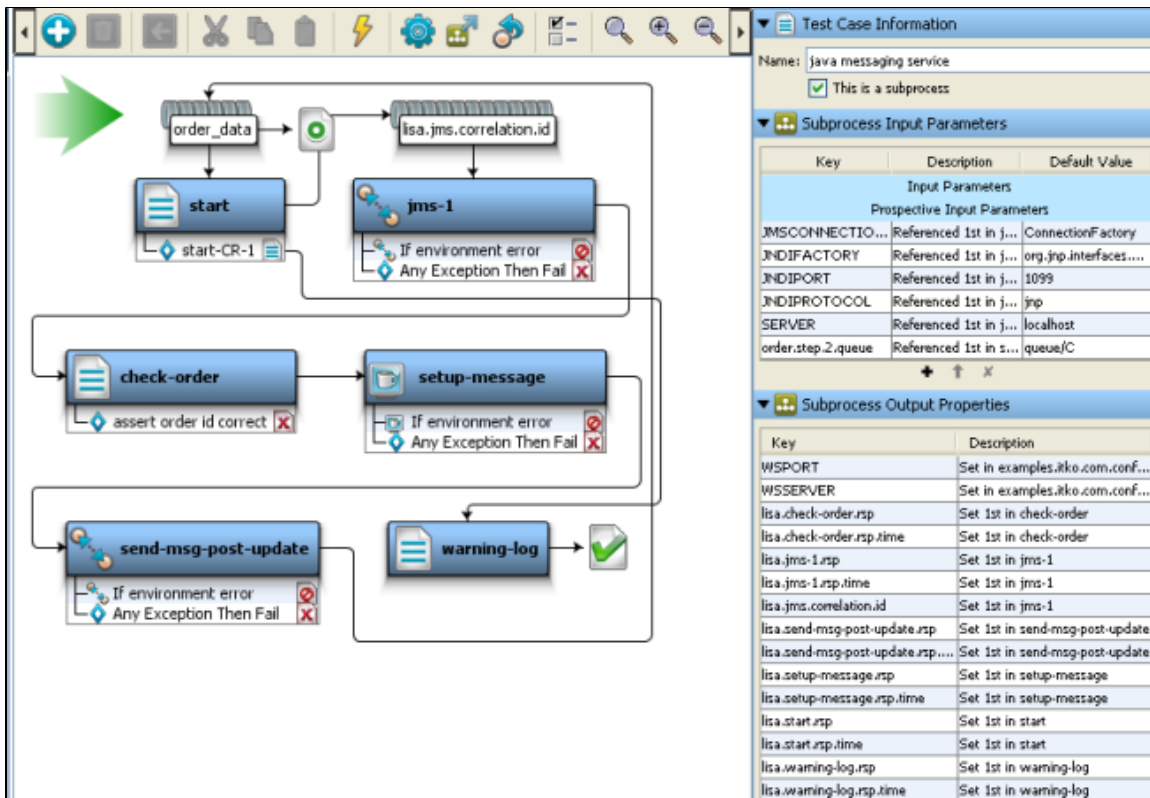
- Remove any properties from configuration files, or a Companion that initialize parameters that are to be input properties of the new sub process.
- Once the input and output properties have been checked, and default values have been given to all the input parameters, the sub process can be run in the ITR.

12.4.3 Sub Process Example

12.4.3 Sub Process Example

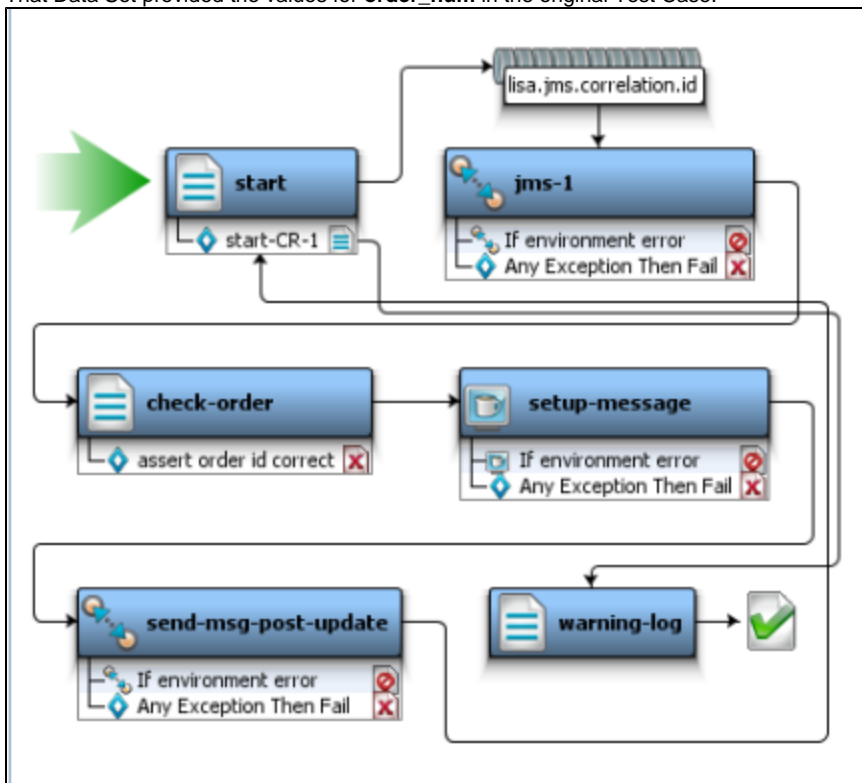
The example below shows a **sub process** that was derived from the **jms (Java Messaging Service)** Test Case in the LISA examples directory, and a Test Case that calls this sub process.

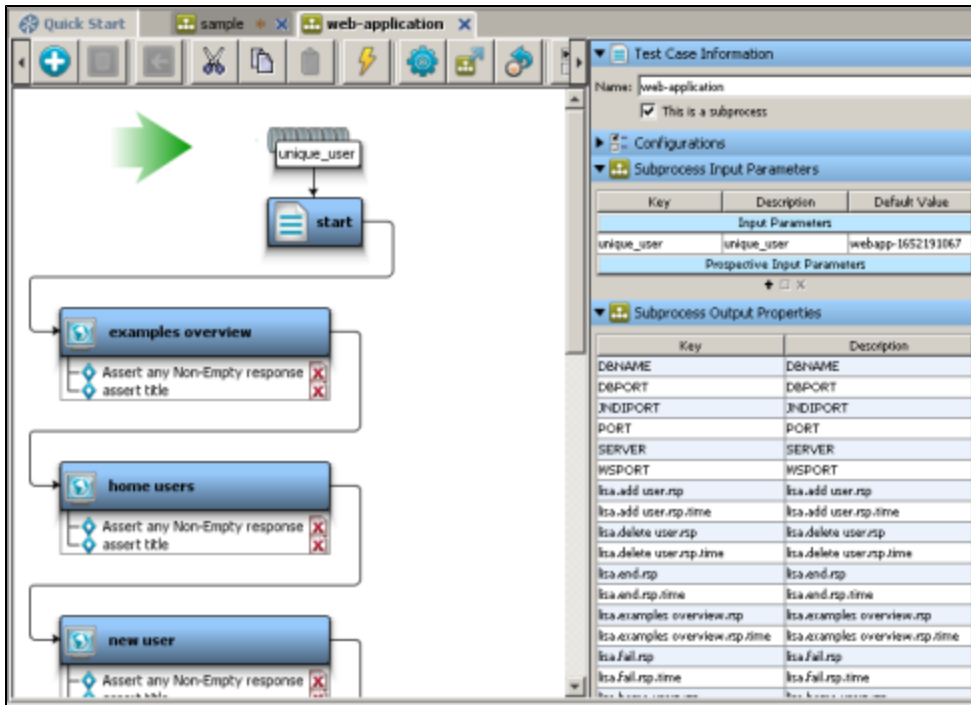
The sub process is shown in the figure below:



One Data Set, **order_data**, was removed from the original Test Case.

That Data Set provided the values for **order_num** in the original Test Case.

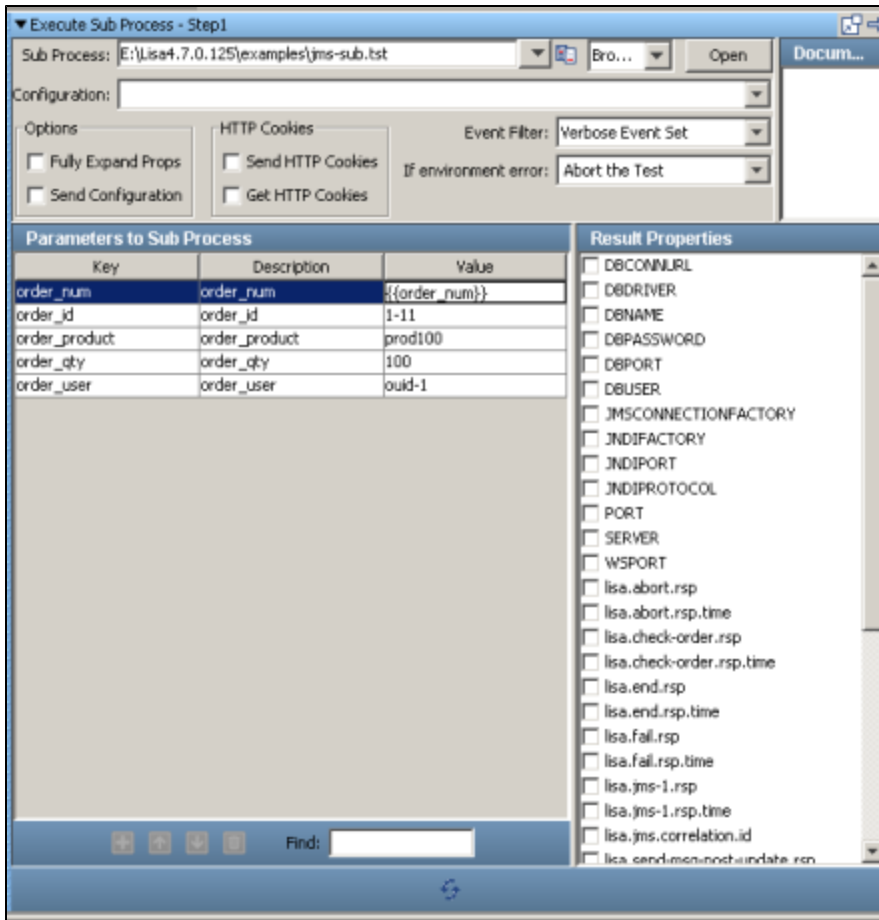




order_num is now an input property.

Test Case Information		
Name: Ex jms sub		
<input checked="" type="checkbox"/> This is a subprocess		
Subprocess Input Parameters		
Key	Description	Default Value
Input Parameters		
Key	<< description >>	<< default value >>
order_num	order_num	112227654
order_id	order_id	1-11
order_product	order_product	prod100
order_qty	order_qty	100
order_user	order_user	oid-1
Prospective Input Parameters		
+ - X		

The figure below shows a Test Case with one test step **Execute Subprocess**



ExecSub step is of type *Execute Sub Process*, and it invokes the sub process *JMS-sub* (which is shown above).

The input property matches, and the calling step has asked for the **lisa.Step3.rsp** property to be made available after the sub process has finished executing.

For details of the rest of the parameters in the **ExecSub** test step, see *Test Steps* in the [LISA Reference Guide](#).

13. Building Staging Documents

13. Building Staging Documents

A Staging Document is a very important document for LISA, as it contains the information about "how" to run the test.

The Staging document allows you to specify some important criteria's for running test cases.

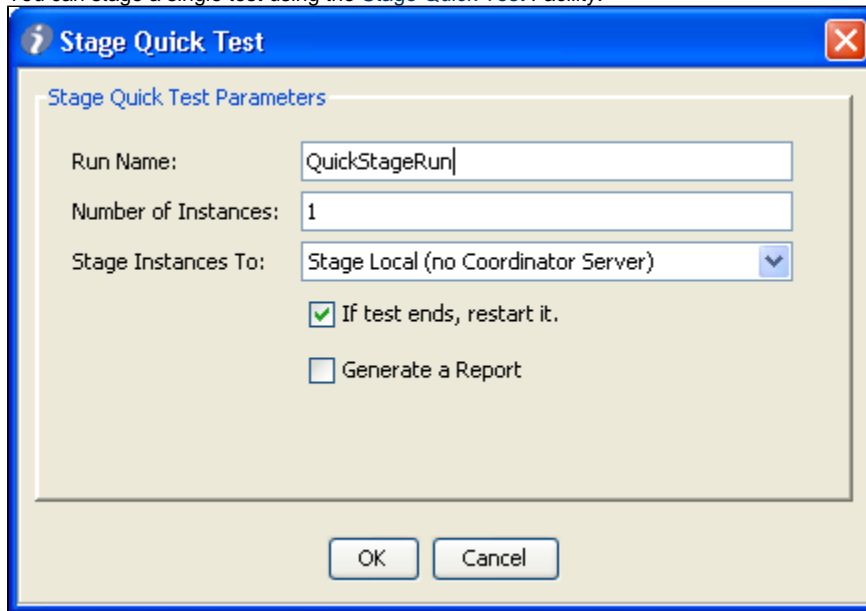
The Staging documents given by LISA can be seen in the Project pane under the Staging documents folder. Staging documents are XML documents saved with the suffix **".stg"**.

In the staging document we can specify the following:

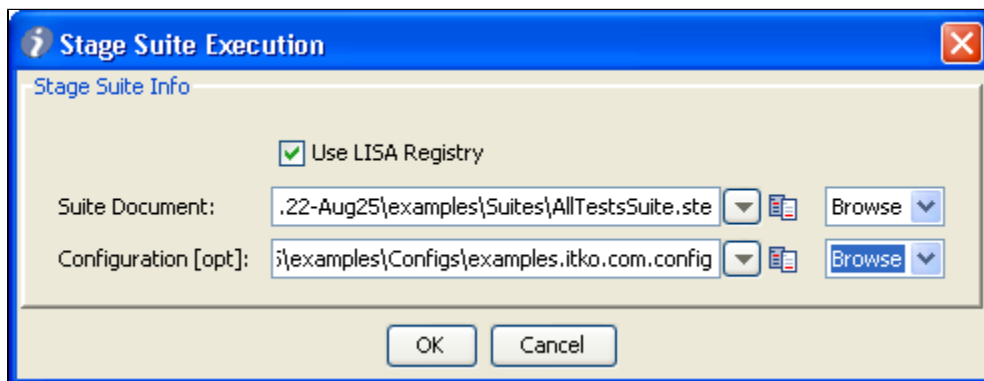
- The duration of the test (elapsed time or number of runs).
- The number of virtual users.
- Loading patterns for the virtual users.
- Distribution patterns of the virtual users on your test platform.
- Global adjustments to think times.
- Information to pace tests, such that a given number of tests complete in a specified time period.
- The Type and number of Reports requested.
- The Events required for inclusion in the reports.
- The Metric requested for performance reporting.
- The sampling information for the metrics.
- IP Spoofing support

The Staging Document editor allows you to specify the parameters required to use these features.

You can stage a single test using the [Stage Quick Test Facility](#).



Or you can stage a group of tests using the [Test Suite](#).



13.1 Creating a Staging Document
28.2 Staging Document Editor

13.1 Creating a Staging Document

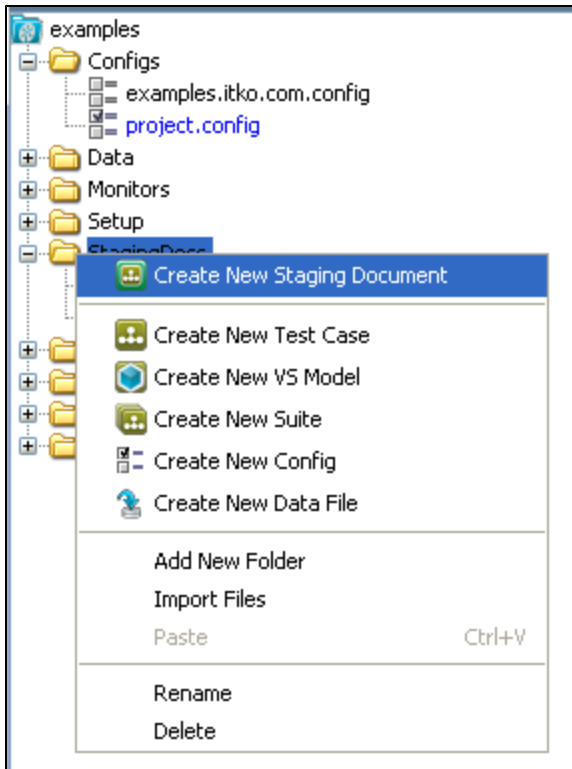
13.1 Creating a Staging Document

To create a new **Staging Document**,

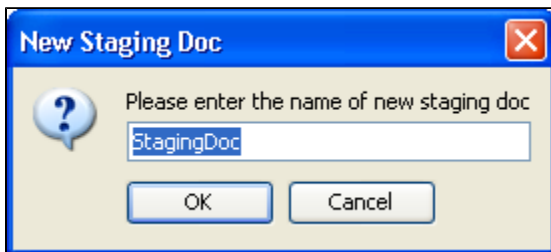
- Select **File->New->Staging Document** from the main menu.

This opens the Staging document Editor as explained in the next section.

- You can also right click the **StagingDoc** folder in the project tree as shown below.



This opens the following dialog, where in you can enter the name of the new staging document:



- Click **OK** to create the new staging document.

Opening a Staging Document

If you have an existing Staging Document, you can open it as

- Select **File->Open->Staging Document**

Note - By default LISA provides some Staging documents in the "Defaults" folder at %LISA_HOME%. Project specific Staging documents are listed in the Project specific directory.

You can also use these documents as the starting point of your new staging document and rename it to save it under a different name.

Staging Notes

If a Test Case contains a global Data Set on the first step and the Data Set is set to end the test when the data set is drained, then - all instances of the Test will end for a staged run, overriding any other staging parameters such as steady state time.

Local Data Sets will not end the staged run in this fashion nor will Data Sets on steps other than the first test.

28.2 Staging Document Editor

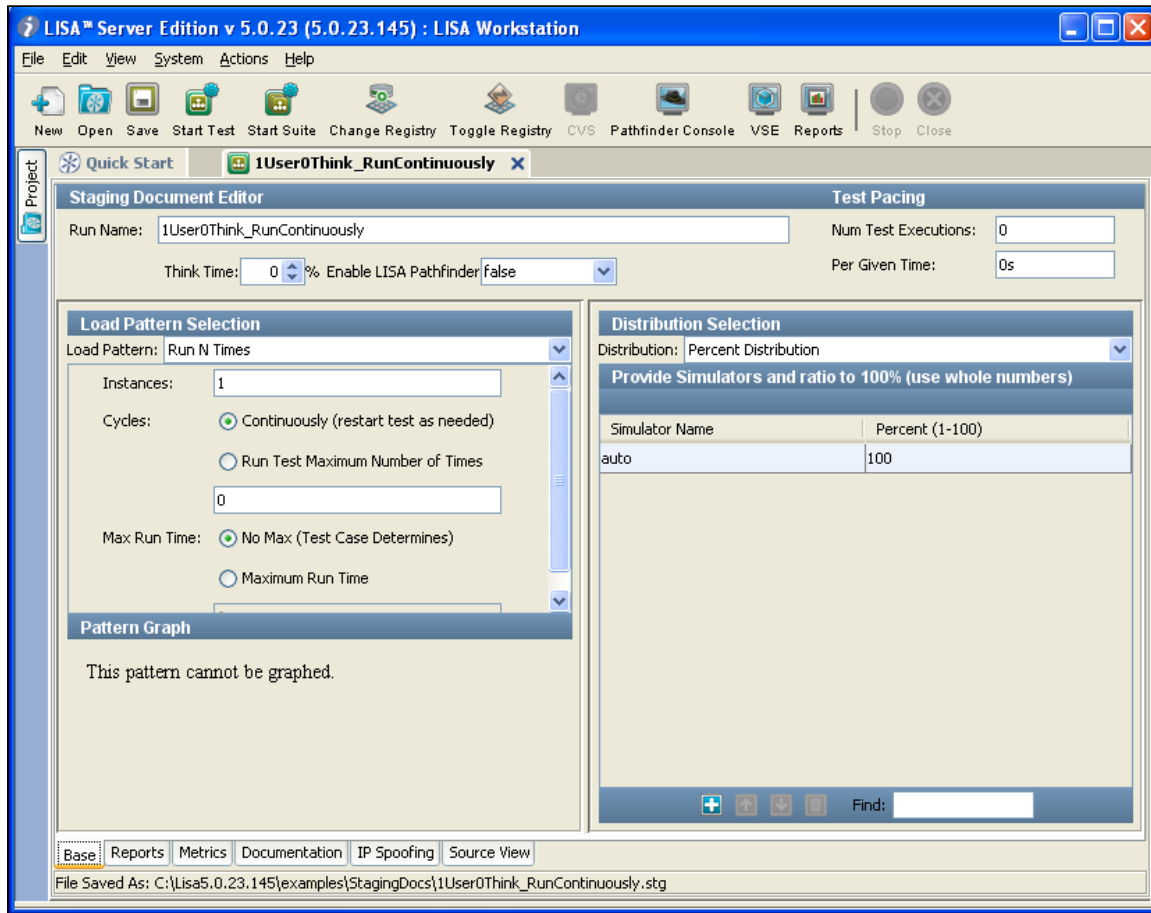
Staging Document Editor

The Staging document editor is a place where you can specify all the criteria for running test cases.

The **Staging Document editor** is shown below:

For the purpose of illustration, we have opened "1User0Think_RunContinuously.stg" document which is in the LISA_HOME/examples/StagingDocs directory.

By default the Staging document editor opens in the Base tab.



At the bottom of the Staging doc editor, there are five tabs which are explained in detail in the next section:

- **Base** : The base parameters of your test.
- **Reports** : Where you select and add your Reports and select the types of Events you want to capture.
- **Metrics** : Where you select your Metrics, and specify your sampling intervals.
- **Documentation** : A text area where you can document your staging document. A very good thing to do!
- **IP Spoofing** : IP spoofing allows multiple IP addresses on a network interface to be utilized when making network requests.
- **Source View** : The Staging document is saved as an XML file with a suffix of .stg. This tab displays the XML equivalent of the current staging document.

The following additional topics are available.

[Base Tab](#)
[Reports Tab](#)
[Metrics Tab](#)
[Documentation Tab](#)

[28.2.4 Documentation Tab](#)
[IP Spoofing](#)

[Source View Tab](#)

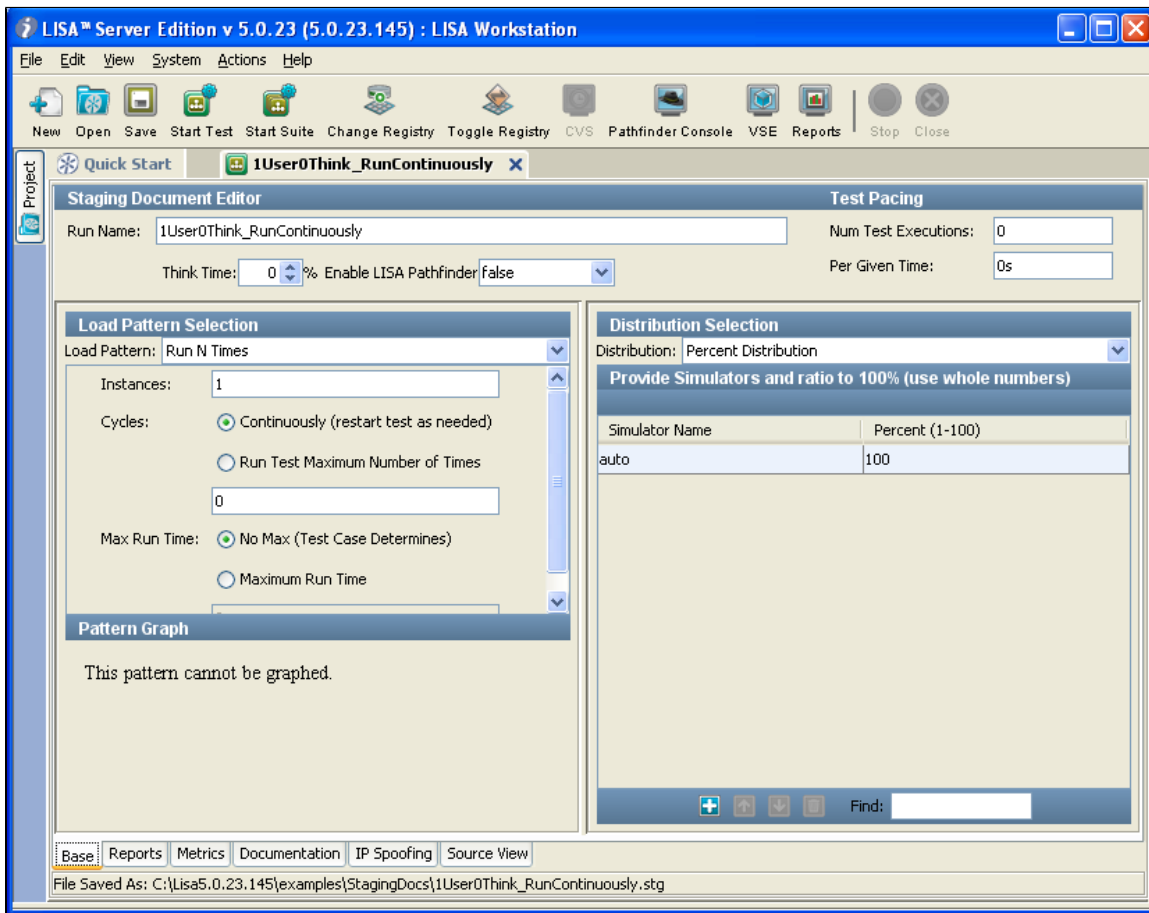
28.2.1 Base Tab

28.2.1 Base Tab

The Base tab describes the basic parameters of a Test Case.

It specifies:

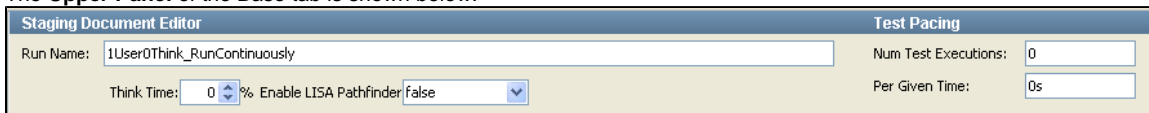
- The duration of the test (elapsed time or number of runs)
- The number of virtual users.
- Loading patterns for the virtual users.
- Distribution patterns of the virtual users on your test platform.
- Global adjustments to think times.
- Information to pace tests, such that a given number of tests complete in a specified time period.



The Base tab of the Staging Document is divided into 3 panels.

Upper Panel

The **Upper Panel** of the Base tab is shown below:



It has the following parameters:

- **Run Name:** The name given to this staging document
- **Think time:** The Think Time in percentage, for all the test steps that are included in the Test Case. Each test step has the ability to declare a think time, in its Base Step Info. Here, a global change can be applied to these think times, as a percentage of their values. Think times can be eliminated by setting the percentage to 0%, halved by setting it to 50%, or doubled by setting it to 200%. For example, a step's think time range of 4-6 seconds would be reduced to 2-3 seconds if the staging document Think Time percentage was set to 50%.
- **Enable LISA Pathfinder: True/False:** You can choose to enable/disable the LISA Pathfinder utility.
- **Num Test Executions:** The number of test executions you want to complete in a given time (Per Given Time).
- **Per Given Time:** The time period (wall-clock), in which you want the tests to run. You can specify h for hours, m for minutes, s for seconds. For example, we can specify that we want LISA to adjust the time such that 2500 Tests complete in 8 minutes.

Notes:

- 1) LISA does not change think times to achieve the required pace.
- 2) If the test pace cannot be achieved because it is too high a pace, LISA will run without any pause between tests. LISA will report in the log that

the test is running at a lower pace than requested.
3) If the test pace cannot be achieved because too few virtual users have been specified, LISA will not add more users. To estimate the number of virtual users needed, see the Optimizer Utility.

Load Pattern Selection Panel

The **Load Pattern Selection** panel allows you to set the duration of the test, the number of virtual users, (instances), and the load pattern for those virtual users (if you have more than one virtual user).

Load Pattern Selection

Load Pattern: Run N Times

Instances:

1

Cycles:

☐ Continuously (restart test as needed)

☒ Run Test Maximum Number of Times

1

Max Run Time:

☒ No Max (Test Case Determines)

☐ Maximum Run Time

0s

Pattern Graph

This pattern cannot be graphed.

By default the LOAD Pattern is "Run N Times". You can select the other LOAD Patterns by clicking on the drop down.

There are **five Load Pattern** choices:

Load Pattern Selection

Load Pattern: Run N Times

Immediately Ramp

Manual Load Pattern

Run N Times

Stair Steps

Weighted Average Pattern

All of these patterns are explained in the [Load Patterns for Virtual Users](#) section.

Distribution Selection Panel

The **Distribution Selection** Panel on the right is as shown below:

Distribution Selection

Distribution: Percent Distribution

Provide Simulators and ratio to 100% (use whole numbers)

Simulator Name	Percent (1-100)
auto	100

+ ↑ ↓ ✖ Find:

It allows you to distribute virtual users (instances) over your running Simulators.

You can Add, Delete and move the simulators within the list by clicking on the icons in the toolbar.

If you are using LISA Workstation you will not have use for this panel, since all your virtual users will be running locally (i.e. using a Simulator that is part of Workstation).

However for LISA Server, with several Simulator Servers active, this panel allows you to specify how you want to distribute your virtual users over these Simulators.

There are three Distribution choices:

Distribution Selection

Distribution: Balanced Based on Instance Capacity
Percent Distribution
Round Robin Distribution

1. Percent Distribution
2. Round Robin Distribution
3. Balanced Based on Instance Capacity

By default the choice of distribution is "Percent Distribution".

All these options are discussed in the [Distribution Selection Options](#) .

28.2.1.1 Base Tab - Load Patterns

28.2.1.1 Load Patterns for Virtual Users

The Base tab describes the basic parameters of a Test Case.

There are **five Load Pattern** choices:

Load Pattern Selection

Load Pattern: Run N Times

- Immediately Ramp
- Manual Load Pattern
- Run N Times
- Stair Steps
- Weighted Average Pattern

- Run N Times
- Immediately Ramp
- Stair Steps
- Weighted Average Pattern
- Manual Load Pattern

All of these patterns are explained below:

Run N Times

The Run N Times pattern is applicable when you are running only one or just a few virtual users (instances), but you want to be able to specify **how many times** the test will run. You are not concerned with any loading pattern. This pattern will start all virtual users at the same time.

▪

To configure this pattern enter the following parameters:

- **Instances:** The number of virtual users
- **Cycles:** Choose between running Continuously until the time in the Max Run Time setting has been reached, or running This Many Times. In the latter case, specify the Number of Times to run.
- **Max Run time:** Choose between No Max, (the time is determined by the Test Case), or, Maximum Run time. In the latter case, specify the Maximum Run Time, (default is seconds). This will overwrite any value entered for Cycles. You can specify h for hours, m for minutes, s (or no letter) for seconds (default).

In the figure above, there is a single user running the Test Case one time to conclusion.

[Back to List of Load Patterns\](#)

Immediately Ramp

The Immediately Ramp pattern is applicable when you are running just a few virtual users (instances), but you want to be able to specify **how long** to run the test. You are not concerned with any loading pattern. This pattern will start all virtual users at the same time.

▪

Load Pattern Selection

Load Pattern: Immediately Ramp

Instances: 10

Max Run Time: ☐ No Max (Test Case Determines) ☒ Maximum Run Time

1h

Pattern Graph

The graph shows a constant load of 10 instances over 3500 seconds. The Y-axis represents the number of instances (0 to 10) and the X-axis represents time in seconds (0 to 3500).

Base Reports Metrics Documentation Source View

To configure this pattern enter the following parameters:

- **Instances:** the number of virtual users
- **Max Run Time:** Choose between No Max, (the time is determined by the Test Case), or, Maximum Run Time. In the latter case, specify the Maximum Run Time, (default is seconds). This will overwrite any value entered for Cycles. You can specify h for hours, m for minutes, s (or no letter) for seconds (default).

In the example above, there are 10 virtual users running the test concurrently for 60 min.

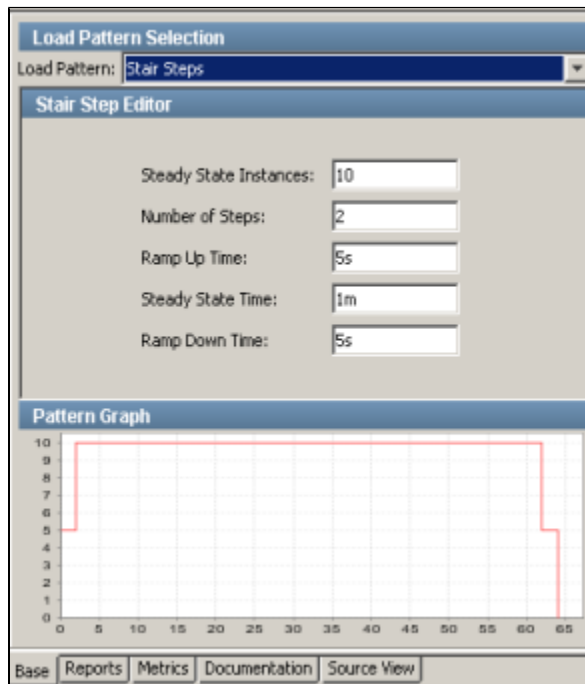
The Pattern Graph shows graphically how the test will be run.

[Back to List of Load Patterns\](#)

Stair Steps

The Stair Step pattern introduces virtual users (instances), to the system in well defined steps rather than all at once. You specify the total number of steady state users, the ramp up and ramp down times, and the number of steps for the ramp:

•



To configure this pattern enter the following parameters:

- **Steady State Instances:** The maximum number of virtual users to run at steady state
- **Number of Steps:** The number of steps to use to reach max number of virtual users. The number of virtual users to introduce at each step is then: Steady State Instances divided by Number of Steps.
- **Ramp Up Time:** The time period over which virtual users are added to get to the Steady State. The time interval between steps is then: Ramp Up Time divided by Number of Steps.
- **Steady State Time:** The time period of the meaningful test run.
- **Ramp Down Time:** The time period over which virtual users are removed. Since the tests will run to completion after a "stop" request, the ramp down times are approximate.

You can enter a time followed by h for hours, m for minutes, s (or no letter) for seconds (default).

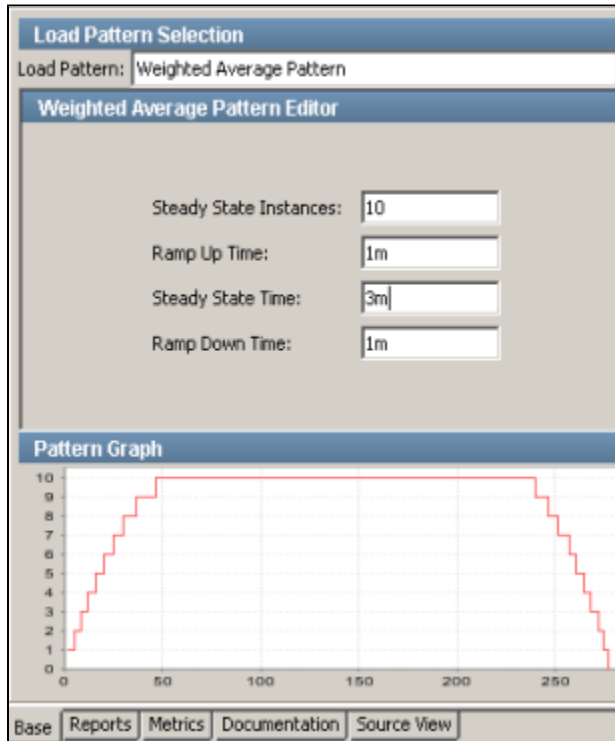
In the example above there are 10 instances and 2 steps, so 5 virtual users will be added each step. The ramp up time is 5 secs and the test will run in steady state for 1 minutes. Two virtual users will be removed approximately every 5 seconds.

The Pattern Graph shows graphically how the test will run.

[Back to List of Load Patterns\](#)

Weighted Average Pattern

The Weighted Average pattern will add and remove virtual users, (instances), based on a statistical calculation. LISA will calculate the number of steps, and the time period between steps, as frequently as every second, by honoring a moving weighted average. This will approximate a bell curve distribution, with the majority of the virtual users being added within 2 standard deviations of the mid-point of the load, or unload ramp time.



To configure this pattern enter the following parameters:

- **Steady State Instances:** The maximum number of virtual users to run at steady state.
- **Ramp Up Time:** The time period over which virtual users are added to get to the Steady State..
- **Steady State Time:** The time period of the meaningful test run.
- **Ramp Down Time:** The time period over which virtual users are removed. Since the tests will run to completion after a "stop" request, the ramp down times are approximate.

You can enter a time followed by **h** for hours, **m** for minutes, **s** (or no letter) for seconds (default).

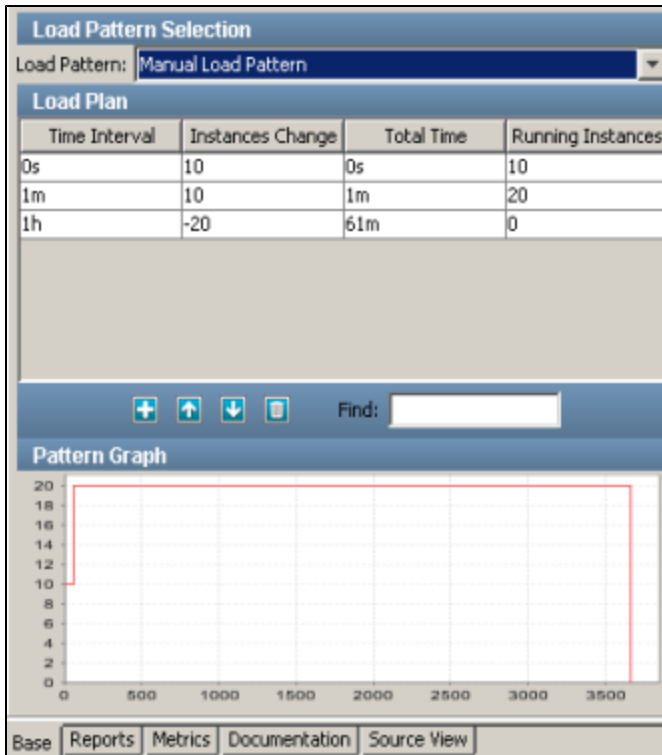
The example above shows the pattern for 10 virtual users ramping up over 1 minute, running in a steady state for 3 minutes, before ramping down over approximately 1 minutes.

The Pattern Graph shows graphically how the test will run.

[Back to List of Load Patterns\](#)

Manual Load Pattern

The Manual Load pattern gives you the most control over the loading and unloading of virtual users, (instances). It is similar to the Stair Step pattern except that it allows you to specify both the number of virtual users to add, and the time interval for each step in the pattern, individually:



To configure this pattern you define each step as a row in a table (see figure above). In each row you define (just) the time interval (Time), and the number of virtual users to add or remove (Instances Change) for that step. LISA will calculate and display the elapsed time (running time), and the total number of virtual users (Running Instances) for each step.

You can enter a time followed by h for hours, m for minutes, s (or no letter) for seconds (default).

Use the Standard Icons to Add, Remove or Change the current order of the steps from the available toolbar at the bottom of the table.

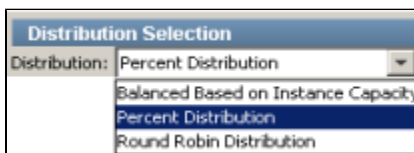
You may have to scroll to see these icons. Alternatively you can select a row and use the following short-cuts:

- Add a line => Ctrl+Shift+A
- Delete a line => Ctrl+Shift+D
- Move line Up => Ctrl+Shift+Up Arrow
- Move Line Down => Ctrl+Shift+Down Arrow
- Extended View => Ctrl+Shift+L

The Pattern Graph will show graphically how the test will run, and is updated dynamically as you add more steps.

28.2.1.2 Base Tab - Distribution Selection

28.2.1.2 Base Tab - Distribution Selection



1) Percent Distribution

The Percent Distribution distributes virtual users (instances), over the Simulators based on the percentages that you choose and is as shown above.

LISA is aware of the names of the running Simulators (plus local), so they are available in a pull-down menu in the Simulator Name column (see figure above).

You build your configuration by choosing a Simulator (Simulator Name), and specifying a percentage of virtual users (Percent), for this Simulator. Repeat this for all the Simulators you want to include until you have a total of 100%. Use integer values for the percentages.

Note: Although "Auto" appears as a choice here, we do not recommend that you use it. It will result in confusing allocations of virtual users, as it will compete with your explicit distribution choices.

2) Round Robin Distribution

The Round Robin Distribution requests that LISA control the allocation of virtual users, (instances), based on a simple round robin distribution pattern.

LISA picks an arbitrary Simulator and adds a user, then goes to the next Simulator and adds a user, continuing to add virtual users as needed. Once all Simulators have been used, the process continues with the first Simulator. This distribution is useful when staging a single, large load test on your system.

There are no parameters needed for this distribution.

3) Balance Based on Instance Capacity

The Balanced Based on Instance Capacity distribution requests that LISA control the allocation of virtual users, (instances), based on an assessment of current loading (percent load on each Simulator).

Each Simulator is initiated with a defined number of virtual users that can be allocated (default is 255). LISA dynamically tracks the percent load and adds virtual users to specific Simulators to try to keep the load percentage as even as possible. So, the Simulator with the lowest percentage load is a candidate for the next virtual user introduced into the system.

This distribution is useful when the system is already running tests from several other testers, and you want to optimize your load distribution.

There are no parameters needed for this distribution.

28.2.2 Reports Tab

28.2.2 Reports Tab

The Reports tab is where you specify the **reports** you want to produce after every test case or test suite run.

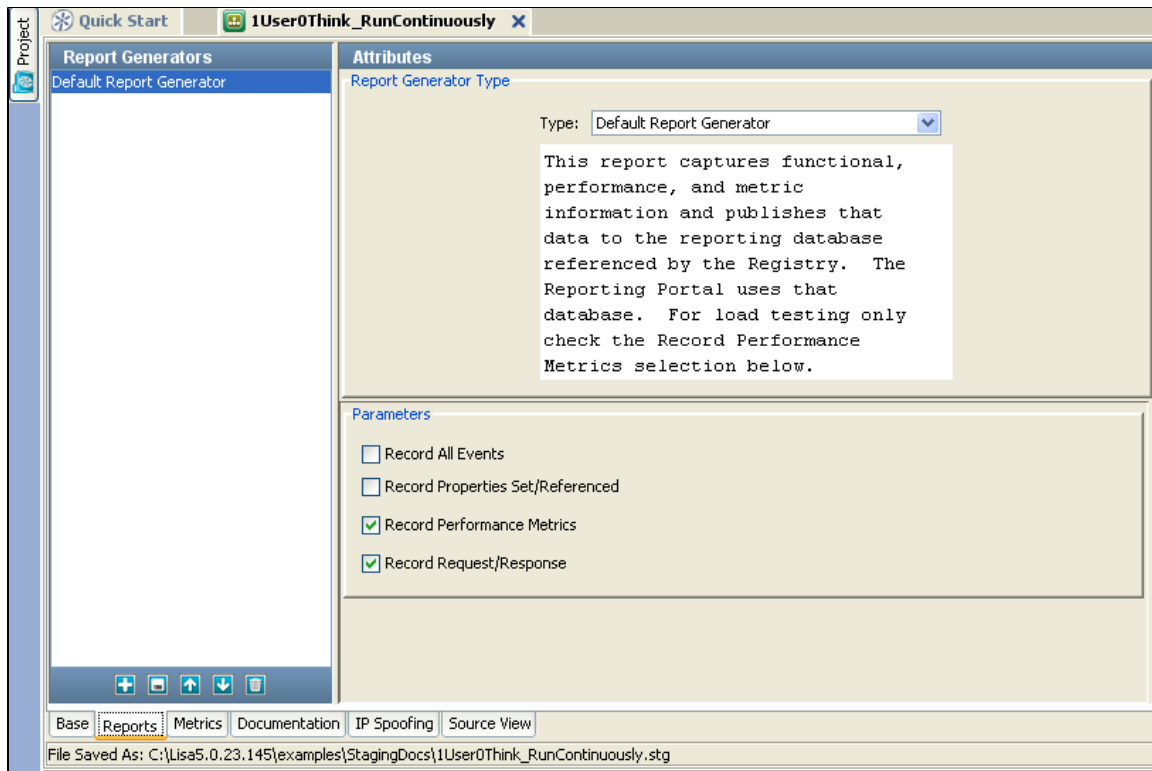
There are two types of report generators available in LISA.

- [Default Report generator](#) - available for LISA 5.x version related reports.
- [Legacy Report generator](#) - available for LISA 4.x version related reports. These reports are marked as (legacy).

Details of the reports available in each report generator, viewing reports, report contents and output options are discussed in detail in [Reports](#) section.

The metrics to capture for the reports are discussed more fully in the next section [Metrics](#).

The Reports tab in a **Staging document** is shown below. By Default, LISA opens the 5.0 Default Report Generator. Some of the reports are added by default in this sample staging document.



When you select a report from the drop down, a brief summary of that report appears in the box below.

According to the selected report, its parameters will change. You need to set the parameters as and where necessary.

The Basic Reports tab is divided into the following:

Right panel - This is where you select the report to be added.

Attributes: This contains the "Report Generator Type" and the required "Parameters" for the report.

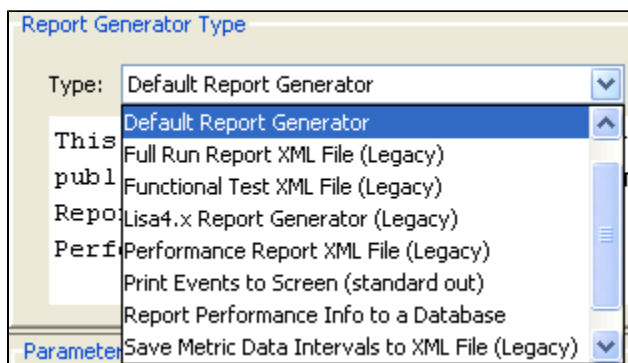
--Report Generator Type - A pull-down menu lists the available Report Types in LISA.

--Parameters - These are the Parameters required to set the selected report.

Left panel - This shows the list of added reports.

Type of Reports

There are the following types of Reports which can be generated:



- Append Events to Comma Delimited Files
- Default Report Generator
- Full Run Report XML File (Legacy)
- Functional Test XML File (Legacy)
- Lisa4.x Report Generator (Legacy)
- Performance Report XML File (Legacy)
- Prints events to screen (Standard Out)

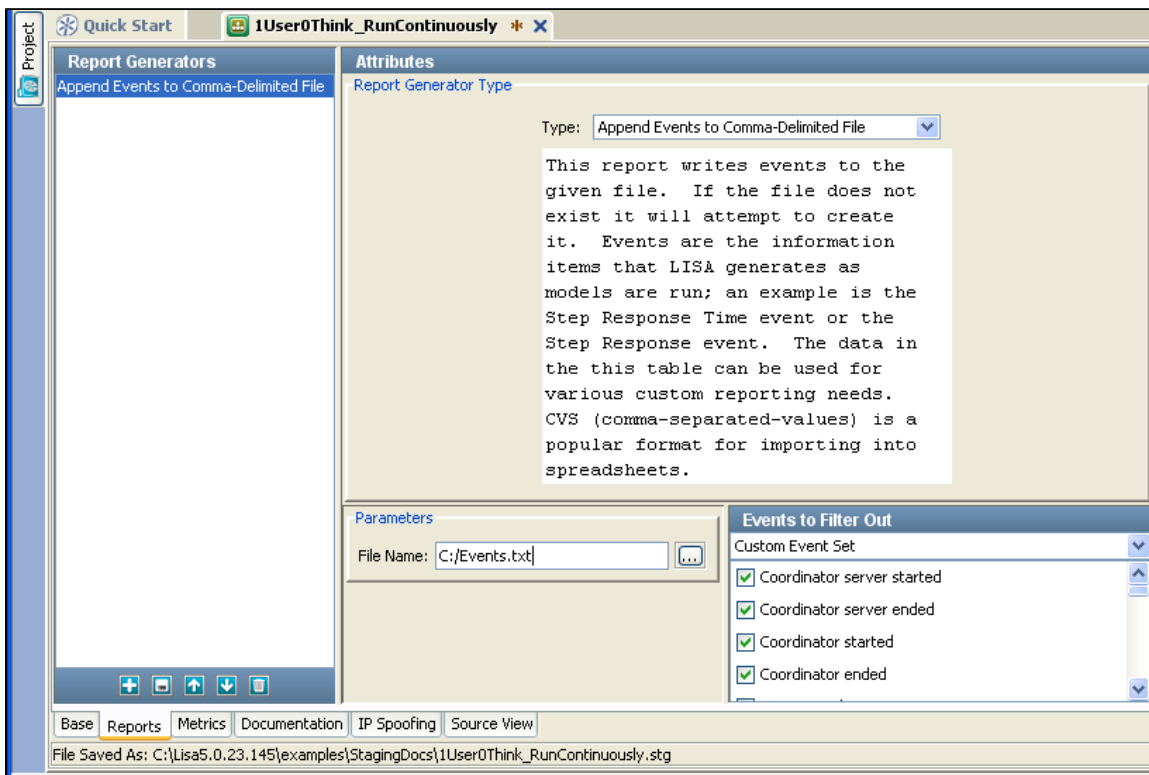
- Report Performance Info to a Database
- Save Metric Data Intervals to XML File (Legacy)
- Write Events to JDBC Data Table

All these Reports are explained in detail in the [Reports](#) Chapter.

Add a Report

To Add a Report,

- Select the required Report to be generated, from the "**Report Generator Type**" drop down.
For example we have selected the "**Append Events to Comma Delimited Files**" Report to be generated. This Report writes Events to the given File.
- Click on the **Add Report** icon on the toolbar at the left bottom to add the Report in the "Report Generator" list as shown below:



You need to fill in the required parameters and select the Events to be Filtered out from the given Event list.

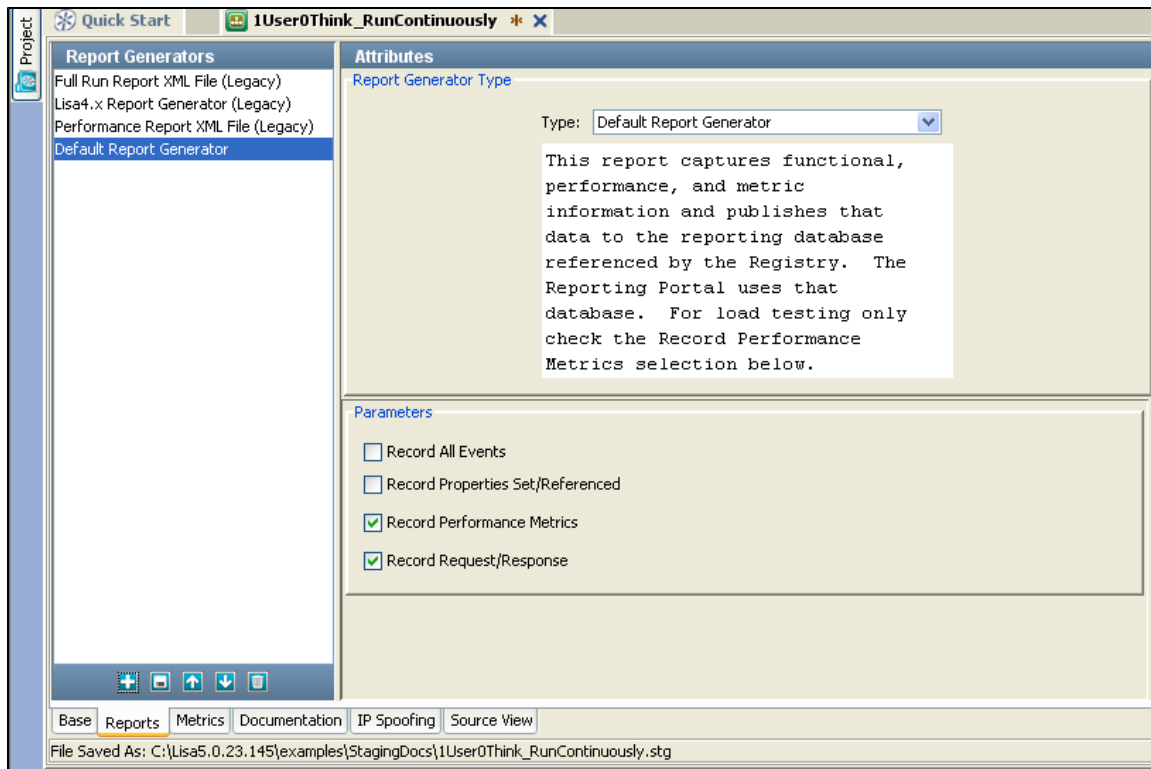
- File name - Enter the file name to which the Report will be written.
- Click on the **Add** icon to add the Report in the Report Generators list on the left.

As seen above, the **Append Events to Comma Delimited Files** report is added to the Report Generator list.

Similarly all other Reports can be added.

For Legacy Reports (LISA 4.x Version)

This the screen for 4.x version Reports:



The Reports tab is divided into the following:

- **Report Generators:** Shows a list of selected report types
- **Attributes:** A pull-down menu of available Report Types in LISA
- **Parameters:** Parameters available for the selected type of Report. This will change for the type of report selected.
- **Auto-generate:** A list of Reports added. Information includes Enable reports, Type of Report, Format of the Report, Filename, Time stamp of the Report and email.

Note - The filename of the auto generated files can now be appended with the timestamp so that the user does not have to give different file names each time. This is a checkbox besides the filename and if unchecked the file name will not be appended with the timestamp.

- **Events to Filter Out:** A entire list of LISA events, which can be filtered out.

LISA 4.x Report Types

In the "**Report Generator Type**" panel, there is a "**Type**" pull-down menu listing all the available Reports.

The following report types are available; the default report generator is shown in bold:

- *Append Events to Comma-delimited File
- Full Run Report (Large, Use with Caution)
- Functional Test Report (Test Case only)
- LISA Default Report Generator (events and metrics written to central embedded database)
- LISA5 Report generator (preview)
- Performance Report XML Document
- Print Events to Screen (standard out).
- Save Metric Data Intervals to XML Document
- **Report Performance Info to a Database:
- **Write Events to JDBC Database Table

In the left panel, a list of selected reports is displayed:

- To add a report, click the **Add** icon at the bottom of the list panel, and select the report type from the Type pull-down menu in the top right panel.

Note: If you choose a report type that requires a CSV file*, or a standalone database**, you will be prompted for additional information, as shown below for the CSV file, and the standalone database, (do not confuse this with the new default report generator database). This is a database of your choice, specified using the screen shown below:

CSV file:

Attributes

Report Generator Type

Type: Append Events to Comma-Delimited File

Parameters

File Name:

Standalone Database:

Attributes

Report Generator Type

Type: Report Performance Info to a Database

Parameters

JDBC Driver:

Connection URL:

User:

Password:

In the right panel, there is **Events to Filter out** list, a list of all the Events available in LISA. Each one has a checkbox. Using this list you can filter out events you **do NOT** want captured in a report. If you place a check next to an event it will NOT be captured by the associated report.

LISA Sub Reports Types

You can choose from the following Sub Reports:

- Report Metric Data Info
- Report Metric Data Info (Grouped)
- Functional Report
- Performance Summary
- Suite Report

To select Events to be included in one of your reports, highlight the report and make sure the check box next to the Event is left unchecked.

Selecting Events

For convenience there are 3 standard sets of events, listed in the pull-down menu above the Event list.

- Terse Event Set
- Common Event Set
- Verbose Event Set

These sets provide a pre-selected list of events that you can use as a starting list. You can then customize the list for each of your report types. Unless there is a specific need to change the Event list it is recommended that you use the Common Event Set.

For more details, refer to the [Events](#) chapter.

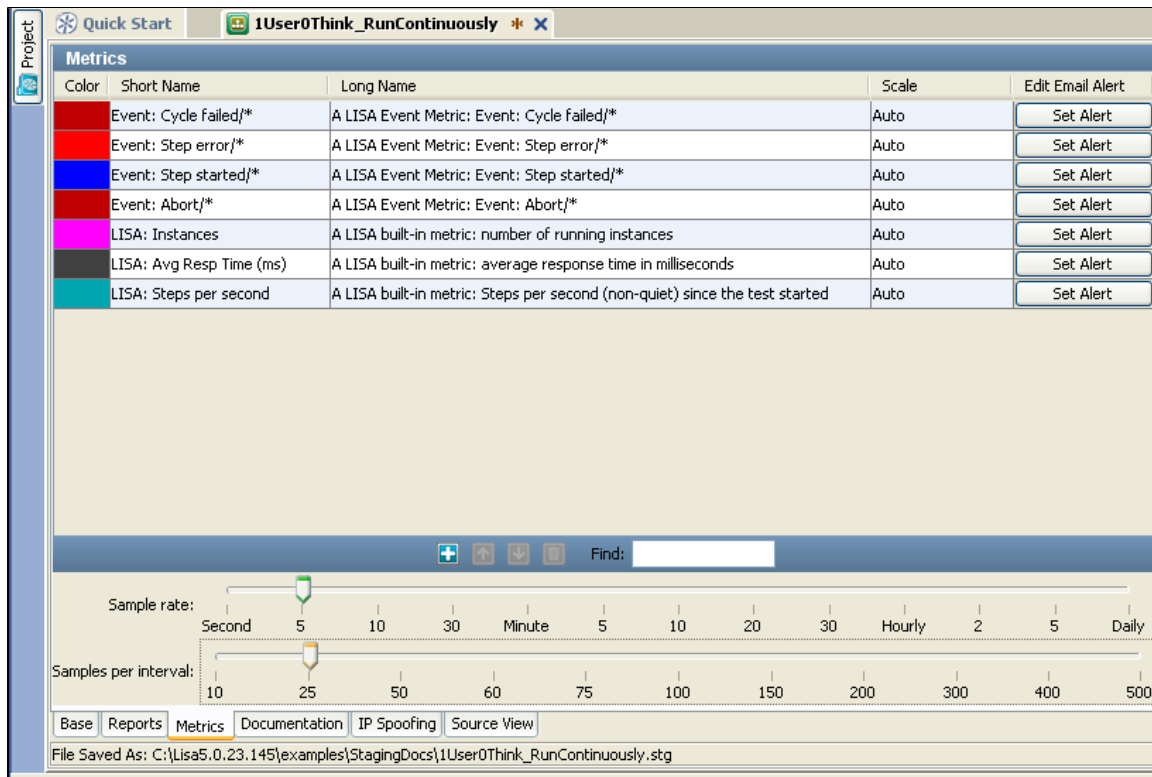
28.2.3 Metrics Tab

28.2.3 Metrics Tab

The Metrics tab is where you select the test metrics you want to record.

You can also set the sampling specifications for the collection of the metrics and set an email alert on any metric that you have selected.

The Metrics tab is shown below:



The Metrics tab is divided into 2 sections.

The **top panel** has the default Metrics listed, differentiated by color code and the **bottom panel** has the sampling parameters.

You can add or delete Metrics and set email alerts in the top panel.

Following are the major Metrics categories available in LISA:

- LISA Whole-Test Metrics
- LISA Test Event Metrics
- SNMP Metrics
- JMX Attribute Reader (JMX metrics)
- TIBCO Hawk
- Windows Perfmon Metrics
- Unix Metrics via SSH

More information about each Metric is available in the Metrics Chapter.

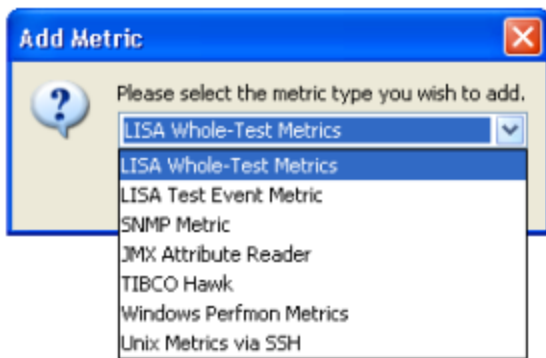
Add a Metric

To Add a Metric,

- Click on the **Add** icon on the toolbar

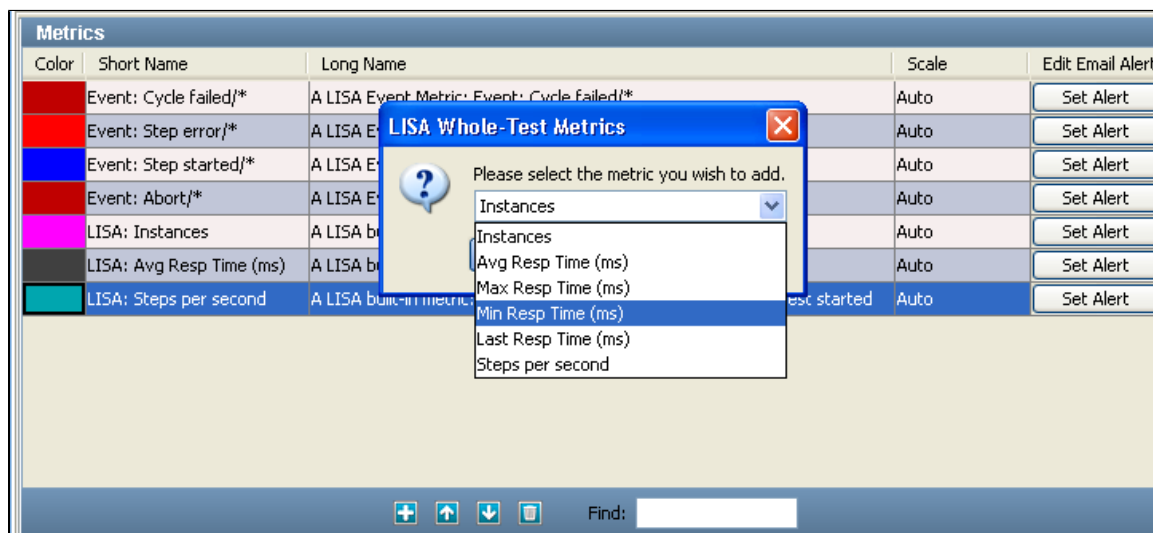


A dialog box to Add metrics will open up with a list of Metrics which can be added:



- Select the desired Metric type and click **OK**.
(For purpose of illustration, we have selected the **LISA Whole-Test Metrics**.)

The Metric Selection dialog box opens as below:



- Select the desired sub-category for this Metrics and click **OK**. Sub categories are different for each Metrics.


The newly added Metrics is seen in the list of already existing Metrics in a different color as shown below: It can be seen as last line added in the Metric list with Black Color code.

Metrics				
Color	Short Name	Long Name	Scale	Edit Email Alert
	Event: Cycle failed/*	A LISA Event Metric: Event: Cycle failed/*	Auto	Set Alert
	Event: Step error/*	A LISA Event Metric: Event: Step error/*	Auto	Set Alert
	Event: Step started/*	A LISA Event Metric: Event: Step started/*	Auto	Set Alert
	Event: Abort/*	A LISA Event Metric: Event: Abort/*	Auto	Set Alert
	LISA: Instances	A LISA built-in metric: number of running instances	Auto	Set Alert
	LISA: Avg Resp Time (ms)	A LISA built-in metric: average response time in milliseconds	Auto	Set Alert
	LISA: Steps per second	A LISA built-in metric: Steps per second (non-quiet) since the test started	Auto	Set Alert
	LISA: Min Resp Time (ms)	A LISA built-in metric: minimum observed response time in milliseconds	Auto	Set Alert

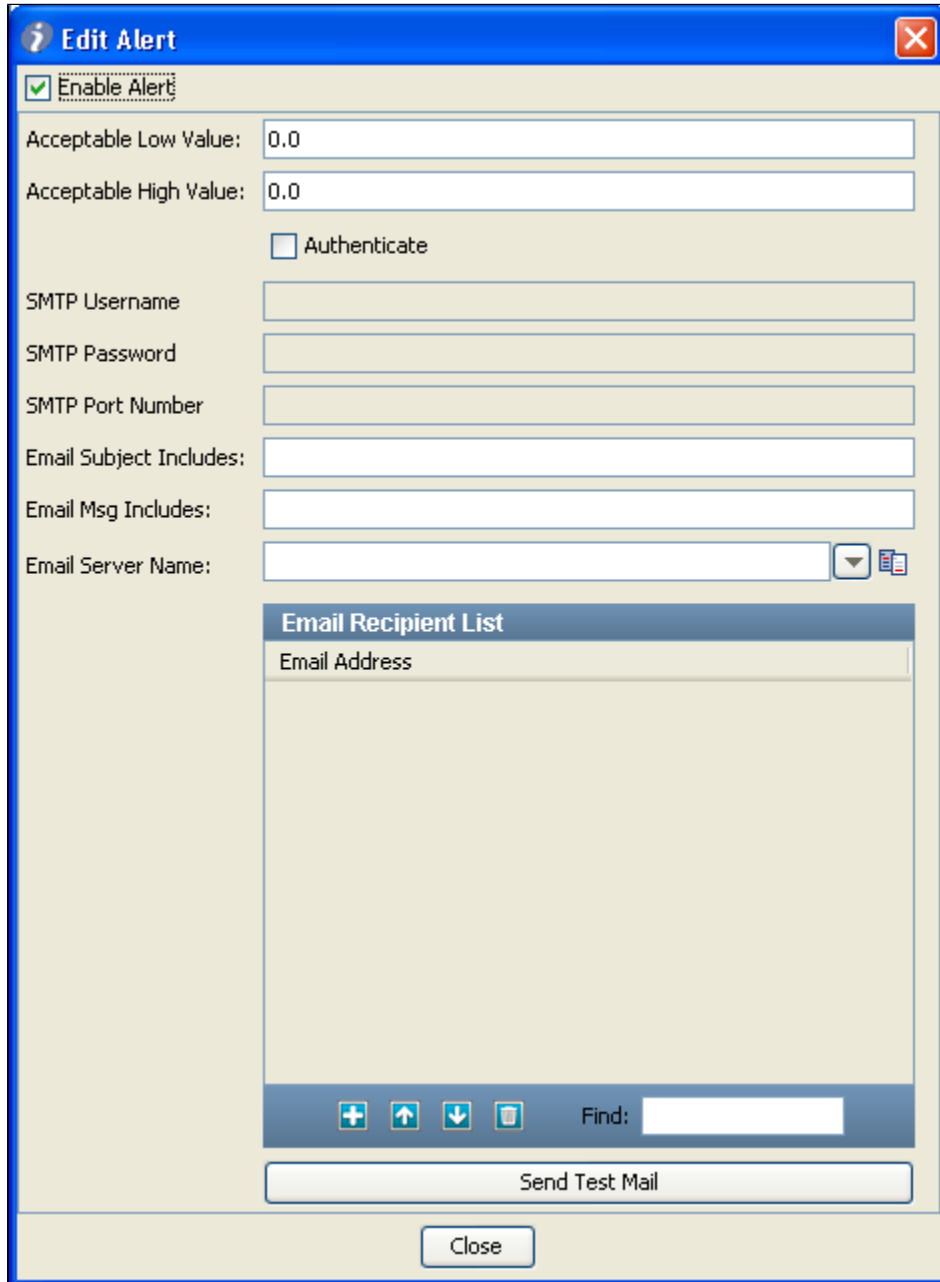
Descriptions of all the Metrics in all categories, and details on how to configure them for inclusion in Reports, can be found in [Metrics](#).

Setting an Email Alert

To set an email alert on an individual metric

- Click the **Set Alert**  button corresponding to that metric.

The following pop-up screen will be displayed:



The **Edit Alert** dialog box is shown. It has a blue title bar with a question mark icon and a close button. The main area is light beige. At the top left, there is a checkbox labeled **Enable Alert** which is checked. Below this are two text boxes: **Acceptable Low Value:** with the value **0.0** and **Acceptable High Value:** with the value **0.0**. Below these is a checkbox labeled **Authenticate** which is unchecked. Then there are three text boxes: **SMTP Username**, **SMTP Password**, and **SMTP Port Number**. Below these are two more text boxes: **Email Subject Includes:** and **Email Msg Includes:**. Below these is a text box for **Email Server Name:** with a dropdown arrow and a small icon to its right. Below the text boxes is a section titled **Email Recipient List** with a list box containing the header **Email Address**. At the bottom of the list box are four icons: a plus sign, an up arrow, a down arrow, and a trash can. To the right of these icons is a **Find:** text box. Below the list box is a **Send Test Mail** button. At the very bottom of the dialog is a **Close** button.

On this screen, you can set the acceptable limits for the metric (low and high value), and the details to be sent in an email. You must provide the name of your email Server and a list of email addresses.

Email alerts added to staging documents store the SMTP password as an encrypted value. Also, if you open an existing staging document and re-save it, the SMTP password will be saved as an encrypted value.

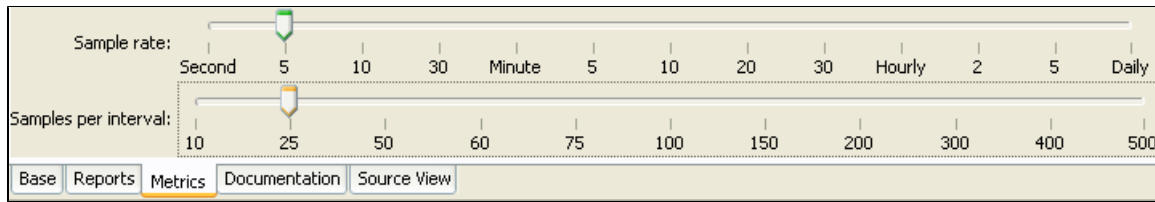
Email addresses are added and removed, using the **Add** and **Delete** icon at the bottom of the window. Click **Close** when finished.

Notice that the Set Alert button is now an Edit Alert button on the Metric tab. This shows which Metrics have alerts set.

Note - For the email alert to work, you also need to set the SMTP server related paths in the lisa.property file.

Sampling Parameters

The bottom panel has two slider bars that allow you to set your **sampling parameters**:



- **Sample Rate:** Specifies how often to take a sample, i.e. record the value of a metric. It is specified as a time period, and is the reciprocal of the sampling rate.
- **Samples per Interval:** Specifies how many samples are used to create an interval for calculating summary values for the metric.

Taking sample values every minute (Sample Rate =1 minute), and averaging every 60 samples (Samples Per Interval=60), will produce a metric value every minute, and a summary value (average) every hour.

For example, the default is a 1 second Sample Rate, and 10 samples per interval (making an interval 10 seconds).

The above example uses 5 seconds per sample, and 25 samples per interval, producing a metric value every 5 seconds and a summary value every 125 seconds.

Metric values are stored in XML files or database tables for inclusion in reports (see the previous section on reports).

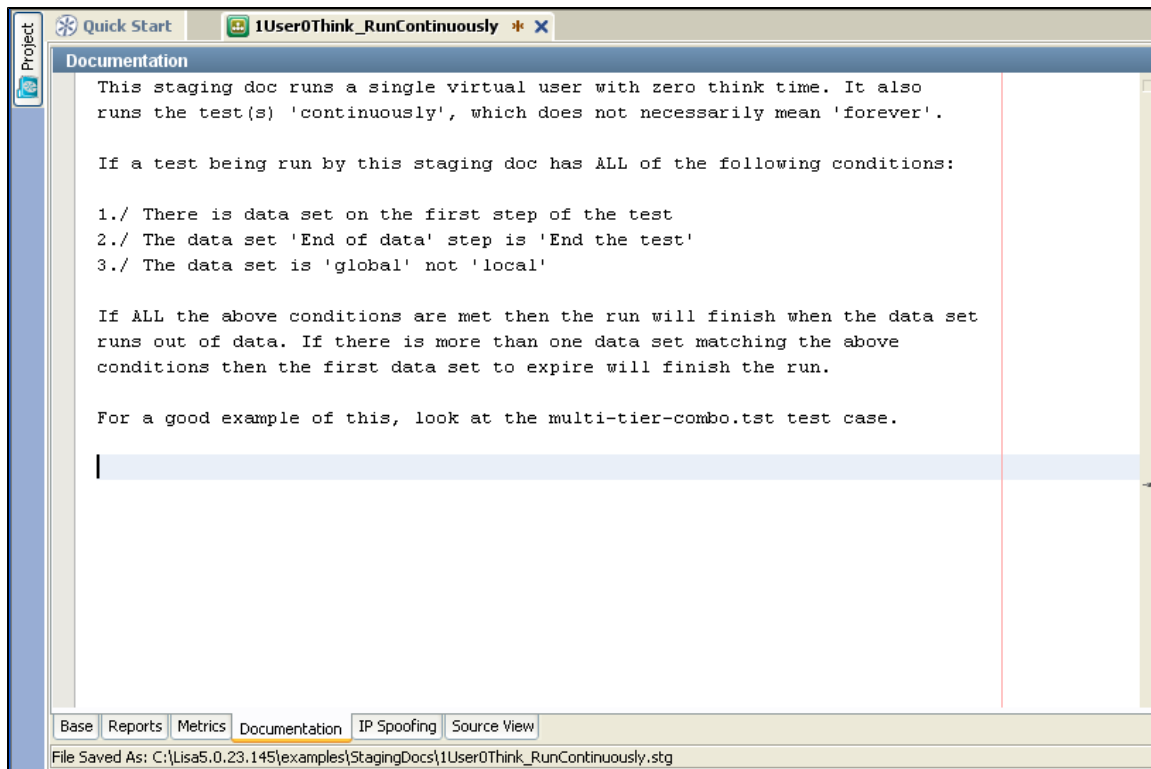
28.2.4 Documentation Tab

28.2.4 Documentation Tab

The Documentation tab will contain the staging document related documentation.

It is a good practice to write documentation, so that other users are able to understand the purpose of the document in a better way.

For the purpose of illustration we have shown below the Documentation tab of the Staging document that we have used - **1User0Think_RunContinuously.stg**



28.2.5 IP Spoofing Tab

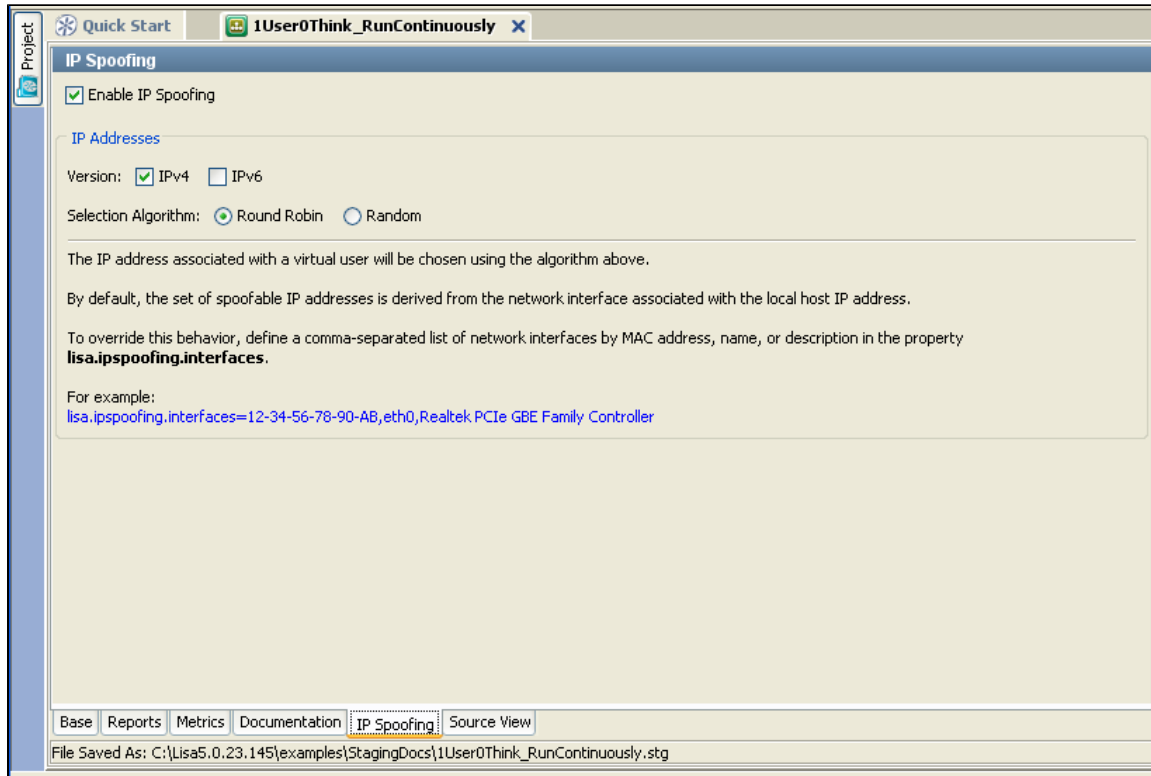
28.2.5 IP Spoofing Tab

IP spoofing support in LISA allows multiple IP addresses on a network interface to be utilized when making network requests.

In a performance testing scenario, enabling IP spoofing against a system under test gives the appearance that requests are originating from many different virtual users. For systems such as web applications, this outcome will often more closely resemble "real-world" behavior.

IP spoofing is introduced in LISA 5.0.23.XX version.

This tab is seen in the staging document:



Enable IP Spoofing: If selected, IP addresses will be spoofed for all supported LISA steps in a test case.

Version

IPv4: If selected, IPv4 addresses will be spoofed.

IPv6: If selected, IPv6 addresses will be spoofed.

Note: At least one of IPv4 or IPv6 must be selected.

Selection Algorithm

Round Robin: Spoofed IP addresses will be selected in a round-robin format.

Random: Spoofed IP addresses will be selected in a random format.

Support

Currently, IP spoofing is only supported for HTTP.

The following LISA steps can be configured to use IP spoofing:

- HTTP/HTML Request
- REST Step
- Web Service Execution (XML)
- Raw SOAP Request

Configuring IP Address in Windows

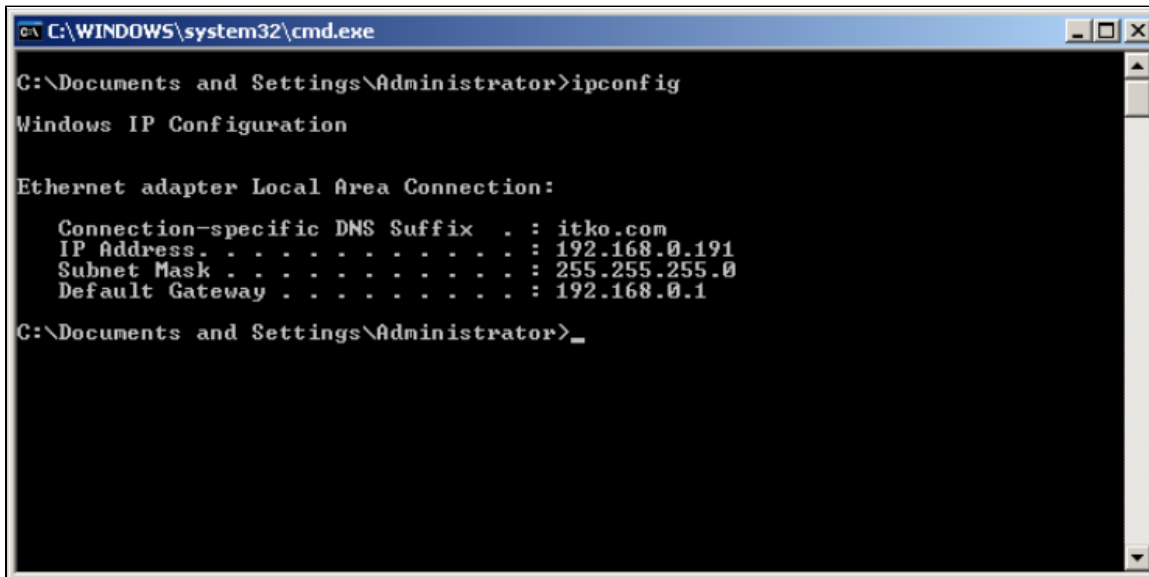
This section provides instructions on how to add IP addresses to a network interface for IP spoofing.

Windows

Note: Before adding IP addresses, make sure that you are an Administrator.

On Windows, IP address information can be obtained using the command-line utility **ipconfig**.

For example, the following screenshot shows the output of **ipconfig** for a server with a single network interface card. It has a single IP address, **192.168.0.191** and is named **Local Area Connection**.

A screenshot of a Windows command prompt window. The title bar reads "C:\WINDOWS\system32\cmd.exe". The command prompt shows the user running "ipconfig" from the directory "C:\Documents and Settings\Administrator". The output displays the configuration for the "Ethernet adapter Local Area Connection:", including the "Connection-specific DNS Suffix" as "itko.com", the "IP Address" as "192.168.0.191", the "Subnet Mask" as "255.255.255.0", and the "Default Gateway" as "192.168.0.1". The prompt ends with a trailing underscore character.

```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrator>ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . : itko.com
    IP Address. . . . .               : 192.168.0.191
    Subnet Mask . . . . .            : 255.255.255.0
    Default Gateway . . . . .         : 192.168.0.1

C:\Documents and Settings\Administrator>_
```

To add a single IP address, 192.168.0.201, the **netsh** command-line utility can be used as below:

```
netsh in ip add address "Local Area Connection" 192.168.0.201 255.255.255.0
```

If many IP addresses are to be added, **netsh** can be used in a loop.

For example, the following command will add 9 more IP addresses between 192.168.0.202 and 192.168.0.210:

```
for /L %i in (202, 1, 210) do netsh in ip add address "Local Area Connection" 192.168.0.%i
255.255.255.0
```

If these commands are successful, the new IP addresses can be verified using the command-line utility **ipconfig /all** as shown below:

```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Administrator>ipconfig /all

Windows IP Configuration

Host Name . . . . . : win2003server
Primary Dns Suffix . . . . . :
Node Type . . . . . : Mixed
IP Routing Enabled. . . . . : No
WINS Proxy Enabled. . . . . : No
DNS Suffix Search List. . . . . : itko.com

Ethernet adapter Local Area Connection:

Connection-specific DNS Suffix . : itko.com
Description . . . . . : VMware Accelerated AMD PCNet Adapter
Physical Address. . . . . : 00-0C-29-E9-37-08
DHCP Enabled. . . . . : No
IP Address. . . . . : 192.168.0.210
Subnet Mask . . . . . : 255.255.255.0
IP Address. . . . . : 192.168.0.209
Subnet Mask . . . . . : 255.255.255.0
IP Address. . . . . : 192.168.0.208
Subnet Mask . . . . . : 255.255.255.0
IP Address. . . . . : 192.168.0.207
Subnet Mask . . . . . : 255.255.255.0
IP Address. . . . . : 192.168.0.206
Subnet Mask . . . . . : 255.255.255.0
IP Address. . . . . : 192.168.0.205
Subnet Mask . . . . . : 255.255.255.0
IP Address. . . . . : 192.168.0.204
Subnet Mask . . . . . : 255.255.255.0
IP Address. . . . . : 192.168.0.203
Subnet Mask . . . . . : 255.255.255.0
IP Address. . . . . : 192.168.0.202
Subnet Mask . . . . . : 255.255.255.0
IP Address. . . . . : 192.168.0.201
Subnet Mask . . . . . : 255.255.255.0
IP Address. . . . . : 192.168.0.191
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.0.1
DNS Servers . . . . . : 209.18.47.61
                       209.18.47.62

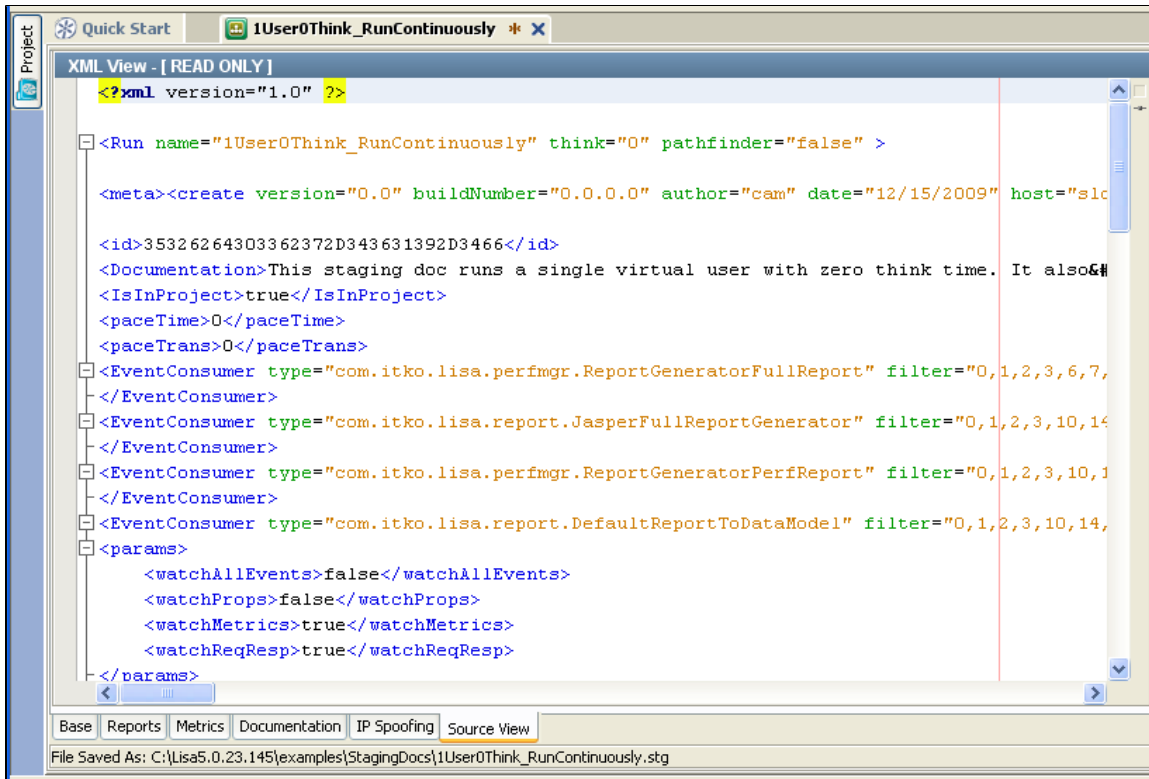
C:\Documents and Settings\Administrator>
```

28.2.6 Source View Tab

28.2.6 Source View Tab

The Source View tab shows the xml source of the staging document.

For the purpose of illustration, we have shown the Source tab of the staging document we have used - **1User0Think_RunContinuously.stg**



14. Building Test Suites

14. Building Test Suites

You can stage a combination of test cases together in LISA. This bunch of test case run together are in a form of a test suite.

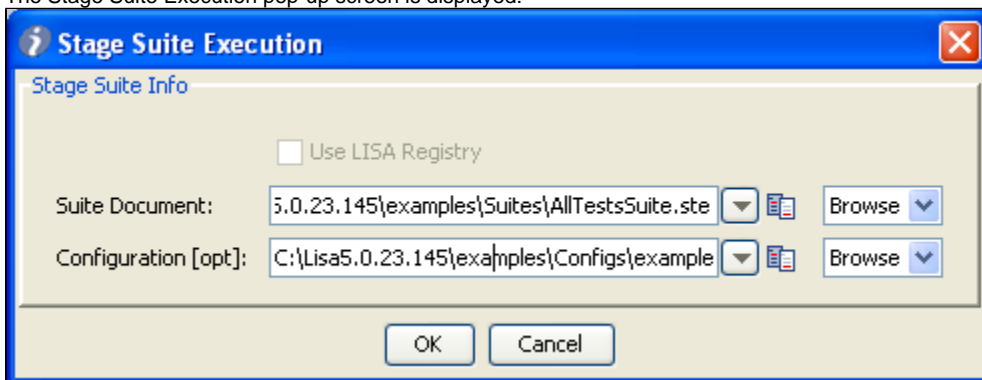
Starting/Running an entire test suite in LISA requires that you are running LISA Workstation.

Within the Workstation, you can configure and stage the test, start the test, monitor the test while it is running, and view the reports at the conclusion of the test.

To Start a Test suite,

- Click the **Start Suite**, icon on the toolbar.
- Or From the main menu, select Actions > Start Test Suite Execution

The Stage Suite Execution pop-up screen is displayed:



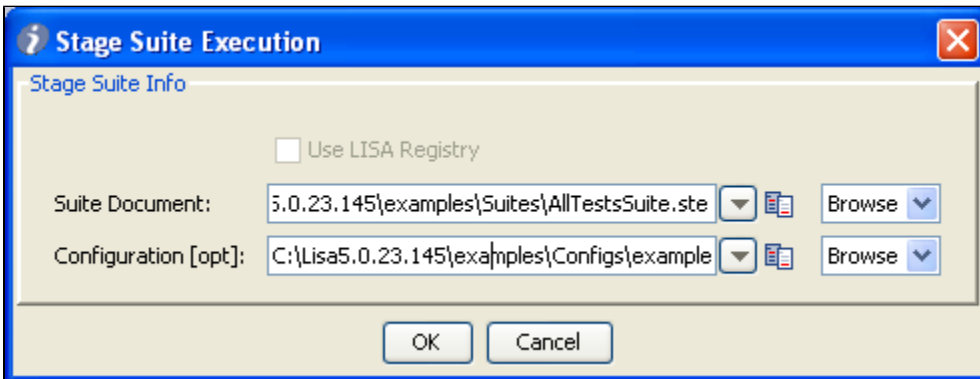
The following topics are available.

14.1 Running a Test Suite
Delete
14.2 Test Registry Monitor for LISA Server

14.1 Running a Test Suite

14.1 Running a Test Suite Document

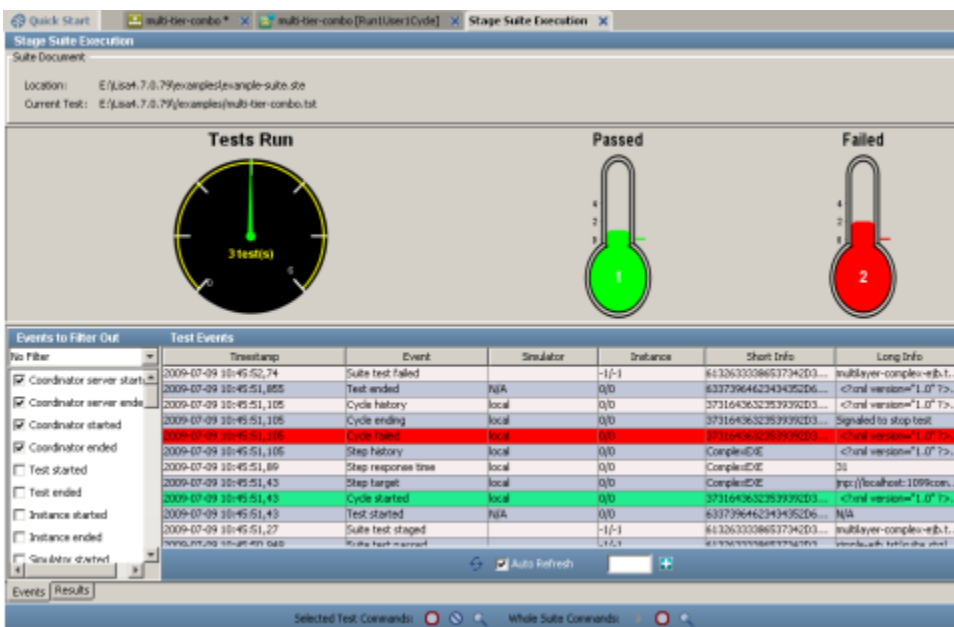
For the purpose of illustration we have taken the **AllTestsSuite.ste** test suite from the examples directory.



Enter the follow parameters:

- **Use LISA Registry:** This field is disabled as, since 5.0 the LISA Registry is mandatory to start LISA Workstation.
- **Suite Document:** The name of the test suite document. The document can be in the file system, on the classpath, or specified by a URL. You can browse the locations using the Browse pull-down menu. For detailed information on building test suites, see [Building Test Suites](#)
- **Configuration [opt]:** The name of the configuration document to use, or the name of a configuration in the Test Case document. The document can be in the file system, on the classpath, or specified by a URL. You can browse the locations using the Browse pull-down menu. Entering a configuration is optional, and if left blank the configurations specified Using Configurations in the staging documents will be used. For detailed information on using configurations, see [Using Configurations](#)
- Click **OK**.

The **Stage Suite Execution** screen is displayed and the tests start:



The **top panel** displays information about the test suite: **location** of the test suite document and the **current test** name.

The **middle panel** shows the **Test Run meter** showing the total number of tests, and the number completed.

To its right are two thermometers, **Passed and Failed tests**, that show the pass count and the fail count in real time. The colors of the thermometer indicate that you have passed our preselected number of cases. The know behavior in Lisa when you run over 50 - 100 testcases in a Suite is:

- If number of test passed > 50 - color of the thermometer is Orange.
- If number of test passed > 75 – color is Red.
- If number of test passed > 100 – color is Gray.

The **lower panel** has two tabs:

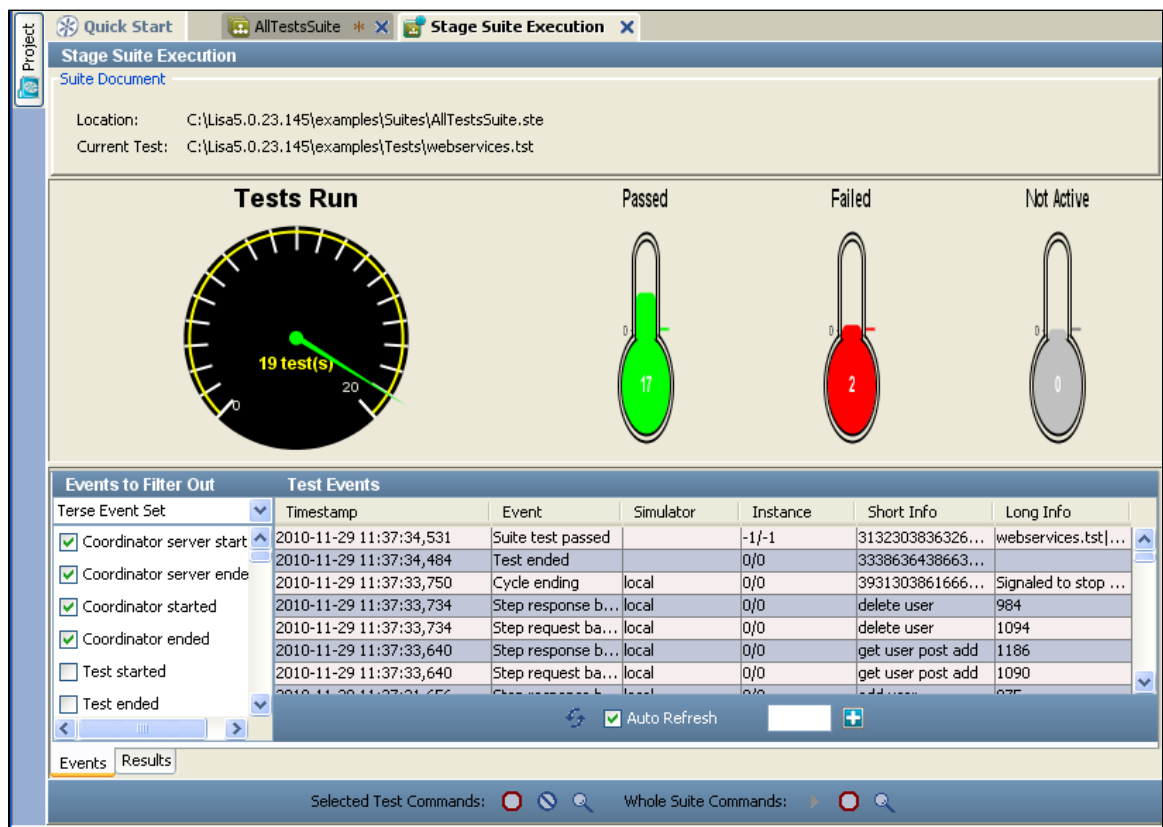
- **Events Tab**: Lists requested events as they are emitted
- **Results Tab**: Shows the status of each individual test

14.1.1 Events Tab

14.1.1 Events Tab

The Events tab is the tab that opens by default in a test suite document.

As shown below, it lists the events that you have selected from the panel on the left of the tab.



It consists of two sections:

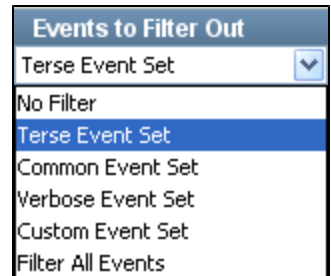
Events To Filter Out

Events to Filter Out panel is a list of all the Events available in LISA. Each one has a checkbox. Use this list to Filter out events you **do NOT** want to monitor.

You can filter out Events by two ways –

Manually – By checking individual Events to be filtered out.

You can choose **"No Filter"** in the drop down and then select the desired Events from the list to filter out individually.



Event Set – LISA provides certain Event "sets" by default. Each Event set has certain Events added in that set. For convenience there are 3 standard sets of Events listed in the pull-down menu.

- Terse Event Set
- Common Event Set
- Verbose Event Set

These sets provide a pre-selected list of events that you can use as a starting list. You can then customize the list.

These sets provide a pre-selected list of events that you can use as a starting list. You can then customize the list.

Test Events Panel

Test Events panel lists the events as they are emitted, with the most recent on the top of the list. You can list events in real time by checking the Auto Refresh box at the bottom of the panel, or you can refresh the list manually by clicking the Refresh icon, with the Auto Refresh box unchecked.

For each event, the following information is posted:

- **Name:** The Event ID of the event
- **Simulator:** The name of the Simulator where the instance is running
- **Instance:** The Instance ID and Run Number (in the form Instance ID/Run Number)
- **Short Info:** The short information for the event
- **Long Info:** The long information for the event

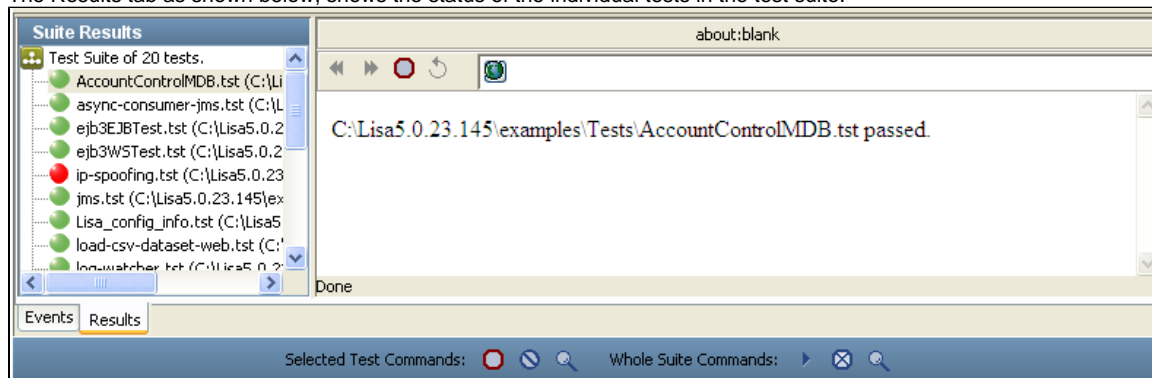
For more information on the event attributes – Event ID, Short Info, and Long Info, see Events in *the LISA Reference Guide*.

14.1.2 Results tab

14.1.2 Results tab

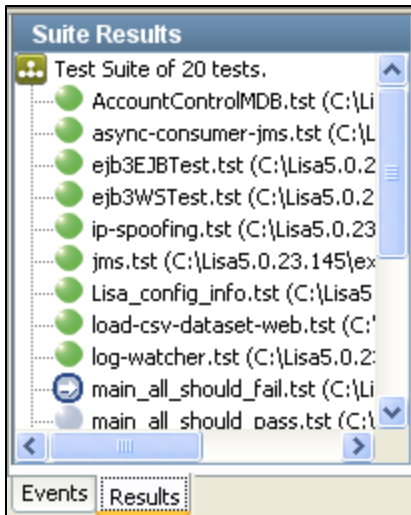
Results Tab

The Results tab as shown below, shows the status of the individual tests in the test suite:



It consists of two sections:

- 1) **Suite Results:** This displays a **list of all the tests** in the suite.



The tests are shown by three icons in the Results tab.

Tests preceded by a **green icon** have completed and passed.

Tests preceded by a **red icon** have completed and failed

Tests preceded by a blue icon and a arrow are still running.

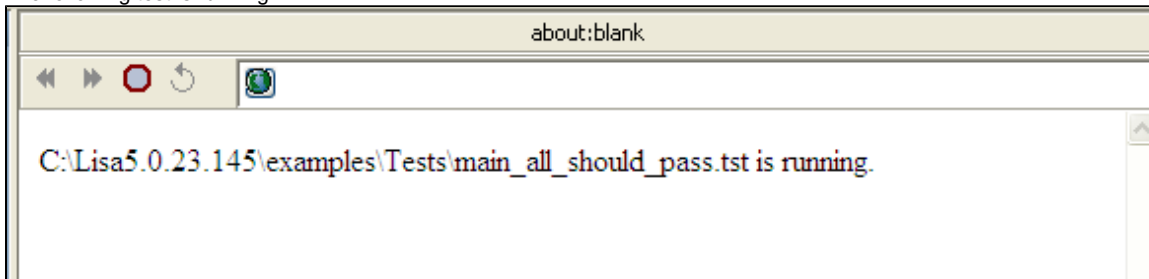


For tests that are still running you can click the **View Test** icon at the bottom of the panel to bring up the test monitor for this test. This is explained in more detail below.

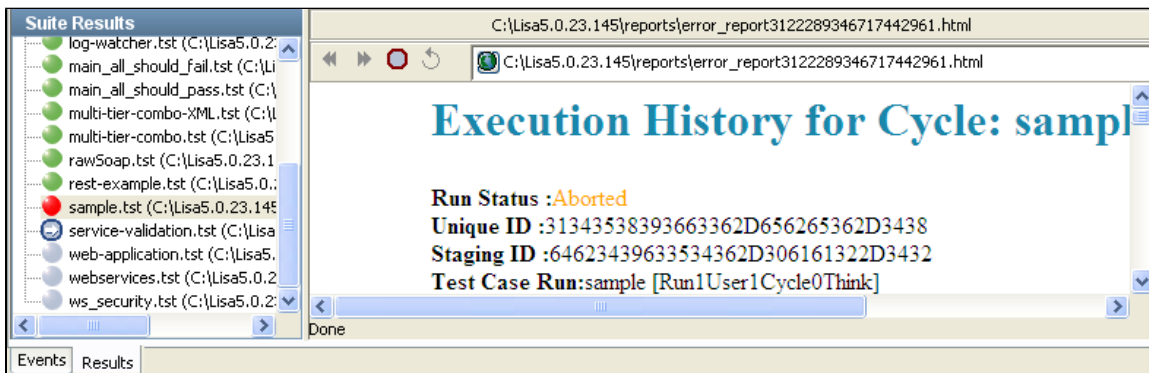
For completed tests, you can highlight the test name to see a status in the text area to the right. This is most useful to see why a given test failed.

2) **Status Window** : This displays the status of the test selected on the left.

The following test is running -



The following test has failed -



The two figures above show examples. A failed test status shows as much information as is available as to why the test failed.

At the bottom of the Stage Suite Execution panel is a toolbar.

The first set of icons **manages individual tests** in the suite.

To use these icons, first **select a test** in the Test List section of the Results tab.

The following functions are available for a particular **selected test**:



Stop Test: Stops the test after it reaches the next logical end/fail. It does not start a new cycle.



Kill Test: Stops the test immediately after the current step completes.



View Test: This will work only for currently running tests. This Launches the Test Monitor for the selected test.

The second set of icons **manages the test suite** itself, or when suite is running:



Run Suite: Starts/restarts the suite test.



Close Suite: Close the Stage Suite Execution Window.



View Test: Launches the Test Monitor for the selected test.

For more information on Test Monitor, refer to [Test Registry Monitor](#) .

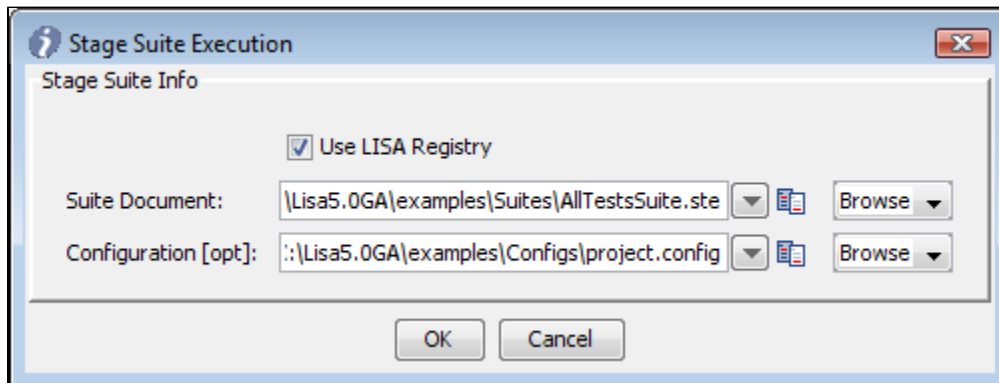
14.2 Test Registry Monitor for LISA Server

14.2 LISA Registry Monitor for LISA Server

The Test Monitor for LISA Server looks very similar to the test monitor for LISA Workstation, and you use it the same way as we have already described earlier in this chapter.


There are some extra features like the **LISA Registry Monitor** Panel and the **Optimizers** Panel as shown below on the left of the Test Monitor:

For the purpose of illustration, we will run a test suite named "**AllTestSuite.ste**" from the LISA examples directory.



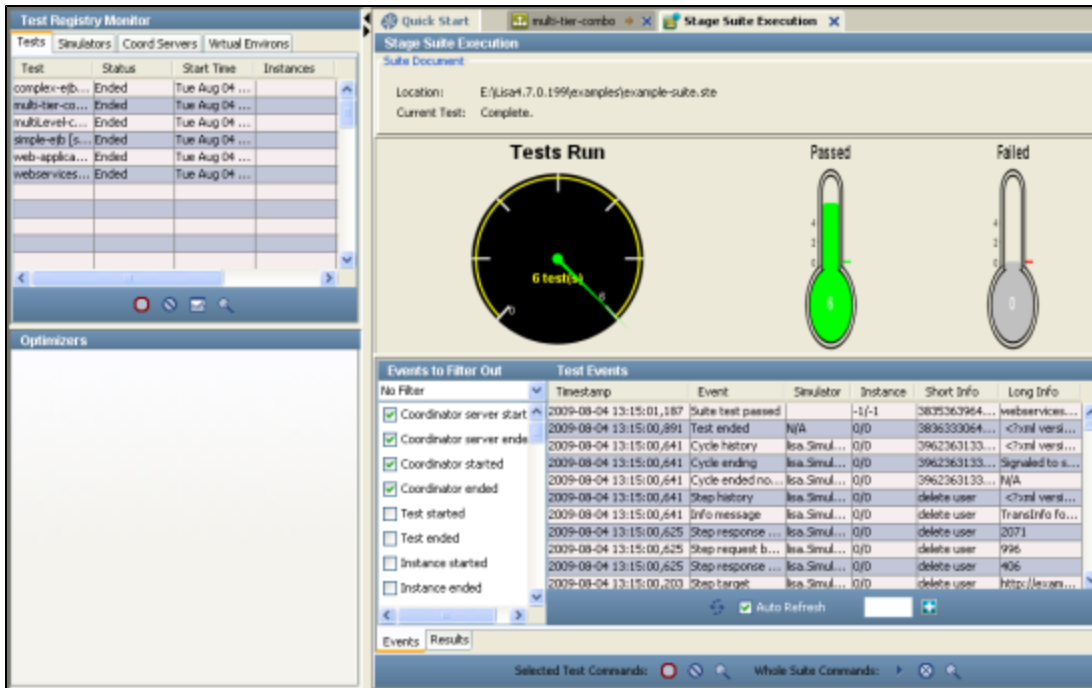
To open the LISA Registry monitor for the suite document,



Click the **Toggle Registry**, , icon on the toolbar.

Or from the main menu, select **View -> Toggle LISA Registry Monitor**

Or use the shortcut **CTRL + SHIFT + Y**



The above screen shows the LISA Registry Monitor and the Optimizers in the left panel.

The top panel, the "LISA Registry Monitor" has four tabs –

- Tests
- Simulators
- Coord Servers
- Virtual Environment

The bottom panel has "Test Optimizer", which is explained later in this section.


Tests Tab




The Tests tab lists information about all tests currently running.

It lists the tests running in this test suite, its status, start time, instances, average time and time left.

Test	Status	Start Time	Instances
complex-ejb...	Ended	Tue Aug 04 ...	
multi-tier-co...	Ended	Tue Aug 04 ...	
multiLevel-c...	Ended	Tue Aug 04 ...	
simple-ejb [s...	Ended	Tue Aug 04 ...	
web-applica...	Ended	Tue Aug 04 ...	
webservices...	Ended	Tue Aug 04 ...	

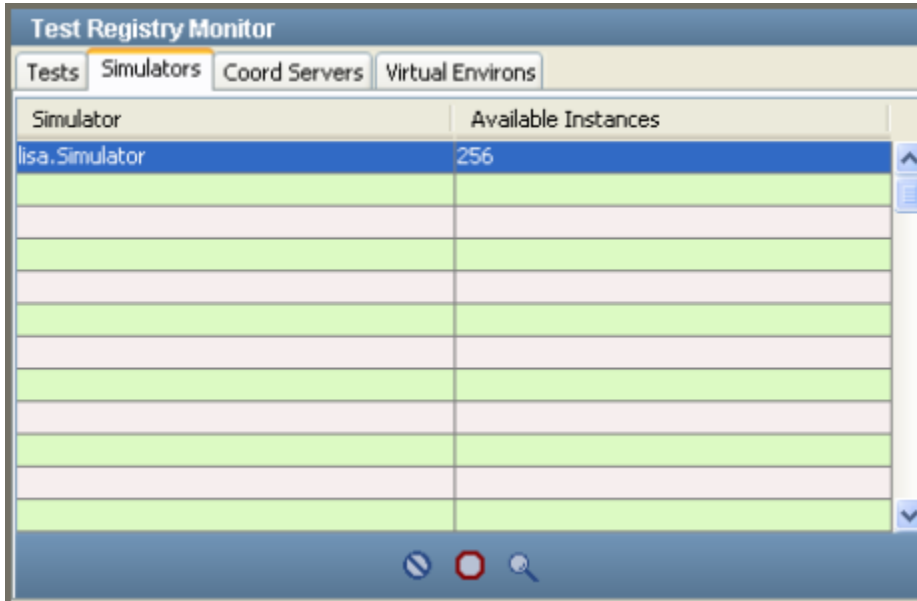
The Toolbar at the bottom of the tab has four icons. These perform actions on a selected test in the list:

- To stop a test, click the Stop  icon.

- To kill a test (stop immediately), click the Kill, , icon.
- To optimize a test, click the Optimize Test, , icon. Test optimization is described below.
- To view a test, click the View Test  icon. The test information is displayed in the Test Monitor.

Simulators tab

The Simulators tab in the Test Monitor allows you to add virtual users in real time to the selected Simulator Server:

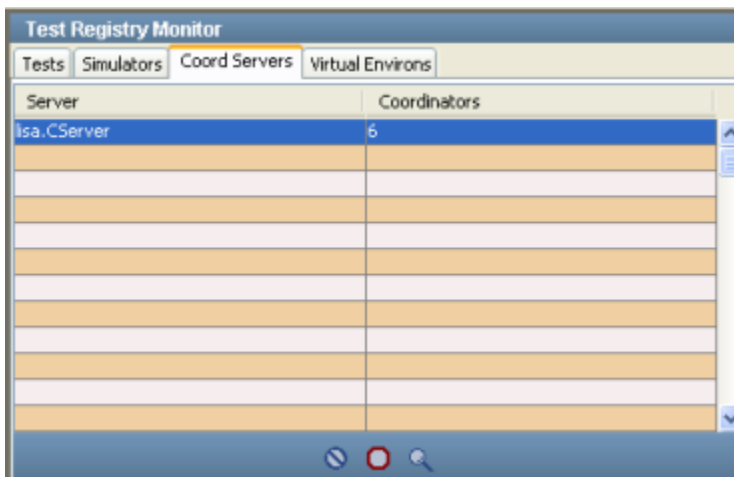


Simulator	Available Instances
lisa.Simulator	256

This will list the Simulator Server and its available instances.



Coordinator Servers Tab


The Coordinator Servers tab lists information about Coordinator Servers that are running:



Server	Coordinators
lisa.CServer	6

The Toolbar at the bottom of the tab has three icons. These perform actions on a selected Coordinator Server:

- To shutdown this service, click the Stop,  icon.
- To reset this service (stop and clear current activities), click the Reset,  icon.

- To view a status message, click the View Status Message  icon.

Virtual Environment Tab

The Virtual Environment tab lists information about the virtual environments if any that are running.

Using the Load Test Optimizer

The Load Test Optimizer allows you to run a simple load test on the system under test. A load test determines how many users the system under test supports.

In the Load Test Optimizer, the system under test is run continuously while more and more simulated users access it, until a specific target is reached, usually a predefined average response time.

The Load Test Optimizer in the LISA Registry Monitor helps determine how many users the system under test supports. An Optimizer can be set on any LISA metric like average response time or a metric pulled from an external source (e.g. SNMP, JMX, Windows Perfmon). It increments the number of users at a preset frequency and informs you when a preset metric threshold is achieved. For example, you can configure the optimizer to increment the number of test instances spawned by the Simulator by five every ten seconds until the average response time hits two seconds.

To configure and start an Optimizer, **select a test from the running tests list**, and click Optimize Test, , icon to open the optimizer for that test.

The Optimizer panel in the LISA Registry Monitor opens with the name of the staging document listed at the top:

The following parameters are listed:

- **Metric:** The metric to use for the optimization. Select from the pull-down list
- **Simulator:** The Simulator used to spawn test instances. Select from the pull-down list
- **Threshold Low:** The metric value at which the optimizer reports that the system will require more virtual users. Enter a numerical value.
- **Threshold High:** The metric value at which the optimizer reports that the system cannot support any more virtual users. Enter a numerical value.
- **Increment (#instances):** The number of additional virtual users to add to the system at the update frequency. Enter a numerical value.
- **Update Frequency (millis):** the number of milliseconds between increments

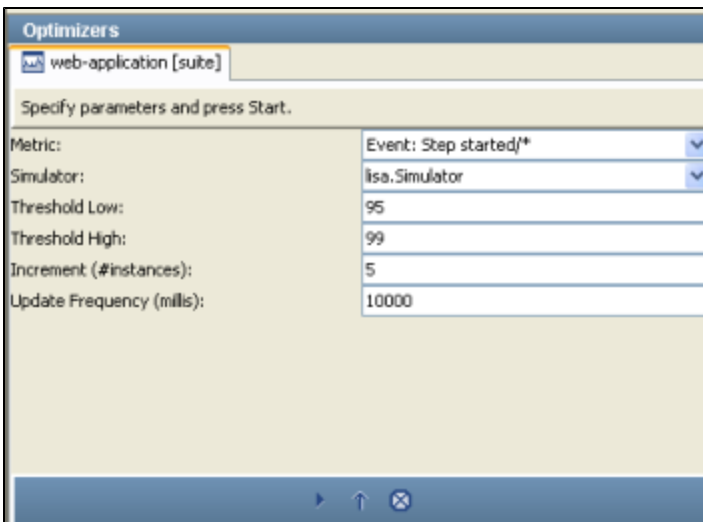


Tip: To optimize the test, you need it in the running stage.




Click Start Optimizer  icon to start the optimizer.

As the optimizer increments the number of virtual users, it displays the number of virtual users on both the Optimizers section and in the Instances column of the Tests tab in the LISA Registry Monitor.



As the optimizer executes, you can change the parameters specified above.

To update the optimizer with the changed information, click the Update,  icon.

To close the optimizer click the Close,  icon.

Delete

15. Building Audit Documents

15. Building Audit Documents

An Audit Document allows you to **set success criteria** for a test, or set of tests in a test suite.

An audit document can track:

- Specific steps that must execute, or must not execute during a test
- Specific events that must occur, or must not occur during a test
- Whether the test takes too little or too much time to complete

The audit document is associated with a particular test, test suite, or directory during the construction of a test suite.

The following topics are available.

[15.1 Creating an Audit Document](#)
[15.2 Sample Audit Document](#)

15.1 Creating an Audit Document

15.1 Creating an Audit Document

To create a new Audit Document,
Select **File->New->Test Audit** from the main menu

Alternatively, if you have an existing Audit Document similar to the one, you are about to create you can open it:
Select **File->Open->Test Audit**

Use this document as the starting point of your new audit document. LISA comes with a standard **built-in Audit Document** that will be used for any test in a test suite that does not specify an audit document of its own. The built-in document can be used without change or, it can be opened and used as the starting point of a new audit document. In this case, remember to save it under a different name. Audit documents are XML documents saved with the suffix **".aud"**.

The LISA built-in audit document can be found on the LISA classpath at: *com.itko.lisa.files.DefaultAudit.aud*.

To configure an Audit document, you need to have a Test Case open in the workstation.

For the purpose of illustration, we open the multi-tier-combo Test Case.

The Audit Document editor is shown below:

Audit Document Name:

Step Audits

Step Name	Must See	Must NOT See	Fail Message	Stop Test
fail	<input type="checkbox"/>	<input checked="" type="checkbox"/>	The test case failed (ex...	<input type="checkbox"/>

Find:

Event Audits

Event ID	Short Desc Contains	Must See	Must NOT See	Fail Message	Stop Test
Cycle started		<input checked="" type="checkbox"/>	<input type="checkbox"/>	The test case neve...	<input type="checkbox"/>
Cycle failed		<input type="checkbox"/>	<input checked="" type="checkbox"/>	Test case failed (rai...	<input type="checkbox"/>
CUSTOM--1		<input type="checkbox"/>	<input checked="" type="checkbox"/>	A error occured tha...	<input type="checkbox"/>

Find:

Run For Audit Info

☐ Audit Run Time

Minimum Time (secs):

Maximum Time (0=ignore):

Failure Message:

The Audit Document editor consists of 3 panels:

- **Step Audits** – audit information for test steps
- **Event Audits** – audit information for events
- **Run For Audit** – audit information for execution time

To configure an audit document, enter the following parameters:

Audit Document Name: The name of the Audit document.

Step Audits Panel

Here, select and specify step(s) that must execute, or must not execute during a test.

Click the **Add** icon from the toolbar, to add a step to the table (see figure above).

Step Name	Must See	Must NOT See	Fail Message	Stop Test
delete user	<input type="checkbox"/>	<input checked="" type="checkbox"/>	The test case failed (ex...	<input type="checkbox"/>
get user	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Failed	<input checked="" type="checkbox"/>
list user	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Failed	<input checked="" type="checkbox"/>

Note: To see the step list, you must have a Test Case open in the Workstation. Else no steps will be seen.

In the new row, select the **step name** from the pull-down menu in the Step Name column.

The table has following parameters:

- **Must See:** Click this box if the step must execute in the test, or,
- **Must Not See:** Click this box if the step must not execute in the test.
- **Fail Message:** Enter a message to be logged if this step audit failed.
- **Stop Test:** Click this box if the step audit failure should "stop and fail" the whole test.

Add additional rows for each step you want to include in the audit.

You can delete rows using the **Delete** icon, and re-arrange rows with the Move Up and Move Down icons.

Event Audits Panel

Specify any events that must occur, or must not occur during a test:

Event Audits					
Event ID	Short Desc Contains	Must See	Must NOT See	Fail Message	Stop Test
Info message		<input type="checkbox"/>	<input checked="" type="checkbox"/>	Failed	<input checked="" type="checkbox"/>
Cycle runtime error		<input type="checkbox"/>	<input checked="" type="checkbox"/>	Failed	<input checked="" type="checkbox"/>
Info message		<input type="checkbox"/>	<input checked="" type="checkbox"/>	Failed	<input checked="" type="checkbox"/>
Info message					
Model definition error					
Cycle ending					
Cycle runtime error					
Step request					
Call made					
Call result					
Suite started					

Click the **Add** icon from the toolbar, to add a event row to the table (see figure above).

In the new row, select the **Event** from the pull-down menu. This pull-down menu lists all the events available in LISA.

Enter the following parameters:

- **Short Desc Contains:** If you want to Filter events based on the value of the events short description, enter the keyword(s) from the short description in this column.

For more information on Event IDs and Short Descriptions, see the [LISA Reference Guide](#).

- **Must See:** Click this box if the event must occur in the test, or,
- **Must Not See:** Click this box if the event must not occur in the test.
- **Fail Message:** Enter a message to be logged if this event audit failed.
- **Stop Test:** Click this box if the event audit failure should "stop and fail" the whole test.

Add additional rows for each step you want to include in the audit.

You can delete rows using the Delete icon, and re-arrange rows with the Move Up and Move Down icons.

Run for Audit Info Panel

You can specify the run time information here.

Run For Audit Info	
<input checked="" type="checkbox"/> Audit Run Time	
Minimum Time (secs):	<input type="text" value="5"/>
Maximum Time (0=ignore):	<input type="text" value="50"/>
Failure Message:	<input type="text" value="test failed"/>

To audit the test run time enter the following parameters:

- **Audit Run Time:** Check if you want to audit the run time, leave unchecked otherwise
- **Minimum Time (secs):** The minimum time (seconds) the test under audit must run for a successful audit.
- **Maximum Time (0=ignore):** The maximum time (seconds) the test under audit can run and still be considered a successful audit. Enter 0 if there is no maximum time constraint.
- **Failure Message:** The message to be logged if the test failed the time audit.

15.2 Sample Audit Document

15.2 Sample Audit Document

Shown below is a complete sample audit document.

The screenshot shows the 'Audit Doc Editor' window. At the top, the 'Audit Document Name' is 'Default: Audit Doc'. Below this are two tables: 'Step Audits' and 'Event Audits'. Each table has columns for 'Step Name' or 'Event ID', 'Short Desc Contains', 'Must See', 'Must NOT See', 'Fail Message', and 'Stop Test'. The 'Step Audits' table has three rows: 'delete user', 'get user', and 'list user'. The 'Event Audits' table has four rows: 'Cycle started', 'Cycle failed', 'CUSTOM--1', and 'Info message'. Below the tables are navigation buttons and a 'Find:' search box. At the bottom, the 'Run For Audit Info' section includes a checked 'Audit Run Time' checkbox, input fields for 'Minimum Time (secs):' (5), 'Maximum Time (0=ignore):' (50), and a 'Failure Message:' field containing 'test failed'.

Step Name	Must See	Must NOT See	Fail Message	Stop Test
delete user	<input type="checkbox"/>	<input checked="" type="checkbox"/>	The test case failed (ex...	<input type="checkbox"/>
get user	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Failed	<input checked="" type="checkbox"/>
list user	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Failed	<input checked="" type="checkbox"/>

Event ID	Short Desc Contains	Must See	Must NOT See	Fail Message	Stop Test
Cycle started		<input checked="" type="checkbox"/>	<input type="checkbox"/>	The test case neve...	<input type="checkbox"/>
Cycle failed		<input type="checkbox"/>	<input checked="" type="checkbox"/>	Test case failed (ra...	<input type="checkbox"/>
CUSTOM--1		<input type="checkbox"/>	<input checked="" type="checkbox"/>	A error occurred th...	<input type="checkbox"/>
Info message		<input type="checkbox"/>	<input checked="" type="checkbox"/>	Failed	<input checked="" type="checkbox"/>

Run For Audit Info

☒ Audit Run Time

Minimum Time (secs):

Maximum Time (0=ignore):

Failure Message:

16. Applying Metrics

16. Applying Metrics

LISA has a wealth of features related to the generation and capture of data for the purpose of reporting results to its users.

Metric collection, LISA's own metric calculation method is an extensible reporting mechanism, and the ability to generate a variety of reports to a variety of outputs.

Metrics allow you to apply **quantitative methods** and **measurements** to the performance and functional aspects of your tests, and the "system under test".

The software "Metric" is a measure of some property of a piece of a software, a hardware system, or their specifications. Quantitative methods using metrics have proved to be very powerful in several areas of computing, and testing is no exception!

LISA provides Metrics that fall into two broad groups: **Gauge** and **Counter**

- **Gauge** – A Gauge provides an instantaneous reading of a value, such as response time or CPU utilization.
- **Counter** – A Counter provides a continuous count of a property, such as the number of failed tests.

For most metrics in LISA, the type of metric, gauge or counter, is already known. When a metric could be used as either, you can specify the type you want.

Metrics can be added to the following LISA elements:

- **Quick Tests** – For use in monitoring the test
- **Staging Documents** – For inclusion in reports
- **Test Suite Documents** – For inclusion in reports
- **Test Monitors** – For use in monitoring tests

The procedure to add metrics is same in each case.

The following topics are available.

- [16.1 Types of Metrics](#)
- [16.2 Adding Metrics](#)
- [16.3 Viewing Metrics in Quick Test](#)

16.1 Types of Metrics

16.1 Types of Metrics in LISA

The following topics are available in this chapter.

- 16.1.1 LISA Whole Test Metrics - This metric lists the Summary data (eg. Average response time)
- 16.1.2 LISA Test Event Metrics - This metric lists the Event values and/or counts as events occur
- 16.1.3 JMX Metrics - This is a standard way to collect metrics on Java-based systems
- 16.1.4 SNMP Metrics - This metric is a standard used on several platforms
- 16.1.5 TIBCO Hawk Metrics
- 16.1.6 Windows Perfmon Metrics - This is a standard way to collect metrics on Windows-based platforms
- 16.1.7 Unix Metrics Via SSH - This is a way to collect metrics on Unix command-line.

16.1.1 LISA Whole Test Metrics

16.1.1 LISA Whole Test Metrics

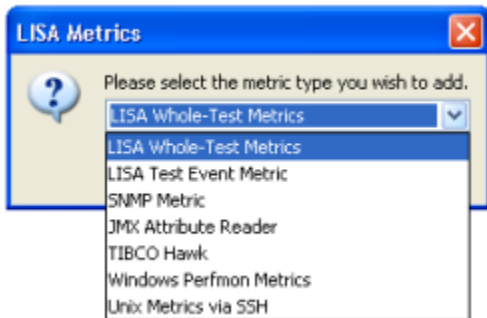
The LISA Whole-Test Metrics, as the name suggests collects all the basic information regarding the test case and hence provides five sub metrics within it.

The Metrics collects basic information such as, the number of virtual users (Instances), and average response times (Avg Resp Time), Minimum Response Time (Min Resp Time), Maximum Response Time (Maximum Resp Time), and Last Response Time (Last Resp Time) and steps per second.

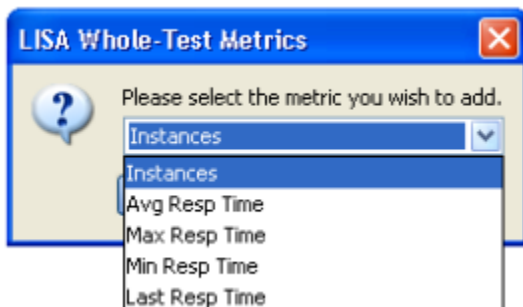
Note - The number of virtual users (Instances), average response times (Avg Resp Time) and Steps Per Second sub metrics are added by default to a staging document's metric list.

To **add** Whole-Test metrics,

- Open a Staging document and click on the Metrics tab.
- Click on the **Add** button in the Metric tab to open the LISA Metrics dialog box as shown below;;
- Select the **LISA Whole-Test Metrics** option from the dialog box:

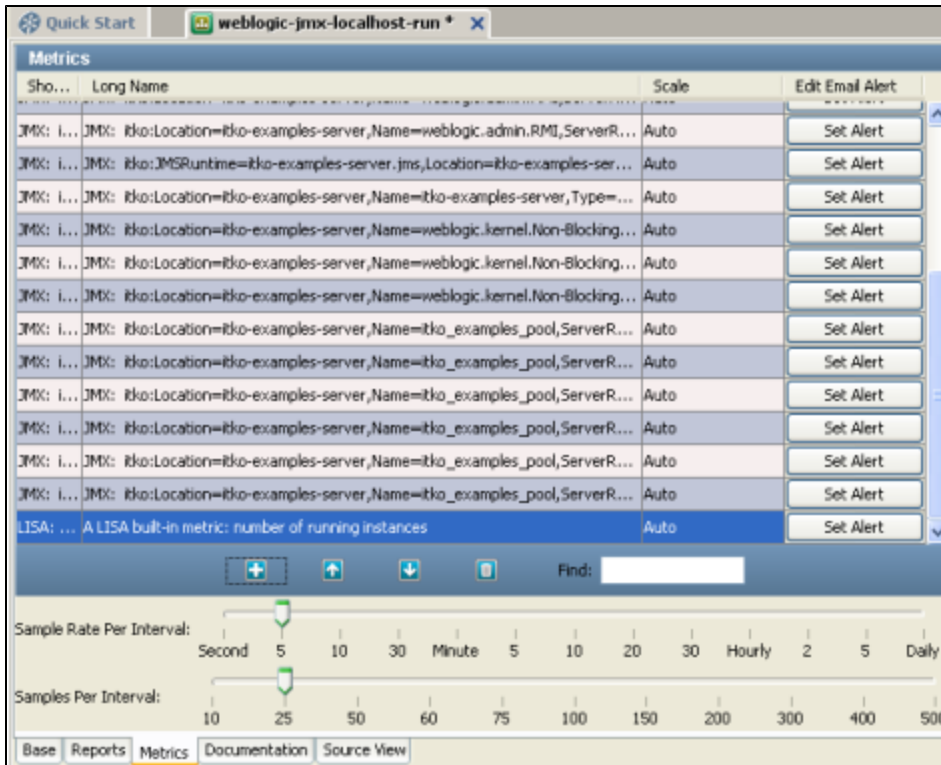


- Click **OK** to display a second dialog box containing the available sub metrics:



- Select the desired sub metric
- Click **OK** to add this metric to the list of metrics.

The newly added Metric is added at the bottom of the list and is highlighted as shown below:



As you can see, a list of Metrics are added in the Metrics list. The one we added is highlighted in the last row.

Repeat until you have added all the desired metrics from this category and then save the Staging document.

16.1.2 LISA Test Event Metrics

16.1.2 LISA Test Event Metrics

The LISA Test Event Metrics provides Metrics based on Events. Hence this Metric has several LISA event sub metrics.

These metrics include both counters and gauges.

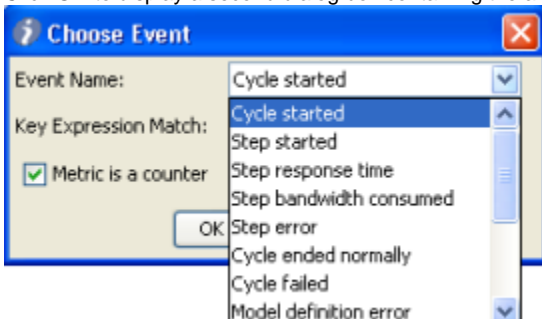
- Typical Counter Metrics that are of use are **Cycle Failed** (EVENT_TESTFAILED), **Step Error** (EVENT_TRANSFAILED), and **Step Started** (EVENT_TRANSACTION).
- Typical Gauge Metrics include **Step Response Time** (EVENT_RESPTIME).

Note: From 5.0 version, LISA will support more intuitive Event ID's. Earlier they used to be mentioned as shown in brackets for example -(EVENT_TESTFAILED), now they are more intuitive like "**Cycle Failed**".

Event metrics allow you to Filter "Events" of a given type by including a regular expression to match the short description field of the event.

To add Test Event metrics,

- Select **LISA Test Event Metrics** from the dialog box.
- Click **OK** to display a second dialog box containing the available sub metrics as shown below:



In addition to selecting the metric, there are certain parameters:

- **Event Name:** Select the desired event name from the drop down list of events.
- **Key Expression Match:** Enter an expression to tell LISA to sample the chosen event only if it has this expression in its short description field. If you leave this blank, or enter *, then every event of this type will be reported.
For example, if you wanted to track the execution time of a given step as its own metric, you would select an event type of EVENT_EXECTIME, and enter the step name in the Key Expression Match field.
- **Metric is a Counter:** If this box is checked, the value counts over time are recorded. If the box is unchecked, the absolute value is recorded (metric is functioning as a gauge).
- Click **OK** to add this metric to the list of metrics.

Repeat until you have added all the desired metrics from this category.

16.1.3 JMX Metrics

16.1.3 JMX Metrics

JMX Metrics

The JMX metrics uses the **Java Management Extension (JMX) API** to provide metrics.

Reference: For information on JMX refer to any of the many external resources on Java JMX.

LISA provides easy set up for the following by providing JMX connectors:

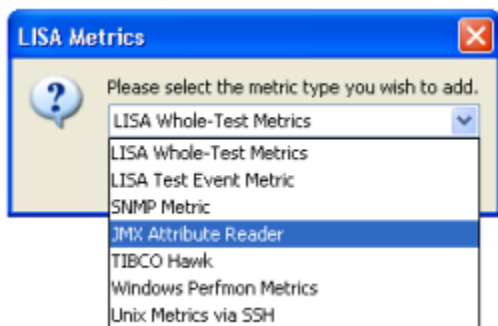
- Any JSR 160 RMI connection
- JBoss 3.2-4.0
- JSE 5 Connector
- Oracle AS (OCJ4)
- Tomcat 5.0.28
- WebLogic 6.1-8.1
- WebLogic 9.x
- WebSphere 5.x
- iTKO JMX Agents

Each of these requires slightly different connection parameters. The values that you require for your particular Server should be available from your Server administrator.

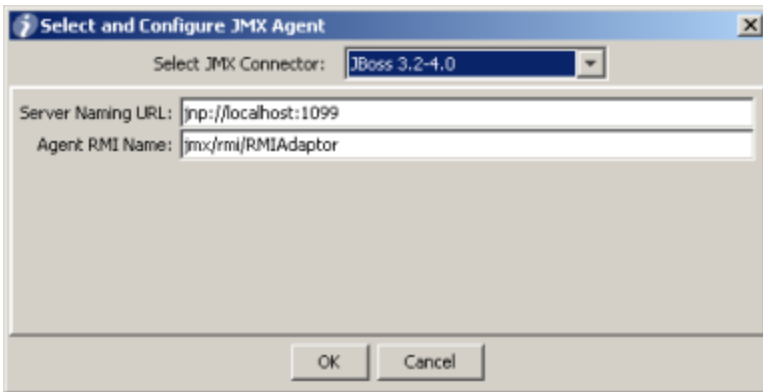
Here we will use JBOSS as our example. There is no agreement on standard metrics, so each provider provides slightly different metrics.

LISA only supports numerical attributes. To use other JMX features, you can invoke them as RMI steps.

To **add** JMX metrics, select **JMX Attribute Reader** from the dialog box:



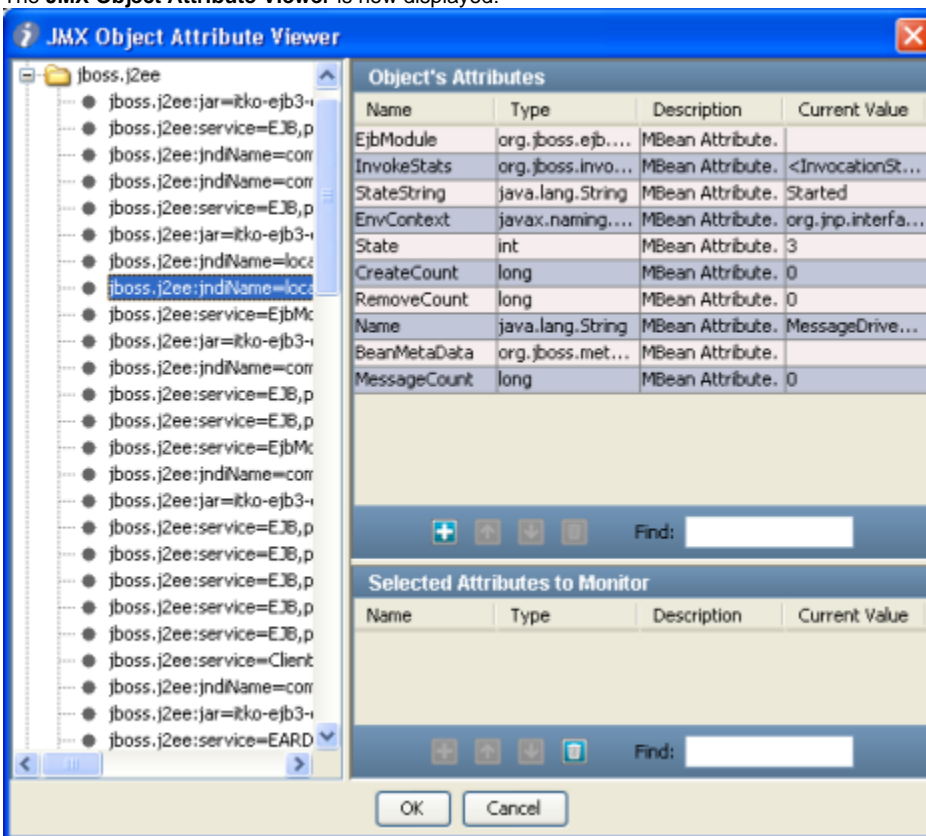
Click **OK** to display a second dialog box, Select and **Configure JMX Agent**, to configure the JMX agent.



Choose the desired connector, **JBoss** in our case, and enter the connection information **Server Naming URL** and **Agent RMI name**, for your particular installation.

Click **OK**.

The **JMX Object Attribute Viewer** is now displayed:



On the left is the **JMX Domain hierarchy** for JBoss. JMX metrics use an object-attribute model where domain objects are an area published by the particular application Server, (for instance 'system'), and attributes are name/value pairs within the object.

When you select an object from this tree, the base attributes of that object are also displayed in the tree. Once you select a base attribute, the rest of the attribute name appears in the list in the top right **Object Attribute panel**.

In the example above, we selected the domain object to be **jboss.system**, the base attribute to be **ServerInfo**, and the rest of the attribute name to be **FreeMemory**. The attributes for ServerInfo are displayed in the **Object Attribute panel**.

To select one of these attributes (metrics) highlight the metric, and click the Add icon, at the bottom of the **Object Attribute panel**. This metric will be added to the list of selected metrics in the bottom panel – **Selected Attributes to Monitor panel**.

To remove an attribute from this panel click the **Delete** icon at the bottom.

Repeat this process until all of your desired metrics appear in your list (bottom panel).

Click **OK** to return to the main metrics panel.

Notice that the JMX metrics we added are now on our list of metrics.

Depending on the application Server, LISA may require vendor-specific jars to enable JMX communication with that application Server. Please visit iTKO forums for specific information.

16.1.4 SNMP Metrics

16.1.4 SNMP Metrics

SNMP Metrics

The SNMP Metrics use the Simple Network Management Protocol (SNMP) to monitor system performance.

Reference: For information on SNMP refer to the following resources:

Windows SNMP Support information on Microsoft Website

Essential SNMP by Douglas R. Mauro and Kevin J. Schmidt

Prerequisite

Setting up SNMP Support

Before you can collect SNMP metrics you must configure SNMP.

For details on setting up SNMP on Unix and Windows see the [LISA Install and Configuration Guide](#).

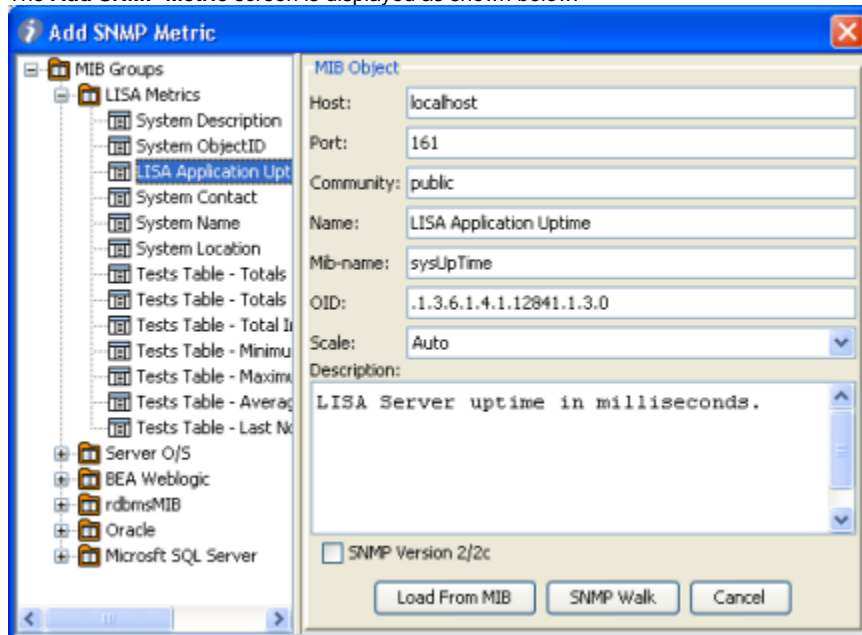
LISA's SNMP Metrics Support - MIB

LISA provides support for SNMP through the ability to monitor SNMP-based metrics.

The SNMP Metrics provide many SNMP metrics.

To add SNMP metrics, select **SNMP Metric** from the dialog box and click **OK**.

The **Add SNMP Metric** screen is displayed as shown below:



There are two panels in this screen:

On the left is a **MIB Groups** tree - **Management Information Bases (MIBS - in SNMP terminology)** that displays all the SNMP metrics that come standard with LISA. The main ones are described below:

LISA provides easy setup for the following by providing **Management Information Bases (MIBS - in SNMP terminology)**. A MIB is a predefined database of a set of metrics on a given domain.

- **Host:** provides system information about a Server hosting a Coordinator Server. Host metrics include **hrProcessorLoad** for CPU utilization and **hrSystemUptime** for the amount of time since this host was last initialized.
- **Server O/S:** provides information related to the system – like up time, date, number of users etc.

- **BEA WebLogic**: provides JDBC, JMS, JVM, socket, servlet and web application information about a Server running BEA WebLogic. BEA WebLogic metrics include **jvmRuntime-HeapFreeCurrent** for the current amount of free memory in the JVM heap in bytes, and **webAppComponentRuntimeOpenSessionsCurrentCount** for the current total number of open sessions in this component.
- **RDBMS**: provides information about a Server running a generic relational database management system. RDBMS metrics include **rdbsSrvInfoPageReads** for the number of physical page reads completed since the RDBMS was last restarted, and **rdbsSrvInfoPageWrites** for the number of single page writes completed since the RDBMS was last restarted.
- **Oracle**: provides information about a Server running Oracle. Oracle metrics include **oraDbSysUserCommits** for the number of user commits, and **oraDbSysUserRollbacks** for the number of times data has rolled back.
- **Microsoft SQL Server**: provides information about a Server running Microsoft SQL Server. Microsoft SQL Server metrics include **mssqlSrvInfoCacheHitRatio** for the percentage of time that a requested data page was found in the data cache (instead of being read from disk), and **mssqlSrvInfoUserConnections** for the number of open user connections.

You can browse through these to select the metric of interest.

When you click the desired metric in the left panel, LISA will fill in the required parameters for this metric in the **MIB Object form** on the right of the panel. A description of the metric will be displayed in the text box. The other information can be either ignored or accepted, if you are not familiar with SNMP.

You must enter the domain name or **IP address** of the host computer (Host) where you are collecting the metrics. Click **OK** to add this metric.

Repeat until you have added all the desired SNMP metrics.

To add SNMP metrics that are **not in the MIB Group tree** you must enter the data (OID) into the **MIB Object form** manually. An OID is the unique identifier of a particular metric, using a tree structured naming scheme. The domain root OID for iTKO is ".1.3.6.1.4.1.12841.1.1", so all SNMP metrics in LISA will start from there.

LISA also provides a **Load From MIB** button to allow you to browse the file system for MIBs, and an **SNMP Walk** button to browse an SNMP tree.

Note: LISA supports all SNMP MIBs. The set of MIBs displayed in the Add SNMP Metric window are only a sample of those supported by LISA. The Add SNMP Metric window makes it easier to understand the Object IDs (OIDs) in those MIBs. However, LISA works with any valid OID, not only those displayed in the Add SNMP Metric window. LISA provides a set of all the standard MIBs from IETF and IANA. LISA stores those MIBs in the \snmp\ietf, and \snmp\iana directories in the LISA install directory.

16.1.5 TIBCO Hawk Metrics

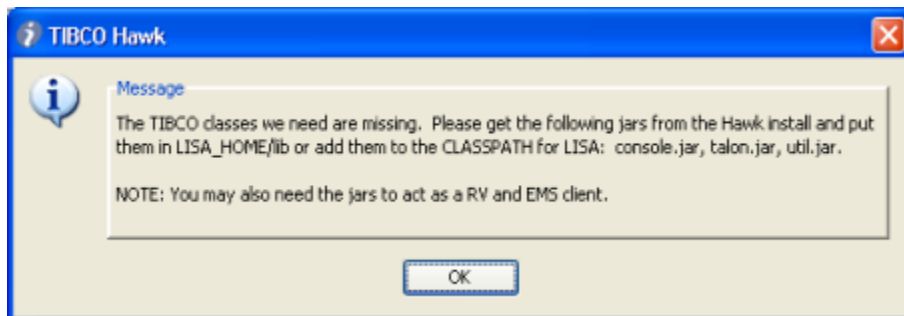
16.1.5 TIBCO Hawk Metrics

TIBCO Hawk Metrics

TIBCO Hawk® is a sophisticated tool for monitoring and managing distributed applications and systems throughout the enterprise.

Prerequisite

You need to copy TIBCO classes (jars) in to the LISA home/lib directory to start this metric. Else you get an error as below:



16.1.6 Windows Perfmon Metrics

16.1.6 Windows Perfmon Metrics

The **Perfmon Metrics** use **Microsoft Windows Perfmon** to provide metrics to monitor system performance on a Windows platform. These metrics are similar to the SNMP metrics that are discussed above.

Reference: Microsoft's web site has more information on Perfmon.

Prerequisite

Setting up Perform

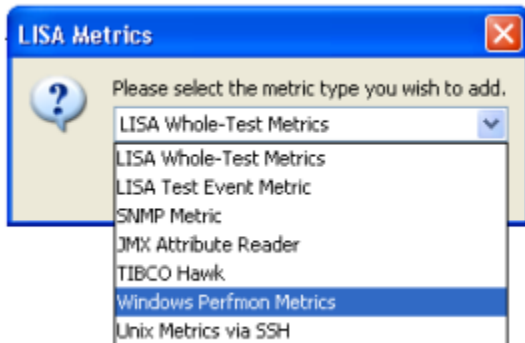
Before you can collect Perfmon metrics you must configure Perfmon on your Windows machine.

Tip: You must have version 1.1 of the Microsoft .NET framework installed and you need to run the **setup-wperfmon.bat** file located in the LISA_HOME/bin directory.

For more details on setting up Perfmon see the [Installation Guide - 4.1 Installing Perfmom](#).

The Windows Perfmon Metrics provides many metrics.

To add Perfmon metrics, select **Windows Perfmon Metrics** from the dialog box and click **OK**.

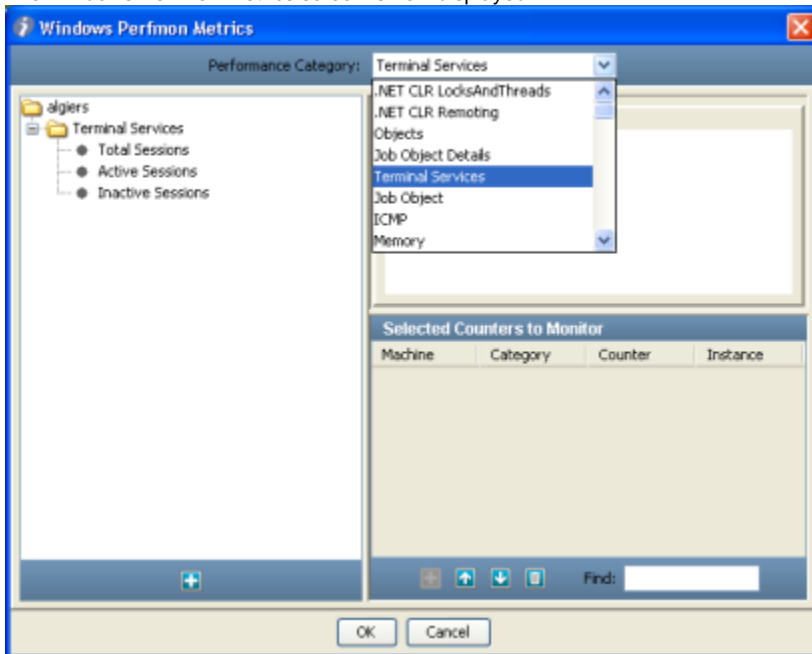


The Windows Perfmon Machine Name dialog box is displayed.

- **Machine Name** - Enter the **Machine Name** of your Windows installation. This can be found by right-clicking My Computer, selecting Properties, and then going to the "Computer Name" tab.
- **User Name** - Enter the user name of the remote machine
- **Password** - Enter the password of the remote machine
- **Domain** - Enter the domain name

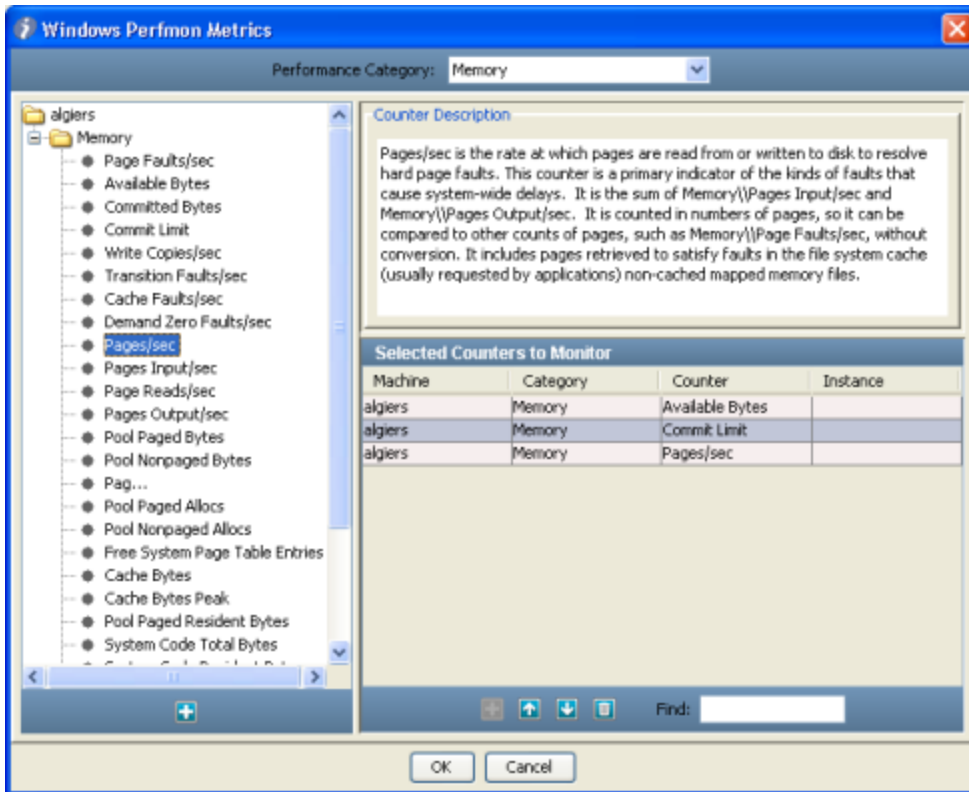
Click **OK** to proceed.

The Windows Perfmon Metrics screen is now displayed:



The Perfmon Metric window has three panels.

Left panel will display the "**Selected Performance**" Category. Top Right panel will display the description of the highlighted category in the "**Counter Description**" section. The Bottom Right panel will list the metric that is highlighted in the left panel.



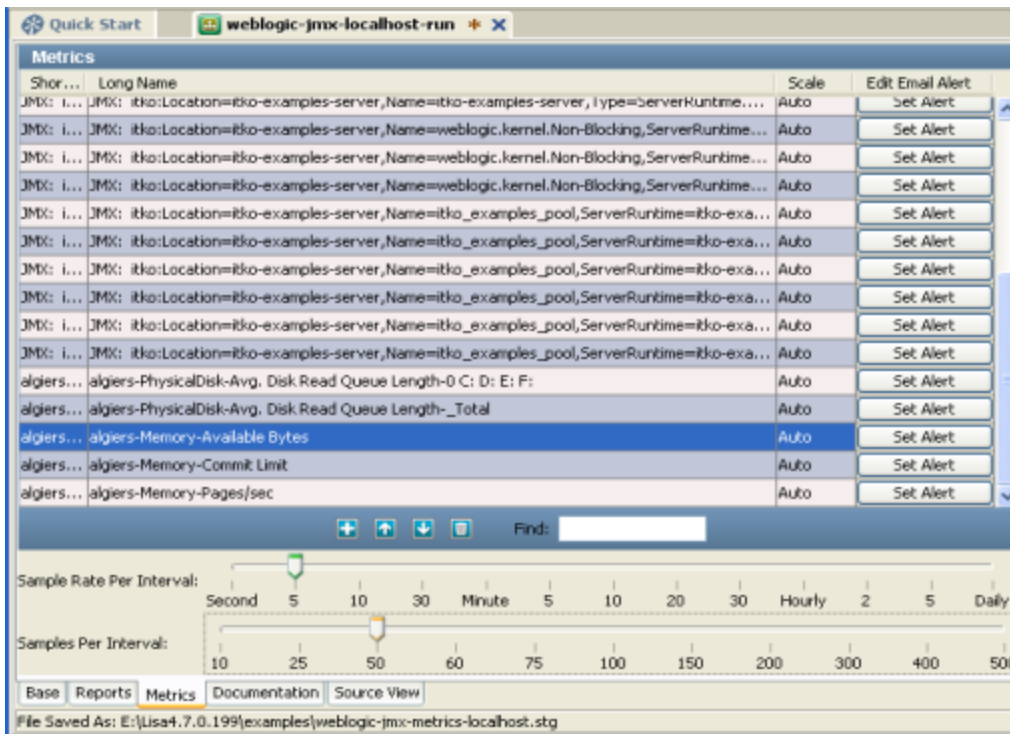
To select a particular metric:

- **Performance Category:** Select a performance category from the pull-down menu. This lists various categories which can be monitored. To name a few like:
 - .NET CLR Remoting/ LocksandThreads
 - .NET CLR Data/Networking
 - Job Objects/Job Object Details
 - Performance/RSVP Service
 - Memory/Print Queue
 - ICMP/Process
 - Outlook/Logical disk
 - IP/Server/Cache
 - And many more.....
- Once you **select** the category, it will be added to the Left panel*.*
- **Double-click** the desired metric in the left panel, to add it to the **Selected Counters to Monitor** table on the bottom right panel.
- **Repeat** until you have added all the desired Perfmon metrics.

Click **OK**.

As shown in the sample below: we have added the Performance Category as "Memory", and selected Available Bytes, Commit Limit and Pages/sec metrics.

Click **OK** to add these to the Metric list on the main page as shown below:



You can also set an **email alert** for this metric, by clicking on the **Set Alert** button.

The 'Edit Alert' dialog box is shown, allowing configuration of an email alert. It includes fields for 'Acceptable Low Value', 'Acceptable High Value', 'SMTP Username', 'SMTP Password', 'SMTP Port Number', 'Email Subject Includes', 'Email Msg Includes', and 'Email Server Name'. There is also a section for 'Email Recipient List' with a table of email addresses and a 'Send Test Mail' button.

Edit Alert

☒ Enable Alert

Acceptable Low Value: 0.0

Acceptable High Value: 0.0

☐ Authenticate

SMTP Username:

SMTP Password:

SMTP Port Number:

Email Subject Includes: Parameter - Available Bytes

Email Msg Includes: Metrics

Email Server Name: localhost

Email Recipient List

Email Address
mailitko@itko.com
gauri.kulkarni@synerzip.com

16.1.7 Unix Metrics Via SSH

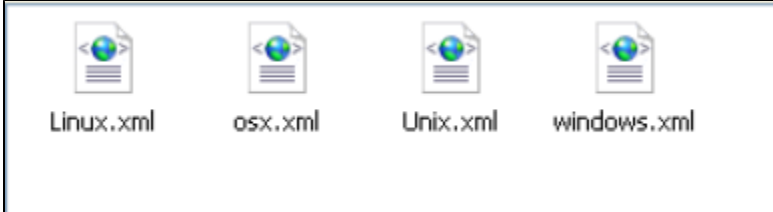
16.1.7 Unix Metrics Via SSH

This metric is used to collect command line type of metrics. This metric will gather inputs like authentication details, host name and the selection of metrics which needs to be collected.

The folder where the metric files are to be placed is mentioned in the `_local.properties` by the key `stats.unix.xml.folder`.

It stores the metric data in an XML file that is stored in the **LISA_HOME/umetrics** directory.

Every file in that directory has to have a unique file name and a unique `<Platform>` tag and can be seen as below:



XML file is used to feed the information to LISA about the command and metrics that needs to be collected on a particular platform. For e.g. to make LISA know the command and metrics on a unix platform, a `unix.xml` file should be placed in `LISA_HOME/metrics` folder.

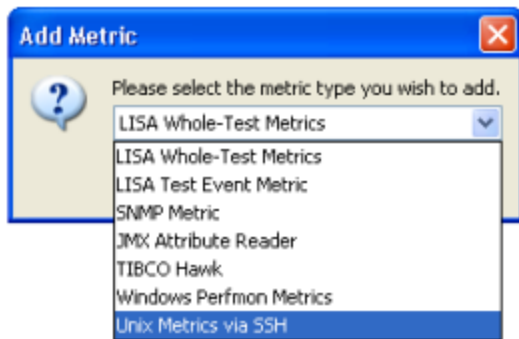
Following is an example of an OSX command parser for "iostat" that collects CPU and disk0 metrics...

```
<UnixMetrics>
  <Platform>OS X</Platform>
  <MetricCollectorSet>
    <Name>iostat</Name>
    <Description>Statistics on the IO for our Unix box</Description>
    <!-- command that will be invoked. Must not require interactive prompts! -->
    <WindowsCommand>cmd ssh john@myserver.itko.com iostat -w 1</WindowsCommand>
    <UnixCommand>ssh john@myserver.itko.com iostat -w 1</UnixCommand>
    <!-- Most commands have a header that we want to know to skip -->
    <IgnoreLineCount>3</IgnoreLineCount>
    <!-- Sometimes headers are repeated, or random break lines appear, so put tokens that warn our
    parser not to consider that line here -->
    <IgnoreTokenList>load,MB</IgnoreTokenList>
    <!-- And here is the metric you want, the assumption is that every "real" line has a value for it -->
    <Metric name="UserCPU" start="38" end="41" decimalPlaces="0" gauge="true">
      Summary of % time spent in user code of across all CPU cores
    </Metric>
    <Metric name="SystemCPU" start="41" end="44" decimalPlaces="0" gauge="true">
      Summary of % time spent in kernel code of across all CPU cores
    </Metric>
    <Metric name="Disk0MBs" start="13" end="19" decimalPlaces="2" gauge="true">
      The first disk MB per second
    </Metric>
    <Metric name="Idle CPU" start="44" end="47" decimalPlaces="0" gauge="true">
      Percent of time CPUs are idle
    </Metric>
  </MetricCollectorSet>
</UnixMetrics>
```

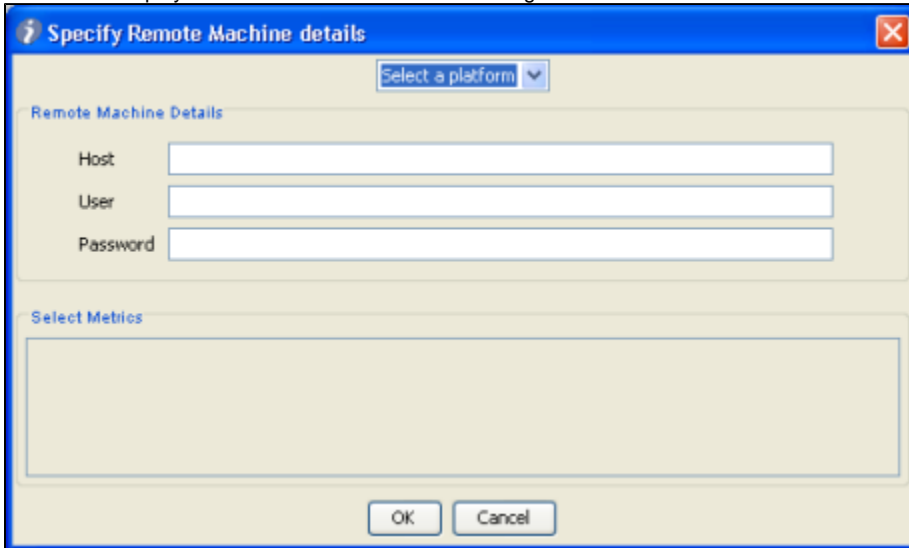
Note: We should store all of the information we need to invoke the command line into the staging document. Do not assume that the XML document that we read at design time will be available at run time.

To **add** this metrics,

- Open the staging document and click on the Metrics tab.
- Select **Unix Metric via SSH Metrics** from the dialog box.

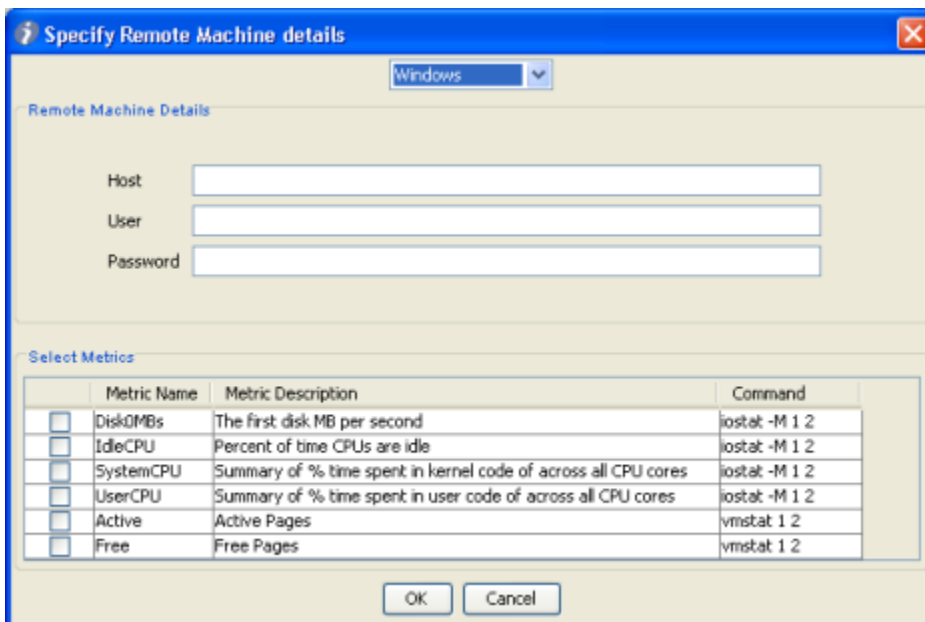


- Click **OK** to display the "Remote Machine Details" dialog box as shown below:



- "Select the Platform"** from the drop down menu and click **OK**.

This opens the "Specify Remote Machines Details" dialog box as shown below:



- Enter the Host name, User name and Password in the Remote machine details dialog box.
- Check the box next to the Metric category to be added and click **OK** to add this metric in the Staging document.

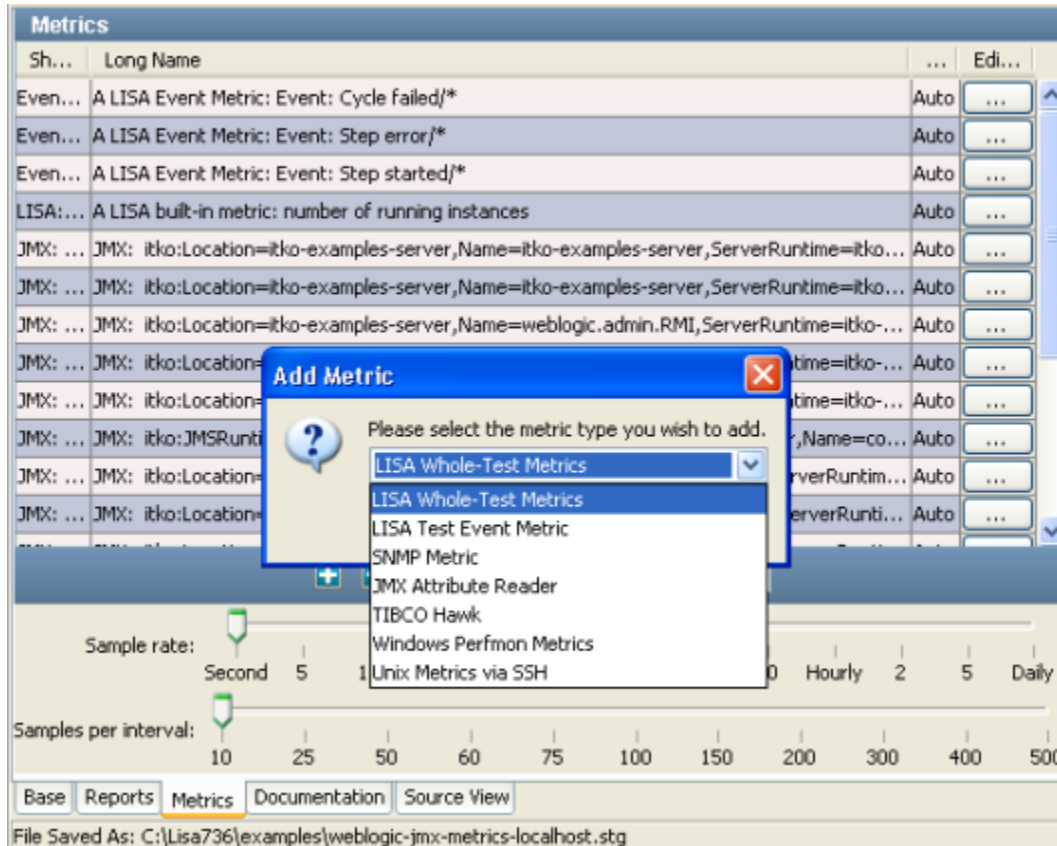
Click on the Staging document to check that this metric is added.

16.2 Adding Metrics

16.2 Adding Metrics

For the purpose of illustration, we have used a Staging document **weblogic-jmx-localhost-run.stg** which is in the LISA examples directory.

- Open the Staging document **weblogic-jmx-localhost-run.stg**.
- Click the **Metrics** tab and Click the **Add** icon in the Metrics panel of the current element, to display an Add Metrics dialog box:



Once you select the type of metric you wish to add, the rest of the procedure will depend on the type of metric you have chosen.

Adding metrics from each of the types is explained in the next section Types of Metrics.

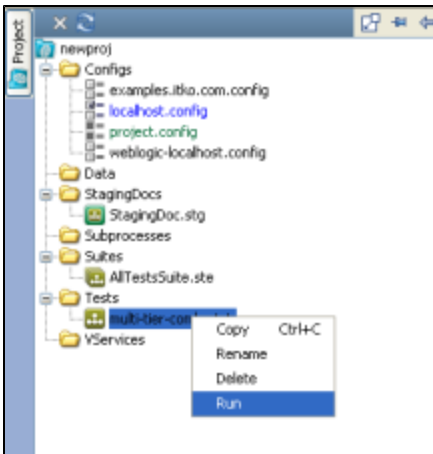
- To **remove** or **reorder** the Metric, use the toolbar at the bottom of the Metric list.

16.3 Viewing Metrics in Quick Test

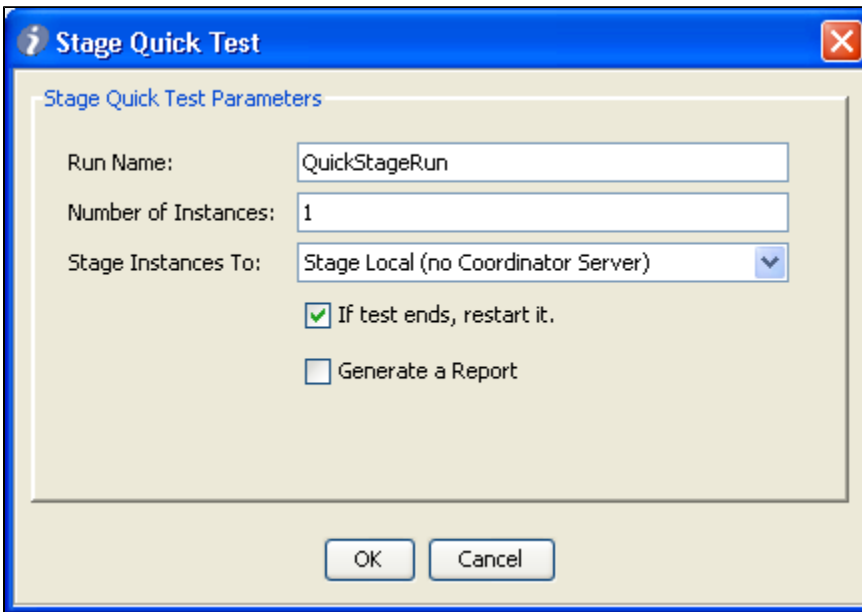
16.3 Viewing Metrics in a Quick Test

To view the added Metrics in a test case, you can run a Quick test within LISA Workstation.

- Select the test case to be run, in the Project tree
- Right click to select **"Run"** as shown below:



The Quick Stage run dialog appears:



- Click OK to run the test case.

When you click **Run**, the test is staged - but not yet running.

You will get a dialog box conveying the same message:



To run the test, click the 'Play' button in the main toolbar when you are ready.

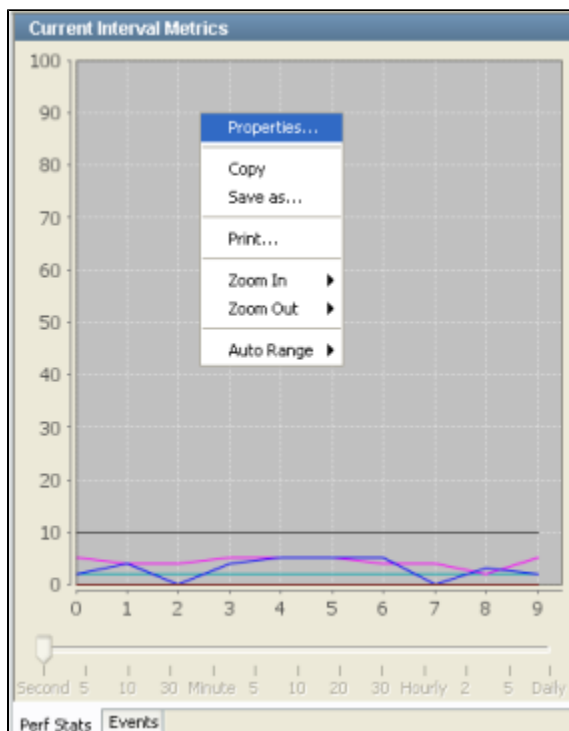
A message appears in the Summary Interval Graphs section confirming the same.

Perf Stats Tab

Current Interval Metrics

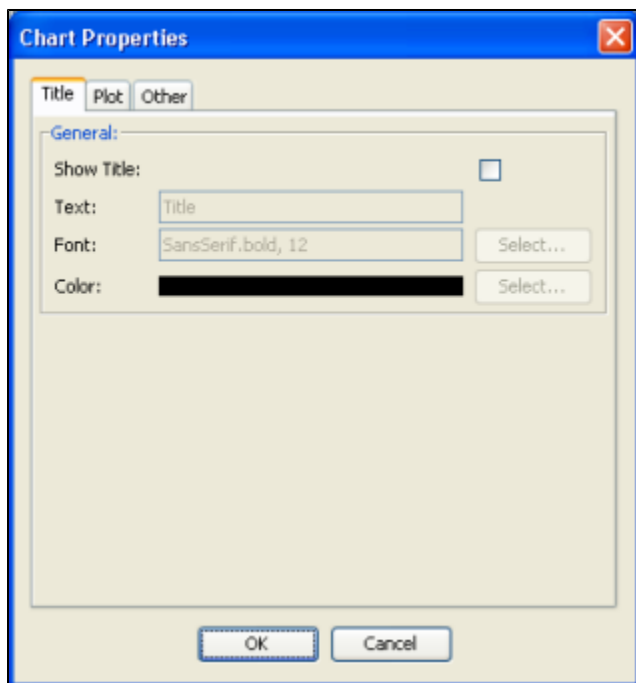
Metrics are defined and collected here.

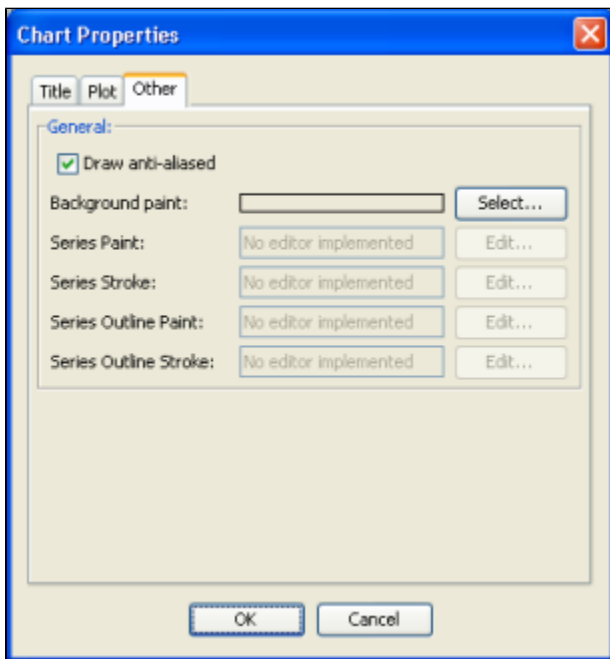
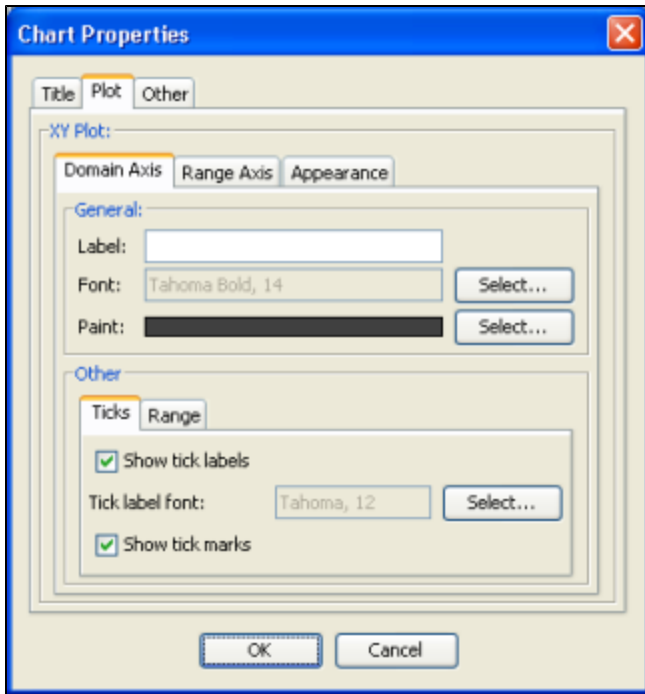
Once the test starts this value can not be changed and hence is disabled.



Right click the graph area to set the Properties of the graph – like line colors, text color & font, labels etc.

This is as shown below:





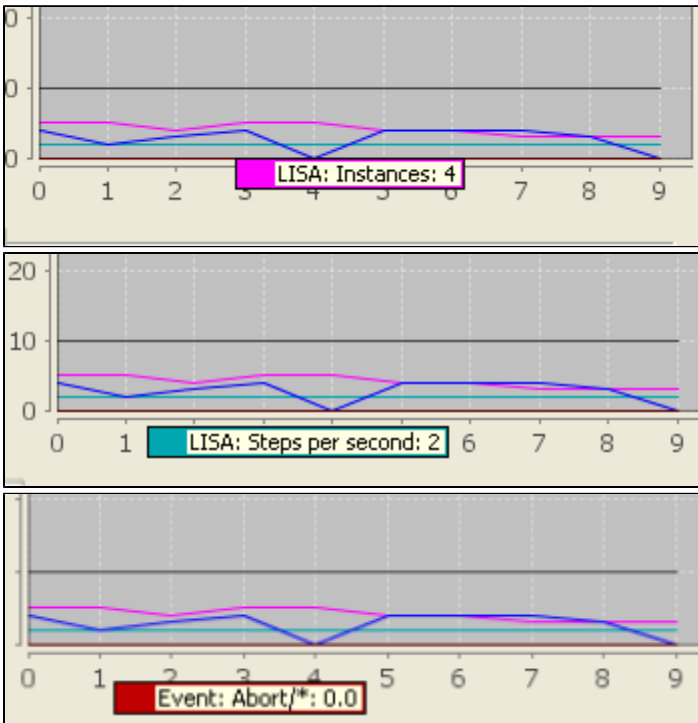
Test Metric Status

The metrics can be added and defined in this area.

Click the **Add** button to add a metric to the existing list. The metrics are identified with different colors as seen below:

Test Metrics Status			
Color	Name	Current Scale	Current Value
Red	Event: Cycle failed/*	Auto (1.0)	0
Red	Event: Step error/*	Auto (1.0)	0
Blue	Event: Step started/*	Auto (1.0)	2940
Red	Event: Abort/*	Auto (1.0)	0
Magenta	LISA: Instances	Auto (1.0)	4
Black	LISA: Avg Resp Tim...	Auto (1.0)	9
Cyan	LISA: Steps per sec...	1	2

The added metrics like LISA Instances, Steps per second, Cycle failed, Aborted steps etc can be seen in the graph with tooltips as shown below:



The steps per second can be seen on the "Steps per second" meter as shown below:



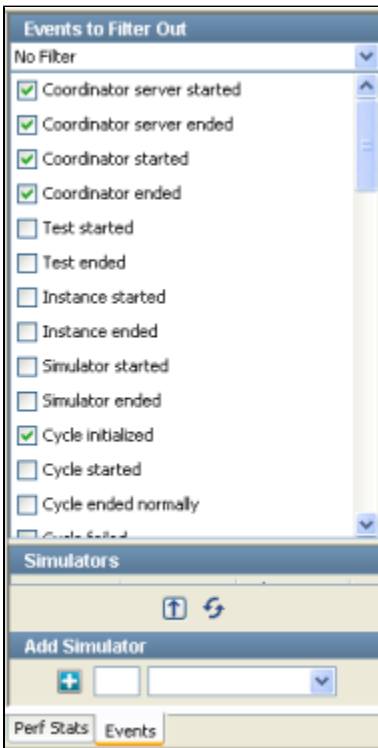
Summary Interval Graphs

How many samples make an Interval? This is graphed here -



Events tab

You can choose the Instances to be filtered out in the **Events tab**



So only the unchecked ones will be seen in the graph shown earlier.

For More information about all these, please refer to Starting a Single Test Case.

17. Understanding Events

17. Understanding Events

An Event is a message broadcast from LISA informing any interested parties, including you, that an action has occurred.

LISA creates events for every major action that occurs in a test case.

An Event is created every time a step is executed, a property is set, a response time is reported, a result is returned, a test succeeds or fails and more.

LISA provides you the ability to see every event or to filter out the events that are not of interest.

Events are important to you when you are monitoring tests or analyzing test results.

The following topics are available...

17.1 Events
17.2 Filtering Events
33.3 Adding_Viewing Events in LISA
17.4 LISA Test Events Table

17.1 Events

17.1 Events

An Event is a message broadcast from LISA informing any interested parties, including you, that an action has occurred. LISA creates events for every major action that occurs in a test case.

An Event is created every time a step is executed, a property is set, a response time is reported, a result is returned, a test succeeds or fails and more. LISA provides you the ability to see every event or to filter out the events that are not of interest.

Events are important to you when you are monitoring tests or analyzing test results.

17.2 Filtering Events

17.2 Filtering Events

By default, all the Events are generated during a Test Case/Suite run.

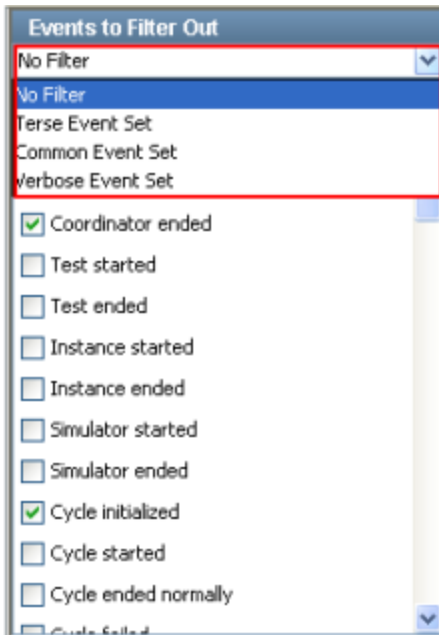
You can view all these Events in the Event tab post execution in the [Test Monitor](#) window.

The Events tab, in the Test Monitor Window, shows a list of Events generated in LISA.

Events to Filter Out		Test Events					
Terse Event Set		Timestamp	Event	Simulator	Instance	Short Info	Long Info
<input checked="" type="checkbox"/>	Coordinator server started	2010-12-17 06:30:45,296	Cycle ending	local	2/0	61346137326...	Signaled to st...
<input checked="" type="checkbox"/>	Coordinator server ended	2010-12-17 06:30:44,734	Step response bandwidth	local	3/1	Delete User	494
<input checked="" type="checkbox"/>	Coordinator started	2010-12-17 06:30:44,718	Step request bandwidth	local	3/1	Delete User	627
<input checked="" type="checkbox"/>	Coordinator ended	2010-12-17 06:30:44,656	Step response bandwidth	local	2/0	Delete User	494
<input type="checkbox"/>	Test started	2010-12-17 06:30:44,656	Step request bandwidth	local	2/0	Delete User	627
<input type="checkbox"/>	Test ended	2010-12-17 06:29:11,484	Step response bandwidth	local	3/1	Add User	1237
<input checked="" type="checkbox"/>	Instance started	2010-12-17 06:29:11,484	Step request bandwidth	local	3/1	Add User	1226
<input checked="" type="checkbox"/>	Instance ended	2010-12-17 06:29:07,390	Cycle started	local	3/1	32616464376...	
<input checked="" type="checkbox"/>	Simulator started	2010-12-17 06:29:07,390	Cycle ending	local	3/0	36633234623...	Signaled to st...
<input checked="" type="checkbox"/>	Simulator ended	2010-12-17 06:29:07,359	Abort	local	3/0	36633234623...	<?xml version=...
		2010-12-17 06:29:07,187	Step error	local	3/0	Get User	HomelessEJBN...
		2010-12-17 06:29:04,312	Step response bandwidth	local	2/0	Add User	1224
		2010-12-17 06:29:04,312	Step response bandwidth	local	4/0	Add User	1227
		2010-12-17 06:29:04,250	Step response bandwidth	local	3/0	Add User	1230
		2010-12-17 06:29:04,250	Step response bandwidth	local	1/0	Add User	1241
		2010-12-17 06:29:04,250	Step response bandwidth	local	0/0	Add User	1225
		2010-12-17 06:29:04,250	Step request bandwidth	local	3/0	Add User	1220
		2010-12-17 06:29:04,234	Step request bandwidth	local	2/0	Add User	1214
		2010-12-17 06:29:04,234	Step request bandwidth	local	4/0	Add User	1215
		2010-12-17 06:29:04,234	Step request bandwidth	local	1/0	Add User	1232
		2010-12-17 06:29:04,234	Step request bandwidth	local	0/0	Add User	1214
		2010-12-17 06:28:53,234	Cycle started	local	1/0	66656263613...	
		2010-12-17 06:28:53,234	Cycle started	local	2/0	61346137326...	
		2010-12-17 06:28:53,234	Cycle started	local	4/0	37353035376...	
		2010-12-17 06:28:53,234	Cycle started	local	0/0	32653862653...	
		2010-12-17 06:28:53,234	Cycle started	local	3/0	36633234623...	
		2010-12-17 06:28:53,187	Test started	N/A	0/0	62303033646...	N/A

Test Events panel lists the events as they are emitted, with the most recent on the top of the list. You can list events in real time by checking the **Auto Refresh** box at the bottom of the panel, or you can refresh the list manually by clicking the Refresh icon, with the Auto Refresh box unchecked.

You can filter out these Events, by clicking on the "Events to Filter Out" drop down box as shown below:



You can filter out Events in two ways --

Manually – By checking individual Events to be filtered out.

You can choose "**No Filter**" in the drop down and then select the desired Events from the list to filter out individually.

Event Set – LISA provides certain Event "**sets**" by default. Each Event set has certain Events added/ excluded in that set.

For convenience there are 3 standard sets of Events listed in the pull-down menu.

- Terse Event Set
- Common Event Set
- Verbose Event Set

These sets provide a pre-selected list of events that you can use as a starting list. You can then customize the list as and when needed.

17.3 Adding_Viewing Events in LISA

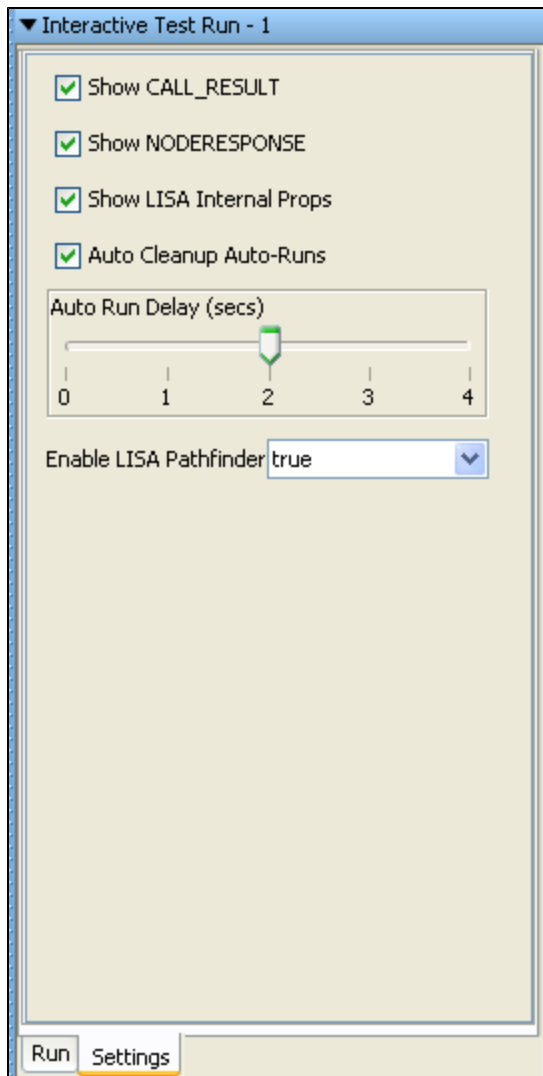
17.3 Adding_Viewing Events in LISA

You can add Events in LISA:

Through ITR and Through Quick Test/Staging Document:

Through ITR

You can enable some Events in the ITR in the Settings tab as shown below:



Check "**Show CALL_RESULT**" and "**Show NODERESPONSE**" check boxes to view those events in the ITR Events Tab.

Through Quick Test/Staging Document

When you start a Test case execution through either Start Test option or Start Suite option, the Test will be staged and relevant Graphs will appear in the Perf Stats Tab.

To view the Test Events, click on the **Events** Tab.

You can select the **Events to Filter out** in the Left panel or choose from the already defined Filter sets: Terse Event Set, Common Event Set or Verbose Event Set.

Check the "**Auto Refresh**" check box in the **Test Events** panel to refresh the Test Events list.

Events to Filter Out

Terse Event Set

Coordinator server started

Coordinator server ended

Coordinator started

Coordinator ended

Test started

Test ended

Instance started

Instance ended

Simulator started

Simulator ended

Simulators

Name	Instances	Change
local	5	

+

+

Add Simulator

+

Perf Stats

Events

Failures(1)

Test Events

Timestamp	Event	Simulator	Instance	Short Info	Long Info
2010-12-17 06:30:45,296	Cycle ending	local	2/0	61346137326...	Signaled to st...
2010-12-17 06:30:44,734	Step response bandwidth	local	3/1	Delete User	494
2010-12-17 06:30:44,718	Step request bandwidth	local	3/1	Delete User	627
2010-12-17 06:30:44,656	Step response bandwidth	local	2/0	Delete User	494
2010-12-17 06:30:44,656	Step request bandwidth	local	2/0	Delete User	627
2010-12-17 06:29:11,484	Step response bandwidth	local	3/1	Add User	1237
2010-12-17 06:29:11,484	Step request bandwidth	local	3/1	Add User	1226
2010-12-17 06:29:07,390	Cycle started	local	3/1	32616464376...	
2010-12-17 06:29:07,390	Cycle ending	local	3/0	36633234623...	Signaled to st...
2010-12-17 06:29:07,359	Abort	local	3/0	36633234623...	<?xml version=...
2010-12-17 06:29:07,187	Step error	local	3/0	Get User	HomelessEJB.N...
2010-12-17 06:29:04,312	Step response bandwidth	local	2/0	Add User	1224
2010-12-17 06:29:04,312	Step response bandwidth	local	4/0	Add User	1227
2010-12-17 06:29:04,250	Step response bandwidth	local	3/0	Add User	1230
2010-12-17 06:29:04,250	Step response bandwidth	local	1/0	Add User	1241
2010-12-17 06:29:04,250	Step response bandwidth	local	0/0	Add User	1225
2010-12-17 06:29:04,250	Step request bandwidth	local	3/0	Add User	1220
2010-12-17 06:29:04,234	Step request bandwidth	local	2/0	Add User	1214
2010-12-17 06:29:04,234	Step request bandwidth	local	4/0	Add User	1215
2010-12-17 06:29:04,234	Step request bandwidth	local	1/0	Add User	1232
2010-12-17 06:29:04,234	Step request bandwidth	local	0/0	Add User	1214
2010-12-17 06:28:53,234	Cycle started	local	1/0	66656263613...	
2010-12-17 06:28:53,234	Cycle started	local	2/0	61346137326...	
2010-12-17 06:28:53,234	Cycle started	local	4/0	37353035376...	
2010-12-17 06:28:53,234	Cycle started	local	0/0	32653862653...	
2010-12-17 06:28:53,234	Cycle started	local	3/0	36633234623...	
2010-12-17 06:28:53,187	Test started	N/A	0/0	62303033646...	N/A

Auto Refresh

+

View Events in ITR

You can view test events in **Interactive Test Run (ITR)**, by clicking the Test Events Tab, as shown below:

Response

Properties

Test Events

EventID	Timestamp	Short	Long
Property set	Thu Nov 05 09:44:14 IST ...	lisa.hidden.LisaFilterBase.processedO...	false
Property set	Thu Nov 05 09:44:14 IST ...	lisa.hidden.lisaint.support	true
Abort	Thu Nov 05 09:44:14 IST ...	33376230383439302D616535372D3466	<?xml vers...
Step started	Thu Nov 05 09:44:14 IST ...	delete user	
Property set	Thu Nov 05 09:44:14 IST ...	lisa.hidden.LisaFilterBase.processedO...	false
Property set	Thu Nov 05 09:44:14 IST ...	lisa.hidden.lisaint.support	true
Info message	Thu Nov 05 09:44:14 IST ...	delete user	com.itko.lis...
Step error	Thu Nov 05 09:44:14 IST ...	Error building URL to Web Service	=====...
Step response t...	Thu Nov 05 09:44:14 IST ...	delete user	31
Step history	Thu Nov 05 09:44:14 IST ...	delete user	<?xml vers...
Cycle ending	Thu Nov 05 09:44:14 IST ...	33376230383439302D616535372D3466	Signaled to ...
Cycle history	Thu Nov 05 09:44:14 IST ...	33376230383439302D616535372D3466	<?xml vers...
Property set	Thu Nov 05 09:44:14 IST ...	lisa.http.obj	<removed>

View Events in Quick Stage Run

You can observe and filter events in a **Quick Test Run** by checking the "Auto Refresh" check box, as shown below:

For more information on LISA Reports, see Chapter [Reports](#) .
For more information on LISA Metrics, see Chapter [Metrics](#) .

You can also have a Functional Test Report showing an EVENT_NODEMSG event being reported.

Note: Internal to LISA, a step may also be referred to as a node, explaining why some events have node in the EventID.

An event is characterized by an EventID, a Short value, and a Long value. EventIDs are keywords that indicate the type of event. The Short values and Long values contain information about the event; their contents will vary with the type of event.

The table below displays, an Event ID for each test event type, a description of the event, and an explanation of the Short and Long values.

The events are listed in the same order as they appear in the Event lists in LISA. An example of the event list can be seen in one of the figures above.

17.4 LISA Test Events Table

17.4 LISA Test Events Table

No.	Events/Event ID Event Description	Short Name	Long Name	
1	Coordinator server started EVENT_COORDSERVERCREATED	The Coordinator Server was created.	The name of the Coordinator Server	
2	Coordinator server ended EVENT_REMOVECOORDSERVER	The Coordinator Server was ended	The name of the Coordinator Server	
3	Coordinator started EVENT_NEWCOORDINATOR	A new Coordinator as given in the short description was created	The name of the Coordinator Server	
4	Coordinator ended EVENT_REMOVECOORDINATOR	The Coordinator given in the short description was removed	The name of the Coordinator Server	
5	Test started EVENT_TESTSTARTED	Sent when the Coordinator starts the test	The name of the test case	
6	Test ended EVENT_TESTEND	Sent when the Coordinator stops the test	The name of the test case	
7	Instance started EVENT_INSTANCESTART	Sent when the Simulator creates an instance	The name of the Simulator	
8	Instance ended EVENT_INSTANCEEND	Sent when the instance in the Simulator has finished	The name of the Simulator	
9	Simulator started EVENT_SIMSTARTED	A Simulator started	The name of the Simulator	
10	Simulator ended EVENT_SIMEND	A Simulator ended	The name of the Simulator	
11	Cycle initialized EVENT_INIT	Sent when the test has been initialized (loaded/staged)	The name of the test case	
12	Cycle started EVENT_START	Indicates that the test instance and cycle has started	The name of the test case	
13	Cycle ended normally EVENT_NORMALEND	Indicates that the test execution completed in a successful state	The name of the test case	
14	Cycle failed EVENT_ABEND	Indicates that the test execution failed. Should mean that there are no exceptions in the test case, but either logic errors in the test case or in the system under test caused the test case not to complete as expected.	The name of the test case	
15	Property set EVENT_SETPROP	A LISA property was set	Name of the property	Value of the property as a string
16	Step Started EVENT_TRANSACTION	A step is being executed (transaction is a synonym for step)	The name of the step	

17	Assertion fired EVENT_ASSERT	An assertion on a step 'fired'	The name of the assertion	The log message of the assertion, or a LISA-generated message if there is no log message set
18	Step response EVENT_NODERESPONSE	Indicates that we have completed a transaction against the system under test.	The name of the step	The response data as a string
19	Step response time EVENT_EXEETIME	Records the amount of time a transaction took to execute against the system under test.	The name of the step	The number of milliseconds to execute the step
20	Step bandwidth consumed EVENT_BANDWIDTH	Approximate amount of data sent and received from the system under test for the step execution	The name of the step	Actual number of bytes read/received
21	Step error EVENT_ERROR	An error has occurred in the system under test. For example, no response from a web server. This is on a per step basis. The EVENT_TESTFAILED refers to the complete test case.	The name of the step	If available, a message to help determine the cause of the failure
22	Log Message EVENT_LOGMSG	Basic logging data that LISA considers valuable to record, but can be turned off, when filtering events, to minimize overhead	The message sent to the log	
23	Info message EVENT_NODEMSG	Basic logging data that LISA considers valuable to record. For example, the HTTP/HTML request step will send this message with the step name in the short field and the URL being sent to the server in the long field	Usually the name of the step during which this message was generated	Usually a message that LISA generated
24	Model definition error EVENT_TESTDEFERROR	A test case error was discovered during execution of the test. For example, the name is constructed from a property that does not exist	Varies but it is usually the name of the test element (i.e. step, data set, or filter) that has the error	Usually a message that explains the error
25	Cycle ending EVENT_STOPTESTSIGNAL	The instances have been instructed to stop testing		
26	Cycle runtime error EVENT_TESTRUNERROR	A LISA error occurred. For example, the Coordinator lost its connection to a Simulator while running a test	Varies but it is usually the name of the test element (i.e. step, data set, or filter) that has the error	Usually a message that explains the error
27	Step request EVENT_REQUEST	Steps that support this event use it to report the actual request made to the system under test	The name of the step	The request data as a string
28	Call made EVENT_CALL	Steps that perform object calls like Web Services or EJBs use this event to report each call that is made on the object	The name of the step	The call as a string, like void addUser([string] Joe, [string] mypass)
29	Call result EVENT_CALLRESULT	Steps that perform object calls like Web Services or EJBs use this event to report the response they get from calls	The name of the step	The call response as a string. If the response is an object, an XML view of the object is presented
30	Suite started EVENT_SUITE_STARTING	A suite is starting to run	The name of the suite	
31	Suite test staged EVENT_SUITE_TESTSTAGED	When a test is staged to run as part of a suite	The name of the suite	The name of the test, path to the test, and other information
31	Suite test passed EVENT_SUITE_TESTPASSED	When a test running as part of a suite ends successfully	The name of the suite	
32	Suite test failed EVENT_SUITE_TESTFAILED	When a test running as part of a suite ends in failure	The name of the suite	

33	Suite test finished EVENT_SUITE_FINISHED	When all the tests running as part of a suite have finished, the suite is completed	The name of the suite	
34	Suite ended EVENT_SUITE_SKIPPED	When a setup test (defined in a suite document) has failed, then the suite will not run the tests defined in the suite. This event informs you that this has happened	The name of the suite	
35	Suite setup/teardown EVENT_SUITE_SETUPTEARDOWN	The suite has a setup or teardown test defined and that test has just started to run	The name of the suite	The name of the test, path to the test, and other information
36	Metric alert EVENT_METRIC_ALERT	A metric has been collected and is reporting its value	The short name of the metric, like LISA: Avg Response Time	The value of the metric collected
37	Pathfinder EVENT_INTEGRATION	The system being tested has LISA integration enabled. This event contains the LISA integration XML data that was captured	The name of the step	The XML representation of the LISA Integration data captured
38	Step request bandwidth EVENT_REQUEST_BANDWIDTH		Step request bandwidth	
39	Step response bandwidth EVENT_RESPONSE_BANDWIDTH		Step response Bandwidth	
40	Subprocess ran EVENT_SUBPROCESS	A subprocess (subtest) was started	A subprocess (subtest) was started	
41	Step Warning EVENT_WARNING	A warning was emitted, like a filter that took the default value b/c it couldn't find the current value	Step warning	
42	VSE server reset EVENT_SERVICE_RESET	A service reset request was made of a server	VSE server resets	
43	VSE server stop EVENT_SERVICE_STOP	A service stop request was made of a server	VSE server Stops	
44	VSE server shutdown EVENT_VS_ENV_SHUTDOWN	A virtual service environment was asked to shut down	VSE Server Shutdown	
45	VS service started EVENT_VIRTUAL_SERVICE_START	A virtual service was started.	VS service started	
46	VS service ended EVENT_VIRTUAL_SERVICE_STOP	A Virtual Service was stopped.	VS Service Stopped	
47	VS transaction match EVENT_VSE_TRANS_MATCH	A virtual service matched a transaction request to at least one recorded response	VS transaction match	
48	VS No transaction match EVENT_VSE_NO_TRANS_MATCH	A virtual service did NOT match a transaction request to at least one recorded response	VS transaction no match	
49	VS log message EVENT_VSE_LOG	VSE internal logging	VS log message	
50	Abort EVENT_ABORT	This event ends a test in a "cannot finish" state. It is a failing type of end event	Event Aborted	
51	Step target EVENT_NODE_TARGET	Every step has a target, like the URL for a web request or the JNDI name of an EJB		
52	Step history EVENT_NODE_HISTORY	Every node has a history event that fires of type com.itko.lisa.test.NodeExecHistory in it's long info.	Step History	
53	Cycle history EVENT_CYCLE_HISTORY	Every model that runs will generate one of these. It's the final trace of all the details of its run.	Cycle History	
54	Suite history EVENT_SUITE_HISTORY	Every suite that runs will generate one of these. It's the final trace of all the details of its run.	Suite History	

55	Assert evaluated EVENT_ASSERT_EVALUATED	Every non-embedded assert will emit either this event if it did not fire, or the EVENT_ASSERT * if it did fire. Firing means it is true and it's consequence will be followed. These * events are the asserts that did not get to execute their consequence	Assert Evaluated	
56	Metric started EVENT_METRIC_START	Metrics that are collected generate real-time events of their values	Metric Started	
57	Metric value EVENT_METRIC_VALUE	Metrics that are collected generate real-time events of their values	Metric value	
58	Data set read EVENT_DATA_SET_READ	Data sets emit an event that makes it clear what values are about to be used	Data Set read	
59	Test not active EVENT_TEST_NOT_ACTIVE	This is emitted when a test is marked inactive in a suite.	Event Test Not Active	This is emitted when a test is marked inactive in a suite.
60	HTTP performance EVENT_HTTP_PERFSTATS	This is emitted for every HTTP transaction LISA executes so we can capture the perf stats	HTTP performance Statistics	This is emitted for every HTTP transaction LISA executes so we can capture the perf stats

PART 4 - Running Test Cases_Suites

PART 4 - Running Test Cases/Suites

Staging and running tests in LISA is relatively straightforward.

Most of the complexity is in the building of the Test Cases (Test Case documents), the collection of the parameters required to communicate with the system under test ([Configurations](#)), and the specification of the instructions on how to perform the test, or tests. ([Staging documents](#) and [Test Suite documents](#)).

Once you have these documents prepared, staging a test simply means telling LISA to stage a test using the appropriate documents. The type of test that you run is defined by the documents you specify to LISA. Often the same Test Cases are used with different staging documents to perform a variety of different tests. That is, a separate Test Case does not have to be written for functional, regression and load testing. Instead, a different staging document is prepared using the same Test Case.

A test suite consists of a group of related tests, and other test suites, to be run in a single test run. Each test in the test suite is configured using its own staging document, audit document, and configuration.

Tests staged with [Test Runner](#) run in a separate Java virtual machine. Test Runner is a headless version of Workstation that allows you to run tests as a batch process. For more information on Test Runner, see the LISA Installation and Configuration Guide.

Tests staged in a LISA Server environment are staged, and monitored with LISA Workstation, but they are managed and run by the LISA Server. A LISA Server can be configured to run on a single CPU, including the one that Workstation is using, or on a large distributed environment incorporating many CPUs. Your testing requirements should dictate the Server architecture to be used. LISA Server installation and configuration is described in the LISA Installation and Configuration Guide.

Tests can be scheduled to run on at regular time intervals using the [CVS Dashboard](#), i.e Continuous Validation Service (CVS). For more details on using the continuous testing service, see CVS Tests run as part of an automatic build process make use of two external Java application: ANT and JUnit.

In this section, the following topics are covered:

- [18. Running a Single Test](#)
- [19. Running a Quick Test](#)
- [20. Running a Test Suite](#)
- [21. Running a Test using ITR](#)

18. Running a Single Test

18. Running a Single Test

Starting a single test in LISA Workstation requires you to run the LISA Workstation.

Within the Workstation, click Start Test, where in you can enter the Coordinator server name (if any), Test case name to be run, the staging document and the configuration to be applied if any.

You can even view the reports at the conclusion of the test in the [Report Viewer](#) .

The following topics are available.

- [18.1 Starting a Single test](#)
- [18.2 Test Monitor Window](#)
- [18.3 Quick Test Toolbar](#)
- [18.4 Example Test Case](#)
- [18.5 View Reports](#)

18.1 Starting a Single test

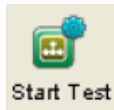
18.1 Starting a Single test

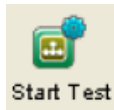
You can start the test case execution by specifying the following documents:

- Test case document - Name of the test case to be executed.
- Staging document - Name of the staging document to be attached.
- Coordinator server - Name of the coordinator server

Specifying a Configuration is optional while running a test case. If not specified, it will apply the Default configuration.

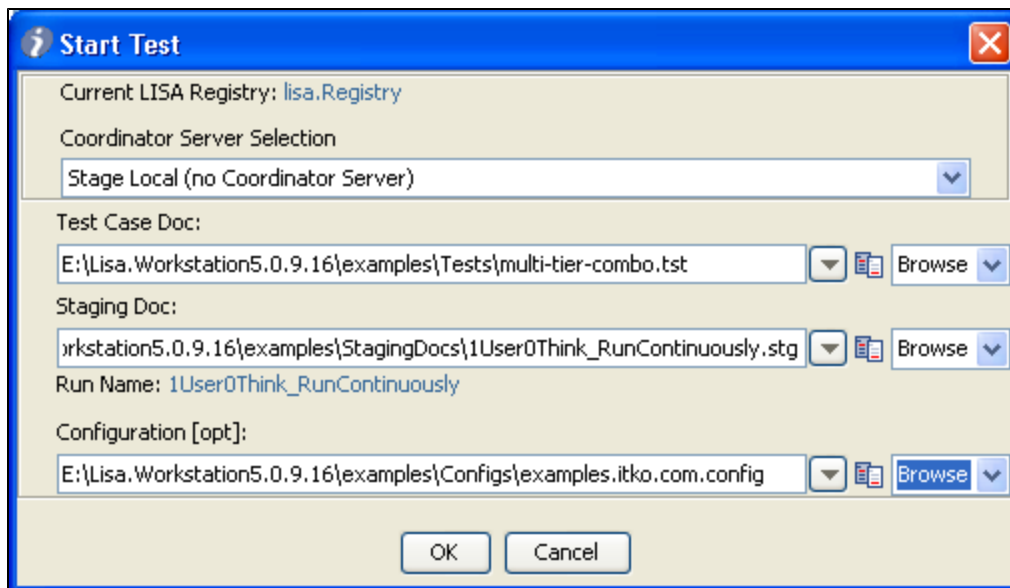
To start a test in LISA Workstation:



Click the **Start Test**,  icon on the toolbar:

Or, from the main menu, select **Actions > Start Test Case Execution**

The **Start Test** screen is displayed:



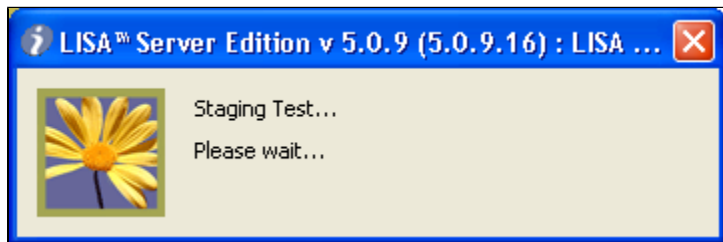
Enter the following parameters:

- **Current LISA Registry:** The attached LISA registry will be seen here.
- **Coordinator Server Selection:** Select the Coordinator Server. There is no choice here as you will always use Stage Local.
- **Test Case Doc:** The name of the Test Case document you want to run/execute. The document can be in the file system, on the classpath, or specified by a URL. You can browse the locations using the Browse pull-down menu. For detailed information on creating Test Cases, see [Part 2 - Building Test Cases](#) of this User Guide.
- **Staging Doc:** The name of the staging document you want to refer to, while executing the test case. The document can be in the file

system, on the classpath, or specified by a URL. You can browse the locations using the Browse pull-down menu. For detailed information on creating staging documents, see [Staging Documents](#).

- **Configuration [opt]:** The name of the configuration document to use, or the name of a configuration in to be applied to the Test Case document. The document can be in the file system, on the classpath, or specified by a URL. You can browse the locations using the Browse pull-down menu. Entering a configuration is optional, and if left blank the default configuration in the Test Case will be used. Note that it is the default configuration that is used, not the active configuration. Make sure that the desired configuration is made active in the Configs list. For detailed information on using configurations, see [Using Configurations](#)
- Click **OK** to open the Test Monitor.

It will start staging the test giving information as follows:



18.2 Test Monitor Window

18.2 Test Monitor Window

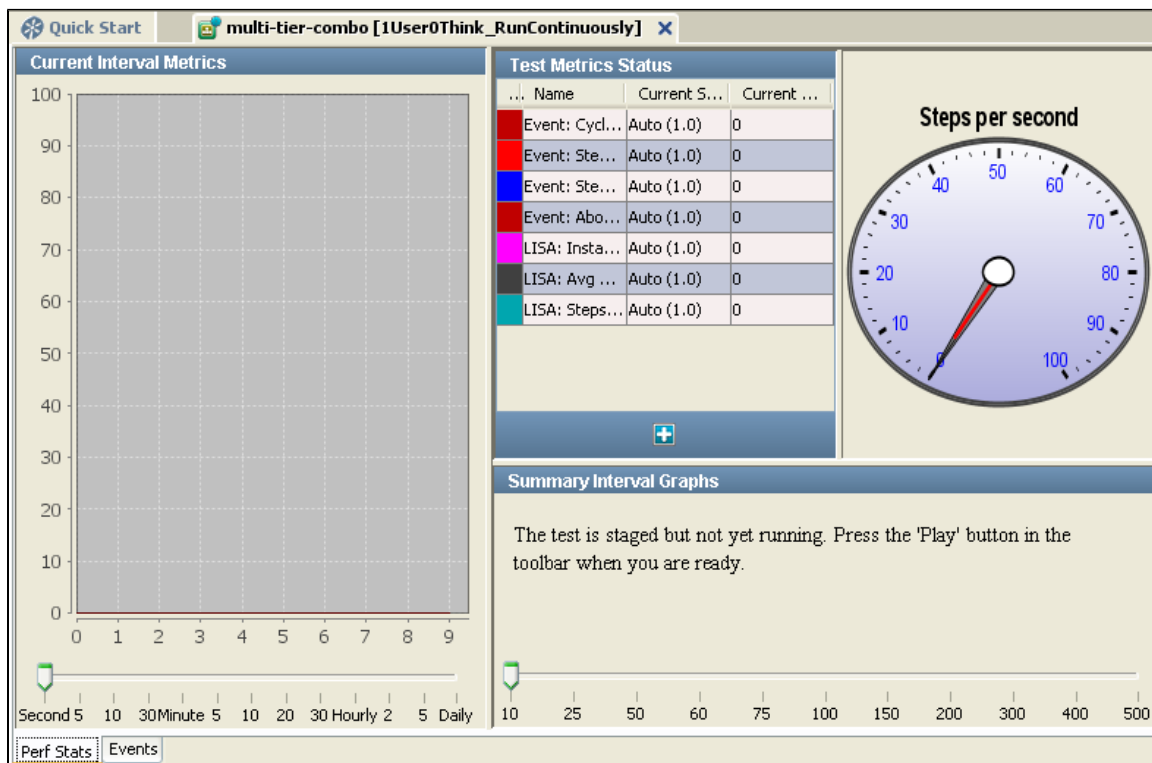
The test monitor window is similar to the one used for Running [Quick Tests](#) .

The difference between staging quick tests and staging a single test is:

In Quick tests, you only stage/run the test that is already open in the LISA workstation. Here in Start test case execution dialog box, you choose the test to be staged, along with others such as staging document.

You can also see the difference when the Test Case monitor opens:

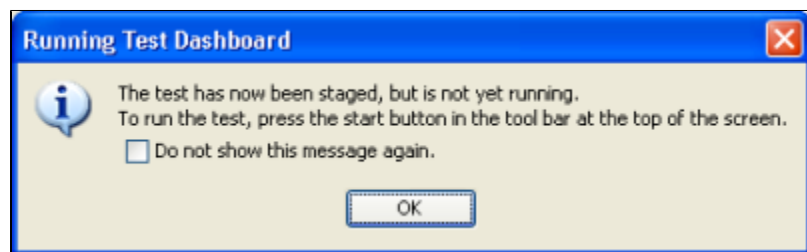
The only difference is when you start the Quick test, the name of the document has a "Quick stage run" defined. Where as for single test case, the name of the document has the staging document name associated with it as seen below **multi-tier-combo[1User0Think_RunContinuously]**:



Note: If you have already read about Quick tests, then all the information mentioned below is the same.

By default the test monitor opens in the Perf Stats tab.

At first, the test is staged, but the test has not started running yet, as it displays the following message:



Click OK to start and click on the **Play** button in the main toolbar.

At this point you can add the metrics and events that you want LISA to monitor during the test run.

The Test Monitor has two tabbed panes, **Perf Stats Tab** and **Events Tab**:

- The **Perf Stats Tab** allows you to choose Metrics, and display them as a function of time.
- The **Events Tab** allows you to choose and display Events as they are emitted during the test run.
- Once you run the test case and incase there are Failures in the test, then a third tab **Failures** will be displayed.

Performance Statistics Tab

The Test Manager opens in the **Perf Stats** tab by default. The Perf Stats tab is shown above.

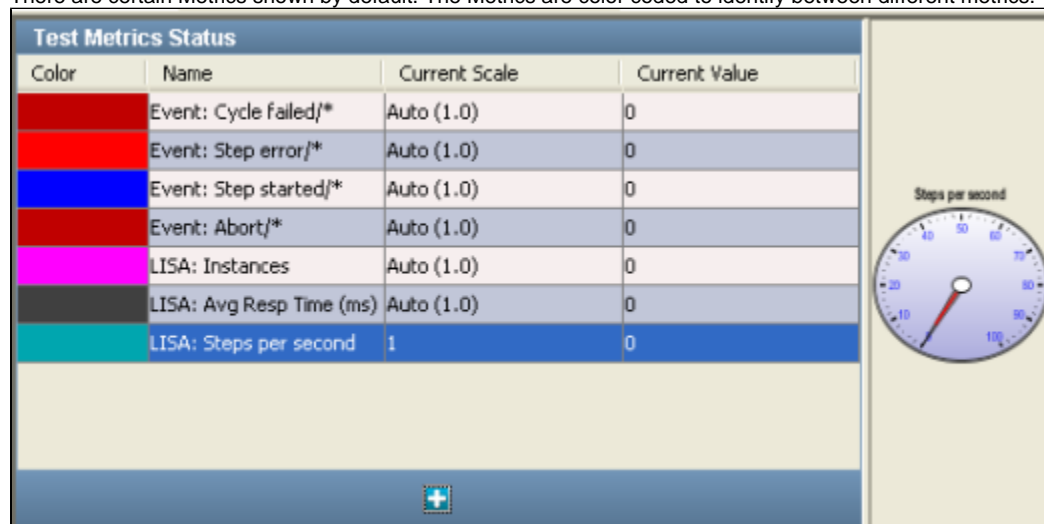
In this tab, you can add the Metrics and Events that you want to monitor during the test run.

The Perf Statistics tab consists of **three panels**. All of these panels are resizable within the screen.

1) Top --Right: The **Test Metrics Status** panel lists the current metrics being monitored.

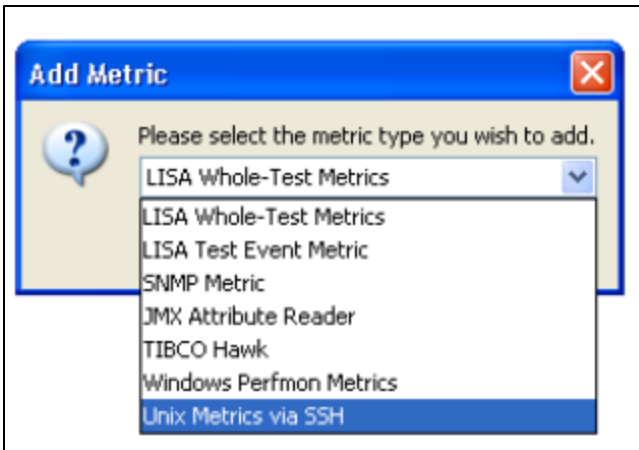
It also contains the "**Steps Per Second**" Meter on the right.

There are certain Metrics shown by default. The Metrics are color coded to identify between different metrics.



You can add additional Metrics by clicking the **Add** icon.

A pull-down menu displays all **metrics categories** that can be added:



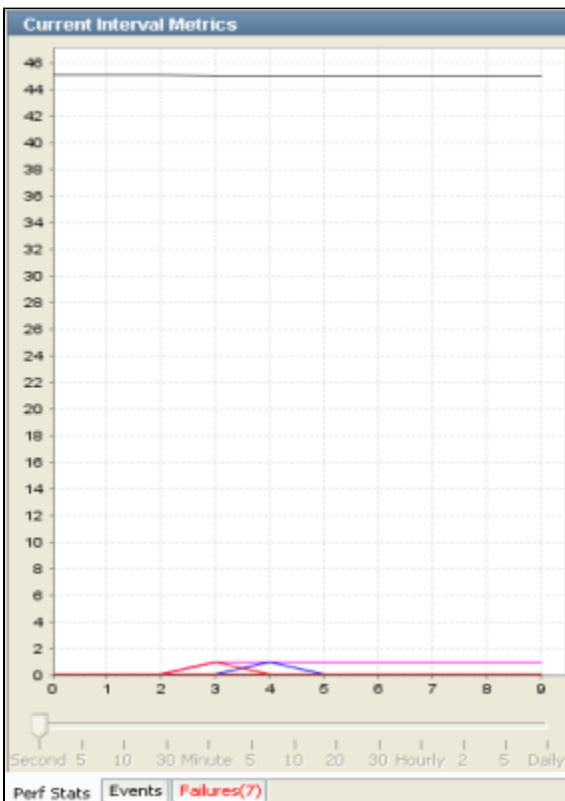
A description of each metric category and detailed instructions on how to configure individual metrics for display here can be found in [Metrics](#).

For each metric listed, LISA posts the following:

- **Color:** The color coding used on the graphs
- **Name:** The name of the metric. If the metric has been Filtered using its short name, then the short name appears after a slash in the name field. An * means use just the event name for the metric. For details, see [Metrics](#).
- **Current Scale:** The vertical scale used on the graphs. You can adjust the graph scale for any metric using the pull-down menu in this column.
- **Current Value:** The instantaneous value of the metric.

2) Top --Left: **Current Interval Metrics** panel displays the (color-coded) value of each metric at a specified time interval.

The time interval is chosen using the slider at the bottom of the panel. This can be adjusted prior to the test run, but not after the run has commenced. This test has failures so the **Failures** tab is seen in **Red**.



3) Bottom --Right: **Summary Interval Graphs** panel displays the (color-coded) value of each metric averaged over a specified time interval.

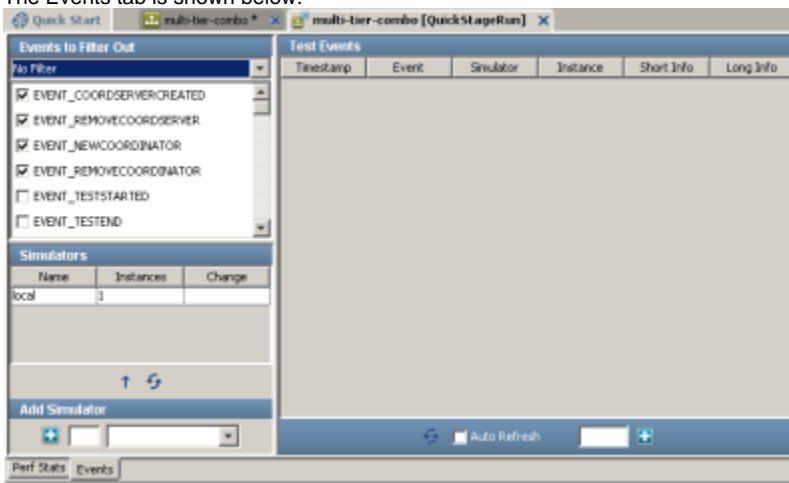
The time interval is chosen using the slider at the bottom of the panel. This can be adjusted prior to the test run, but not after the run has commenced.



Events Tab

The Events tab is used to filter Events.

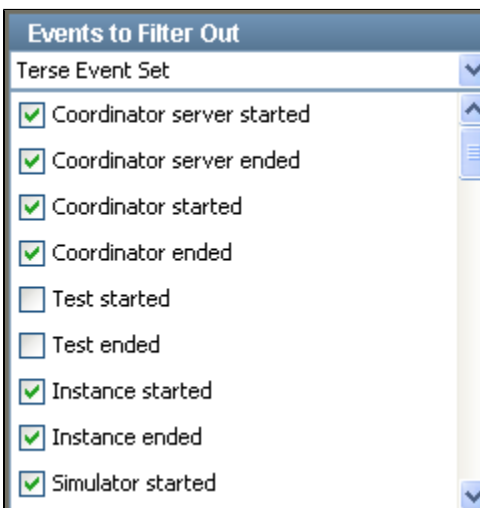
The Events tab is shown below:



It consists of three panels:

1) Top --Left: **Events to Filter Out** panel is a list of all the Events available in LISA.

Each one has a checkbox. Use this list to **Filter out** events you **do NOT want** to monitor. Choose the events to Filter out by clicking the checkbox associated with that event.



For convenience there are 3 standard sets of events listed in the pull-down menu above in the Event list.

Test Events					
Timestamp	Event	Simulator	Instance	Short Info	Long Info
2010-04-21 12:04:37,990	Test ended		0/0	3230356166...	
2010-04-21 12:04:37,521	Cycle ending	local	0/15	6163326266...	Signaled to s...
2010-04-21 12:04:37,521	Cycle started	local	0/15	6163326266...	
2010-04-21 12:04:37,521	Cycle ending	local	0/14	3231366562...	Signaled to s...
2010-04-21 12:04:37,521	Abort	local	0/14	3231366562...	<?xml versi...
2010-04-21 12:04:37,521	Step error	local	0/14	Get User	HomelessEJB...
2010-04-21 12:04:31,913	Step response bandwidth	local	0/14	Add User	1228
2010-04-21 12:04:31,913	Step request bandwidth	local	0/14	Add User	1236
2010-04-21 12:04:31,100	Cycle started	local	0/14	3231366562...	
2010-04-21 12:04:31,100	Cycle ending	local	0/13	3831646132...	Signaled to s...
2010-04-21 12:04:31,100	Abort	local	0/13	3831646132...	<?xml versi...
2010-04-21 12:04:31,85	Step error	local	0/13	Get User	HomelessEJB...
2010-04-21 12:04:25,288	Step response bandwidth	local	0/13	Add User	1231
2010-04-21 12:04:25,288	Step request bandwidth	local	0/13	Add User	1240
2010-04-21 12:04:24,491	Cycle started	local	0/13	3831646132...	
2010-04-21 12:04:24,491	Cycle ending	local	0/12	6137626465...	Signaled to s...
2010-04-21 12:04:24,491	Abort	local	0/12	6137626465...	<?xml versi...
2010-04-21 12:04:24,491	Step error	local	0/12	Get User	HomelessEJB...
2010-04-21 12:04:18,601	Step response bandwidth	local	0/12	Add User	1242
2010-04-21 12:04:18,601	Step request bandwidth	local	0/12	Add User	1251
2010-04-21 12:04:17,742	Cycle started	local	0/12	6137626465...	
2010-04-21 12:04:17,742	Cycle ending	local	0/11	6361663839...	Signaled to s...
2010-04-21 12:04:17,742	Abort	local	0/11	6361663839...	<?xml versi...
2010-04-21 12:04:17,742	Step error	local	0/11	Get User	HomelessEJB...

Only those Events, which have not been filtered out are seen in the above list.

For each event, the following information is posted:

- **Time Stamp:** The time of the Event
- **Event:** Name of the Event
- **Simulator:** The name of the Simulator where the instance is running
- **Instance:** The Instance ID and Run Number (in the form Instance ID/Run Number)
- **Short Info:** The short information for the event
- **Long Info:** The long information for the event

For more information on the event attributes – EventID, Short Info, and Long Info, see Chapter 7 Events in the LISA Reference Guide.

18.3 Quick Test Toolbar

18.3 Quick Test Toolbar

Quick tests are started, stopped and are reset using icons in the toolbar. The icons will change depending on the status of the quick test.

To start a quick test, click the **Play** icon on the main toolbar:



Once the test starts playing, the Play button, changes to Stop button.

So to stop a quick test, click the **Stop** icon on the toolbar:



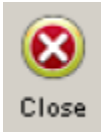
Note: If you click the "Stop" icon again you will be asked if you want to kill the test run.

When you stop a test you are saying "do not start any new instances". When you kill a test you are saying "stop all running instance as soon as possible".

To replay a quick test, click the **Replay Test** icon in the toolbar:



To close a quick test, click the **Close** icon in the toolbar:

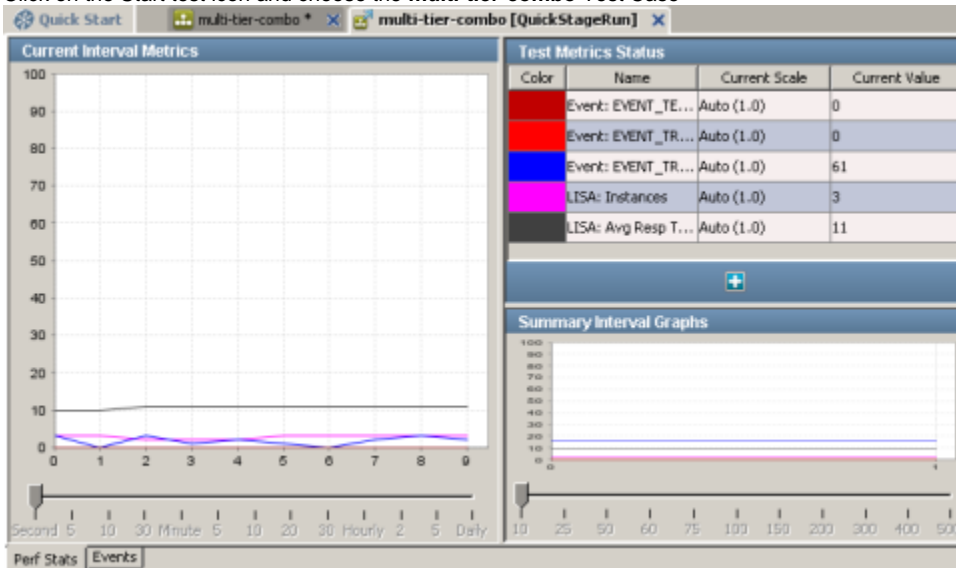


18.4 Example Test Case

18.4 Example Test Case

For the purpose of illustration, we will look at the Test using **multi-tier-combo** Test Case in the examples directory (multi-tier-combo.tst).

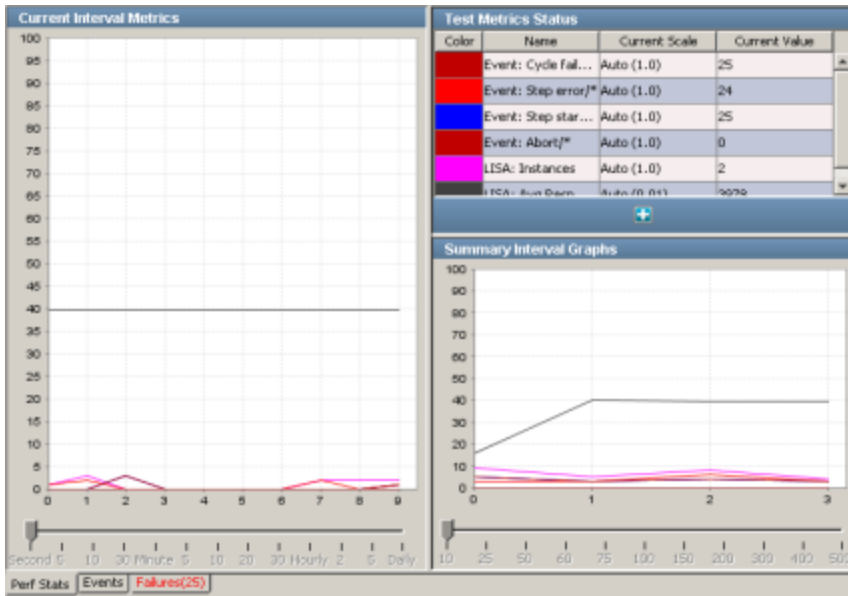
Click on the Start test icon and choose the **multi-tier-combo** Test Case



It was configured to run 3 instances, once each. A performance report was requested.

Performance Statistics Tab

The Performance of the test when staged is as shown below:



Events Tab

The Events List shows starting of an Event in **Green**. Failed events in **Red** and Assertion in **Magenta** color as shown below:

Events to Filter Out		Test Events				
No Filter		Timestamp	Event	Simul...	Insta...	Short Info
<input checked="" type="checkbox"/> Property set		2009-07-08 10:10:...	Cycle ending	local	2/30	delete user
<input checked="" type="checkbox"/> Step started		2009-07-08 10:10:...	Step history	local	2/30	delete user
<input checked="" type="checkbox"/> Assertion fired		2009-07-08 10:10:...	Step error	local	2/30	delete user
<input checked="" type="checkbox"/> Step response		2009-07-08 10:10:...	Info message	local	2/30	delete user
<input type="checkbox"/> Step response time		2009-07-08 10:10:...	Step response ban...	local	2/30	delete user
<input checked="" type="checkbox"/> Step bandwidth consumed		2009-07-08 10:10:...	Step request band...	local	2/30	delete user
<input checked="" type="checkbox"/> Step error		2009-07-08 10:10:...	Step response time	local	2/30	delete user
<input type="checkbox"/> Log message		2009-07-08 10:10:...	Cycle failed	local	0/30	delete user
<input type="checkbox"/> Info message		2009-07-08 10:10:...	Assertion fired	local	0/30	delete user
<input type="checkbox"/> Model definition error		2009-07-08 10:10:...	Info message	local	0/30	delete user
<input type="checkbox"/> Cycle ending		2009-07-08 10:10:...	Cycle failed	local	1/31	delete user
<input type="checkbox"/> Cycle runtime error		2009-07-08 10:10:...	Assertion fired	local	1/31	delete user
<input checked="" type="checkbox"/> Step request		2009-07-08 10:10:...	Info message	local	1/31	delete user
Simulators		2009-07-08 10:10:...	Data set read	local	1/31	delete user
Name		2009-07-08 10:10:...	Data set read	local	0/30	delete user
Instances		2009-07-08 10:10:...	Cycle started	local	1/31	delete user
Change		2009-07-08 10:10:...	Cycle started	local	0/30	delete user
local		2009-07-08 10:10:...	Cycle history	local	1/30	delete user
Add Simulator		2009-07-08 10:10:...	Cycle ending	local	1/30	delete user
		2009-07-08 10:10:...	Step history	local	1/30	delete user
		2009-07-08 10:10:...	Cycle history	local	0/29	delete user
		2009-07-08 10:10:...	Cycle ending	local	0/29	delete user
		2009-07-08 10:10:...	Step history	local	0/29	delete user
		2009-07-08 10:10:...	Step error	local	1/30	delete user
		2009-07-08 10:10:...	Step error	local	0/29	delete user
		2009-07-08 10:10:...	Info message	local	1/30	delete user
		2009-07-08 10:10:...	Info message	local	0/29	delete user

Failures Tab

You can see that there are 134 failures in this Test Case.

Timestamp	Event	Simulator	Instance	Short Info	Long Info
2009-07-30 14:4...	Abort	local	0/33	abort	<?xml version=...
2009-07-30 14:4...	Abort	local	0/32	abort	<?xml version=...
2009-07-30 14:4...	Abort	local	0/31	abort	<?xml version=...
2009-07-30 14:4...	Abort	local	0/30	abort	<?xml version=...
2009-07-30 14:4...	Abort	local	0/29	abort	<?xml version=...
2009-07-30 14:4...	Abort	local	0/28	abort	<?xml version=...
2009-07-30 14:4...	Abort	local	0/27	abort	<?xml version=...
2009-07-30 14:3...	Abort	local	0/26	abort	<?xml version=...
2009-07-30 14:3...	Abort	local	0/25	abort	<?xml version=...
2009-07-30 14:3...	Abort	local	0/24	abort	<?xml version=...
2009-07-30 14:3...	Abort	local	0/23	abort	<?xml version=...
2009-07-30 14:3...	Abort	local	0/22	abort	<?xml version=...
2009-07-30 14:3...	Abort	local	0/21	abort	<?xml version=...
2009-07-30 14:3...	Abort	local	0/20	abort	<?xml version=...
2009-07-30 14:3...	Abort	local	0/19	abort	<?xml version=...
2009-07-30 14:3...	Abort	local	0/18	abort	<?xml version=...
2009-07-30 14:3...	Abort	local	0/17	abort	<?xml version=...
2009-07-30 14:3...	Abort	local	0/16	abort	<?xml version=...
2009-07-30 14:3...	Abort	local	0/15	abort	<?xml version=...
2009-07-30 14:3...	Abort	local	0/14	abort	<?xml version=...
2009-07-30 14:3...	Abort	local	0/13	abort	<?xml version=...
2009-07-30 14:3...	Abort	local	0/12	abort	<?xml version=...
2009-07-30 14:3...	Abort	local	0/11	abort	<?xml version=...
2009-07-30 14:3...	Abort	local	0/10	abort	<?xml version=...

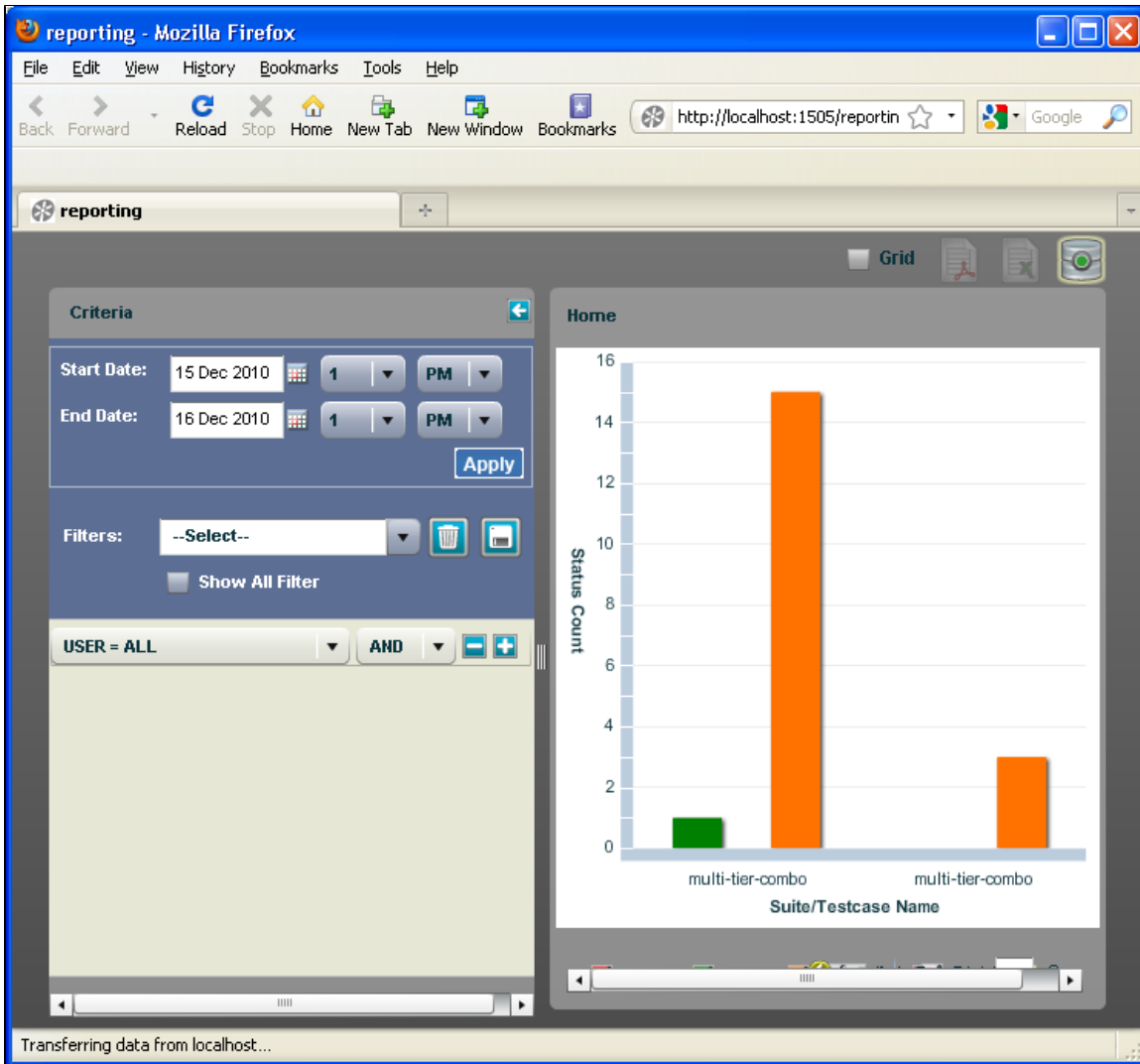
about:blank	
Done	
Perf Stats	Events
	Failures(34)

18.5 View Reports

18.5 View Reports

You can view the Reports in the LISA Reporting Portal.

- Select View > Reporting Console
- Or Click on the Reports icon on the toolbar
- Or Open the Web browser and enter <http://localhost:1505/reporting/> to open the Reports Viewer as shown below:



For more information, see Reports 5.0 section.

19. Running a Quick Test

19. Running a Quick Test

The **Quick Test** utility allows you to run an a test case quickly with minimal set up.

For the Quick test utility, you need to have a test case already open within LISA workstation.

The Quick Test utility runs the test that is currently in memory rather than loading a Test Case document, and uses a simple pre-built staging specification with very few options.

The test has minimal instances so its easy to stage/run, but lacks much of the functionality afforded by a test staged with a [Staging Document](#) as in case of a proper [Test Case Execution](#) .

A Quick Test allows you to select and monitor events and metrics, and view a standard performance report.

The following topics are available...

- [19.1 Running a Quick Test](#)
- [19.2 Test Monitor window](#)
- [19.3 Quick Test Toolbar](#)
- [19.4 Example Test Case](#)
- [19.5 View Report](#)

19.1 Running a Quick Test

19.1 Running a Quick Test

Open a Test Case in LISA workstation.

For the purpose of illustration, we will look at the **multi-tier-combo** Test Case in the examples directory (multi-tier-combo.tst).

To Stage a Quick Test,



Click the **Stage a Quick Test** icon on the toolbar
Or from the main menu, select **Actions->Stage a Quick Test...**

The **Stage Quick Test** screen is shown below:

NOTE - You can run a quick test in LISA, only if you have a Test Case open in the Workstation.

If you have coordinators and simulators attached to a LISA Registry, it will show you the Coordinator or Simulator Servers attached as shown below:

A screenshot of the 'Stage Quick Test' dialog box. The title bar is blue with a question mark icon and a close button. The main area is titled 'Stage Quick Test Parameters'. It contains four input fields: 'Run Name' with the text 'QuickStageRun', 'Number of Instances' with the value '5', and 'Stage Instances To:' with a dropdown menu showing 'Stage Local (no Coordinator Server)' selected. Below the dropdown is a list box containing 'Stage Local (no Coordinator Server)' and 'Isa.CServer'. At the bottom of the list box is a checkbox labeled 'Generate a Report' which is currently unchecked. At the very bottom of the dialog are 'OK' and 'Cancel' buttons.

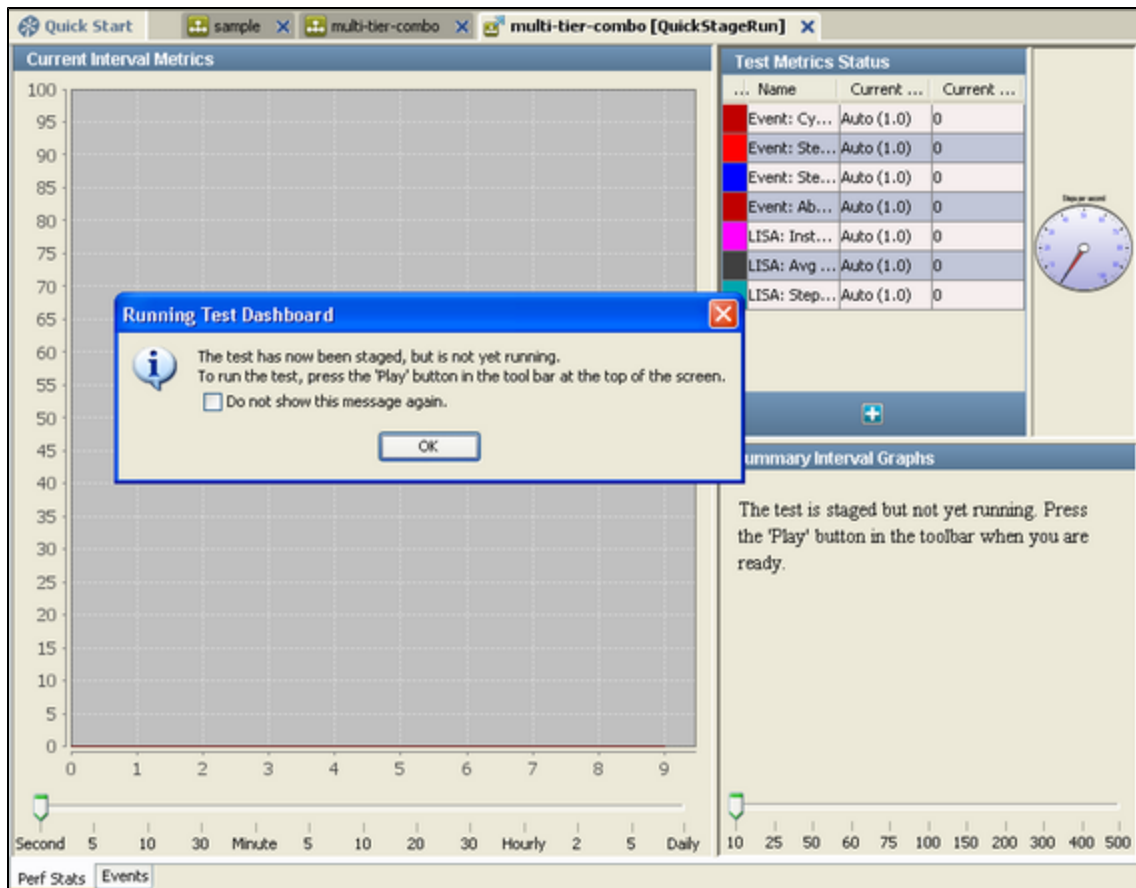
Enter the following parameters:

- **Run Name:** The name given to this quick test.
- **Number of Instances:** The number of concurrent users (instances) to be used.

NOTE: The maximum number of users you can specify is determined by your LISA license.

- **Stage Instances to:** The name of the coordinator server, where the test will be staged.
- **If test ends, restart it:** Check this box if you want to run the test continuously (until you stop it manually).
- **Generate a Report:** Check this box if you want a performance [Report](#) generated. This report can be viewed in the Report Viewer.
- Click **OK** to proceed.

The Test Monitor is displayed, but the test has not been started yet as shown below:

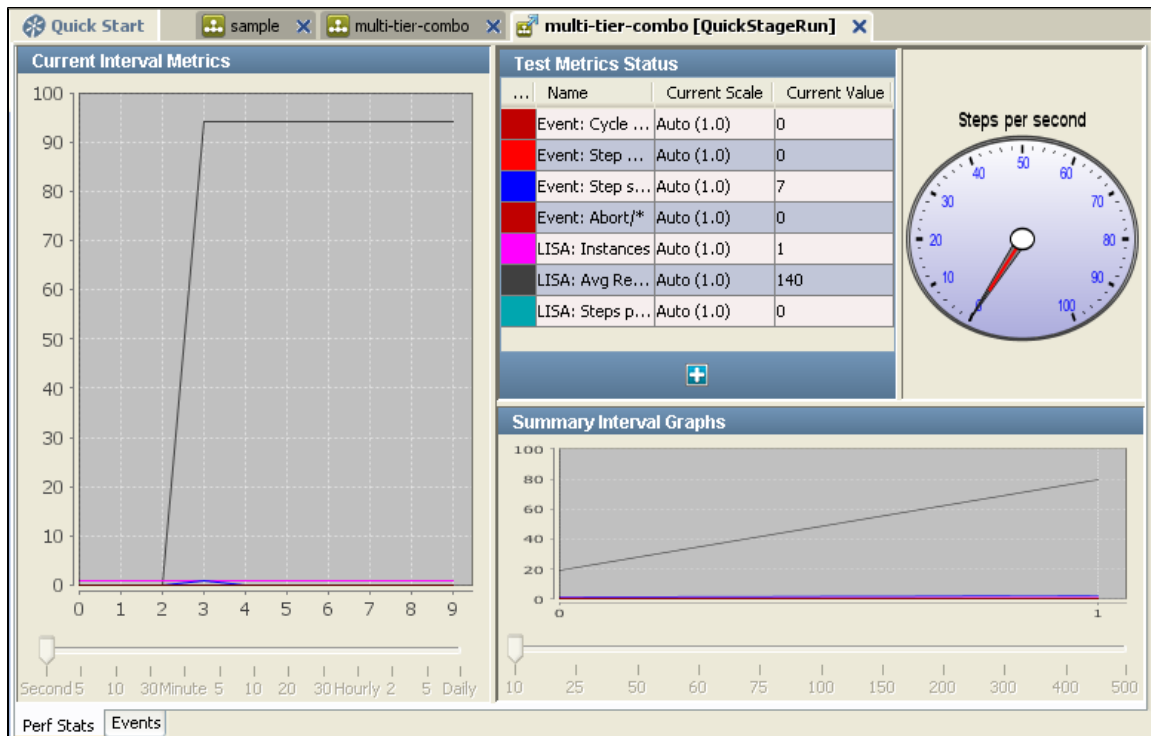


Click **OK** to open the **Test Monitor**.

19.2 Test Monitor window

19.2 Test Monitor window

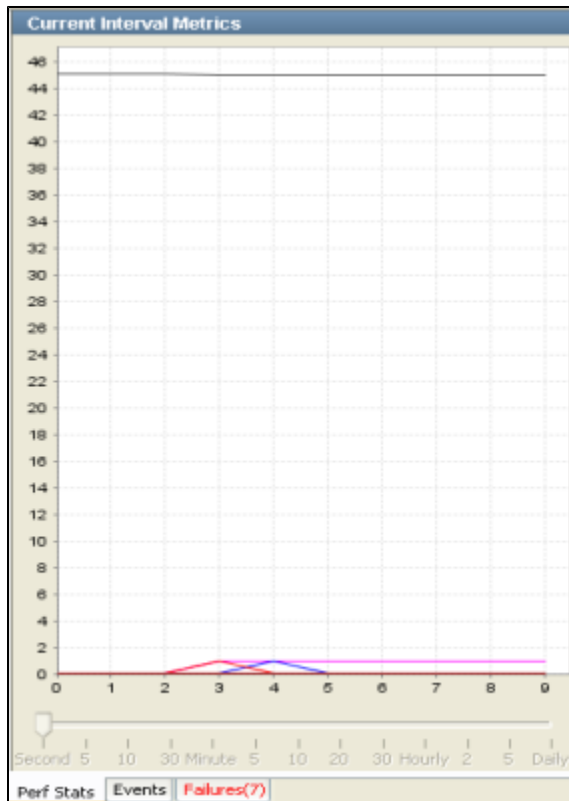
The Test Monitor will open as one of the several tabs in the LISA workstation.



The Test Monitor has two tabbed panes at the bottom, **Perf Stats Tab** and **Events Tab**:

- The **Perf Stats Tab** - Displays collected Metrics as a function of time. Once the test starts, this cannot be changed.
- The **Events Tab** - Displays events as they are emitted during the test run.

Note: Once you run the test and incase there are Failures in the test, then a third tab **Failures** will be displayed.

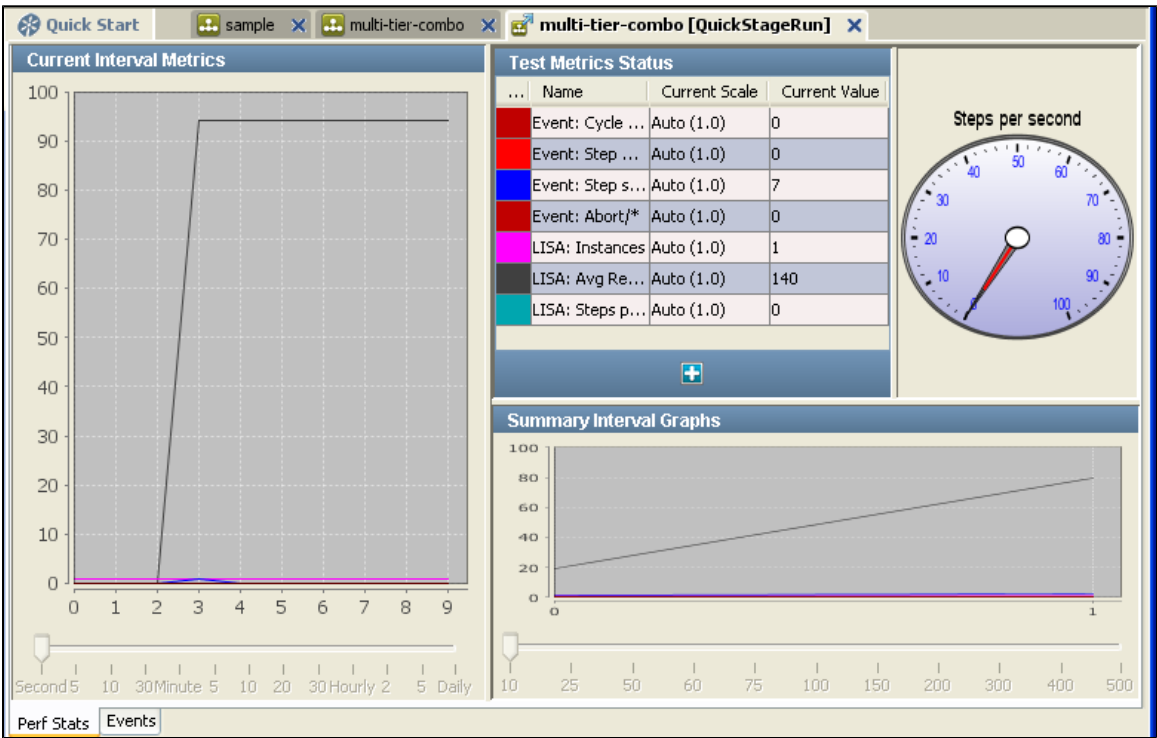


Perf Stats Tab

The Test Manager opens in the **Perf Stats** tab by default. The Perf Stats tab is shown above.

In this tab, you can add the Metrics and Events that you want to monitor during the test run.

The Perf Statistics tab consists of **three panels**. All of these panels are resizable within the screen.

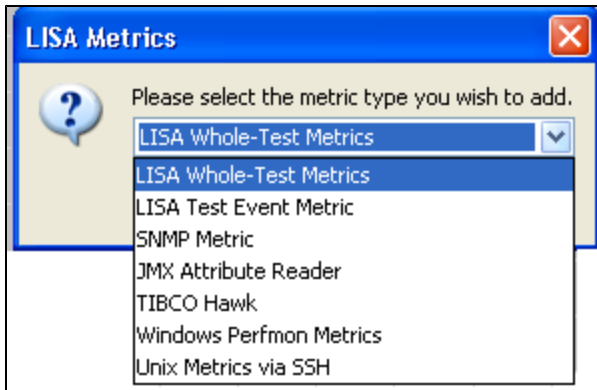


It consists of **three panels**:

1) Top --Right: **Test Metrics Status** panel lists the current metrics being monitored.

Test Metrics Status			
Color	Name	Current Sc...	Current V...
	Event: Cycle failed/*	Auto (1.0)	0
	Event: Step error/*	Auto (1.0)	0
	Event: Step started/*	Auto (1.0)	0
	Event: Abort/*	Auto (1.0)	0
	LISA: Instances	Auto (1.0)	0
	LISA: Avg Resp Time (ms)	Auto (1.0)	0
	LISA: Steps per second	Auto (1.0)	0

You can add additional metrics by clicking the **Add**  icon. A pull-down menu displays **six metrics categories**:



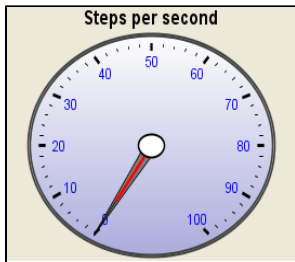
Click on any one to Add the selected metric.

A description of each metric category and detailed instructions on how to configure individual metrics for display here can be found in [Metrics](#).

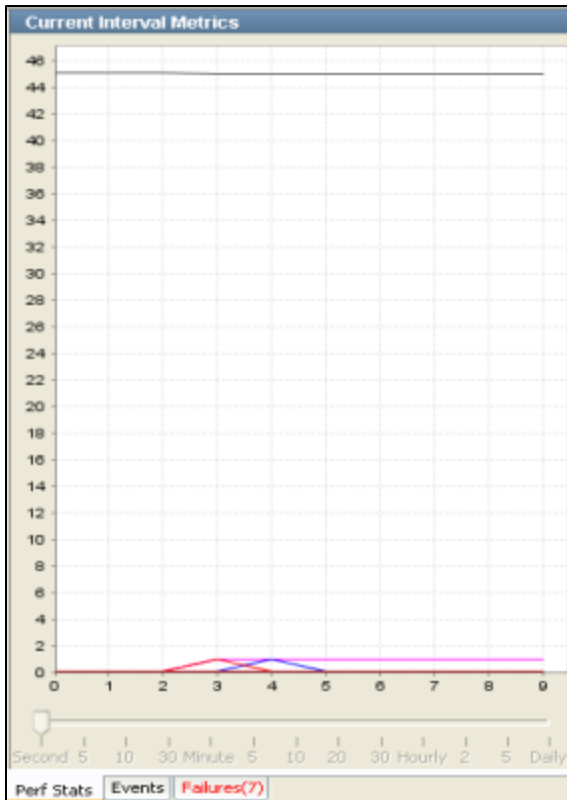
For each metric listed, LISA posts the following:

- **Color:** The color coding used on the graphs
- **Name:** The name of the metric. If the metric has been Filtered using its short name, then the short name appears after a slash in the name field. An * means use just the event name for the metric. For details, see Metrics.
- **Current Scale:** The vertical scale used on the graphs. You can adjust the graph scale for any metric using the pull-down menu in this column.
- **Current Value:** The instantaneous value of the metric.

It also displays a meter showing the Steps executed per second as shown below:



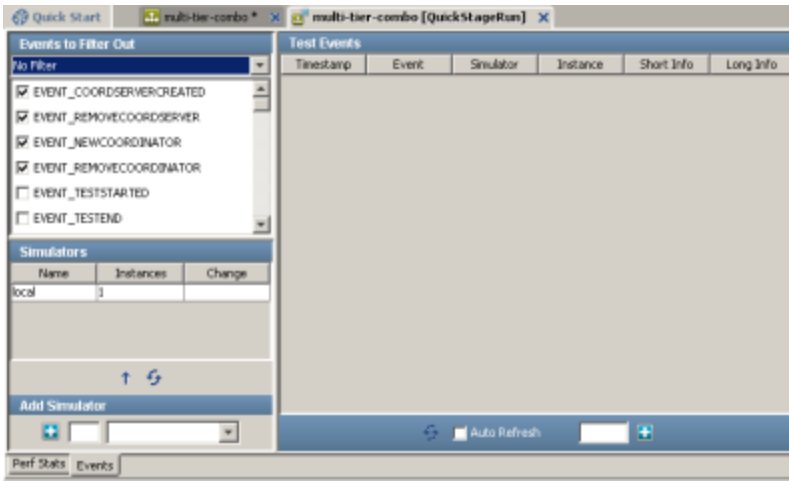
2) Top --Left: **Current Interval Metrics** panel displays the (color-coded) value of each metric at a specified time interval. The time interval is chosen using the slider at the bottom of the panel. This can be adjusted prior to the test run, but not after the run has commenced. This test has failures so the **Failures** tab is seen in **Red**.



3) Bottom --Right: **Summary Interval Graphs** panel displays the (color-coded) value of each metric averaged over a specified time interval. The time interval is chosen using the slider at the bottom of the panel. This can be adjusted prior to the test run, but not after the run has commenced.



Events Tab



It consists of three panels:

1) Top --Left: **Events to Filter Out** panel is a list of all the Events available in LISA. Each one has a checkbox. Use this list to Filter out events you **do NOT want** to monitor. Choose the events to Filter out by clicking the checkbox associated with that event.

For convenience there are 3 standard sets of events listed in the pull-down menu above the Event list.

- Terse Event Set
- Common Event Set
- Verbose Event Set

These sets provide a pre-selected list of events that you can use as a starting list. You can then customize the list.

2) Bottom -Left: **Simulators** panel shows the status of the Simulators in use. This is covered in depth in [PART 5 - Starting, Staging & Running Tests](#). In a quick test, it is just showing how many instances are running, which in our current example is 1, on local (i.e. in Workstation).

3) Top --Right: **Test Events** panel lists the events as they are emitted, with the most recent on the top of the list. You can list events in real time by checking the Auto Refresh box at the bottom of the panel, or you can refresh the list manually by clicking the Refresh icon, with the Auto Refresh box unchecked.

For each event, the following information is posted:

- **Time Stamp:** The time of the Event
- **Event:** Name of the Event
- **Simulator:** The name of the Simulator where the instance is running
- **Instance:** The Instance ID and Run Number (in the form Instance ID/Run Number)
- **Short Info:** The short information for the event
- **Long Info:** The long information for the event

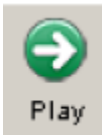
For more information on the event attributes – EventID, Short Info, and Long Info, see [LISA Reference Guide](#).

19.3 Quick Test Toolbar

19.3 Quick Test Toolbar

Quick tests are started, stopped and reset using icons in the toolbar. The icons will change depending on the status of the quick test.

To start a quick test, click the **Play** icon on the main toolbar:



Once the test starts playing, the Play button changes to Stop button.

To stop a quick test, click the **Stop** icon on the toolbar:

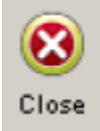


Note: If you click the "Stop" icon again you will be asked if you want to kill the test run. When you stop a test you are saying "do not start any new instances". When you kill a test you are saying "stop all running instance as soon as possible".

To replay a quick test, click the **Replay Test** icon in the toolbar:

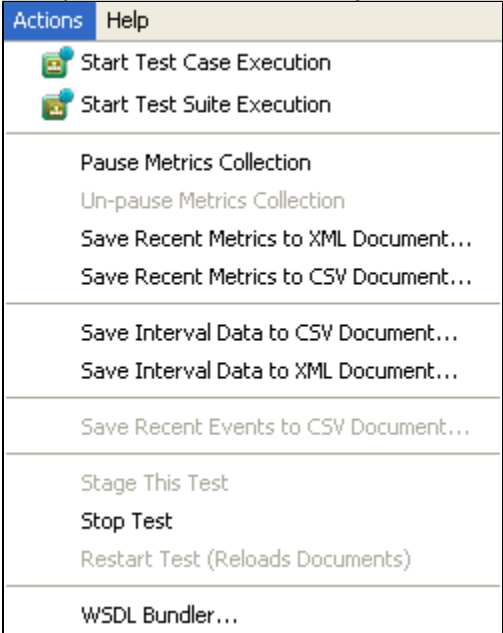


To close a quick test, click the **Close** icon in the toolbar:



Actions Menu

When you run a test and click the **Play** button to actually run it, the Actions main Menu changes as shown below:



These menus are explained below:

- **Pause Metric Collection** – Click to pause the metric collection
- **Un-pause Metric Collection** – Click to un-pause the metric collection
- **Save recent Metrics to XML document** - Click to save the recent metrics to a XML file
- **Save recent metrics to CSV document** – Click to save the recent metrics to a CSV file
- **Save Internal Data to CSV document** – Click to save the internal data to a CSV file
- **Save Internal Data to XML document** – Click to save the internal data to a XML file
- **Save recent Events to CSV documents** – Click to save recent events to CSV documents
- **Stage this test** – Click to stage this test
- **Stop this test** – Click to stop this test
- **Restart Test (Reload Documents)** – Click to reload the documents and Restart the test.

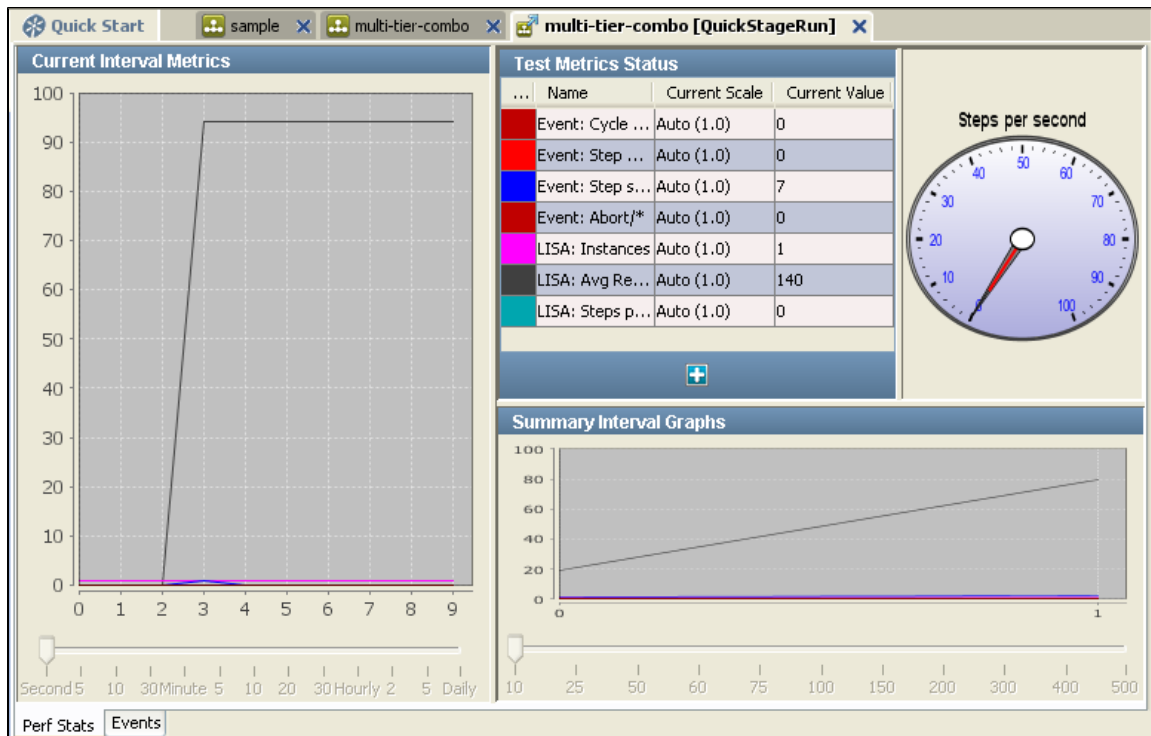
19.4 Example Test Case

19.4 Example Test Case

For the purpose of illustration, we will look at the Quick Test using **multi-tier-combo** Test Case in the examples directory (multi-tier-combo.tst). It was configured to run 3 instances, once each. A performance report was requested.

Perf Stats Tab

The Performance of the test when staged is as shown below:



Events Tab

The Events List shows starting of an Event in **Green**. Failed events in **Red** and Assertion in **Magenta** color as shown below:

Events to Filter Out		Test Events				
No Filter		Timestamp	Event	Simul...	Insta...	Short Info
<input checked="" type="checkbox"/> Property set		2009-07-08 10:10:...	Cycle ending	local	2/30	28373036...
<input checked="" type="checkbox"/> Step started		2009-07-08 10:10:...	Step history	local	2/30	delete user
<input checked="" type="checkbox"/> Assertion fired		2009-07-08 10:10:...	Step error	local	2/30	Invocatio...
<input checked="" type="checkbox"/> Step response		2009-07-08 10:10:...	Info message	local	2/30	delete use...
<input type="checkbox"/> Step response time		2009-07-08 10:10:...	Step response ban...	local	2/30	delete user 0
<input checked="" type="checkbox"/> Step bandwidth consumed		2009-07-08 10:10:...	Step request band...	local	2/30	delete user 1015
<input checked="" type="checkbox"/> Step error		2009-07-08 10:10:...	Step response time	local	2/30	delete user 4062
<input checked="" type="checkbox"/> Log message		2009-07-08 10:10:...	Cycle failed	local	0/30	65616130... <?xml ve...
<input type="checkbox"/> Info message		2009-07-08 10:10:...	Assertion fired	local	0/30	start [lis...
<input type="checkbox"/> Model definition error		2009-07-08 10:10:...	Info message	local	0/30	start Starting L...
<input type="checkbox"/> Cycle ending		2009-07-08 10:10:...	Cycle failed	local	1/31	37323863... <?xml ve...
<input type="checkbox"/> Cycle runtime error		2009-07-08 10:10:...	Assertion fired	local	1/31	start [lis...
<input checked="" type="checkbox"/> Step request		2009-07-08 10:10:...	Info message	local	1/31	start Starting L...
		2009-07-08 10:10:...	Data set read	local	1/31	unique_user unique_us...
		2009-07-08 10:10:...	Data set read	local	0/30	unique_user unique_us...
		2009-07-08 10:10:...	Cycle started	local	1/31	37323863... <?xml ve...
		2009-07-08 10:10:...	Cycle started	local	0/30	65616130... <?xml ve...
		2009-07-08 10:10:...	Cycle history	local	1/30	38613637... <?xml ve...
		2009-07-08 10:10:...	Cycle ending	local	1/30	38613637... Signaled t...
		2009-07-08 10:10:...	Step history	local	1/30	delete user <?xml ve...
		2009-07-08 10:10:...	Cycle history	local	0/29	33623736... <?xml ve...
		2009-07-08 10:10:...	Cycle ending	local	0/29	33623736... Signaled t...
		2009-07-08 10:10:...	Step history	local	0/29	delete user <?xml ve...
		2009-07-08 10:10:...	Step error	local	1/30	Invocatio...
		2009-07-08 10:10:...	Step error	local	0/29	Invocatio...
		2009-07-08 10:10:...	Info message	local	1/30	delete use...
		2009-07-08 10:10:...	Info message	local	0/29	delete use...

Failures Tab

You can see that there are 134 failures in this Test Case.

Timestamp	Event	Simulator	Instance	Short Info	Long Info
2009-07-30 14:4...	Abort	local	0/33	abort	<html version=...
2009-07-30 14:4...	Abort	local	0/32	abort	<html version=...
2009-07-30 14:4...	Abort	local	0/31	abort	<html version=...
2009-07-30 14:4...	Abort	local	0/30	abort	<html version=...
2009-07-30 14:4...	Abort	local	0/29	abort	<html version=...
2009-07-30 14:4...	Abort	local	0/28	abort	<html version=...
2009-07-30 14:4...	Abort	local	0/27	abort	<html version=...
2009-07-30 14:3...	Abort	local	0/26	abort	<html version=...
2009-07-30 14:3...	Abort	local	0/25	abort	<html version=...
2009-07-30 14:3...	Abort	local	0/24	abort	<html version=...
2009-07-30 14:3...	Abort	local	0/23	abort	<html version=...
2009-07-30 14:3...	Abort	local	0/22	abort	<html version=...
2009-07-30 14:3...	Abort	local	0/21	abort	<html version=...
2009-07-30 14:3...	Abort	local	0/20	abort	<html version=...
2009-07-30 14:3...	Abort	local	0/19	abort	<html version=...
2009-07-30 14:3...	Abort	local	0/18	abort	<html version=...
2009-07-30 14:3...	Abort	local	0/17	abort	<html version=...
2009-07-30 14:3...	Abort	local	0/16	abort	<html version=...
2009-07-30 14:3...	Abort	local	0/15	abort	<html version=...
2009-07-30 14:3...	Abort	local	0/14	abort	<html version=...
2009-07-30 14:3...	Abort	local	0/13	abort	<html version=...
2009-07-30 14:3...	Abort	local	0/12	abort	<html version=...
2009-07-30 14:3...	Abort	local	0/11	abort	<html version=...
2009-07-30 14:3...	Abort	local	0/10	abort	<html version=...

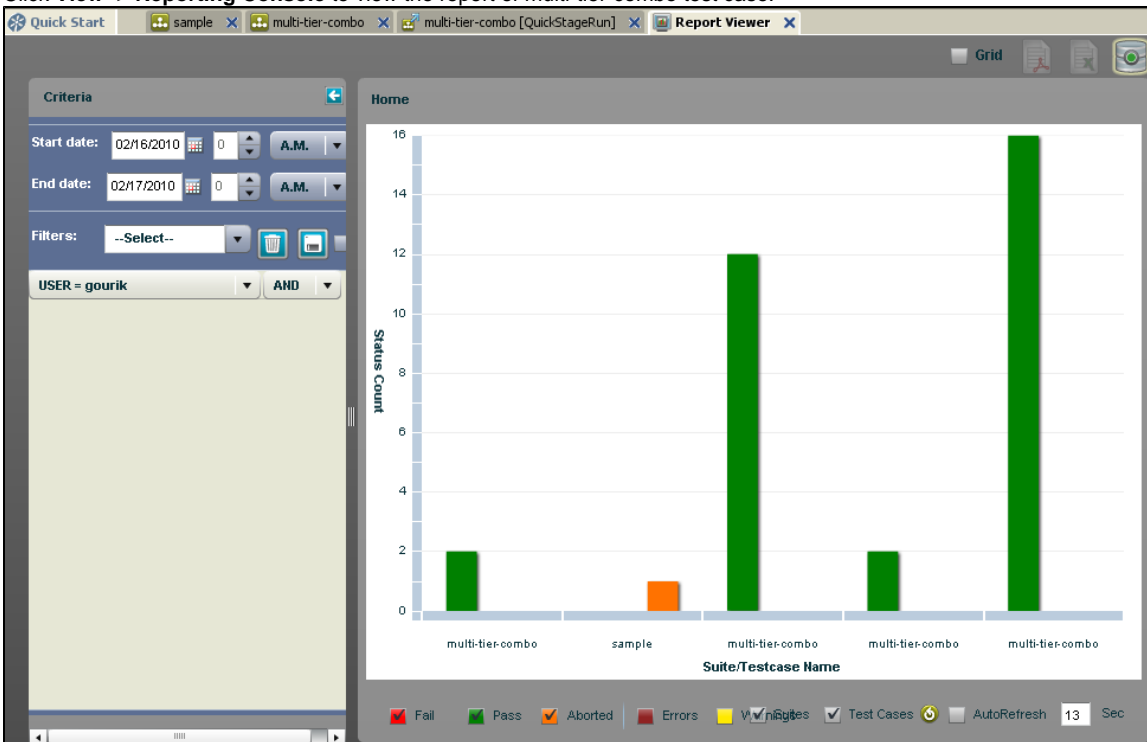
19.5 View Report

19.5 View Report

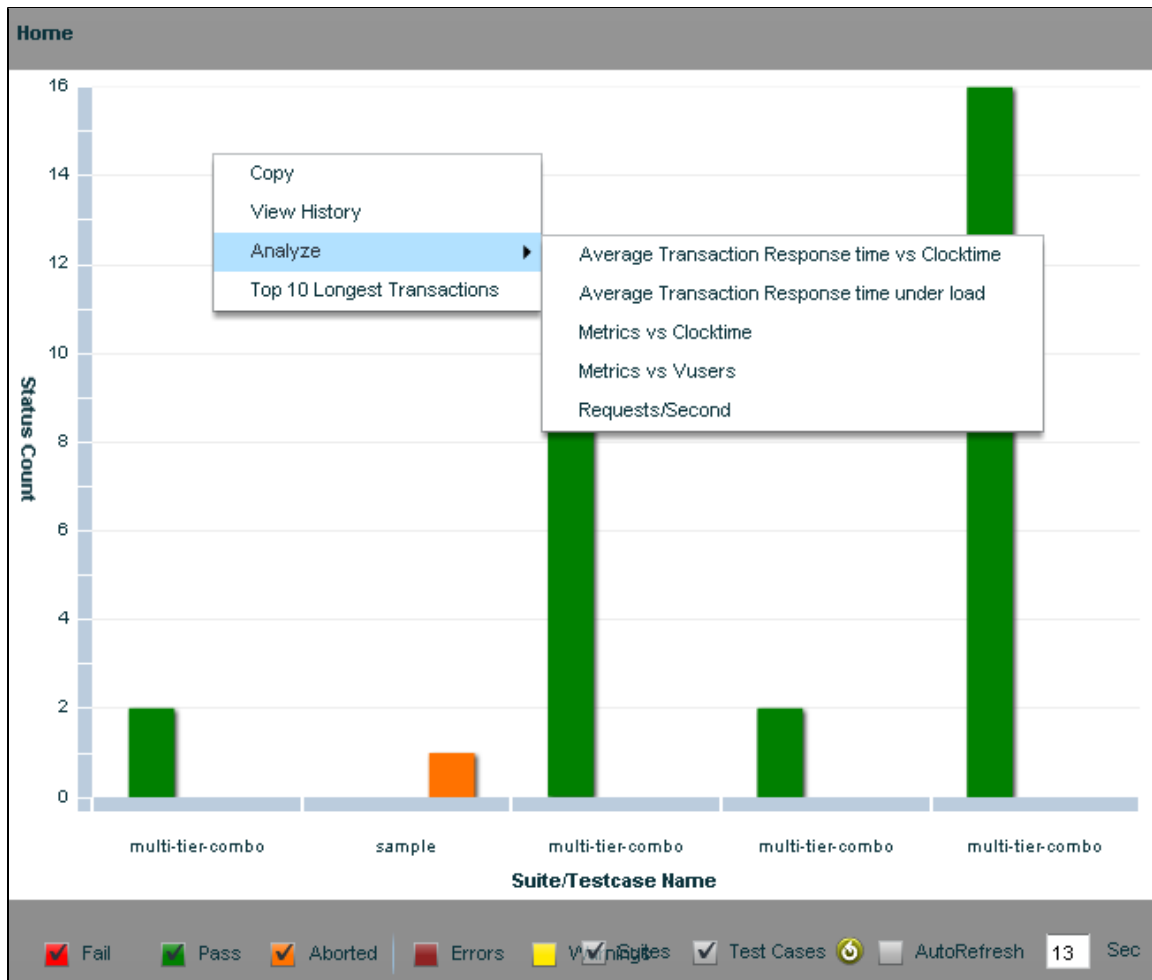
If you have selected to View Reports in the first dialog box, at the end of the test you are given the opportunity to choose and view the report.

The report data is persisted in the report database so that you can view it at a later date also.

Click **View -> Reporting Console** to view the report of multi-tier-combo test case.

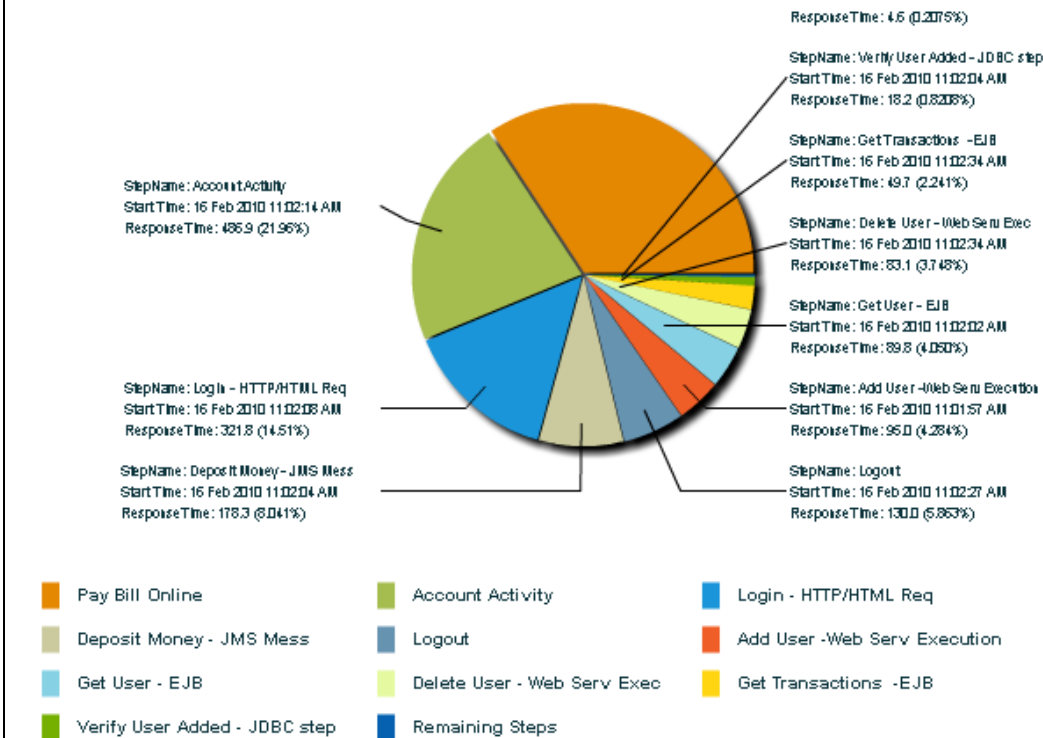


You can also analyze the report by right clicking on the graph as shown below:



Click to see the 10 longest transactions as shown below:

10 Longest Transactions



For more information on reports, see [Reports](#).

20. Running a Test Suite

20. Running a Test Suite

A **Test Suite** allows you to group some related Test Cases and test suites, and run them as a single test. A Test Suite Document specifies the contents of the suite, the reports to generate, and the metrics to collect. These reports and metrics relate to the suite as a whole; each test within the suite will still produce its own reports and metrics. Each test still retains the ability to use its own staging document, configuration, and audit document. Therefore tests in a suite can be run in a distributed environment if desired.

When a suite is included within a suite, the individual tests in the included suite are extracted from the suite and run as individual tests in the current suite. The defaults from the included suite, and the startup and teardown settings will be ignored.

The following topics are available.

[20.1 Creating a Test Suite](#)
[20.2 Test Suite Editor](#)

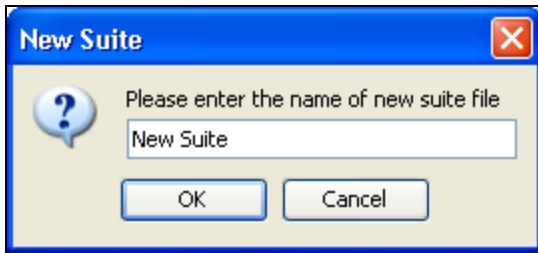
20.1 Creating a Test Suite

20.1 Creating a Test Suite

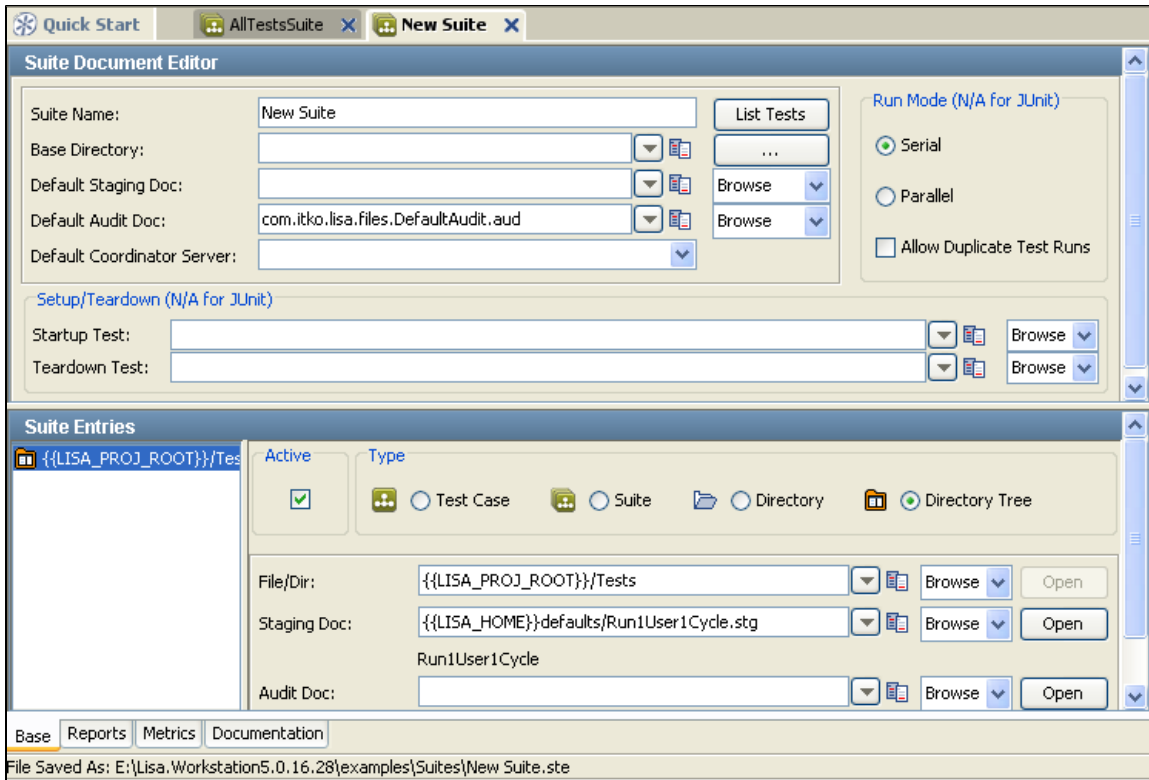
To create a Test Suite,

- Click **File > New > Suite** from the main menu,
- Or Right click on the **Suites** folder in the Project pane to open a menu and Click **Create a new Suite**

The **New Suite** dialog box opens:



- Enter the name of the new Suite and click ok to open the **Test Suite Editor** as shown below:



There are four tabs at the bottom of the Suite Editor:

By default the Test suite document opens in the Base tab.

- Base tab
- Reports tab
- Metrics tab
- Documentation tab

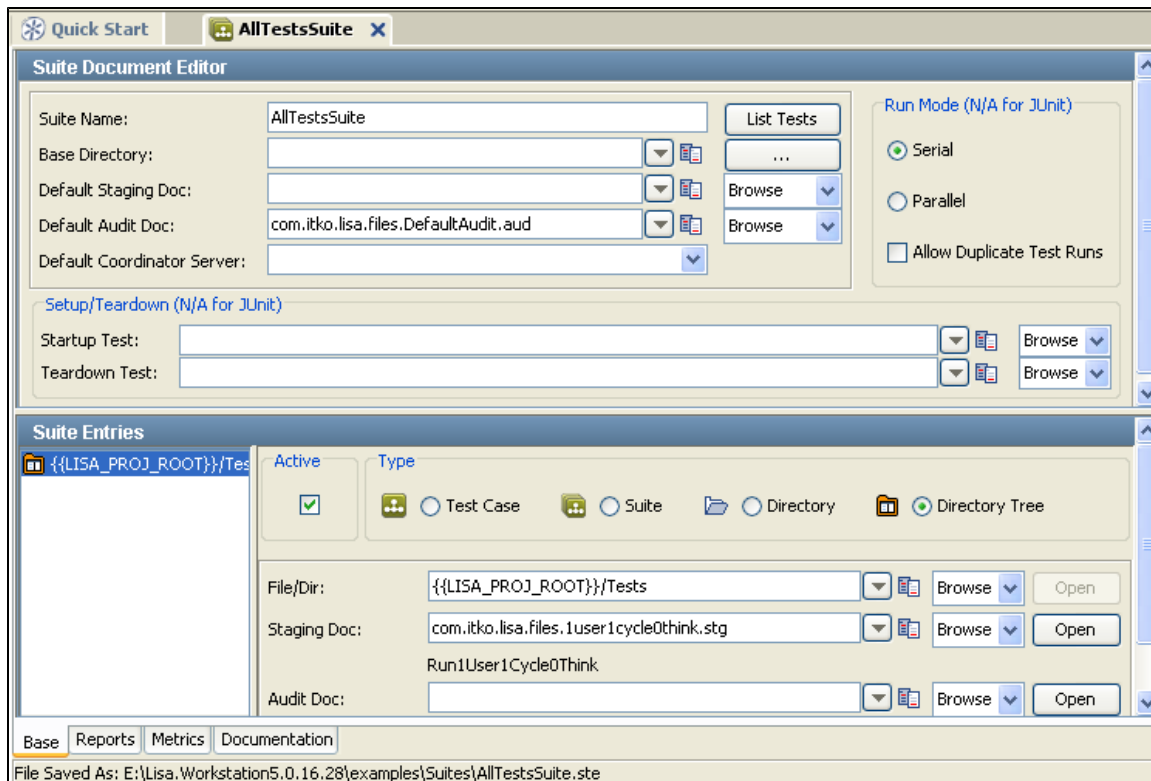
All these tabs are explained in detail in the next section [Test Suite Editor](#) .

20.2 Test Suite Editor

20.2 Test Suite Editor

Once you open a test suite, its editor is as shown below:

For the purpose of illustration, we will take a look at the **AllTestsSuite.ste** located in the %LISA_HOME%/examples/suites/**AllTestsSuite.ste**



There are four tabs at the bottom, which are further explained in the following sections.

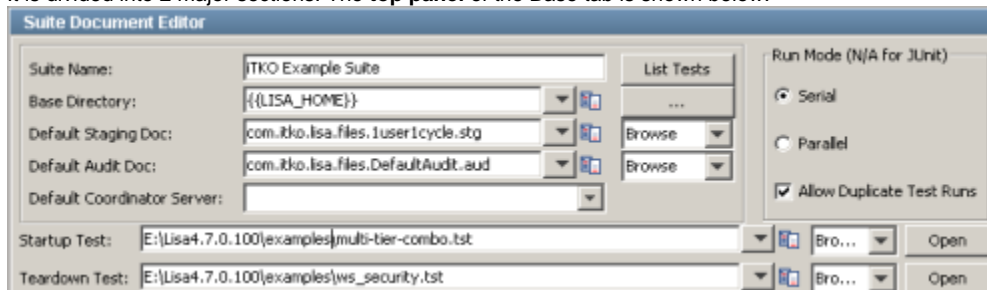
- 20.2.1 Test Suite - Base Tab
- 20.2.2 Test Suite - Reports Tab
- 20.2.3 Test Suite - Metrics Tab
- 20.2.4 Test Suite - Documentation Tab

20.2.1 Test Suite - Base Tab

20.2.1 Test Suite - Base Tab

The Base tab of the Suite Document editor is shown in the figure above.

It is divided into 2 major sections. The **top panel** of the Base tab is shown below:



This panel contains the configuration parameters for the suite, and the default values that are used for tests that do not specify their own values for these parameters.

To configure a Suite enter the following parameters:

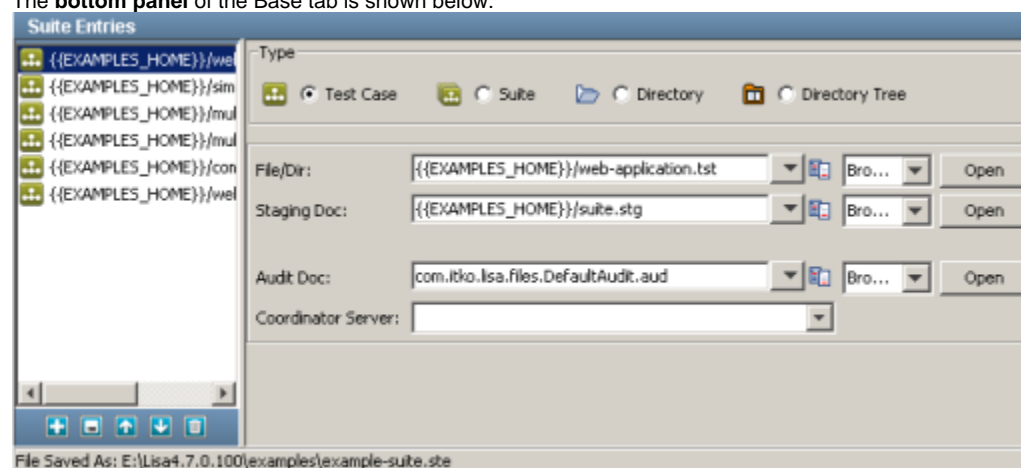
- **Suite Name:** The name given to this suite.
- **Base Directory:** The name of the directory that will be assumed to be the base directory for any individual test that does not contain a complete path, or, if LISA cannot find a test elsewhere it will look in this directory. If you do not specify a Base Directory, a default will be created for you when the suite document is saved. Enter a name, select from the pull-down menu or browse to the directory.
- **Default Staging Doc:** The name of the staging document to use if one is not specified for an individual test. Enter a name, select from the pull-down menu or browse to the document.
- **Default Audit Doc:** The name of the audit document to use if one is not specified for an individual test. This is initially assumed to be

com.itko.lisa.files.DefaultAudit.aud. This can be modified if desired. Accept the default, enter a name, select from the pull-down menu or browse to the document.

- **Default Coordinator Server:** The Coordinator Server to use if one is not specified for an individual test. Available Coordinator Servers are listed in the pull-down menu. This parameter is only needed if you are running a LISA Server.
- **Startup/Teardown:** These features are designed to allow you to run a startup test before your test suite runs, and a teardown test after your test suite has completed. This allows you to prepare the environment before the test starts, and to clean up after the test has finished. Neither the startup test nor the teardown test is included in any test statistics, and events in these tests will appear in the reports. Test suite testing will not continue if the startup test fails.
- **Run Mode:** Choose between running the tests one after another (serial), or starting all the tests at the same time (parallel).
- **Serial:** Tests will be run one after another in the order they show up in the suite document. This is the setting you would use for functional and regression testing.
- **Parallel:** Tests will be run at the same time. This is the setting you would use for Load and Performance testing. In this mode you must have enough virtual users to be able to run all tests concurrently!
- **Allow Duplicate Test Runs:** Allows you to choose if you want the same test to run more than once. Duplicate tests can occur in a suite if the same test appears in an included suite, a directory, or a directory tree. Click the box if you want duplicate tests to be run.
- **List Tests:** The List Tests button displays a pop-up window with a list of the tests currently included in your suite.

Note: You will have to save your suite for this list to be current.

The **bottom panel** of the Base tab is shown below:



This panel is where you build your suite by adding individual tests, existing suites, and directories and directory trees that contain tests. A list of the suite entries is displayed in the panel on the left. Suite entries are added individually by entering the following parameters:

- **Type:** Select the entry type as one of the following:
- **Test Case:** The individual Test Case name
- **Suite:** The suite document name. LISA will extract all the tests from this suite and run them as individual tests.
- **Directory:** The name of the directory that contains tests to be included in the suite. LISA will use the same staging document, audit document and Coordinator Server for all the tests in this directory. If new Test Cases are added to the directory, they will automatically be included in this suite.
- **Directory Tree:** The name of the directory that contains tests to be included in the suite. LISA will look in the named directory and recursively through all the sub-directories in the tree. LISA will use the same staging document, audit document and Coordinator Server for all the tests in this directory tree. If new Test Cases are added to the directory tree, they will automatically be included in this suite.
- **File/Dir:** The file name or directory name for this entry. Enter the name or select from the pull-down menu or browse to the file or directory. Once selected, click **Open** to open the respective file in the Workstation.
- **Staging Doc:** The name of the staging document for this entry. Enter the name, select from the pull-down menu or browse to the document. If you leave this entry blank, LISA will use the default staging document. Once selected, click **Open** to open the Staging document in the Workstation.
- **Audit Doc:** The name of the audit document for this entry. Enter the name, select from the pull-down menu or browse to the document. If you leave this entry blank, LISA will use the default audit document. Once selected, click **Open** to open the Audit document in the Workstation.
- **Coordinator Server:** The name of the Coordinator Server to use with this entry. Select from the list of available Coordinator Servers listed in the pull-down menu. This parameter is only needed if you are running a LISA Server.

Click the **Add** icon to add this entry to the list shown in the left panel.

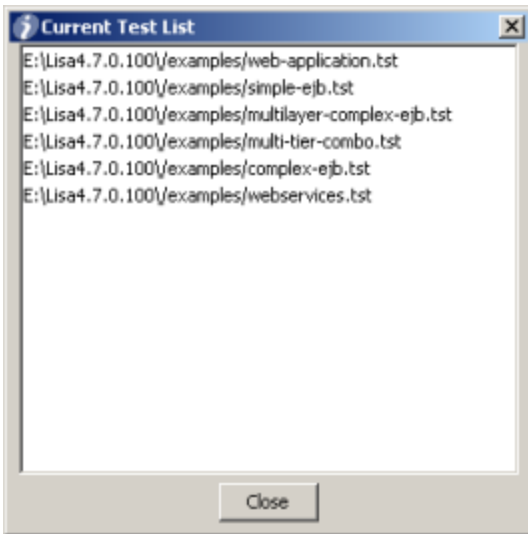
You can delete entries using the **Delete** icon, and re-arrange entries with the **Move Up** and **Move Down** icons.

Once you have entered all your test entries, save your suite document.

Click the **List Tests** button in the top panel to display a complete list of the individual tests in your suite.

Note: You will have to save your suite for this list to be current.

A sample listing is shown below:



All suites, directories and directory tree entries have been expanded in this view to show the tests explicitly.

20.2.2 Test Suite - Reports Tab

20.2.2 Reports Tab

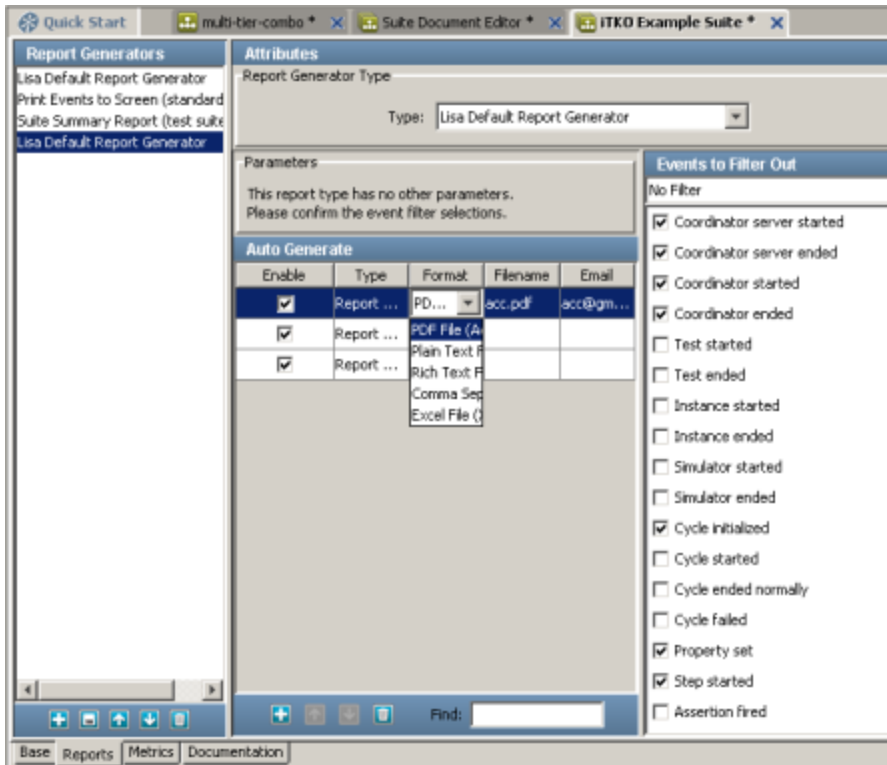
The Reports tab is where you specify the test reports you want to produce, and the events you want to capture at the test suite level. These reports are additional to any reports requested in a staging document. There are two types of report generators available in LISA.

The first generator stores the information in XML documents, (with some options allowing data to be stored in CSV files*, or standalone databases*. **See the list below**). **This generator has always been available in LISA. We will refer to these reports as *XML Reports.**

The second generator stores the requested event and metric information in a central embedded database that is installed as part of LISA. Reports are generated from this data when they are requested for viewing. The generator became available in LISA version 3.6, and is the recommended way to produce reports. We will refer to these reports as **Default Reports**.

In this section we concentrate on selecting the report types, and the events to capture for presentation in the reports. Details of the reports available in each report generator, viewing reports, report contents and output options are discussed in detail in [Reports](#). The metrics to capture for the reports are discussed in the next section, and more fully, in Using [Metrics](#).

The Reports panel is shown below:



There are four panels in this tab:

- **Report Generators** – A list of selected report types
- **Attributes** – A pull-down menu of available report types
- **Parameters** – Parameters required for the report
- **Events to Filter Out** – A list of all available LISA events

The **Attributes panel**, has a pull-down menu listing of all the available report types for both report generators.

The following report types are available, (the default report generator is shown in bold):

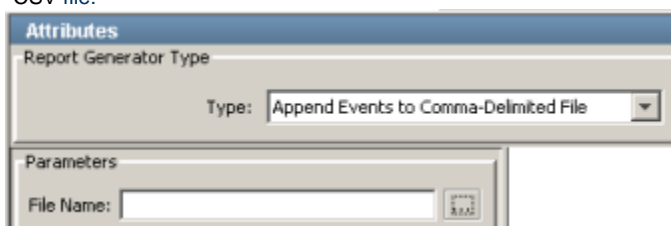
- *Append Events to Comma-delimited File
- Full Run Report (Large, Use with Caution)
- LISA5 Report generator
- LISA Default Report Generator (events and metrics written to central embedded database)
- Performance Report XML Document
- Print Events to Screen (standard out).
- Save Metric Data Intervals to XML Document
- **Report Performance Info to a Database:
- Save Metric Data Intervals to XML Document
- Suite Summary Report (test suite only)
- **Write Events to JDBC Database Table

The **Report Generator panel**, has a list of the reports that have been selected:

- To add a report, click the Add icon at the bottom of the list panel, and select the report type from the Type pull-down menu in the center of the Reports panel.
- To delete a report, highlight the report in the list, and click the Delete icon.

Note: If you choose a report type that requires a CSV file*, or a standalone database**, you will be prompted for additional information, as shown below for the CSV file, and the standalone database, (do not confuse this with the new default report generator database). This is a database of your choice, specified using the screen shown below:

CSV file:



Standalone Database:

Attributes

Report Generator Type

Type: Report Performance Info to a Database

Parameters

JDBC Driver: [Browse]

Connection URL: [Browse]

User: [Text Box]

Password: [Text Box]

The **Events to Filter out** panel, has a list of all the Events available in LISA. Each one has a checkbox. Using this list you can Filter out events you do NOT want captured in a report. To select Events for one of your reports, highlight the report and choose the events to Filter out by checking in the checkbox associated with that event.

For convenience there are 3 standard sets of events, listed in the pull-down menu above the Event list.

- Terse Event Set
- Common Event Set
- Verbose Event Set

These sets provide a pre-selected list of events that you can use as a starting list. You can then customize the list for each of your report types. Details of the reports available in each report generator, viewing reports, report contents and output options are discussed in detail in [24. Reports](#).

20.2.3 Test Suite - Metrics Tab

20.2.3 Test Suite - Metrics Tab

The Metrics tab is where you select the test metrics you want to record, and set the sampling specifications for the collection of the metrics. You can also set an email alert on any metric that you have selected.

The Metrics tab is shown below:

Quick Start AllTestsSuite

Metrics

Color	Short Name	Long Name	Scale	Edit Email Alert
Red	Event: Cycle failed/*	A LISA Event Metric: Event: Cycle failed/*	Auto	Set Alert
Red	Event: Step error/*	A LISA Event Metric: Event: Step error/*	Auto	Set Alert
Blue	Event: Step started/*	A LISA Event Metric: Event: Step started/*	Auto	Set Alert
Red	Event: Abort/*	A LISA Event Metric: Event: Abort/*	Auto	Set Alert
Magenta	LISA: Instances	A LISA built-in metric: number of running instances	Auto	Set Alert
Grey	LISA: Avg Resp Time (ms)	A LISA built-in metric: average response time in milliseconds	Auto	Set Alert
Cyan	LISA: Steps per second	A LISA built-in metric: Steps per second (non-quiet) since the test started	Auto	Set Alert

Find: [Text Box]

Sample rate: [Slider from Second to Daily, currently at 5]

Samples per interval: [Slider from 10 to 500, currently at 25]

Base Reports Metrics Documentation

File Saved As: E:\Lisa.Workstation5.0.16.28\examples\Suites\AllTestsSuite.ste

The Metrics tab is divided into 2 sections.

At the bottom of the panel are two slider bars that allow you to set your sampling parameters:

- **Sample Rate:** Specifies how often to take a sample, i.e. record the value of a metric. It is specified as a time period, and is the reciprocal of the sampling rate.
- **Samples Per Interval:** Specifies how many samples are used to create an interval for calculating summary values for the metric.

So, taking sample values every minute (Sample Rate Per Interval=1 minute), and averaging every 60 samples (Samples Per Interval=60), will produce a metric value every minute, and a summary value (average) every hour.

For example, the default is a 1 second Sample Rate, and 10 samples per interval (making an interval 10 seconds).

Our example above uses 5 seconds per sample, and 25 samples per interval, producing a metric value every 5 seconds and a summary value every 125 seconds.

Metric values are stored in XML files or database tables for inclusion in reports (see the previous section on reports).

To add a Metric

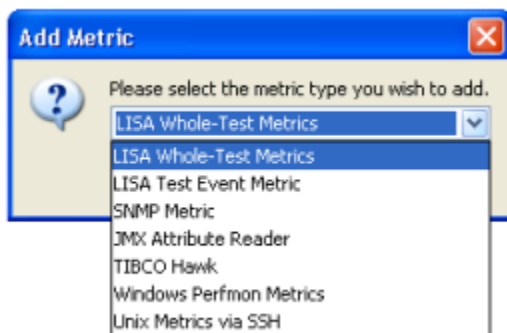
At the top of the Metrics panel is where you add or remove specific metrics, and set email alerts.

Following major metrics categories are available in LISA:

- LISA Whole-Test Metrics
- LISA Test Event Metrics
- JMX Attribute Reader (JMX metrics)
- SNMP Metrics
- TIBCO Hawk
- Windows Perfmon Metrics
- Unix Metric Via SSH

To add and delete metrics, use the Add and Delete icons at the bottom of this panel.

Click the **Add** icon to open a dialog box with a pull-down menu:



Select the desired metric and click **OK**. You are now ready to configure metrics from this category.

Descriptions of all the metrics, in all five categories, and details on how to configure them for inclusion in your reports, can be found in [Using 16. Applying Metrics](#).

Repeat the procedure to configure metrics from the other categories.

To set an email alert, on an individual metric click the **Set Alert** button corresponding to that metric. The following pop-up screen will be displayed:

Edit Alert

☐ Enable Alert

Acceptable Low Value:

Acceptable High Value:

☐ Authenticate

SMTP Username:

SMTP Password:

SMTP Port Number:

Email Subject Includes:

Email Msg Includes:

Email Server Name:

Email Recipient List

Email Address

Find:

On this screen, you can set the acceptable limits for the metric (low and high value), and the details to be sent in an email. You must provide the name of your email Server, and a list of email addresses.

Email addresses can be added and deleted by using the Add icon and Delete icon at the bottom of the window.

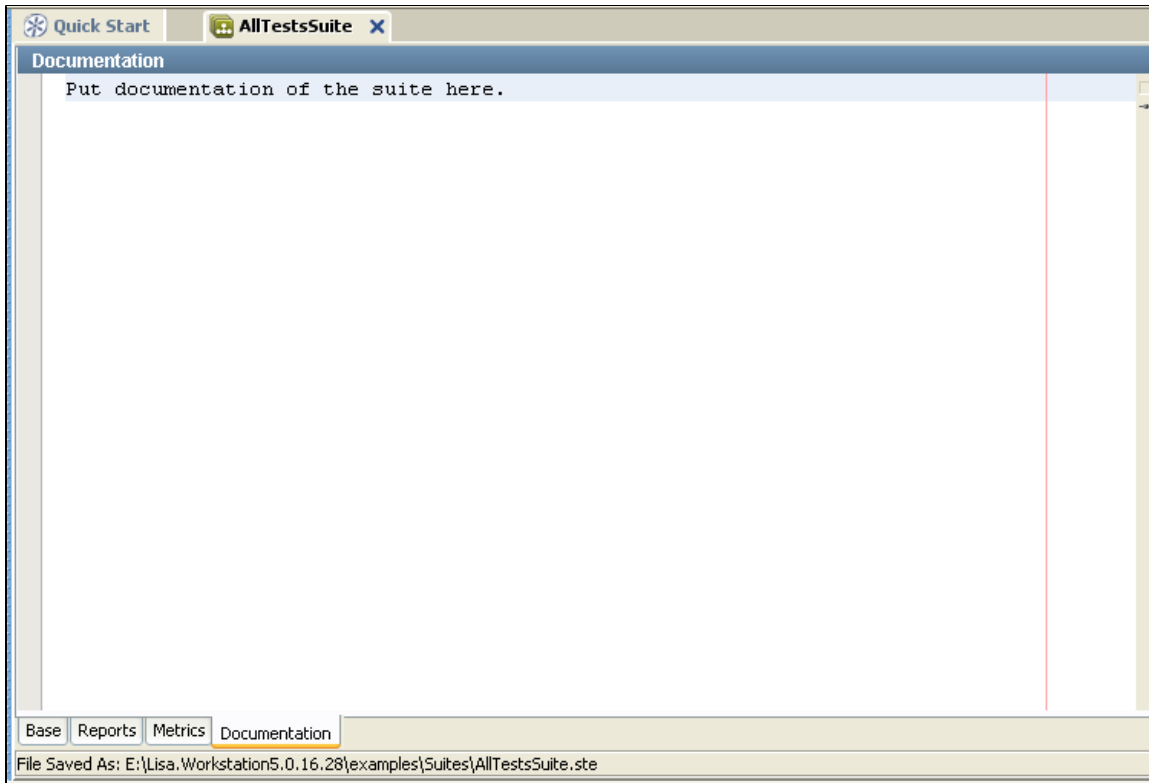
Click **Close** when finished.

Note: The **Set Alert** button is now an Edit Alert button. This is how to tell which metrics have alerts set.

20.2.4 Test Suite - Documentation Tab

20.2.4 Documentation Tab

The documentation tab allows you to enter the related documentation.



Enter the suite related documentation here.

21. Running a Test using ITR

21. Running a Test using ITR

The **Interactive Test Run (ITR)** facility allows you to walk through, and verify, a Test Case - step by step. You can therefore ensure that Test Cases are running correctly before staging them. The ITR runs the test that is currently in memory rather than loading a Test Case document. You can make changes in real time to alter test properties or the workflow.

For instance, you can choose to run a particular test step out of the natural workflow sequence if you wish. The real time changes that you make as the test is running will be integrated into the Test Case, and it will continue running without requiring a restart, unless the change was global in nature, like changing the Active Configuration. So, let's assume you execute a step and get an unexpected result, perhaps because you provided a bad parameter. You can leave the ITR open, edit the offending test step, then go back to the ITR and execute the same step again.

The ITR allows you to save the current ITR state, and/or load an ITR state that was previously saved. This allows you to communicate an error, in context, to another member of your team. You can send the Test Case and the saved ITR state to provide them with the exact actions, and the results that you observed.

The following topics are available.

- [21.1 Running the Interactive Test Run \(ITR\)](#)
- [21.2 Interpreting the Results in an ITR Run](#)
- [21.3 Graphical Diff Utility](#)

21.1 Running the Interactive Test Run (ITR)

21.1 Running the Interactive Test Run (ITR)

If your Test Case is not open in Workstation, open it now see [Creating Test Cases](#).

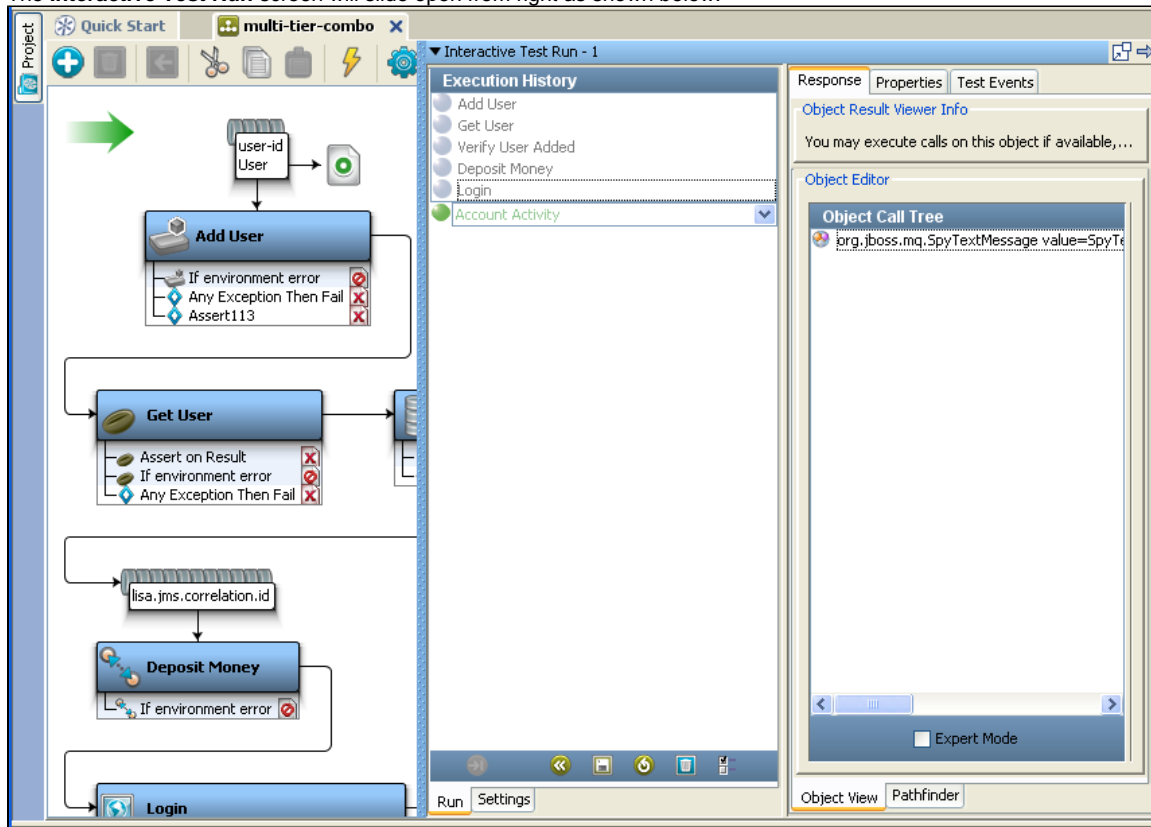
For the purpose of illustration, we will look at the **multi-tier-combo** Test Case in the examples directory (multi-tier-combo.tst).

To start the ITR,

Click the  on the toolbar:

Or from the main menu, select **Actions > Start a New Interactive Test Run**,

The **Interactive Test Run** screen will slide open from right as shown below:



We have selected the multi-tier-combo test case in the Projects pane.

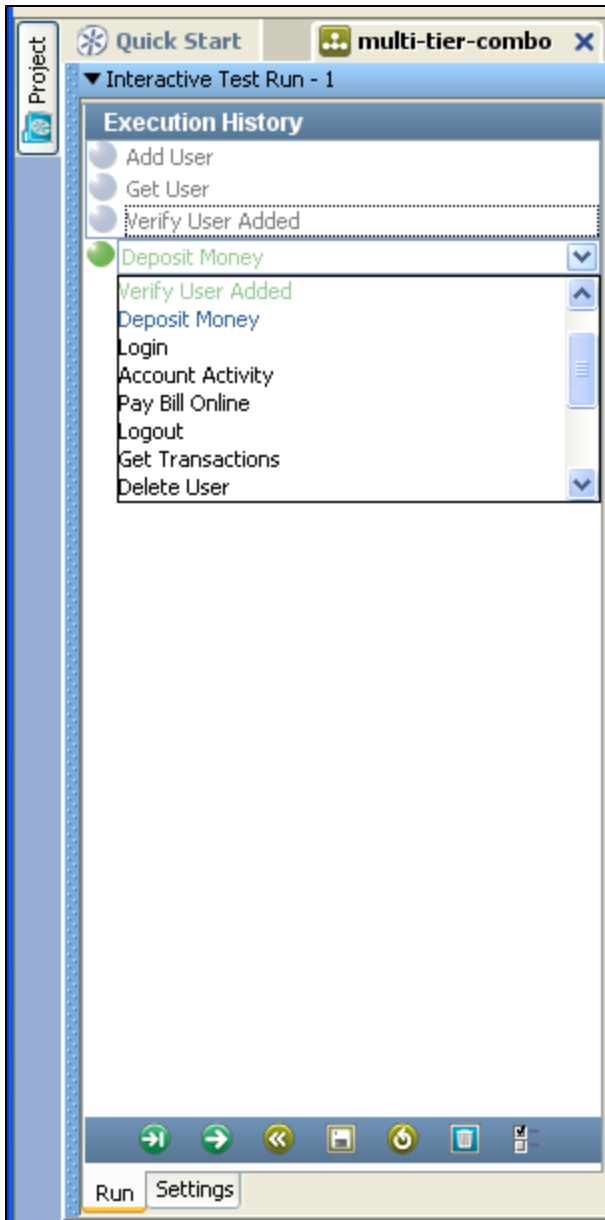
The **Test Case workflow** for the **multi-tier-combo** Test Case can be seen in the Model Editor.

When you click on the ITR icon, the ITR panel slides in from the right hand side as shown above.

There are two panels that make up the actual ITR facility:

Execution History Panel

The left panel is the **Execution History** panel. This shows the steps that have been executed (in Grey), and the next step that will be executed (in Green).



Here, every step will have a pull-down menu containing a list of the steps in the current Test Case. This is where you can change the next step to execute.

In the figure above, the ITR has been opened and the Test Case is run to complete first 3 steps. You will see the fourth step (Deposit money), which is about to be executed is shown in Green. The steps yet to be executed are shown in Green.

Below that is the [ITR toolbar](#). The icons in the toolbar are described below.

There are two tabs at the bottom of the panel.

The [Run tab](#) opens by default and is the one we are currently displaying. The [Settings tab](#) is described later in the section.

Step Information Panel

The right panel is the **Step Information** panel.

This panel displays information about the step that is selected (highlighted) in the Execution History panel. There are three tabs at the top of the panel: Response, Properties and Test Events.

Execution History

- Add User
- Get User
- Verify User Added
- Deposit Money
- Login
- Account Activity
- Pay Bill Online
- Logout
- Get Transactions
- Delete User
- Verify User Deleted
- End the Test

Response

Key	Value	Previous Value
EJBSERVER	localhost	localhost
LISA_LAST_STEP	end	Verify User Deleted
lisa.hidden.ejbbobj...	jboss.j2ee:jar=itko-e...	jboss.j2ee:jar=itko-e...
lisa.ws.current.oper...	deleteUser	deleteUser
LISA_DOC_PATH	C:\Lisa5.0.23.145\ex...	C:\Lisa5.0.23.145\ex...
JMSCONNECTIONFA...	ConnectionFactory	ConnectionFactory
lisa.Add User.rsp	<?xml version="1.0" ...	<?xml version="1.0" ...
lisa.Login.http.respo...	200	200
ENDPOINT1	http://localhost:8080...	http://localhost:8080...
lisa.Deposit Money.li...	Integrator of type co...	Integrator of type co...
lisa.msg.sharingEngi...	SharingEngines [Size...	SharingEngines [Size...
lisa.end.rsp	The test has ended	
lisa.Delete User.lisaint	Integrator of type co...	Integrator of type co...
lisa.Delete User.lisain...	<?xml version="1.0" ...	<?xml version="1.0" ...
lisa.Add User.httphe...	HTTP/1.1 200 OK=8...	HTTP/1.1 200 OK=8...
User_RowNum	1	1
lisa.Verify User Adde...	172	172
lisa.hidden.jms.jnp:/{...	NotSerializableState...	NotSerializableState...
lisa.hidden.ejb.3066...	SerialNum=330, of cl...	SerialNum=330, of c...
lisa.Delete User.http...	HTTP/1.1 200 OK=8...	HTTP/1.1 200 OK=8...
lisa.Deposit Money.rsp	SpyTextMessage {He...	SpyTextMessage {H...
lisa.Login.lisaint	Integrator of type co...	Integrator of type co...
lisa.ws.connprops	org.apache.axis.com...	org.apache.axis.com...
lisa.Pay Bill Online.ht...	200	200
order.step.2.queue	queue/C	queue/C
lisa.Logout.lisaint.xml	<?xml version="1.0" ...	<?xml version="1.0" ...
JNDIEXPORT	1099	1099
LISA_COOKIE_localh...	JSESSIONID=20EA3...	JSESSIONID=20EA3...

These three tabs are explained in the next topic [Interpreting the Results in an ITR Run](#). For every test step run, there will be different tabs at the bottom of the panel Ex: View, Source, DOM tree and Pathfinder etc.

Each of these tabs contains information about the selected step and describes the state of the selected step. If the selected step has not been run, there will be less information in the tabs than when the step has been executed.


There are two icons at the bottom of the right panel, that are used to **Save the ITR State** and **Launch the Test Case in External Browser**.



ITR Toolbar

You manage the ITR using the icons in the toolbar at the bottom of the Execution History panel:



To execute the step shown in the Execution History list, click the **Execute Next Step**  icon. Once the step has executed, you can examine the results from the step execution using the tabs in the right panel (see section below). Click the **Execute Next Step** icon again to execute the next step listed, or choose a different step to execute as described above.

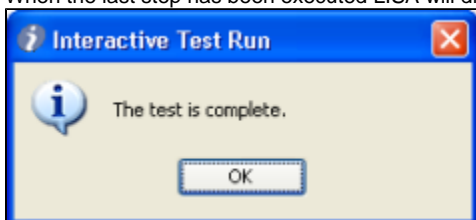


To execute all the steps automatically, click **Automatically Execute Test** icon. This will run the Test Case to completion and end. While the test is running, the Automatically Execute Test icon will change into a Stop icon. You can stop the test by clicking the Stop icon.



To restart the test all over again, click the Restart test icon. This is useful if you have made a change to your Test Case and want to execute it from the beginning.

When the last step has been executed LISA will display a **test complete** dialog box:





To save the **current ITR state**, click the Save icon. In the dialog box, enter the name of the save file. The suffix **.itr** will be appended to your name. Subsequent saves will overwrite this file.



To Load a saved ITR, click on the **Load Saved ITR State** icon. and browse to the .itr file you wish to load. The Saved ITR State will contain the information displayed in the tabs, thereby capturing all the testing performed. To make use of the Saved State, you must first open the Test Case that was used when the original tests were run.

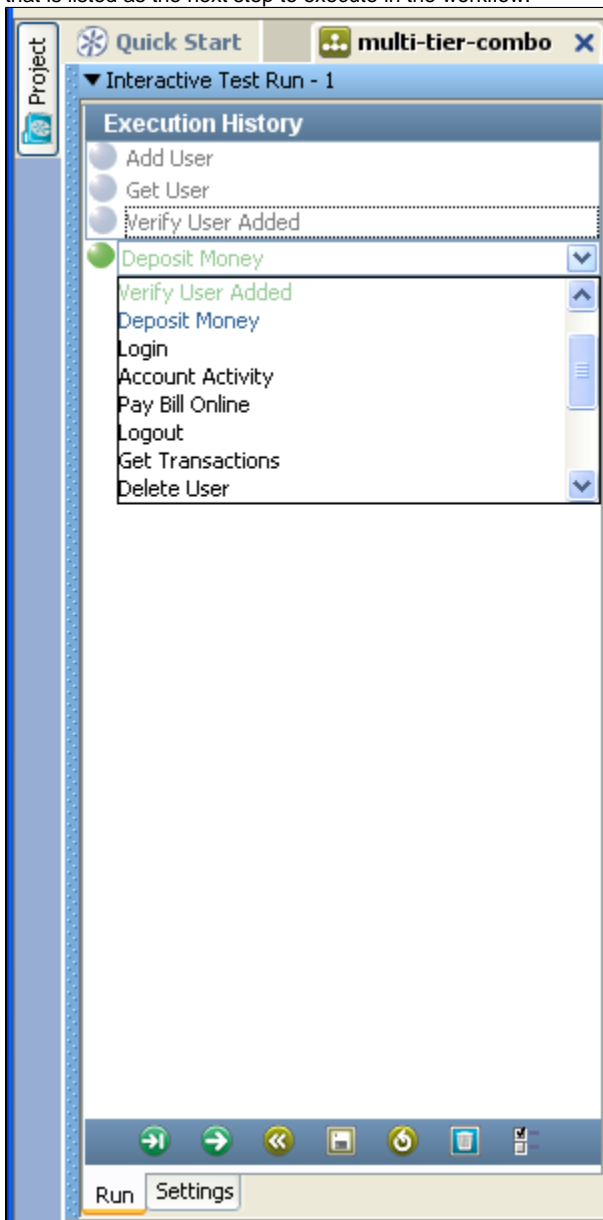


Click on the **Add Properties to Watch window** icon, to open the ITR property watch window to add and/or watch properties.

The ITR pauses briefly between each step execution. To change the pause interval between steps, go to the **Settings tab** at the bottom of the panel and use the slider to adjust the **Auto Run Delay** (secs), as described below in the Settings Tab section. Think time is not honored in the ITR.

Run Tab

In the figure below, the **Execution History** panel shows a Test Case in which three steps have been executed (in Grey) and a fourth (in Green), that is listed as the next step to execute in the workflow:

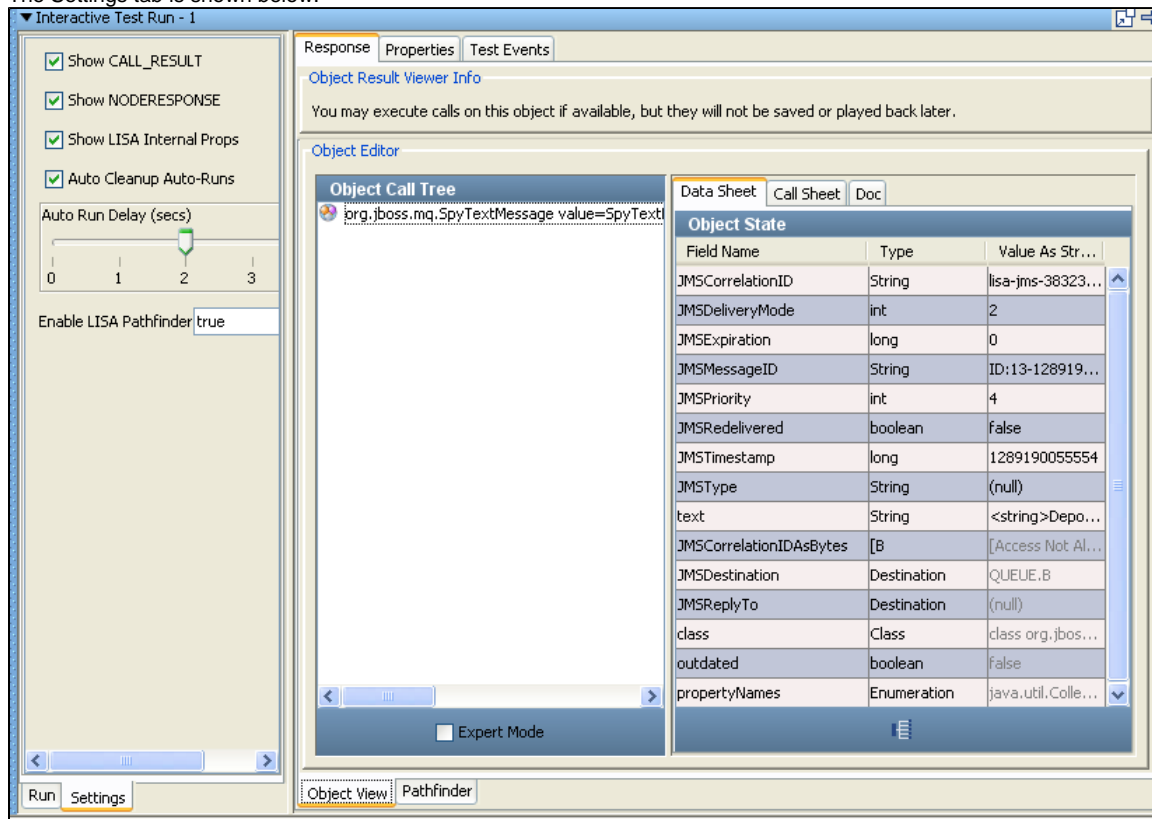


To change the next step, use the **Change Next Step** pull-down as shown below. This will replace the current next step with one of your choice. Once you change the step, the workflow will continue from the new step.

Settings Tab

The Settings tab allows you to change some of the ITR settings.

The Settings tab is shown below:



The four checkboxes allow you to Filter out events and properties from the results tabs. There are some verbose events that you may not want to see:

- **Show CALL_RESULT**: Click to include EVENT_CALL_RESULT events in the Test Events list.
- **Show NODERESPONSE**: Click to include EVENT_NODERESPONSE events in the Test Events list.

Likewise, you may not always want to clutter the property list with LISA internal properties:

- **Show LISA Internal Props**: Click to include all LISA internal events in the property lists in the Initial State and Post Exec State tabs.
- **Auto Clean up Auto Runs**: Click to clean up the Auto Runs.

The **Auto Run Delay (secs)** slider allows you to adjust the wait time between each step execution in the Automatically Execute Test mode. Think time is not honored in the ITR, so this setting allows you to add a constant delay between each step execution.

- **Enable LISA Pathfinder** - Select **true** to enable LISA Pathfinder or **false** to diable it. Once enabled you can see the Pathfinder tab opened as shown above.

21.2 Interpreting the Results in an ITR Run

21.2 Interpreting the Results in an ITR Run

You can examine the results of the execution of a particular test step, either during the run or between two steps or at the conclusion of the run.

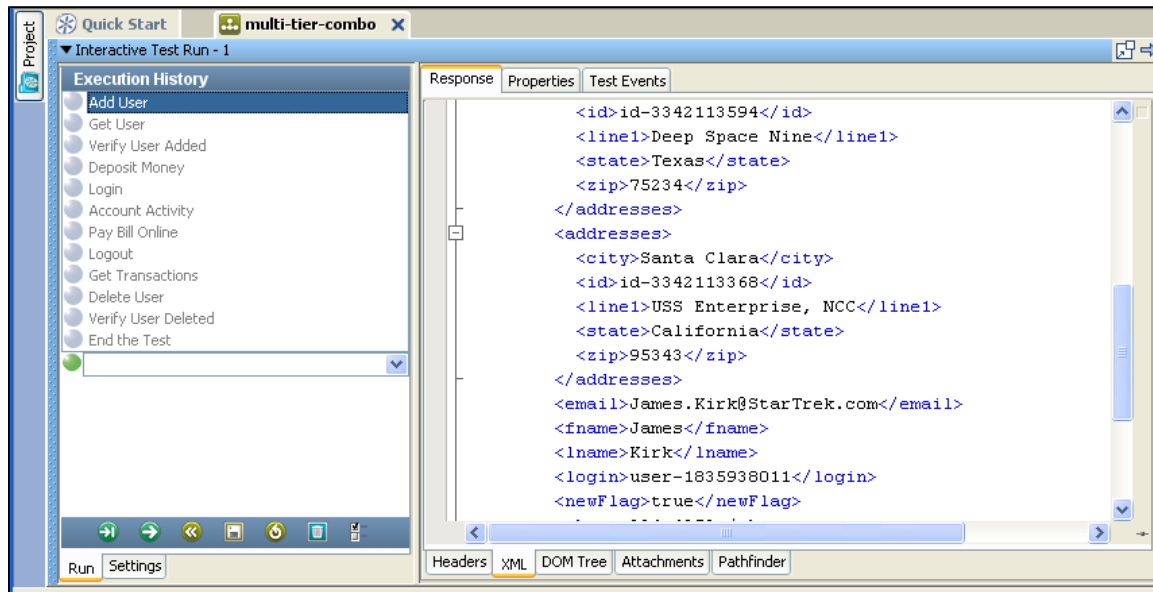
To do so, highlight the step in the Execution History list and examine the information in each of the three tabs ([Response Tab](#), [Properties Tab](#), [Test Events Tab](#)) in the right-hand panel.

Response Tab

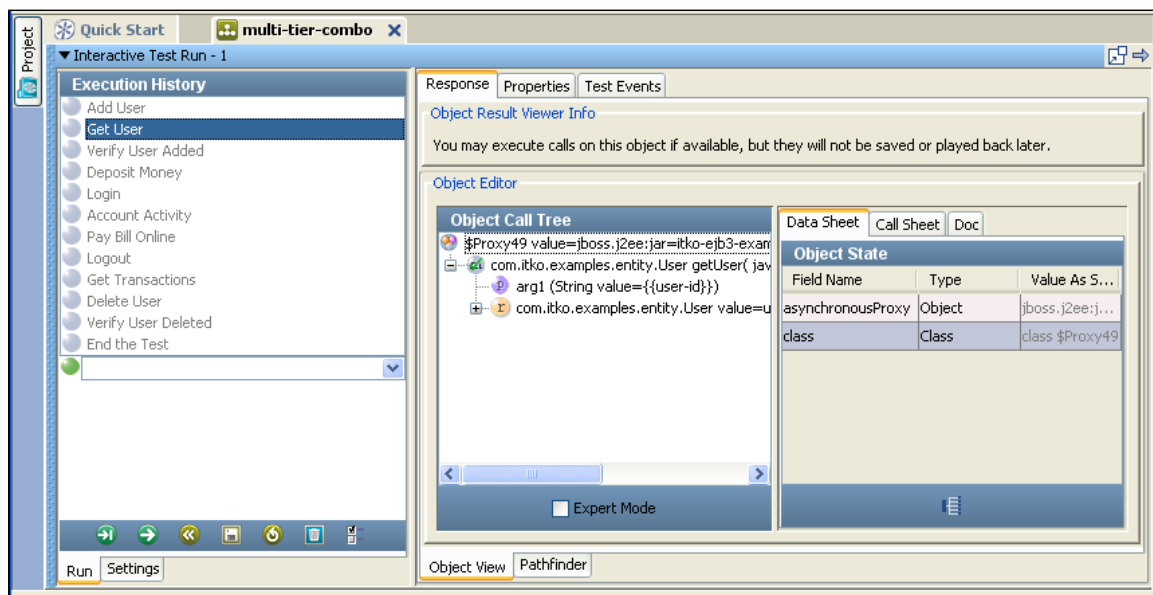
The Response tab displays the step response after executing a step.

The display will use the editor appropriate for the response type.

For example, the **add user** step invokes the html/xml response:



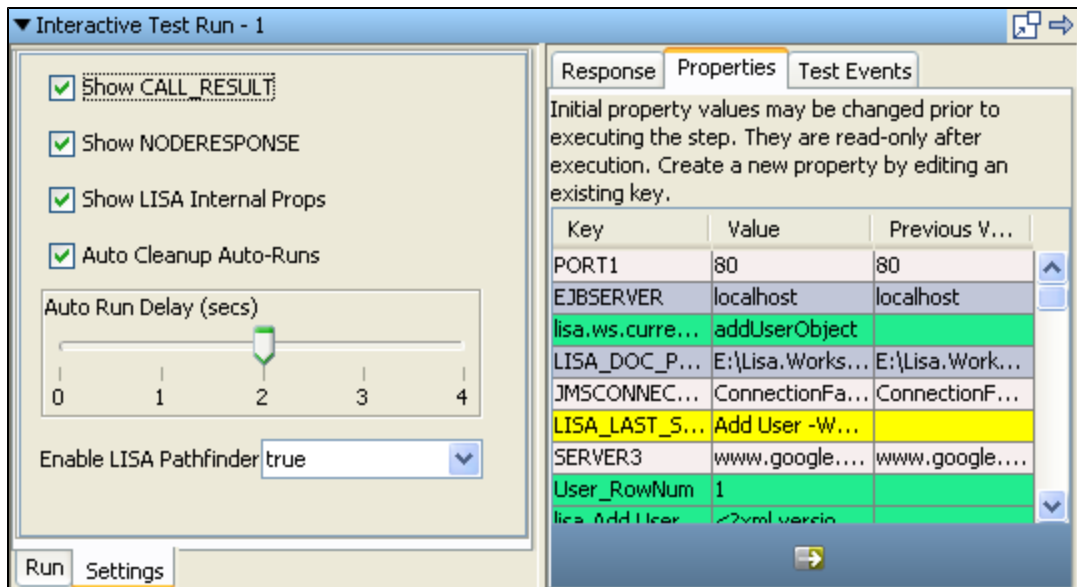
Similarly, the **get user** step invokes an EJB that returns an object that is displayed in the Complex Object Editor:



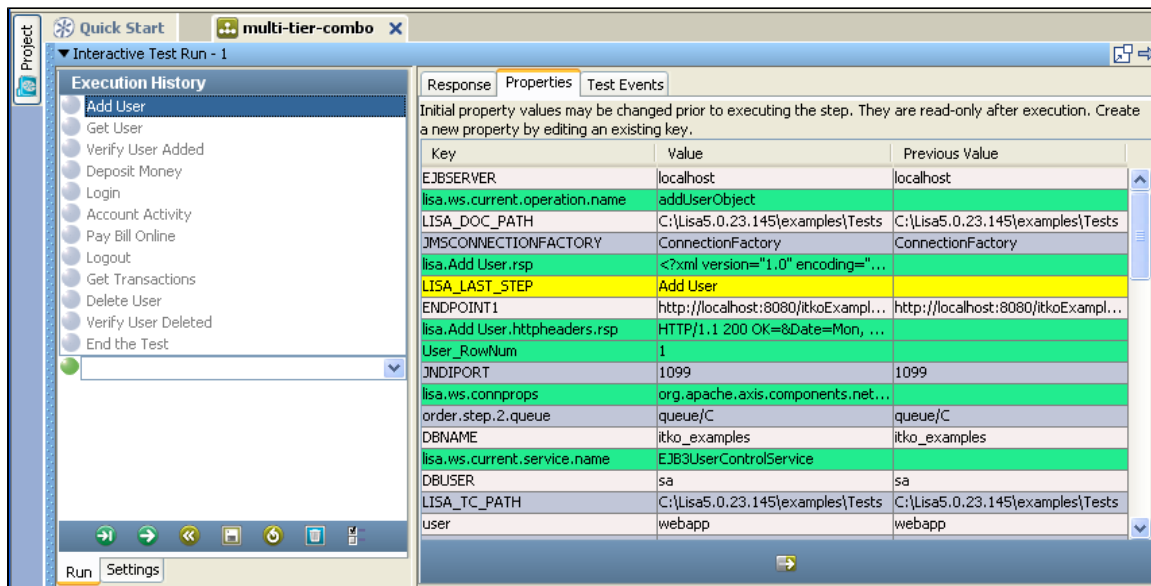
Properties Tab

The Properties tab displays the state of the step after execution (Value) and immediately before execution (Previous Value).

To show/hide LISA internal properties, check/uncheck the **Show LISA Internal Props** box in the Settings Tab as shown below:



LISA **Properties** that were newly set by the step are highlighted in 'Green'.
Properties that were **modified** in the step are highlighted in 'Yellow' color.

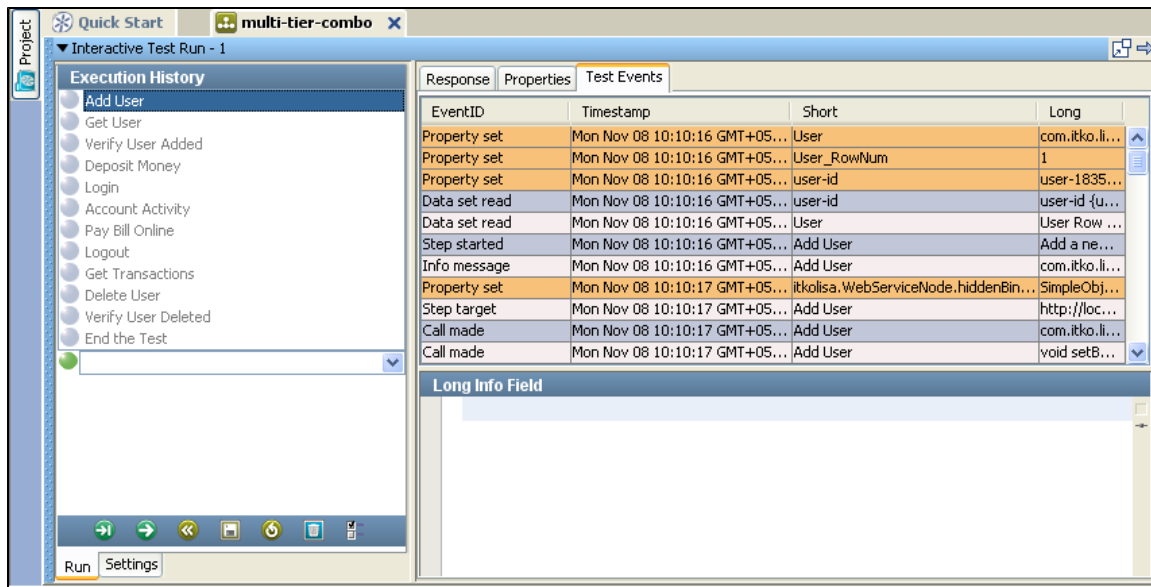


Test Events Tab

The Test Events tab displays the events that were fired when the step was executed, in the order that the events were fired.

To display/hide the events, check/uncheck the Event boxes in the **Settings** tab of the Execution History panel as shown above. (Show CALL_RESULT, Show NODE_RESPONSE)

All Events have meaning full names. For Ex: "**Properties set**" events are highlighted in 'Orange'.



The information provided in the Test Events tab is:

- **Event ID:** The ID of the event that fired.
- **Timestamp:** The time the event fired.
- **Short:** The Short description of the event.
- **Long:** The Long description of the event.

You can enter the complete or full description of the Event in the **Long Info Field**. The Long description field can be truncated in the display.

To view the full text, click on the cell and its full contents will be displayed in the Long Info Field panel above.

Note: An Event is generated on all Assertions that are fired, as well as evaluated.

For more details on Event ID, Short and Long see the [LISA Reference Guide](#).

The example in the Events list, a failed test is shown in Red color and an Assertion event is highlighted in 'lilac' color.

21.3 Graphical Diff Utility

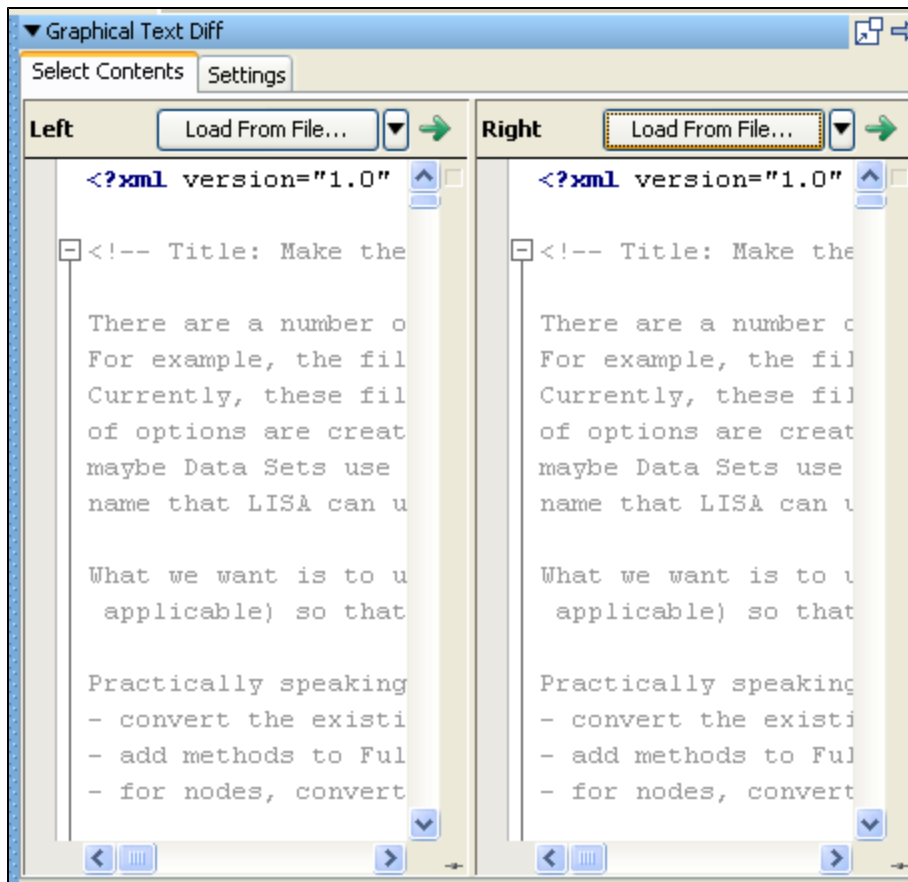
21.3 Graphical Diff Utility

In LISA 5.0, a new graphical XML diff engine and visualizer has been implemented. The new engine is an improvement over XmlUnit, the engine used in LISA until now for XML differencing. You will be now be able to see differences in a graphical viewer.

To start the Diff viewer from the Main menu,

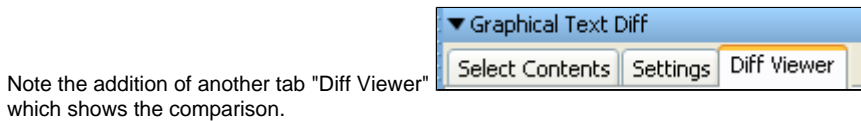
From the main menu, select **Actions > Graphical Text Diff**

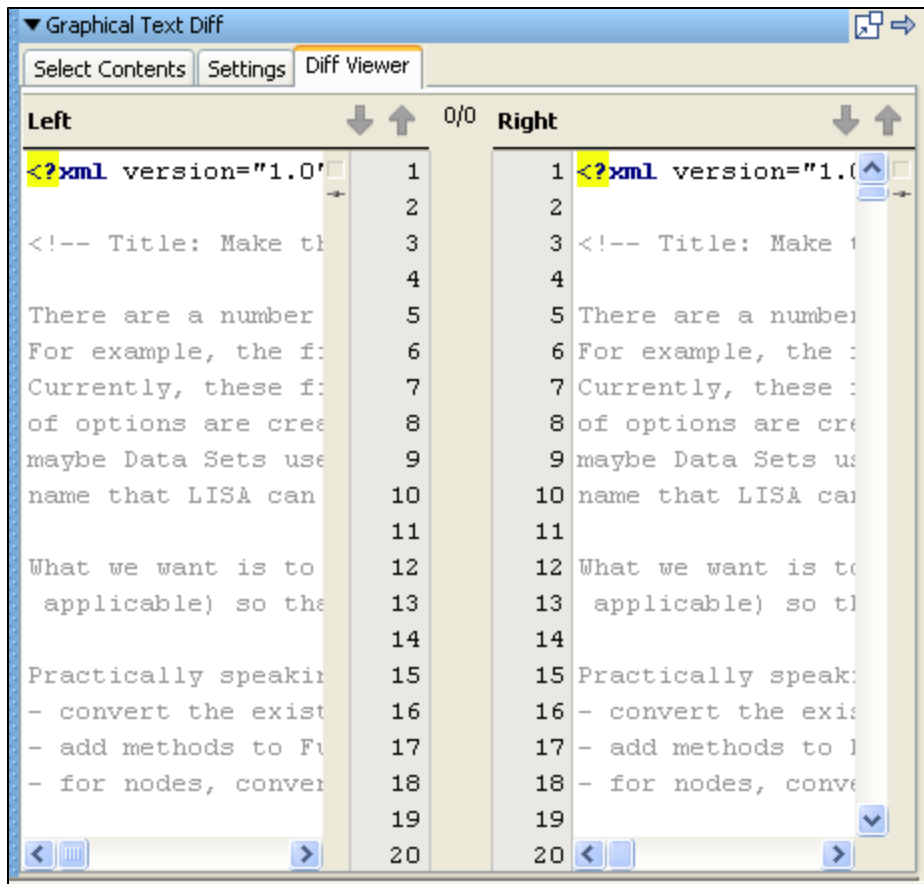
Here, you can compare the contents of the left and right xml files.



Click on the Compare icon  to start the Compare.

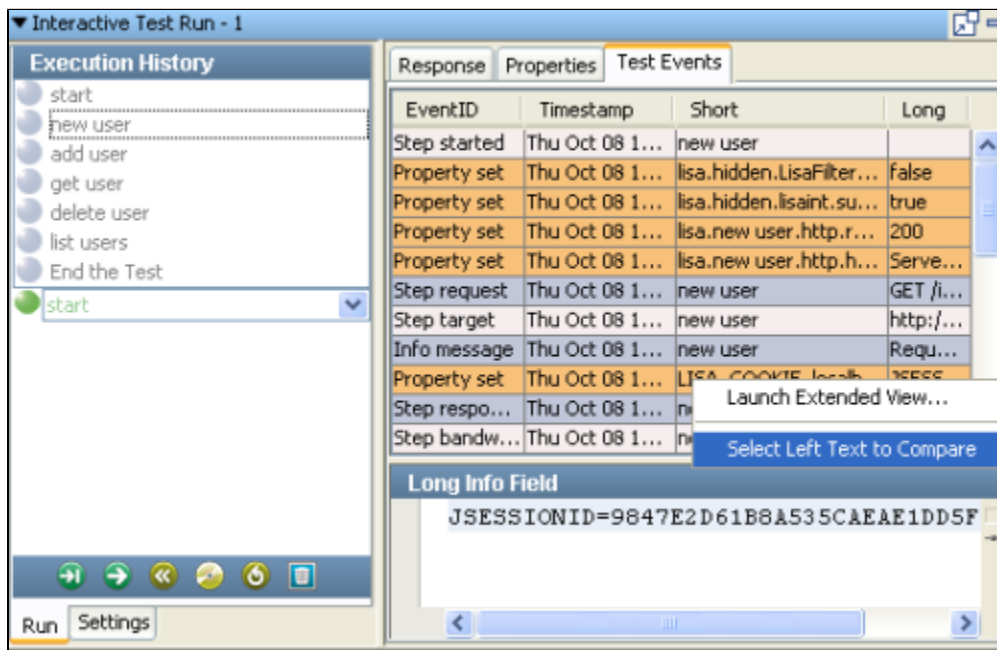
The following screen appears which shows the actual comparison.



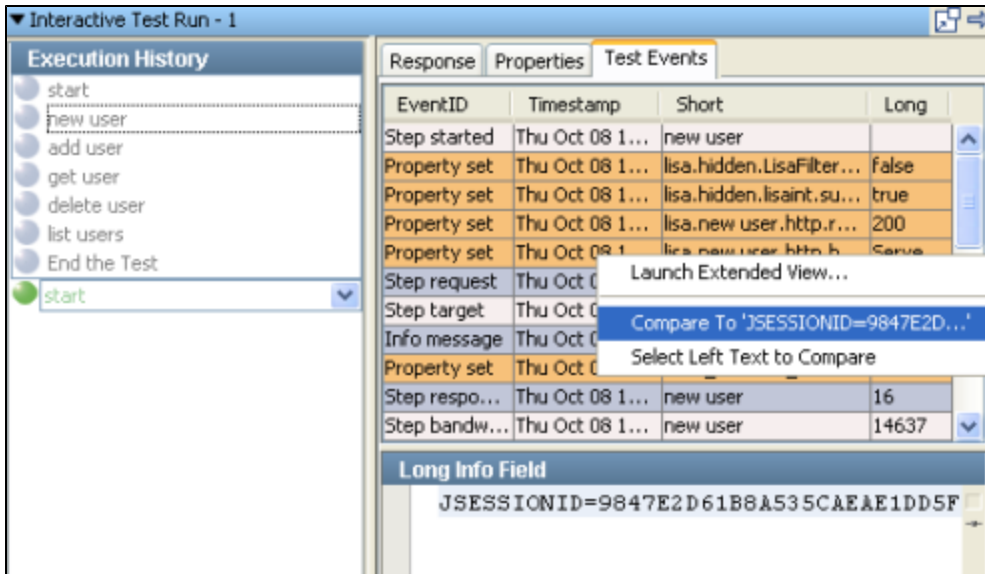


To start the Diff viewer from the ITR,

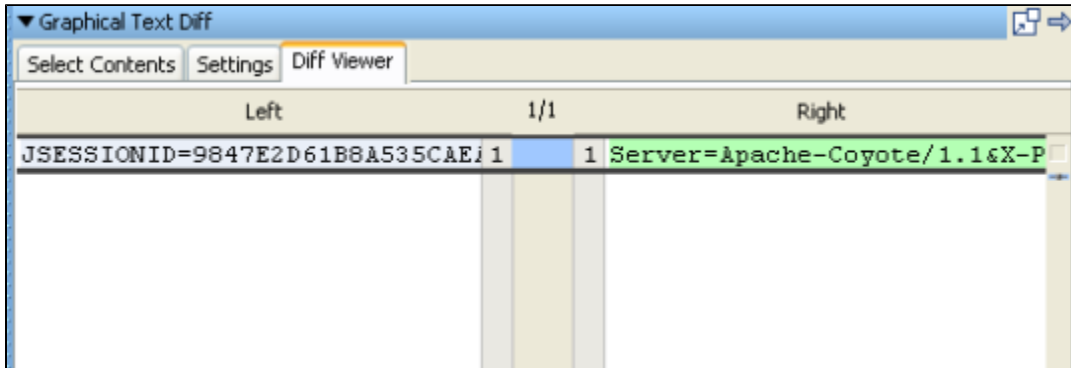
- Select the **Test Events** tab in the ITR,
- Right click a table row to open the following menu:



- Click "Select Left Text to Compare" option.
- Click on a second row to compare with and right click to open the same menu with added option.

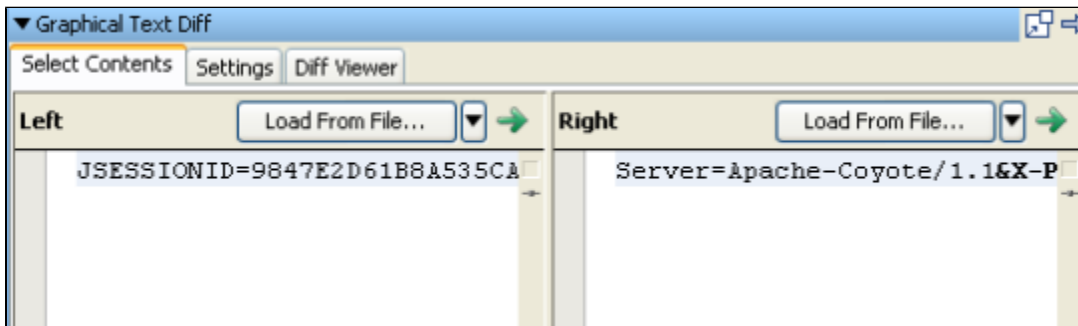


- You can see "Compare to..." menu option here.
- Click the "Compare to..." option
- This will open the **Graphical Text Diff** for comparison as shown below:.



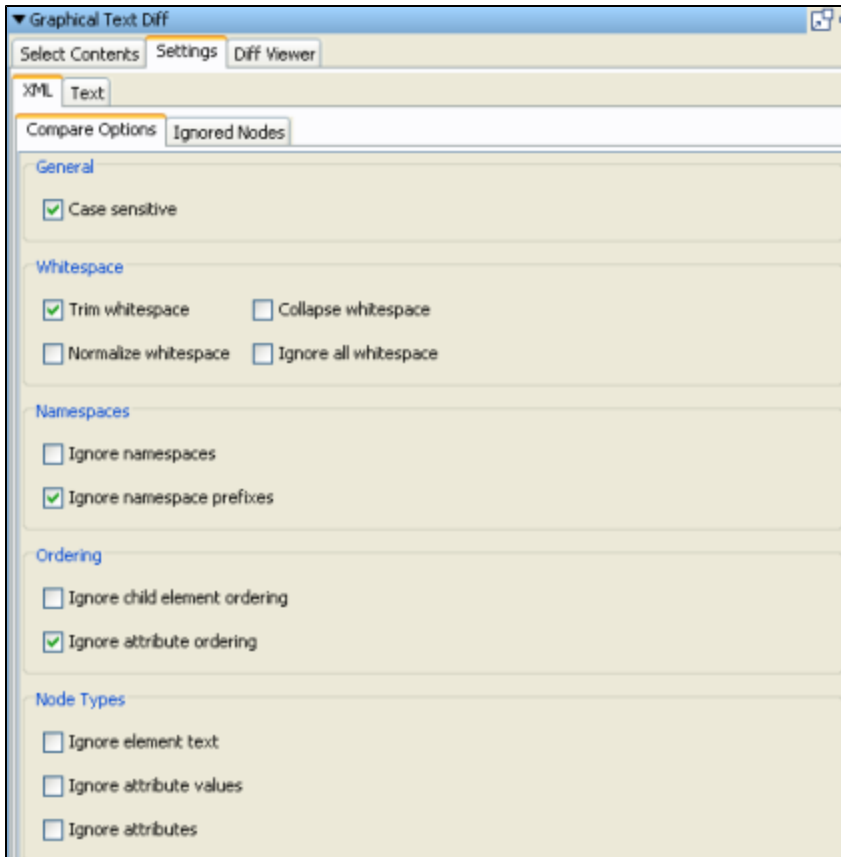
You can also load the contents by selecting a File.

Click the "**Select Contents**" tab and click "Load from File" to load the file.



The results of the diff are then displayed in the global tray panel component.

You can also set the compare options in the "**Settings**" tab as shown below:



PART 5 - Viewing LISA Portals

Part 5 - Viewing LISA Portals

Main functionality of LISA is used to combine different test cases, apply certain rules and make them run.

LISA also provides functionality to tweak test cases and monitor the running of individual or group of test cases. You can also view the changes occurring post a test case run.

The utilities are -

LISA Pathfinder - Pathfinder is a utility in LISA, which allows you to **probe** into the system under test, to examine the components behind the initial request or method call.

CVS Dashboard - The Continuous Validation Service, CVS Dashboard utility, within LISA, allows you to schedule Tests and Test suites to run on a regular basis, over an extended time period and monitor them.

LISA Reports - LISA Reporting tool, has a wealth of features related to the Generation and Capture of data for the purpose of analyzing results.

From LISA 5.0, all these utilities in LISA are web enabled, i.e.; they can also be accessed through the web.

Note - Like previous versions, all these utilities can also be accessed through LISA workstation.

In this section, the following topics are covered:

- 22. [Pathfinder](#)
- 23. [CVS Dashboard](#)
- 24. [Reports](#)

22. Pathfinder

22. Pathfinder

Pathfinder is a utility in LISA, which allows you **to probe** into the system under test, to examine the components behind the initial request or method call.

For example, when making a request to a web page, you would like to be able to follow the request through the execution chain, and then follow the response, or results back out. Pathfinder allows you to exactly do that and it also collects important information along the way.

On some application Servers you may need to configure a web service handler, usually by adding a one line annotation in your java code. That is it! LISA then takes advantage of built-in mechanisms in your application Server to perform the instrumentation of your code.

The Pathfinder requires a [Registry](#) to run, which will invoke the broker and communicate all transaction related details to the Derby database.

For agent-based Pathfinder, the Registry maintains the database of Paths that are captured along with other information.

Note - For testing purpose, please start the LISA demo server which has agent installed or run the LISA bank application.

For running with any other Java application, you will need to configure the Lisa Agent.

For more information on Agent, pl see the [LISA Agent guide](#).

The Pathfinder database is located at **LISA_HOME/database/pathfinder.db**

From LISA 5.0, Pathfinder can also be opened via a web portal.

To open the Portal, click <http://localhost:1505> in a web browser.

Note - Registry needs to be running to start the Portal.

Prerequisites

The LISA Agent is a **must** to run LISA Pathfinder utility.

Note - In the LISA Bank application, Agent is installed by default.

For any other application, you will need to [install the Agent](#) and [start the Registry](#).

For details on how to configure common application Servers, see [LISA Installation Guide](#).

Pathfinder and Pathfinder Pro Licensing

Pathfinder and Pathfinder Pro are licensed separately

Pathfinder allows you to search for transactions and paths.

Pathfinder Pro adds the Dataset / Baseline / Virtualize functionality, as well as the Defect Tracking.

Please see LISA Agent Guide for more information.

Activating Pathfinder

Pathfinder can be activated only after you have installed LISA Agent.

You also must do some transactions within the application server, so that they can be seen and traced within the Pathfinder Console.



- Within LISA Workstation, click on the Pathfinder Console icon to start the Pathfinder Console.
- Or Run <http://localhost:1505/pathfinder/> in your browser to open the LISA pathfinder portal.

De-activating Pathfinder

For those, not using the Pathfinder agents, can turn off all Pathfinder support.

You can disable the Pathfinder by modifying the **lisa.properties** file.

In lisa.properties file, modify the following text:

```
lisa.pathfinder.on=true
```

Change 'true' to "false" in the above line, to disable the Pathfinder.

The following topics are available.

22.1 Exploring Pathfinder
22.2 Pathfinder Console
22.8 Integrating Pathfinder Broker and Portal Server
22.9 Pathfinder Summary

22.1 Exploring Pathfinder

22.1 Exploring Pathfinder

It is much easier to explain and explore pathfinder by making references to an actual example.

For the purpose of illustration, we will take a look at the LISA Bank Application.

Note - The LISA Bank application needs the Demo Server running.

For more information on LISA Bank, please see [Tutorial 3.6 Run LISA Bank Application](#).

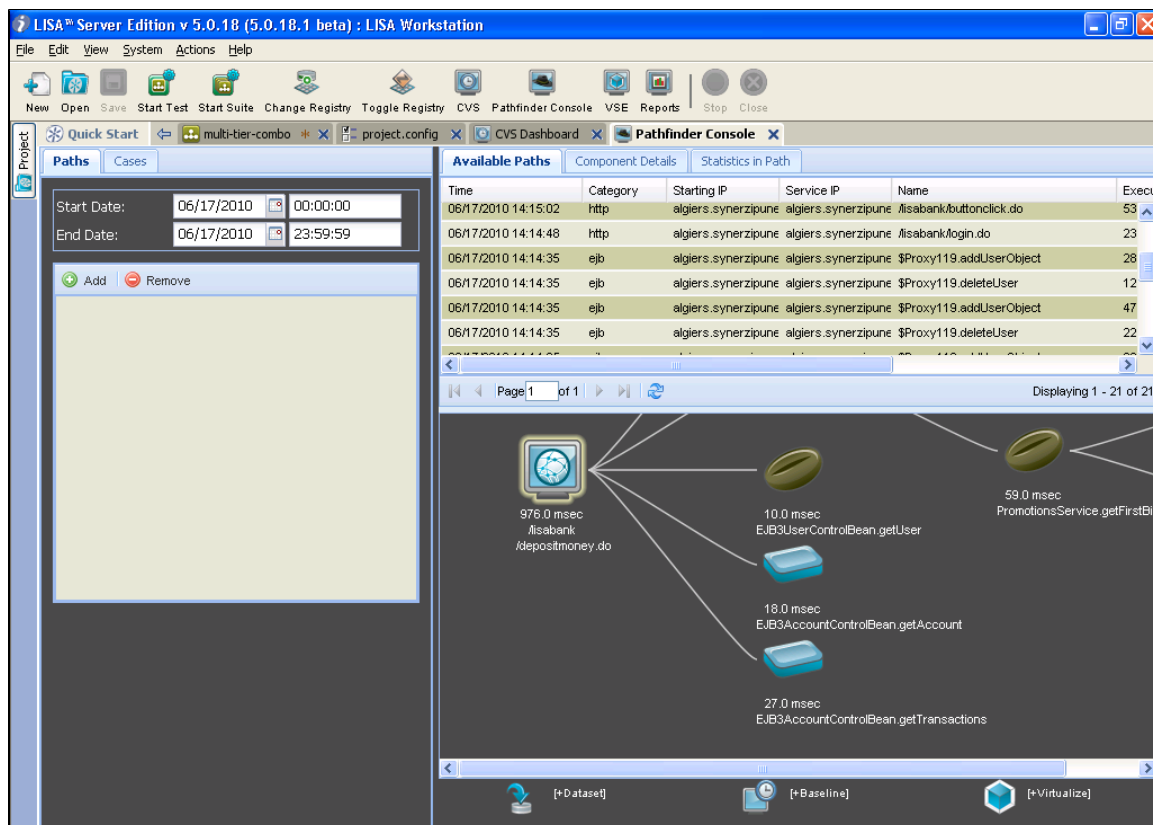
- Start the Demo Server and a web browser.
- Enter <http://localhost:8080/lisabank> to open the LISA Bank application in the browser. (Replace localhost with your machine IP address)
- Login using the following credentials
 - **Login** -> lisa_simpson
 - **Password** -> golisa
- Do some transactions within the LISA Bank application like - opening an savings account, depositing money or withdrawing money from account etc.
- Logout from LISA Bank application.

You can open the Pathfinder console in two ways:



By clicking the Pathfinder Console icon within LISA Workstation.

This will open the Pathfinder Console within LISA as shown below:

The screenshot shows the LISA Workstation interface. The title bar reads "LISA™ Server Edition v 5.0.18 (5.0.18.1 beta) : LISA Workstation". The menu bar includes File, Edit, View, System, Actions, and Help. The toolbar contains icons for New, Open, Save, Start Test, Start Suite, Change Registry, Toggle Registry, CVS, Pathfinder Console, VSE, Reports, Stop, and Close. The main window has a tabbed interface with "Quick Start", "multi-tier-combo", "project.config", "CVS Dashboard", and "Pathfinder Console" selected. The "Pathfinder Console" tab is active, showing a table of available paths. The table has columns for Time, Category, Starting IP, Service IP, Name, and Execu. The data rows show various HTTP and EJB requests. Below the table, there is a diagram showing the execution flow of a transaction, with nodes representing different components and their execution times. The diagram shows a sequence of calls: /lisabank /depositmoney.do (976.0 msec) calls EJB3UserControlBean.getUser (10.0 msec), which calls EJB3AccountControlBean.getAccount (18.0 msec), which calls EJB3AccountControlBean.getTransactions (27.0 msec). There is also a call to PromotionsService.getFirstBill (59.0 msec). The bottom of the window has a status bar with "[+Dataset]", "[+Baseline]", and "[+Virtualize]" buttons.

You can also open Pathfinder Console by entering <http://localhost:1505/pathfinder> in a web browser.

This will open the **Pathfinder Console** in a web browser and will show all the transactions done in the LISA Bank application as shown below:

The screenshot shows the LISA Pathfinder Console interface. The top panel displays a list of transactions under the 'Available Paths' tab. The table below shows the data:

Time	Category	Starting IP	Service IP	Name	Execution Time
06/08/2010 12:06:35	http	algiers.synerzipune	algiers.synerzipune	/lisabank/buttonclick.do	156.0 ms
06/08/2010 12:06:17	http	algiers.synerzipune	algiers.synerzipune	/lisabank/login.do	286.0 ms
06/08/2010 12:00:43	http	algiers.synerzipune	algiers.synerzipune	/lisabank/buttonclick.do	149.0 ms
06/08/2010 12:00:37	http	algiers.synerzipune	algiers.synerzipune	/lisabank/depositmoney.do	626.0 ms
06/08/2010 12:00:19	http	algiers.synerzipune	algiers.synerzipune	/lisabank/buttonclick.do	361.0 ms
06/08/2010 12:00:11	http	algiers.synerzipune	algiers.synerzipune	/lisabank/viewtransactions.do	6437.0 ms
06/08/2010 12:00:08	http	algiers.synerzipune	algiers.synerzipune	/lisabank/createaccount.do	592.0 ms
06/08/2010 11:59:58	http	algiers.synerzipune	algiers.synerzipune	/lisabank/buttonclick.do	1333.0 ms
06/08/2010 11:59:43	http	algiers.synerzipune	algiers.synerzipune	/lisabank/login.do	1664.0 ms

The bottom panel shows a path graph with nodes representing components and edges representing transactions. The graph shows a path from 'Alisabank' to 'EJB3AccountControlBean' with a duration of 6437.0 msec. The graph also shows a path from 'EJB3AccountControlBean' to 'EJB3AccountControlBean' with a duration of 480.0 msec. The graph is labeled 'Alisabank /viewtransactions.do'.

As you can see, the LISA bank application is in one tab of the browser and we have opened the Pathfinder in the other tab. The transactions done in the LISA Bank application are loaded in the Pathfinder console and can be seen.

Creating a Dataset , **Creating a Baseline** and **Creating a VSM** are features of Pathfinder Pro. Pathfinder Pro requires a separate license.

For more information, contact ITKO support.

22.2 Pathfinder Console

22.2 Pathfinder Console

Once you open the Pathfinder Console, you will see the following screen:

There are two panels in the Pathfinder Console -

In the right panel, there are transactions listed which are loaded in the pathfinder from the LISA Bank application.

There are three tabs here -

- **Available Path Tab** - By default Pathfinder opens in this tab. This tab lists the available paths in pathfinder. For more details, see [Available Path Tab](#).
- **Component Details Tab**- When a component is selected in the Path Graph below, this tab lists the component at a detailed level. It has six sub tabs giving complete information about the transaction. For more details, see [Component Details tab](#).
- **Statistics in Path Tab**- This tab gives a detailed look into the Statistics of the selected transaction like the Agent name, Time, CPU time, Memory, Heap/Non heap usage etc. For more details, see [Statistics Tab](#).

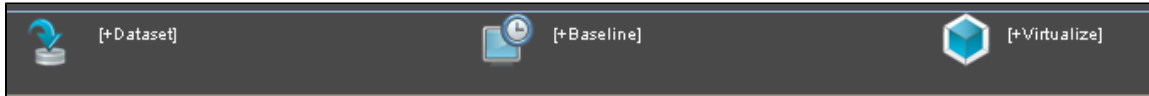
In the Left panel there are two tabs -

- **Paths** - All the transaction paths are listed in the Available Paths tab. In the Paths tab, you can "filter" a particular component type here and know more details about the same in the right panel. This is like a search facility within Pathfinder, where we filter/search for a particular component category, service ID, Operation, Transaction ID etc. For more information, see [Filtering Paths](#).

- **Cases** - This tab is used for defect tracking. You can either create a new defect or search for an old identified or closed defect. For more information, see [Filtering Cases](#).
- Note** - This is a **Pathfinder Pro** feature and can be used in a licensed version only. Please contact ITKO support for more information regarding the same.

When you select a particular transaction row in the Available Paths tab, you can see relevant component details in the Path Graph below. Path Graph is explained in the next section.

There is a panel of icons at the bottom of the Pathfinder screen as shown below:



These are used for [Creating a Dataset](#) , Creating a Baseline and Virtualizing the data in Pathfinder.

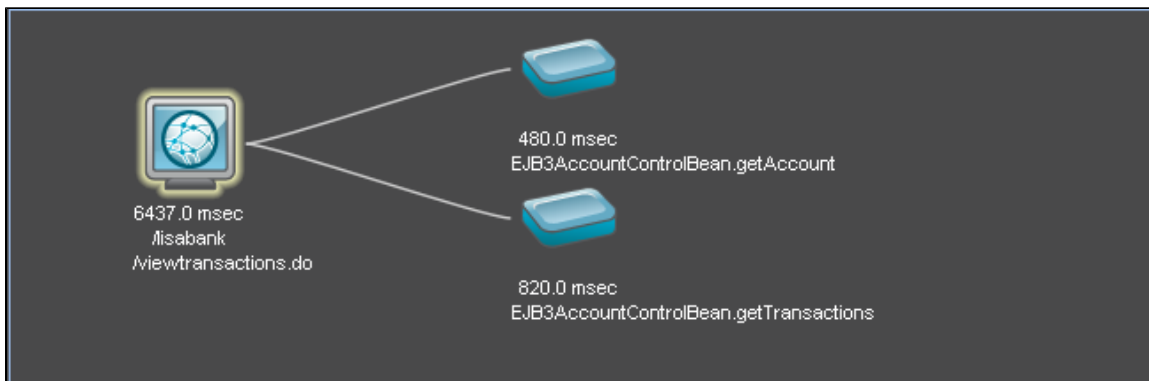
Note - These are **Pathfinder Pro** features and can be used in a licensed version only. Please contact ITKO support for more information regarding the same.

This is the standard pathfinder view. There is a lot of information in the tabs that open in the top panel.

Path Graph

Once you select a particular transaction in the Available Path tab, the lower panel shows the **Path Graph** related to that transaction as discovered by pathfinder.

This graph shows all the paths within the selected transaction.



The selected component is highlighted and shows the path of its transaction with other components.

Each Component has a different icon to identify its category within the Path Graph. There can be different components within a transaction like HTTP, EJB, JDBC, JMS etc.

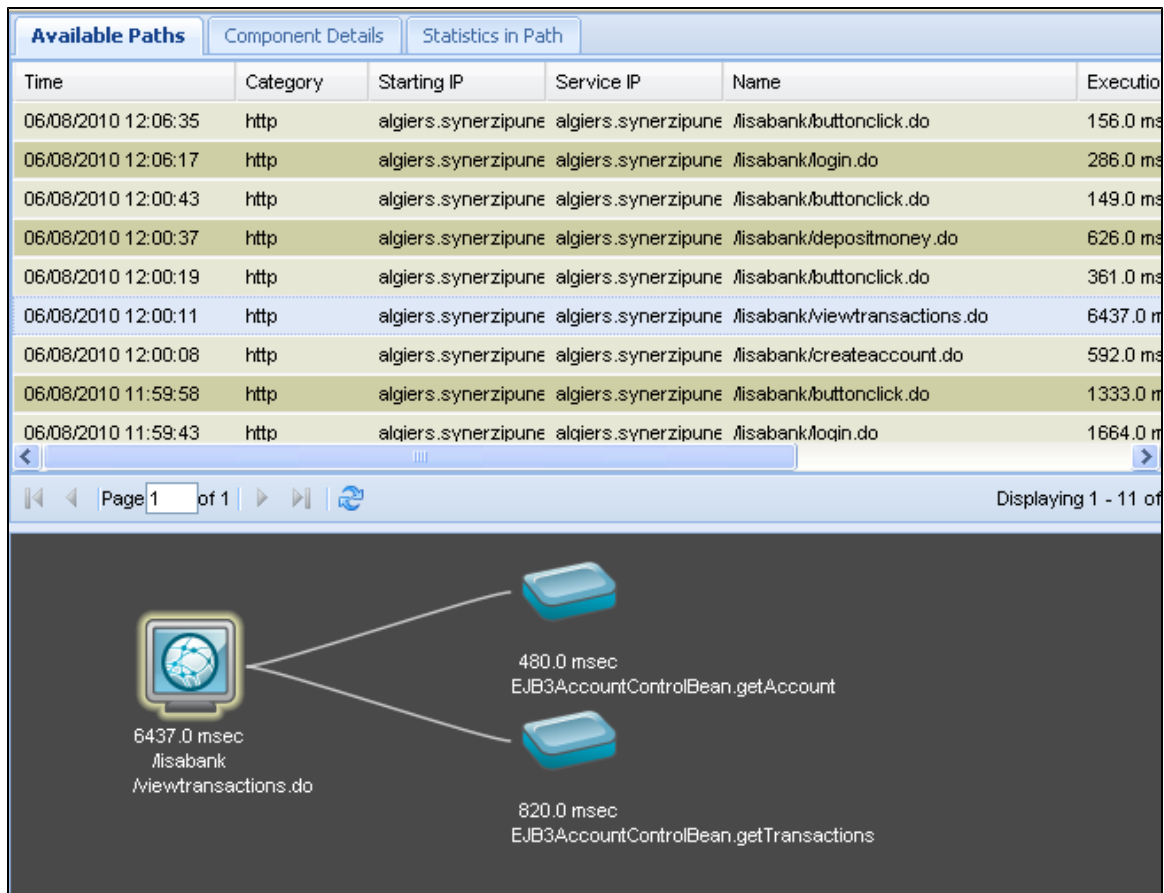
All the component icons within the path graph show the **Transaction name and Time** depicting the actual time required to complete the transaction. The main component depicts the total time taken for the entire transaction.

You can roll over each component icon to see the transaction information.

22.3 Available Paths Tab

22.3 Available Paths Tab

Once you open the Pathfinder Console, by default it opens in the **Available Paths** tab as shown below:



Each row in the table depicts a particular transaction.

Once you select any row from the table, the transaction map below changes to show details related to the selected transaction.

For any transaction, you can find information related to:

- **Time** - The time when the transaction took place
- **Category** - The category of the component ex: HTTP, Web Service, EJB etc
- **Starting IP** - The starting IP number
- **Service IP** - The service IP number
- **Name** - The name of the transaction
- **Execution** - Execution time required given in milliseconds
- **SQL Statement** - The statement executed if any.
- **Elements** - The number of elements in that transaction.

This is the standard pathfinder view. There is a lot of information in all the tabs that open in the top panel.

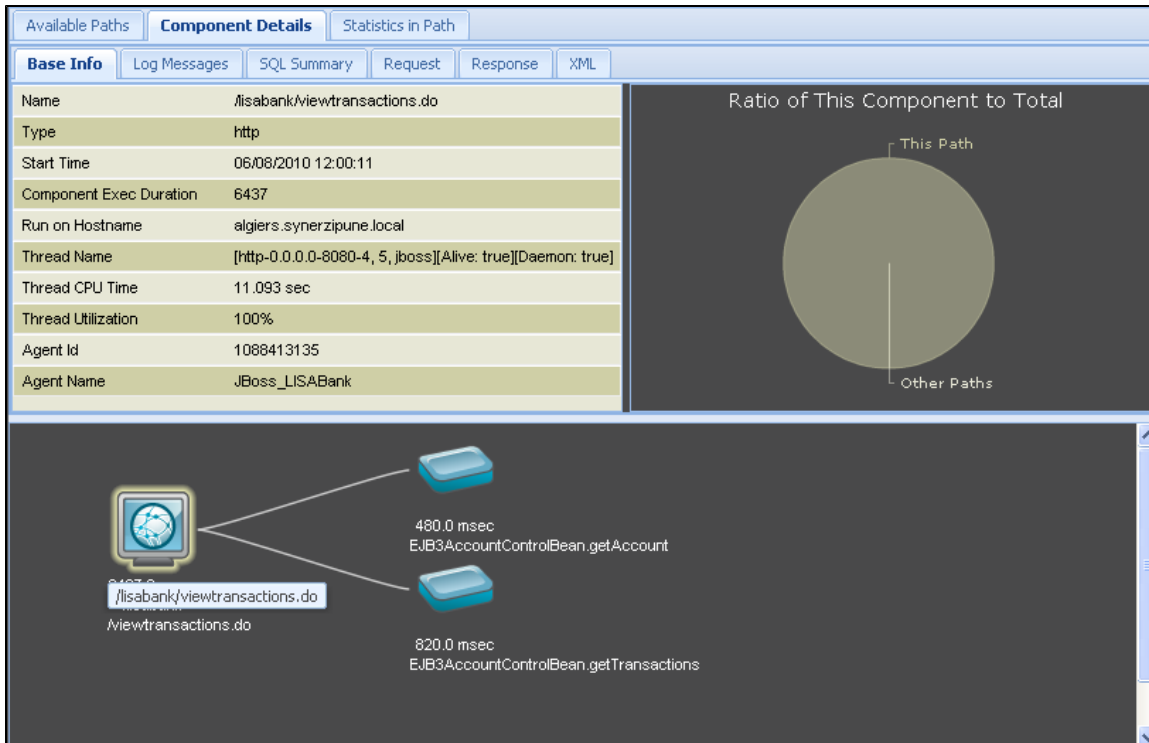
22.4 Component Details Tab

22.4 Component Details Tab

You need to select a component in the Path Graph to view the Component related details.

The **Component details** tab in the Pathfinder console lists all the details like basic information, log information, SQL query, Request & Response related information.

The Component details tab is as shown below: By default it opens in the Base info tab.



The Component details tab has following sub tabs:

- **Base Info Tab**
- **Log Messages Tab**
- **SQL Summary Tab**
- **Request Tab**
- **Response Tab**
- **XML Tab**

Each component when selected, will have information in all these tabs and will give a complete understanding of the component in the pathfinder view.

The general features of each sub tab will be explained by using our example test case of LISA Bank as shown below:

The sub tabs show information about the component that you highlight in the Path Graph.

- Click a component in the Path Graph, to make the component **Active** in the pathfinder view.
- When you select the highlighted component, it shows the transaction made as the tooltip as shown above.

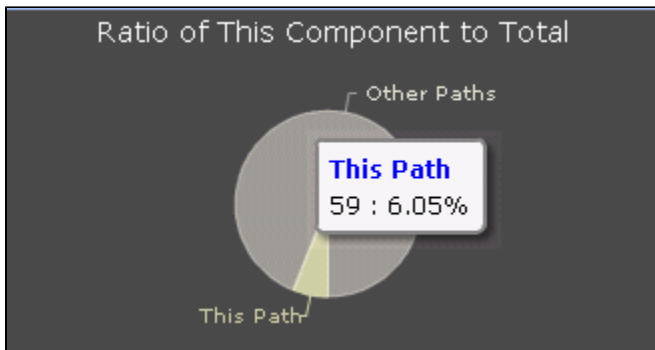
The pathfinder view by default opens in the Base Info tab.

Base Info Tab

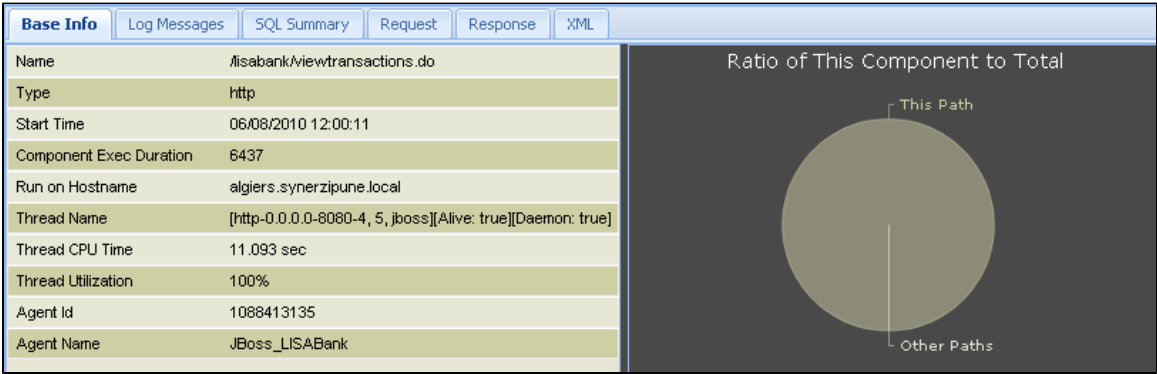
This tab is opened by default when you click on the Component details tab.

In this tab, Pathfinder shows the Basic data of the selected component, from the Java virtual machine (JVM).

A pie chart view of the relative execution time of each step as compared to all transactions, expressed as a ratio, is displayed on the right.



Lisa reports the absolute times in each branch, and then displays the relative times as a pie chart.



In the Base Info tab, Pathfinder list the details of the selected component like the Name, Type of component, Start Time, Total execution time, name of the host on which it runs, Thread name, Thread CPU time, Thread utilization, Agent name and ID.

Log Messages Tab

This is the Log Messages tab.

Pathfinder reports any log messages here.

The example below shows the level of Information, the Logger name if any and the relevant message.

Available Paths			Component Details	Statistics in Path		
Base Info			Log Messages	SQL Summary	Request	Response XML
Level	Logger	Message				
info		Web Service end point: http://localhost:8080/itkoExamples/EJB3AccountControlBean?wsdl				
info		View Account Activity successfully				

SQL's Summary Tab

This is the SQL summary tab.

Pathfinder provides a summary of the SQL information here. It displays the relative executions of both the SQL calls and the non database interaction times for this component.

The figure below shows the SQL Summary tab from the LISA Bank application.

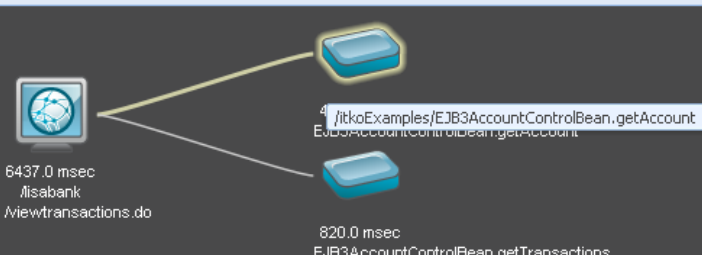
Available Paths | **Component Details** | Statistics in Path

Base Info | Log Messages | **SQL Summary** | Request | Response | XML

SQL Statements

ID	SQL	Row Count	Invocations	Avg. Time (ms)	Total Time (ms)
[sql id 1]	select account0_account_id as account1_0_0_, account0_name as name0_0_, accou	1	1	4	4

Results



6437.0 msec
/isabank
/viewtransactions.do

820.0 msec
EJB3AccountControlBean.getAccount

The SQL Summary tab lists the SQL ID, SQL stat, Invocations, Row Count, Average Time, Total Time etc

Clicking on the **SQL** column will open the SQL Query as shown below:

SQL Statements

ID	SQL	Row Count	Invocations	Avg. Time (ms)	Total Time (ms)
[sql id 1]	select user0_login as login4_2_, user0_pwd as pwd4_2_,	1	1	4	4

Results - Content Viewer

```
select user0_login as login4_2_, user0_pwd as pwd4_2_, user0_newFlag
as newFlag4_2_, user0_fname as fname4_2_, user0_lname as lname4_2_,
user0_email as email4_2_, user0_phone as phone4_2_, user0_roleKey as
roleKey4_2_, user0_ssn as ssn4_2_, accounts1_USER_ID as USER6_4_,
accounts1_account_id as account1_4_, accounts1_account_id as
account1_0_0_, accounts1_name as name0_0_, accounts1_type as
type0_0_, accounts1_version as version0_0_,
accounts1_AVAILABLE_BALANCE as AVAILABLE5_0_0_,
addresses2_USER_ID as USER7_5_, addresses2_address_id as
address1_5_, addresses2_address_id as address1_1_1_
```

Close

11.0 msec
EJB3UserControlBean.getUser

Note - All the columns in the SQL Statements table can be sorted.

Request Tab

This is the Request Tab.

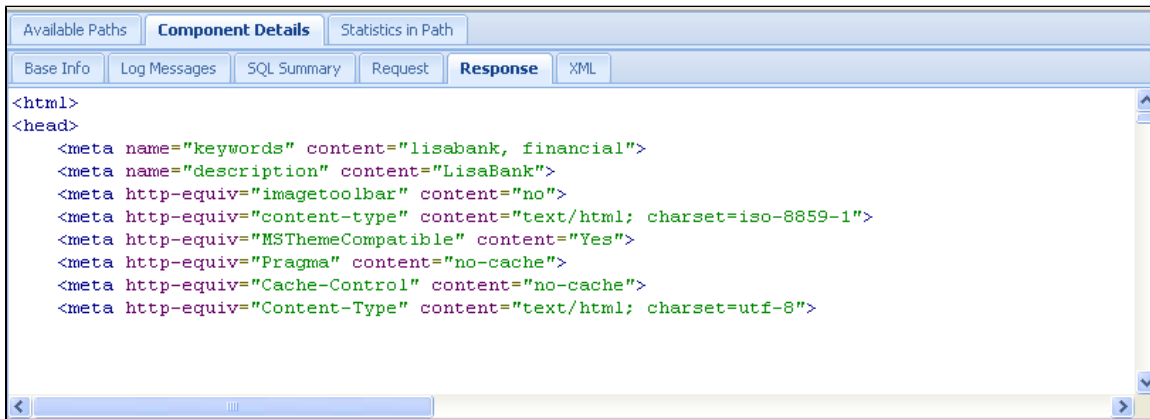
Pathfinder displays the actual Request that was sent by the highlighted component to the component on it's right.



Response Tab

This is the Response tab.

Pathfinder displays the actual response that was received by the highlighted component from the component on it's left.

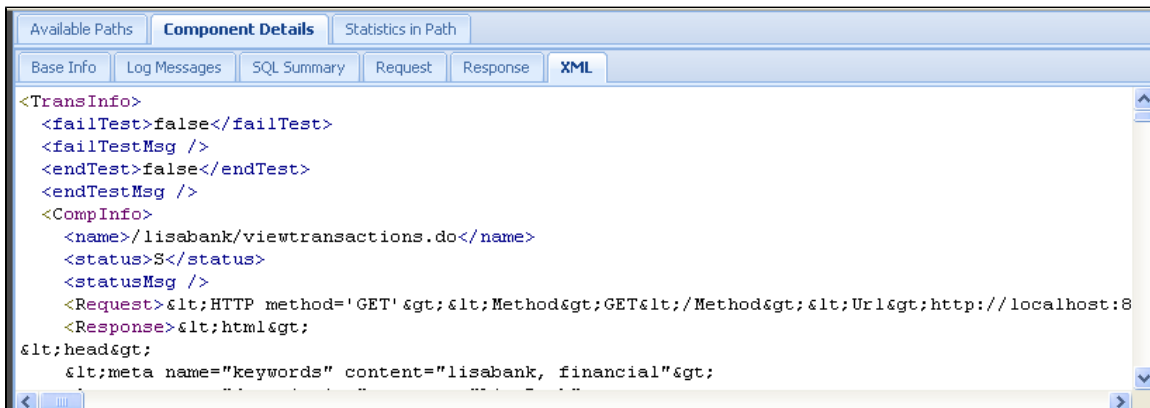


XML Tab

This is the XML Tab.

Pathfinder displays the raw XML of the instrumentation response in the component.

This is additive, so the components to the right contain less information that those on the left, and the first component contains the complete response.



22.5 Statistics in Path Tab

24.5 Statistics in Path Tab

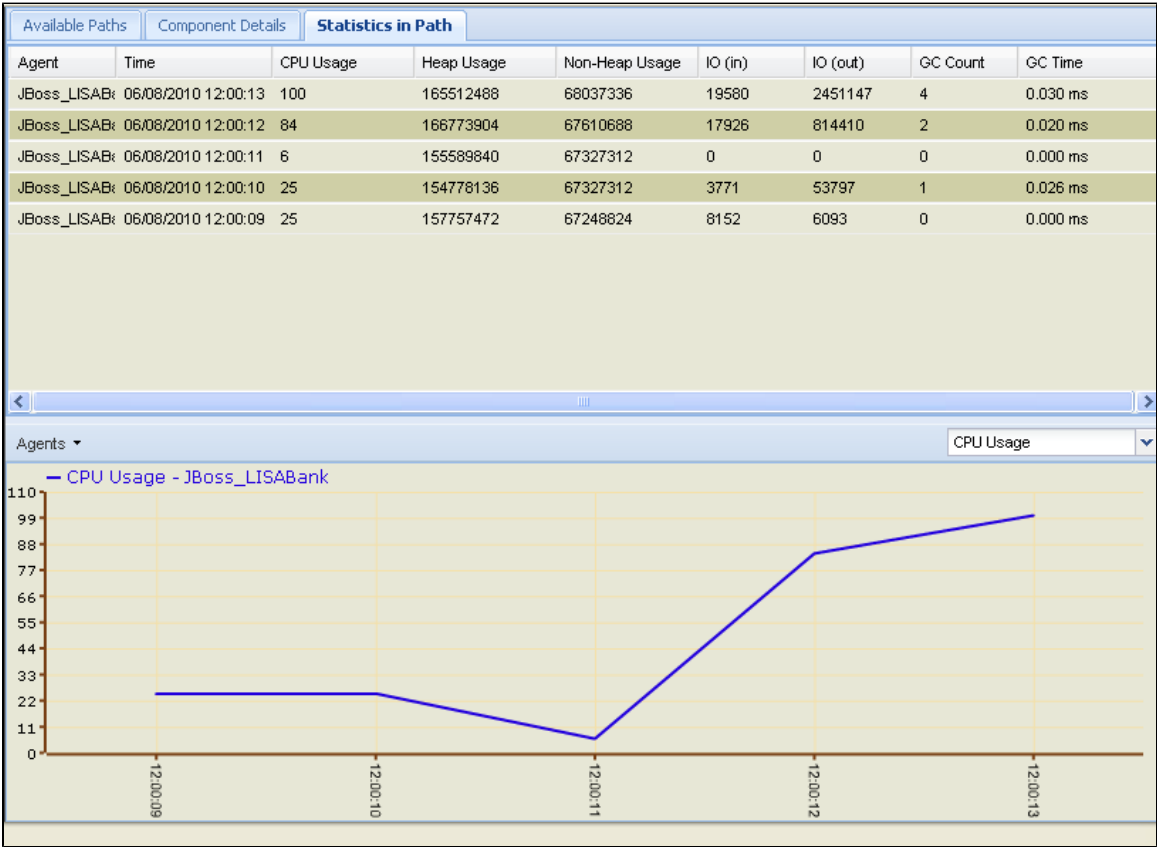
To find out the Statistics in each path, click on the third tab - **Statistics in Path** in the Pathfinder console.

This lists down the statistics details for the transaction selected in the "Available Paths" tab.

The Statistics in Path tab is as shown below:

Each row in the table depicts the transaction of the Agent.

If there are multiple Agents connected, the Agent list shows all the Agents names that are connected.

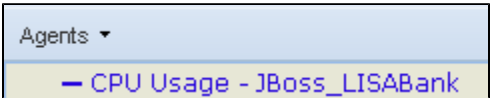


Once you select any row from the table, the statistics shown below changes to show details related to that Agent.

For any Agent, you can find information related to:

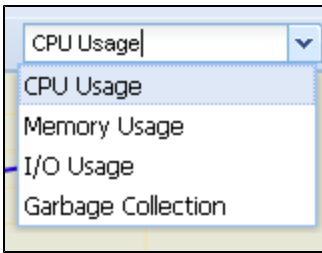
- **Agent Name** - The name of the Agent
- **Time** - The time of the execution
- **CPU Usage** - The CPU usage of that Agent
- **Heap Usage** - Lists the Heap Usage Lists the Heap Usage
- **Non Heap Usage** - Lists the non Heap Usage Lists the non Heap Usage
- **IO (In)** - Lists the input Count Lists the input Count
- **IO (Out)** - Lists the output Count Lists the output Count
- **GC Count** - Garbage collection Count
- **GC Time** - Garbage collection Time

If there are multiple Agents connected, you can also choose to select any Agent from the Agents drop down list as shown below:



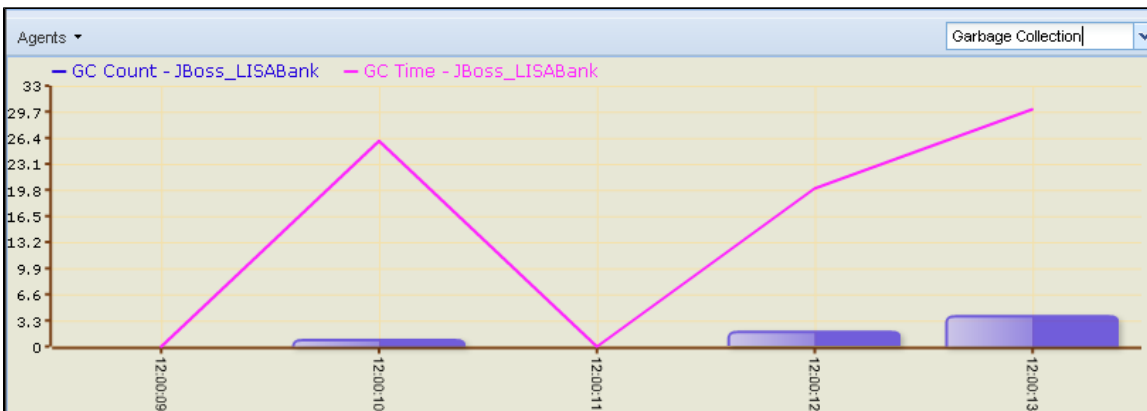
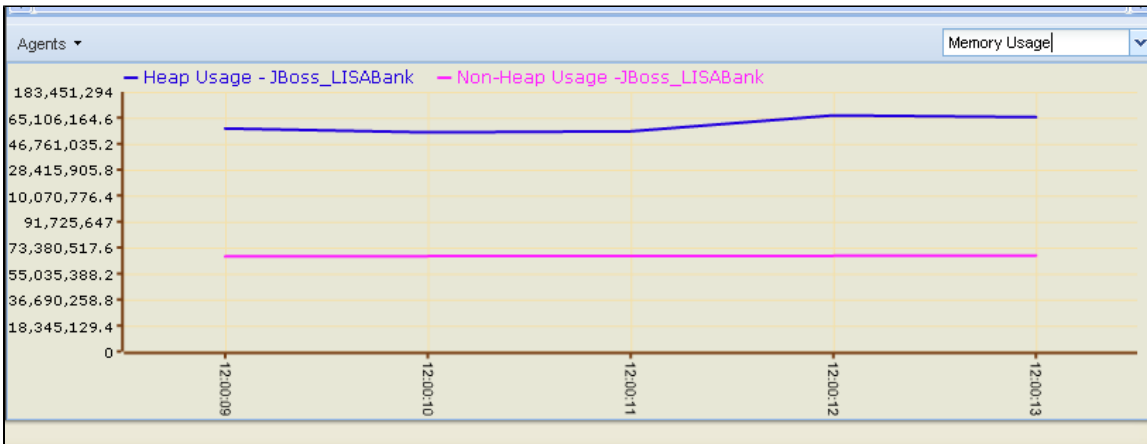
The Agents drop down list will give you a list of available Agents.

There are **four type of statistics** that you can choose for each Agent as shown below:



- CPU Usage - Will show the CPU Usage
- Memory Usage - Will show the Memory usage
- I/O Usage - Will show the Input/Output usage
- Garbage Collection - Will show the details of Garbage collection.

You can choose any one of these from the drop down to view more details regarding the same as shown below:



22.6 Filtering Paths

22.6 Filtering Paths in Pathfinder

You can filter the Path of any transaction in Pathfinder.

- Open the Pathfinder Console
- Click on the **Paths** tab in the left panel.
- Enter the **Start Date/End Date and Exact Time** to filter the transactions within that date and Time.

For the purpose of illustration, we have selected the Start and End Date, Time and have Added the **Type** Filter and Selected **HTTP** Component from the drop down in the example shown below:

Click **Save** to save the criteria and view all the HTTP related transactions within the Pathfinder in the Available Paths tab.

Time	Category	Starting IP	Service IP	Name	Execution Time
06/14/2010 19:08:33	http	algiars.synerzipune	algiars.synerzipune	/alisabank/buttonclick.do	157.0 ms
06/14/2010 19:08:27	http	algiars.synerzipune	algiars.synerzipune	/alisabank/createaccount.do	666.0 ms
06/14/2010 19:08:09	http	algiars.synerzipune	algiars.synerzipune	/alisabank/buttonclick.do	514.0 ms
06/14/2010 19:08:00	http	algiars.synerzipune	algiars.synerzipune	/alisabank/login.do	2178.0 ms
06/14/2010 19:07:38	http	algiars.synerzipune	algiars.synerzipune	/alisabank/home.do	2555.0 ms
06/14/2010 19:07:37	http	algiars.synerzipune	algiars.synerzipune	/alisabank/	538.0 ms

As shown above, there are many Criteria to select from, like Type, Exec Time, Class Name, Operation, Service IP, Starting IP and Transaction IP to choose from.

Depending on the selection in the first column, the second column will change.

For example, we have now selected the Type **Operation**, this will show all the different operations that have taken place during all the transactions in Pathfinder as shown below:


Similarly, you can define different Types and search related paths.

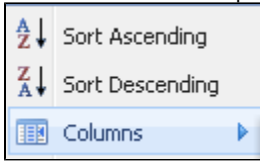
22.7 Sorting & Selecting Columns

22.7 Sorting and Selecting Columns

You can sort the columns within the Pathfinder tabs to get a analyzed view of the data.

- Open the Pathfinder Console
- Check that there are some transactions done by clicking on the Available path tab.
- Click on the **Component Details** tab and click on the SQL Summary tab.

- When you click on any column, a  button appears.
- Click on this button to expand and see options like



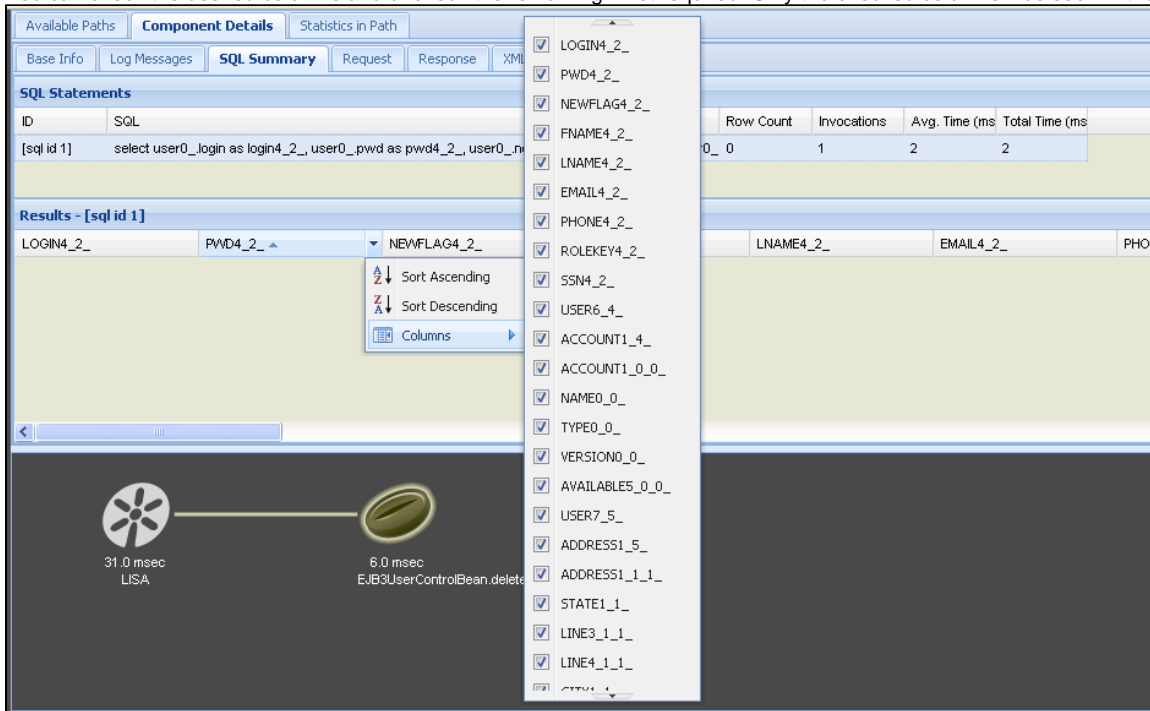
Sorting

You can sort the columns either in Ascending or Descending manner by clicking on the selection.

Selecting Columns

Clicking on the Columns arrow > will display a list of available columns as shown below:

You can check the desired columns and uncheck the remaining if not required. Only the checked columns will be seen in the row.



The screenshot shows a software interface with a 'SQL Summary' tab. A dropdown menu is open, displaying a list of available columns for selection. The columns listed are:

- LOGIN4_2_
- PWD4_2_
- NEWFLAG4_2_
- FNAME4_2_
- LNAME4_2_
- EMAIL4_2_
- PHONE4_2_
- ROLEKEY4_2_
- SSN4_2_
- USER6_4_
- ACCOUNT1_4_
- ACCOUNT1_0_0_
- NAME0_0_
- TYPE0_0_
- VERSION0_0_
- AVAILABLES_0_0_
- USER7_5_
- ADDRESS1_5_
- ADDRESS1_1_1_
- STATE1_1_
- LINE3_1_1_
- LINE4_1_1_

The interface also shows a table with columns: Row Count, Invocations, Avg. Time (ms), and Total Time (ms). The table has one row with values: 0, 1, 2, 2.

As shown above, there are many fields to select from, like Type, Version, Address, State, Account, User e

For each column, there will be different set of fields as shown below:

The screenshot shows the 'SQL Summary' tab with a table of SQL statements. The first statement is 'select user0_login as login4_2_, user0_0' with a row count of 2. A context menu is open over the 'Row Count' column, showing options: ID, SQL, Row Count, Invocations, Avg. Time (ms), and Total Time (ms), all of which are checked. The 'Results' section is empty.

22.8 Integrating Pathfinder Broker and Portal Server

22.8 Integrating LISA Pathfinder Broker and Portal Server

Integrating Pathfinder Broker and Portal Server with LISA Registry

When you run the LISA Registry, it will attempt to start the Pathfinder Broker by default.

You'll see console output as shown below:

```
[TestRegistry] LISA_HOME set to /Users/john/projects/sandbox/lisa/dist/ [TestRegistry] LISA Version :: 0.0 (0.0.0.0) [TestRegistry]
[TestRegistry] Creating a LISA Registry, name=lisa.Registry [TestRegistry] [INFO][9659][main][09:00:40 (667)] LISA AGENT: No
custom rules loaded from /Users/john/projects/sandbox/lisa/rules.properties [TestRegistry] Oct 12, 2009 9:00:41 PM
org.apache.activemq.broker.BrokerService start [TestRegistry] INFO: Using Persistence Adapter:
AMQPersistenceAdapter(activemq-data/localhost) [TestRegistry] Oct 12, 2009 9:00:41 PM
org.apache.activemq.store.amq.AMQPersistenceAdapter start [TestRegistry] INFO: AMQStore starting using directory:
activemq-data/localhost [TestRegistry] Oct 12, 2009 9:00:41 PM org.apache.activemq.kaha.impl.KahaStore initialize [TestRegistry]
INFO: Kaha Store using data directory activemq-data/localhost/kr-store/state [TestRegistry] Oct 12, 2009 9:00:41 PM
org.apache.activemq.store.amq.AMQPersistenceAdapter start [TestRegistry] INFO: Active data files: [] [TestRegistry] Oct 12, 2009
9:00:41 PM org.apache.activemq.broker.BrokerService getBroker [TestRegistry] INFO: ActiveMQ null JMS Message Broker
(localhost) is starting [TestRegistry] Oct 12, 2009 9:00:41 PM org.apache.activemq.broker.BrokerService getBroker [TestRegistry]
INFO: For help or more information please see: http://activemq.apache.org/ [TestRegistry] Oct 12, 2009 9:00:42 PM
org.apache.activemq.kaha.impl.KahaStore initialize [TestRegistry] INFO: Kaha Store using data directory
activemq-data/localhost/kr-store/data [TestRegistry] Oct 12, 2009 9:00:42 PM
org.apache.activemq.transport.TransportServerThreadSupport doStart [TestRegistry] INFO: Listening for connections at:
tcp://Larry.local:61616 [TestRegistry] Oct 12, 2009 9:00:42 PM org.apache.activemq.broker.TransportConnector start [TestRegistry]
INFO: Connector tcp://Larry.local:61616 Started [TestRegistry] Oct 12, 2009 9:00:42 PM
org.apache.activemq.broker.TransportConnector start [TestRegistry] INFO: Connector vm://localhost Started [TestRegistry] Oct 12,
2009 9:00:42 PM org.apache.activemq.broker.BrokerService start [TestRegistry] INFO: ActiveMQ JMS Message Broker (localhost,
ID:Larry.local-56909-1255399242018-0-0) started [TestRegistry] [INFO][9659][main][09:00:42 (579)] LISA AGENT: Started Broker
tcp://localhost:61616 [TestRegistry] LISA Registry Ready.
```

To prevent this, pass "-nobroker" on the command line when you launch the Registry.

Configuration of the Broker inside the Registry is accomplished with lisa.properties entries shown below:

```
# Our Pathfinder Broker can be launched within the Registry; these settings are required for that lisa.pathfinder.broker.port=61616
lisa.pathfinder.broker.db=org.apache.derby.jdbc.ClientDriver:jdbc:
derby://localhost:1527/reports/pathfinder.db:create=true::pathfinder::pathfinder
```

The port is used by the agents, consoles, and broker to communicate back and forth (as it is messaging-based). The DB entry defines where the broker will store historic data about the transactions it saw and statistics, etc. If you use LISA's embedded Derby database, you have nothing to do. To change that database to your own, change this line by creating your own entry in **local.properties** or **site.properties**. The database and user account shown must already exist in your target database instance. The broker will create the tables for you.

22.9 Pathfinder Summary

22.9 Pathfinder Summary

We have purposely taken a simple example to help illustrate the capabilities of Pathfinder, and to facilitate explaining how to use Pathfinder.

To reiterate what was said earlier in this chapter,

- Pathfinder can help you track transactions and their Paths.
- Pathfinder Pro can help you create a Data Set, Baseline Test, Virtual Service image or even create a defect case and track the transactions.
- Pathfinder produces all the information with very little effort on behalf of the tester, and no modification to the system under test. Once the application Server has been configured for pathfinder, it is just a matter of running the application.
- To disable Pathfinder -
For those who are not using Pathfinder agents, they can turn off all Pathfinder support. To do so,

Open lisa.properties file and change the setting of lisa.pathfinder.on=false

For more information on how to configure common application Servers, see the [LISA Installation and Configuration Guide](#).

23. CVS Dashboard

23. CVS (Continuous Validation Service) Dashboard

The Continuous Validation Service (CVS) utility within LISA, allows you to schedule Tests and Test suites to run on a regular basis, over an extended time period.

The CVS utility has a **CVS Dashboard** that maintains a **list of all scheduled tests (services)** and the **status** of each one. From the CVS Dashboard, you can select a test and monitor each test run.

In CVS, a Monitor contains a single test or an entire test suite and a Service contains one or many Monitors within itself.

You can either choose to run a service or individual monitors within it from the CVS Dashboard.

You can open the CVS Dashboard from the LISA main toolbar or from the View main menu.

From LISA 5.0, CVS dashboard can also be opened via a web portal.

To open the Portal, click <http://localhost:1505/cvsdashboard> in a web browser.

NOTE: Registry needs to be running to start the Portal.

Prerequisite

CVS runs in a LISA Server environment.

There must be a Coordinator Server and a Simulator Server running, registered with a LISA Registry.

For details on setting up the LISA Server environment see the [LISA Installation and Configuration Guide](#).

Following sections are available in this chapter:

[23.1 Starting CVS](#)
[23.2 Understanding CVS Dashboard](#)
[23.3 Monitor Tab](#)
[23.4 Graphs Tab](#)
[23.5 Events Tab](#)
[23.6 Examples](#)
[23.7 CVS Notes](#)

23.1 Starting CVS

23.1 Starting the Continuous Validation Service (CVS)

To start the CVS utility,



- Click the CVS icon on the main toolbar
- Or from the main menu select **View->CVS Dashboard**
- Or by entering <http://localhost:1505/cvstdashboard/> in a web browser.

You will need to run the Registry to launch the CVS Dashboard.

LISA Registry

When you start the LISA Workstation, you will be prompted to set the LISA Registry.

If you are not attached to a LISA Registry you can set the Registry from main menu.

You can choose either a LISA Registry or a local registry to attach to.

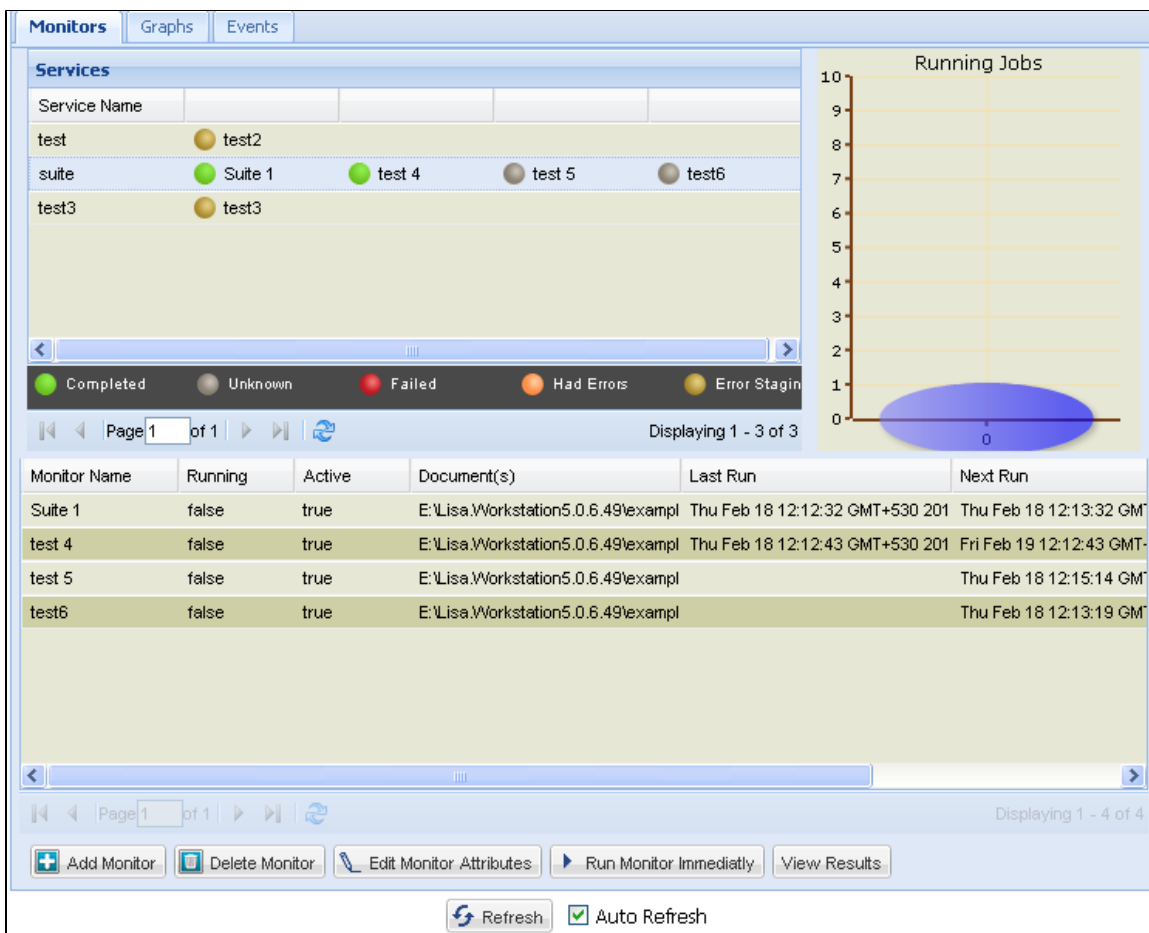
For more information on Registry, refer to the [LISA Registry](#) in the Install Guide.

23.2 Understanding CVS Dashboard

23.2 Understanding CVS Dashboard

The CVS Dashboard is divided into 2 panels:

One for adding monitors into the dashboard under the Services and one for viewing the running the monitors.



The Top panel in the CVS Dashboard has three tabs:

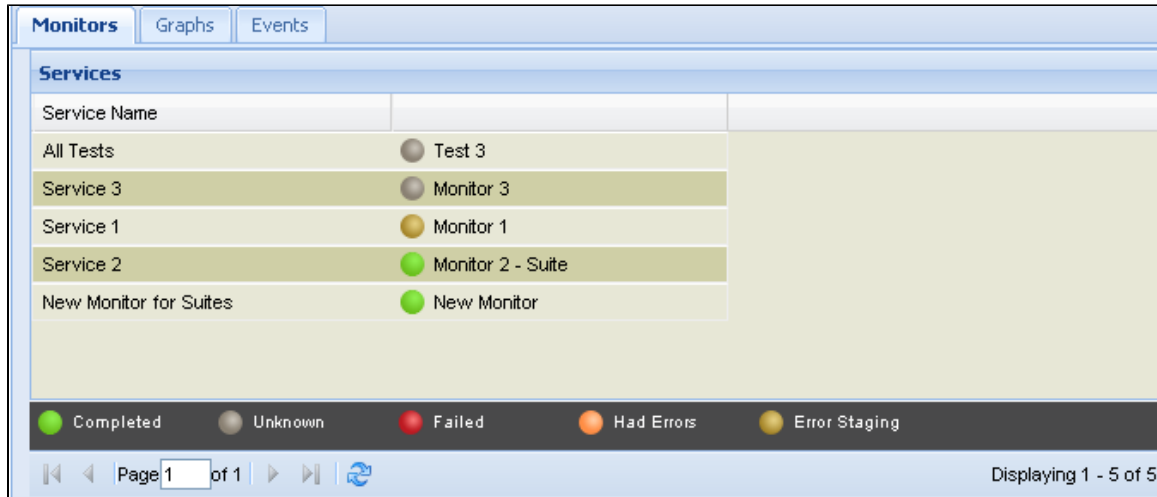
- **Monitors** - By default the CVS Dashboard opens in this tab. This tab will show a list of all the Monitors (Tests/Test Suites) that are added to the CVS Dashboard.
For more information, refer to [Monitor tab](#).
- **Graphs** - This tab displays the Dashboard status graphically. It will also show the percentage of the tests passed or failed.

For more information, refer to [Graphs tab](#).

- **Events** - This tab displays the status events emitted by the monitors.
For more information, refer to [Events tab](#).

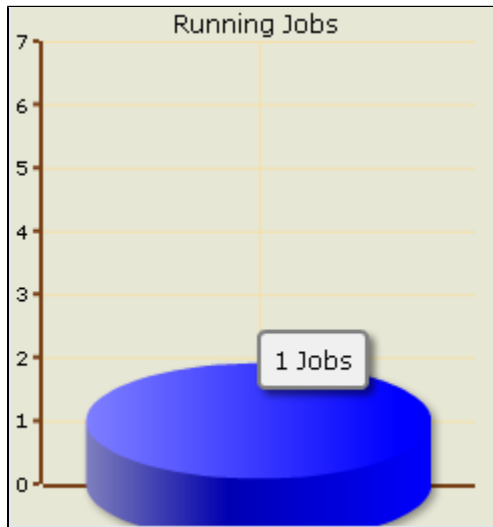
The top panel also has icons below, to depict the Test Run Completion, Failure, Error, Stage Doc Error or Unknown Error:

These icons depict the state of the monitor once it has run as shown below:



At the right end, the top panel shows the graphical representation of the jobs that are currently running.

For example, the below graph shows only one job that is currently running.

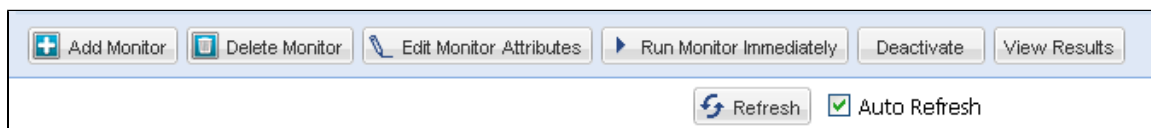


The bottom panel shows all the monitors that are running or have finished there last run.

The monitors that have finished running completely will not be seen in this list.

Dashboard Toolbar

The CVS Dashboard has a toolbar as below to Add, Delete, Edit the Monitors.



- **Add Monitor** - Click to Add a Monitor to the Dashboard. Monitor is a single test case or a suite of test cases. For more information, refer to [Add Monitor](#).
- **Delete Monitor** - Click to Delete the Monitor from the Dashboard.

- **Edit Monitor Attributes** - Click to Edit the Monitor attributes.
- **Run Monitor Immediately** - Click to Run the monitor immediately.
- **Deactivate** - Click to Deactivate the monitor. Deactivate means it remains in the list, but will not run.
- **View Results** - Click to view the results of the run in the Report Viewer.
- **Refresh** - Click to Refresh the list on the Dashboard.
- **AutoRefresh** - Check this to auto refresh the list on the Dashboard after a certain period.

23.2.1 Adding a Monitor

23.2.1 Adding a Monitor in the Dashboard

To schedule a new test run in the CVS utility, you need to add a new Monitor in the CVS dashboard.

You can add the monitor either through the CVS Dashboard UI or through command line [CVSManager](#).

To add a new monitor through CVS Dashboard,

- Click the **Add *Monitor** button,  which is at the bottom of the CVS Dashboard.

LISA displays the configuration window (Monitor Editor) for the new monitor as shown below:

Enter the following in the Monitor Editor:

Monitor Information -

- **Monitor Name:** Name of the new Monitor
- **Note:** Comments or note about that Monitor

Job Type -

- **Job Type:** Select whether you are scheduling a single test or a test suite. Settings will changes depending on the type you select.

Job Priority: Select the priority of the specified job, from High/Medium/Low. According to the set priority, tests will be run.

Job Parameters -

These will change according to the selection of Job type - Single test or Suite

- For **Single** test case the parameters are:

Job Type Job Type: <input checked="" type="radio"/> One Test <input type="radio"/> Suite		Job Priority Priority: High <input type="button" value="v"/>
Job Parameters Test Case: <input type="text"/> <input type="button" value="Browse"/>		
Staging Document: <input type="text"/> <input type="button" value="Browse"/>		
Coordinator Server: <input type="text"/>		

Test Case: Name of the Test Case to run

Staging Document: Name of the staging document to be attached

Coordinator Server: Name of the coordinator server

- For **Test Suite**, the parameters are:

Job Type Job Type: <input type="radio"/> One Test <input checked="" type="radio"/> Suite		Job Priority Priority: High <input type="button" value="v"/>
Suite Document Suite: <input type="text"/> <input type="button" value="Browse"/>		

- **Suite:** Enter the name of the test suite document to run.

Configurations -

- **Service Name:** Enter the name of the service. This will be seen on the CVS Dashboard. A Service holds one or many Monitors within it.
- **Run-time Configuration:** Enter the name of a configuration, or Enter the path name of the external configuration file. If specified, this configuration will be used for the test, or the tests in a test suite. If left blank, the configuration active in the Test Case will be used.
- **Notification email:** Enter the email address for email notification of the test run result.
For email settings, please refer to [Notification Email Settings Delete](#).

Schedule -

Specify the testing schedule here:

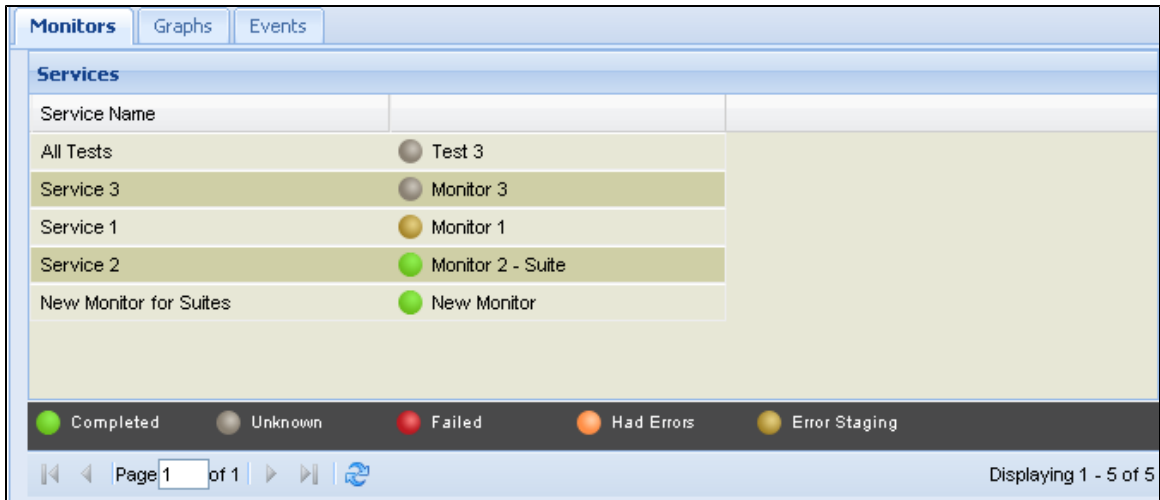
Schedule Start: Stop: Run:	<input type="text" value="One Time"/> <input type="text" value="Every"/> <input type="text" value="Daily"/> <input type="text" value="Weekly"/> <input type="text" value="Monthly"/> <input type="text" value="CRON"/> <input type="text" value="One Time"/> <input type="button" value="v"/>	Time: <input type="text" value="11:11"/> Time: <input type="text" value="11:11"/>
--	---	--

- **Start:** Enter the date and time of the monitor which is first scheduled to start.
-To set the date, click the calendar button, to show a pop-up calendar. Select the day, month and year from the pop-up calendar.
- **Stop:** Enter the date and time that the monitor should expire, using the same method as you used for the start time.
- **Time:** Enter the time of the scheduled tests.
-To set the time, enter the time using a 24 hour clock.
- **Run:** Select the frequency of the monitor to run from the various options given. Depending on the selection type as below, other relevant options are seen.
- **One Time:** The test will be run only one time, at the time specified in the start time.
- **Daily:** The test will be run once every day, at the time specified in the start time.
- **Every few minutes:** Enter the test frequency in minutes. CTS will attempt to run the test at this frequency. However, if the previous run has not completed, CTS will skip a test run and try again at the next scheduled time. CTS will never terminate a test after it has been started.
- **Weekly:** Select the days of the week to run the test using the checkboxes. The test will be run once every day, at the time specified in the start time.
- **Monthly:** Enter the days of the month as a comma delimited list. Entering 1, 15, 30 will run the test on the 1st, 15th and 30th of each month. If a day is specified that does not occur for a particular month, for example February 30th, that day is ignored for that month. The test will be run once every day, at the time specified in the start time.
- **Cron:** Enter a standard 'cron' specification into the text box. Cron is a standard UNIX syntax for specifying time intervals. This allows the most flexibility when specifying a run schedule.

Remember the LISA Registry must be able to locate these documents as well as any documents, such as external Data Sets referenced in these documents. The best practice is to parametrize the document names using properties.

For a discussion of this best practice see the [LISA Installation and Configuration Guide](#)

The monitors added in the Dashboard are shown below:



23.2.2 Running a Monitor

23.2.2 Running a Monitor

Once you add the services in the top panel, they will be run automatically in the background at the scheduled time intervals.

In case you want to run a particular monitor immediately, you can do so from the toolbar below.

To Run a Monitor Immediately,

- Select the Monitor by double clicking it from the Services list
- When it gets listed in the bottom panel, select it again.
- Click **Run Monitor Immediately** to run the Monitor at that instance.

The bottom panel of the CVS Dashboard, lists the monitor that is run, with a suffix **-now** (EX: Test 3-now) and has information regarding that in a table as shown below:

Monitor Name	Running	Active	Document(s)	Last Run	Next Run	Timing
Test 3	false	false	E:\Lisa.5.0.20.81\examples\SuitesV	Tue Jul 20 13:48:05 GMT+530 201C	Tue Jul 20 13:49:05 GMT+530 201C	every 1 minutes.
Test 3-now	false	false	E:\Lisa.5.0.20.81\examples\SuitesV	Tue Jul 20 13:48:05 GMT+530 201C	Tue Jul 20 13:49:05 GMT+530 201C	one time only

The bottom panel also includes a toolbar with buttons: Add Monitor, Delete Monitor, Edit Monitor Attributes, Run Monitor Immediately, Activate, and View Results. The pagination bar shows 'Page 1 of 1' and 'Displaying 1 - 2 of 2'.

The tests which have run, can be seen listed in the Last Run column.

23.2.3 Viewing Test Details

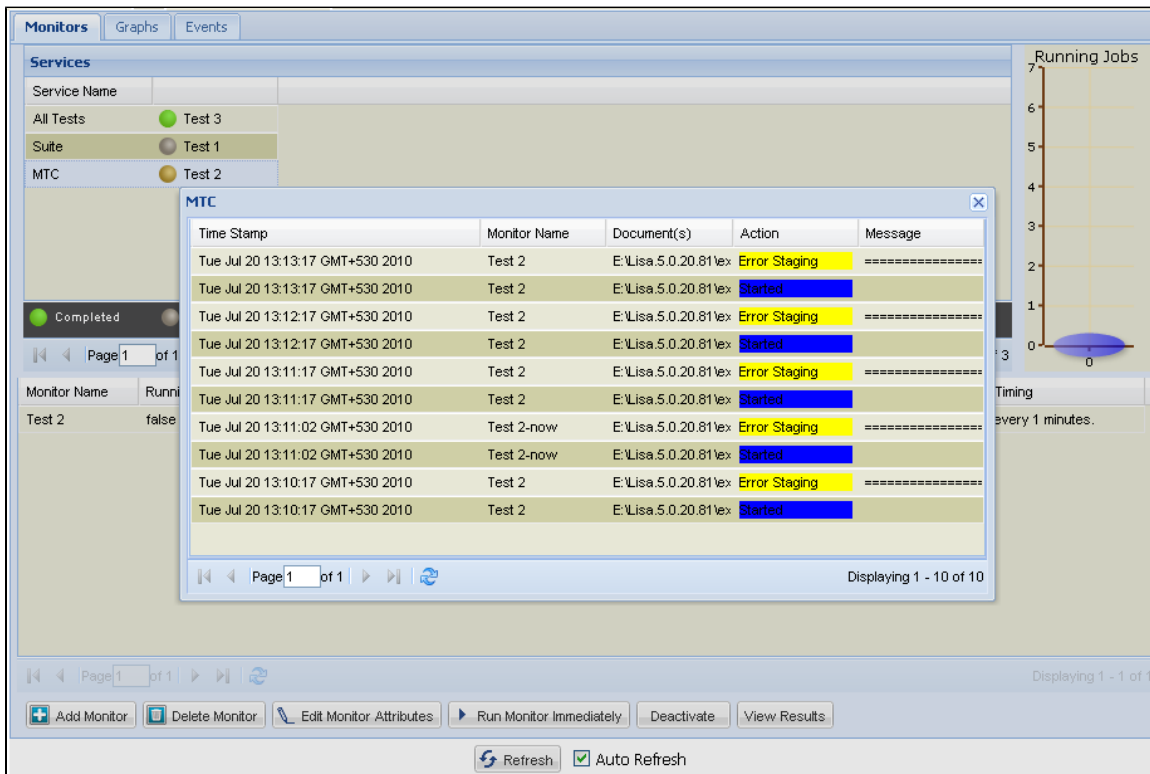
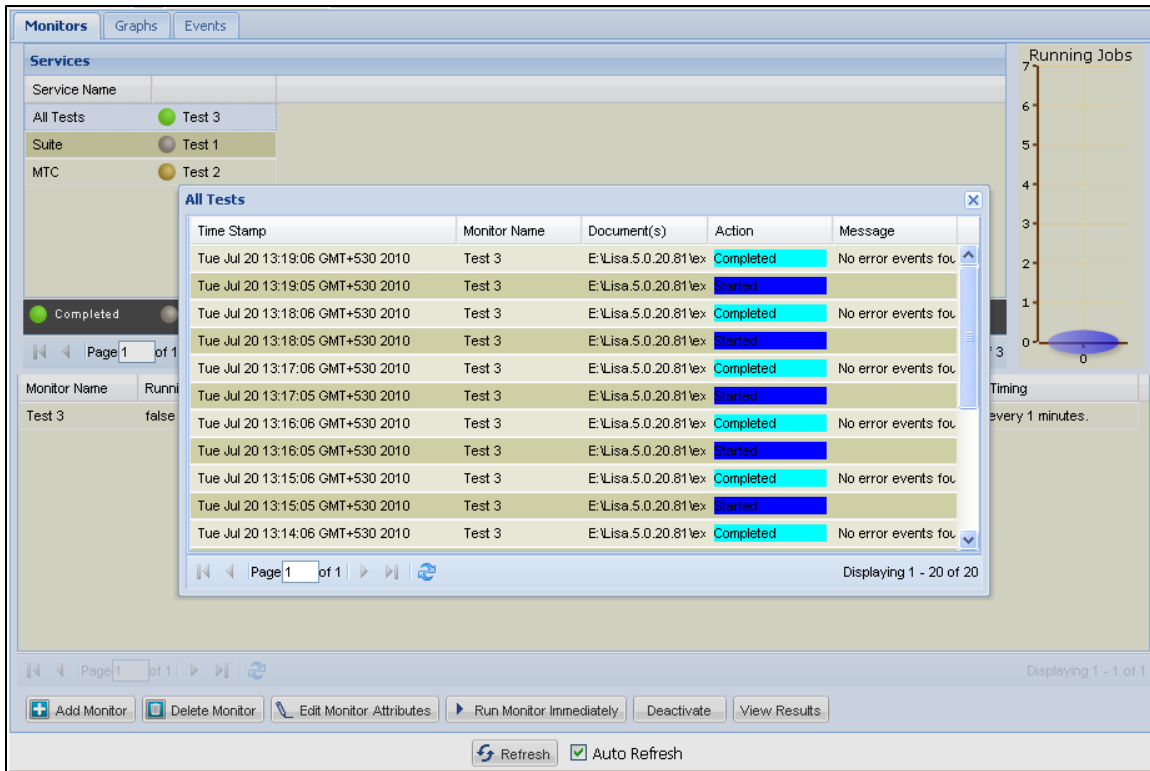
23.2.3 Viewing Test Details

You can view the details of a test run within CVS Dashboard.

To view details,

- Double click the Service to see the Service related details.
- Double click the monitor whose details you want to view in the top panel.

This will open the test related detail window as shown below:



23.2.4 Email Notification Settings

23.2.4 Email Notification Settings

Every time you schedule a new test in the CVS utility, you need to add a New Monitor in the CVS dashboard.

Within the Monitor window, you can also set the **email address**, so that you get a email notification every time the Monitor is run within the specified time period.

To set the email notification,

You need to uncomment the following portion of the **lisa.properties** file.

You also need to change the **mail server configuration** and enter the mail host as shown below:

If you use performance monitoring alerts, this is the "from" email address of those alerts
#lisa.alert.email.emailAddr=lisa@itko.com

And this is the email server we will attempt to route emails with (smtp server)
#lisa.alert.email.defHosts=mail.itko.com

Please uncomment out this information in the lisa.properties file.

The above example shows the mail server configurations for the ITKO mail server. You will need to enter the mail server settings of your mail server.

Restart LISA, to set the mail server configuration.

Notification email: Enter the email address for email notification of the test run result.

Every time the test is run as per schedule, you will receive a email informing the same.

23.3 Monitor Tab

23.3 Monitor Tab

CVS Dashboard by default opens in the Monitor tab.

The Monitor tab consists of a top panel that displays the number of services running and the right corner consists of a job graph as shown below:

Here, you can add many monitors to run in one or many services.

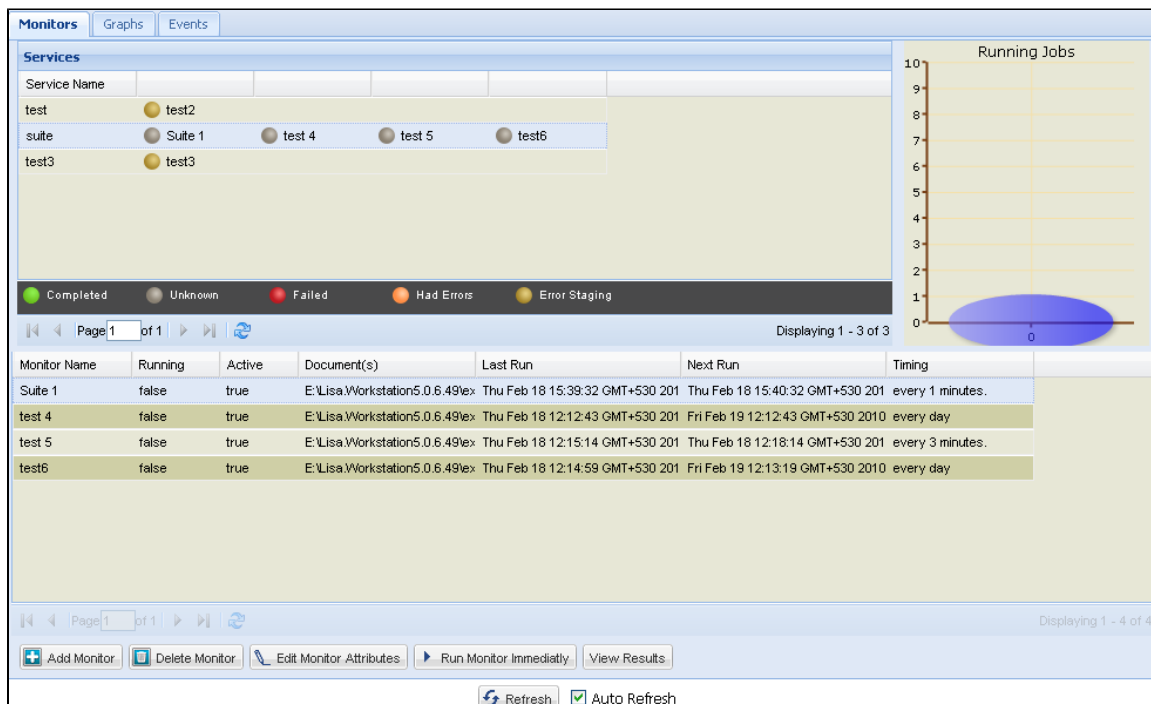
Note - A Service contains one or many Monitors.

You can basically schedule to run a Service. All monitors in that Service are run at scheduled time.

The Services are run in the background at scheduled intervals.

As shown below, you can either Add the monitors in one Service (Ex: Suite) or add the monitors in different services (EX: test, suite or test 3)

All the monitors added in one service (EX: Suite) are shown added after that Service name.











CVS Dashboard Top Panel


All the tests in a particular service which have run are depicted with a colored ball.


For Ex: The completed tests are shown by a Green ball, Failed tests are shown by a Red ball, tests which have error in staging document are shown with yellow ball etc.


Services


Service Name										
test		test2								
suite		Suite 1		Suite 1-now		test 4		test 5		test6
test3		test3								






 Completed

 Unknown

 Failed

 Had Errors

 Error Staging

  Page 1 of 1   

Displaying 1 - 3 of 3

CVS Dashboard Bottom Panel

In the CVS Dashboard, the bottom panel of the Monitor tab displays the status of each monitor run in a Service, is seen as below:

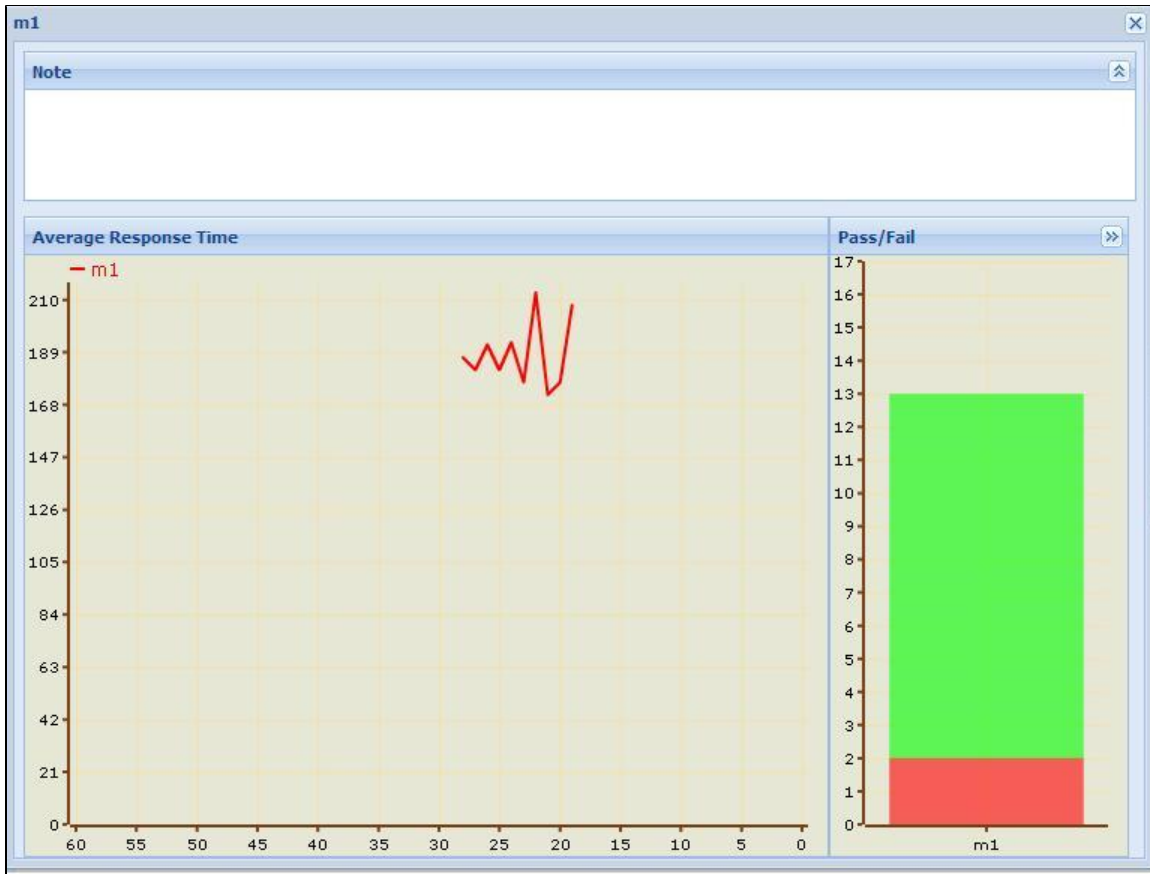
Monitor Name	Running	Active	Document(s)	Last Run	Next Run	Timing
Suite 1	false	true	E:\Lisa\Workstation5.0.6.49\ex	Thu Feb 18 15:39:32 GMT+530 201	Thu Feb 18 15:40:32 GMT+530 201	every 1 minutes.
test 4	false	true	E:\Lisa\Workstation5.0.6.49\ex	Thu Feb 18 12:12:43 GMT+530 201	Fri Feb 19 12:12:43 GMT+530 2010	every day
test 5	false	true	E:\Lisa\Workstation5.0.6.49\ex	Thu Feb 18 12:15:14 GMT+530 201	Thu Feb 18 12:18:14 GMT+530 201	every 3 minutes.
test6	false	true	E:\Lisa\Workstation5.0.6.49\ex	Thu Feb 18 12:14:59 GMT+530 201	Fri Feb 19 12:13:19 GMT+530 2010	every day

The table shows the following information:

- **Monitor Name:** The name given to the monitor
- **Running:** Shows whether this monitor is currently running (True/False)
- **Active:** Shows whether this monitor is currently Active (True/False)
- **Documents:** The names of the documents, such as Test Case, staging, and suite documents associated with this monitor.
- **Last Run:** The last run date and time of this monitor.
- **Next Run:** The next scheduled start time of this monitor.
- **Timing:** The schedule details for this monitor e.g. every 2 minutes, daily, weekly etc

Monitor Information

Double click a Monitor to see the Monitor related information -



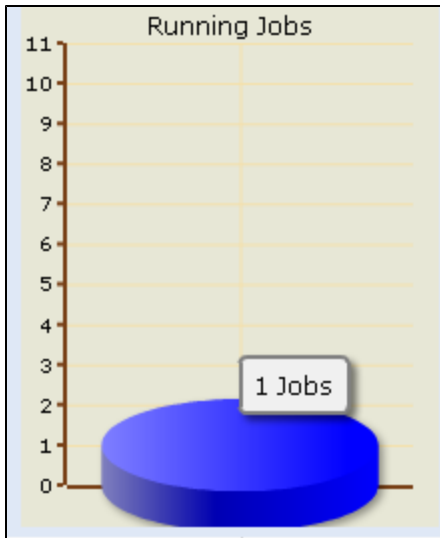
Customizing Columns

In the bottom panel, You can also customize the columns like arranging them in sorted order or showing/hiding of the columns as shown below:

The screenshot shows the 'LISA CVSDashboard Console' interface. The 'Monitors' tab is active, displaying a list of services (s1, s2) and monitors (m1, m2, m3). A table below shows monitor details for 'm2', including 'Active' status, 'Document(s)', 'Next Run', and 'Timing'. A context menu is open over the 'Columns' header, showing options for sorting and column visibility. The 'Columns' menu is checked, and a sub-menu is visible with checkboxes for 'Monitor Name', 'Running', 'Active', 'Document(s)', 'Last Run', 'Next Run', and 'Timing'. On the right, a 'Running Jobs' graph shows a single bar at 1. The bottom of the interface includes buttons for 'Add Monitor', 'Delete Monitor', 'Edit Monitor Attributes', 'Run Monitor Immediately', 'Deactivate', 'View Results', 'Refresh', and 'Auto Refresh'.

Currently Running Jobs

The graph on the right as shown below, shows only the jobs that are currently running at that instance.

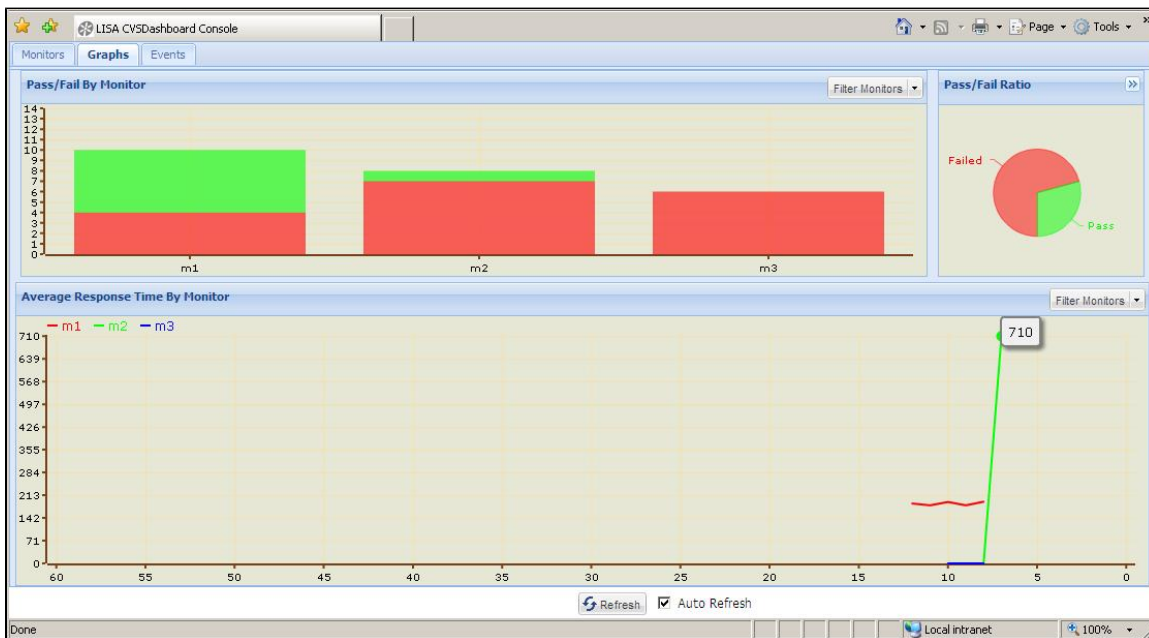


23.4 Graphs Tab

23.4 Graphs Tab

The Graph tab displays the tests in the CVS Dashboard in a graphical format.

It gives you a graphical summary of the tests which have passed and failed as shown below:



The Graph tab consists of three graphical displays, which show and track the last 60 minutes of activity.

Filter Monitor - You can filter the monitors for viewing from the **Filter Monitor** button.

Filter Monitors

- ☐ Suite example
- ☒ Multi-tier combo
- ☐ Service 1

- Click on the Filter Monitor button, to get a list of available monitors.
- Choose one to view the details regarding that monitor.

Pass/Fail by Monitor: The Pass Fail by Monitor graph shows stacked histograms of Pass/Fail results for each monitor. Moving the cursor over the histogram shows the result in text form.

In the top "Pass/Fail By Monitor" panel, you can choose to see either all the tests or selective tests/suites by clicking on the "Filter Monitor" button, which will allow you to choose the tests/suites.

Pass/Fail Ratio: The Pass/Fail Ratio graph displays the percentage of total number of passing tests (green), and the total number of failing tests (red) as a pie chart.

Average Response Time by Monitor: Average Response Time by Monitor graphs the average response time for each monitor over the last 60 minutes of activity. Each monitor is color coded.

You can choose to display the test/suite to be seen in the graphical format in the Pass/Fail By Monitor.

23.5 Events Tab

23.5 Events Tab

The Events tab displays various events corresponding to the stages of the monitor run, like starting of a test run, ending, or failing etc.

The Events tab is as shown below: The following events are generated for Monitor "Test3".

Monitors

Graphs

Events

Monitor Events

Time Stamp	Monitor Name	Service	Document(s)	Action
Fri Feb 19 11	test3	test3	E:\Lisa\Workstation5.0.6.49\examples\Tests\yms.tst -- E:	Completed
Fri Feb 19 11	test3	test3	E:\Lisa\Workstation5.0.6.49\examples\Tests\yms.tst -- E:	Started
Fri Feb 19 11	test2	test	E:\Lisa\Workstation5.0.6.49\examples\Tests\multi-tier-co	Started
Fri Feb 19 11	test3	test3	E:\Lisa\Workstation5.0.6.49\examples\Tests\yms.tst -- E:	Completed
Fri Feb 19 11	test3	test3	E:\Lisa\Workstation5.0.6.49\examples\Tests\yms.tst -- E:	Started
Fri Feb 19 11	test3	test3	E:\Lisa\Workstation5.0.6.49\examples\Tests\yms.tst -- E:	Completed
Fri Feb 19 11	test3	test3	E:\Lisa\Workstation5.0.6.49\examples\Tests\yms.tst -- E:	Started
Fri Feb 19 11	test3	test3	E:\Lisa\Workstation5.0.6.49\examples\Tests\yms.tst -- E:	Completed
Fri Feb 19 11	test2	test	E:\Lisa\Workstation5.0.6.49\examples\Tests\multi-tier-co	Completed
Fri Feb 19 11	test3	test3	E:\Lisa\Workstation5.0.6.49\examples\Tests\yms.tst -- E:	Started
Fri Feb 19 11	test3	test3	E:\Lisa\Workstation5.0.6.49\examples\Tests\yms.tst -- E:	Completed

Page 1 of 2

Displaying 1 - 20 of 24

Refresh

☒ Auto Refresh

The following information is displayed in the table:

- **Timestamp:** The time the event was emitted.
- **Monitor Name:** The name of the monitor in which the event occurred.
- **Service:** The name of the service in which the monitor resides.
- **Document(s):** A list of the documents associated with the monitor in which the event occurred.
- **Action:** The action reported by the event. Start events are colored Blue, staging error events Yellow, completion events Green, and failed events are shown as Red.
- **Message:** The message reported by the event. If the text is larger than the message cell, you can right-click the cell to invoke the extended view window.

You can display the events in real time by checking the **Auto Refresh** box at the bottom of the panel, or you can refresh the list manually by clicking the Refresh icon, with the Auto Refresh box unchecked.

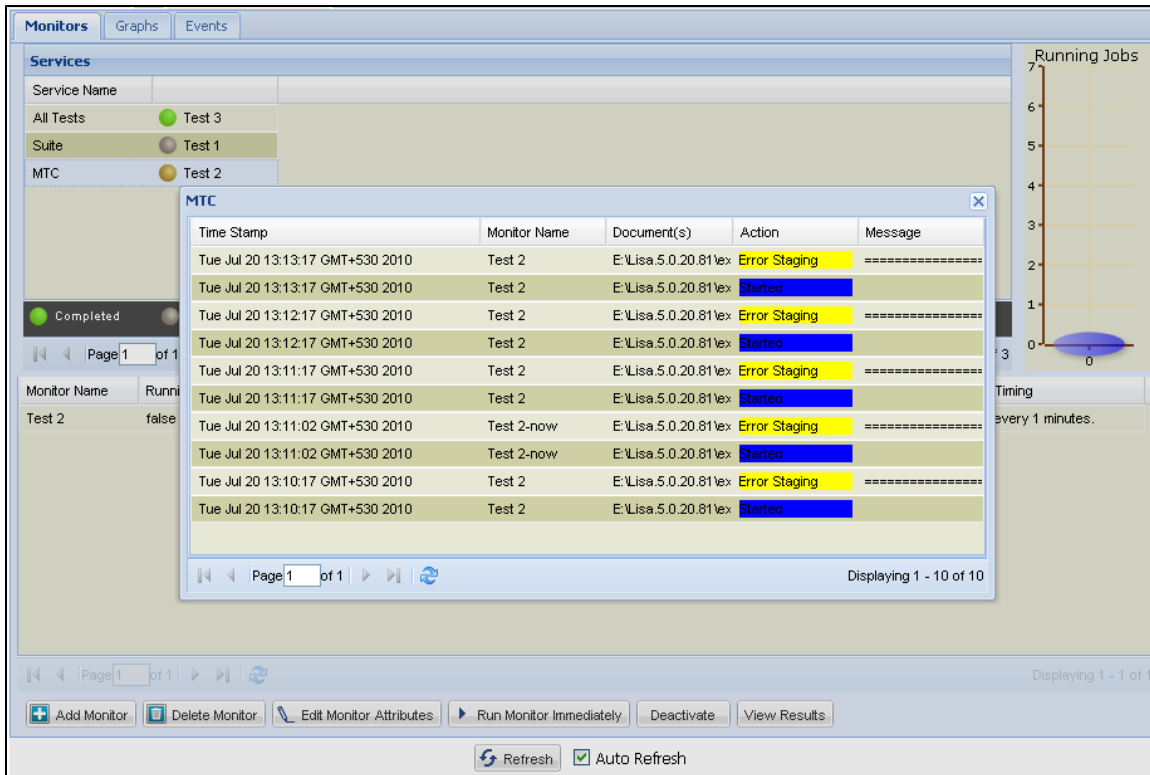
Test Cycle with Errors

At times, the test cycle may fail. The cycle that has errors can be seen in the Actions list in different color.

If it is a staging document related error, it will be shown in yellow color as seen below:

You can double click any event, to see the "Message" in an expanded popup.

The following events are generated for Monitor "Test 2" in Service "MTC" .



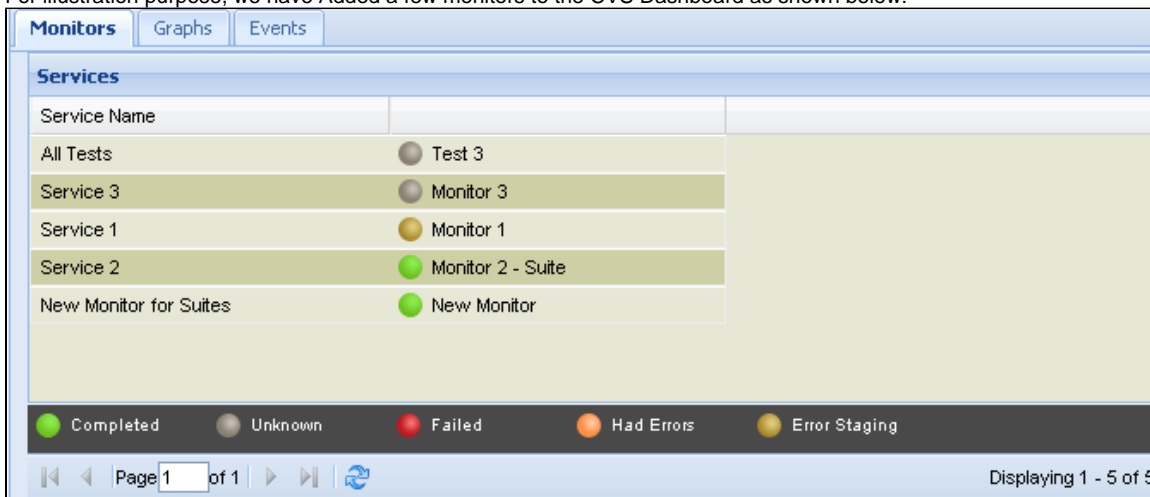
23.6 Examples

23.6 Examples

For illustration purpose, some examples of Adding a Monitor and Running the Monitor are shown below:

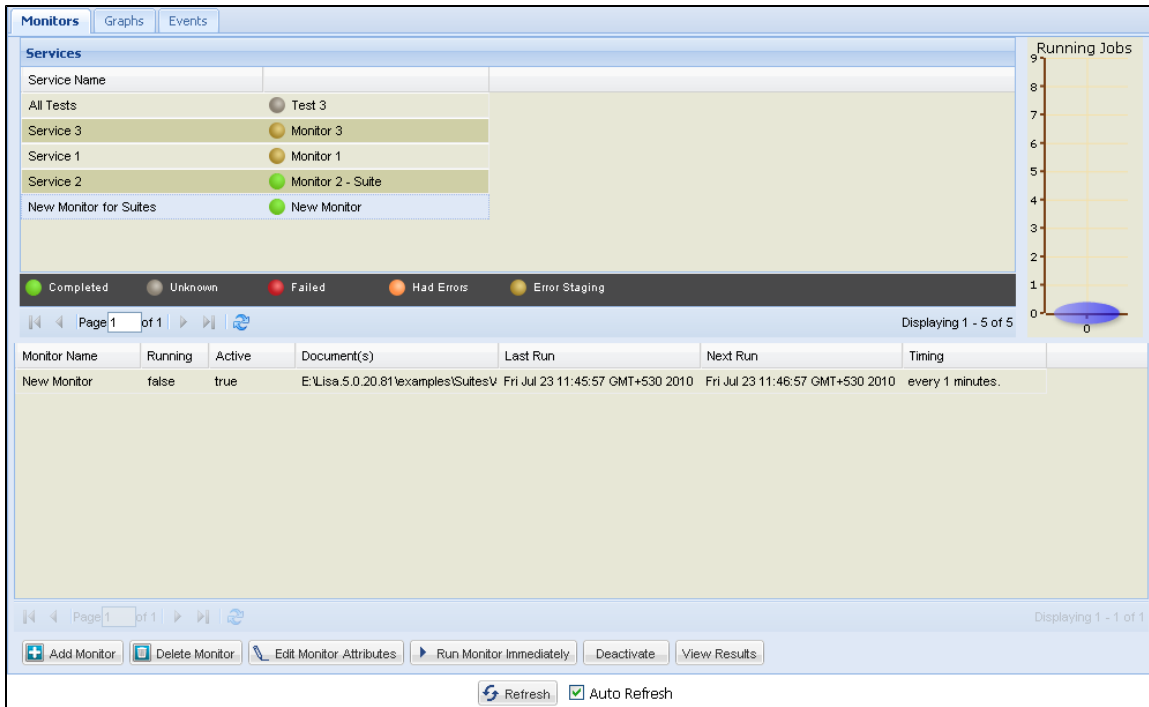
Example 1

For illustration purpose, we have Added a few monitors to the CVS Dashboard as shown below:



Now we select one of these monitors to run for ex - New Monitor.

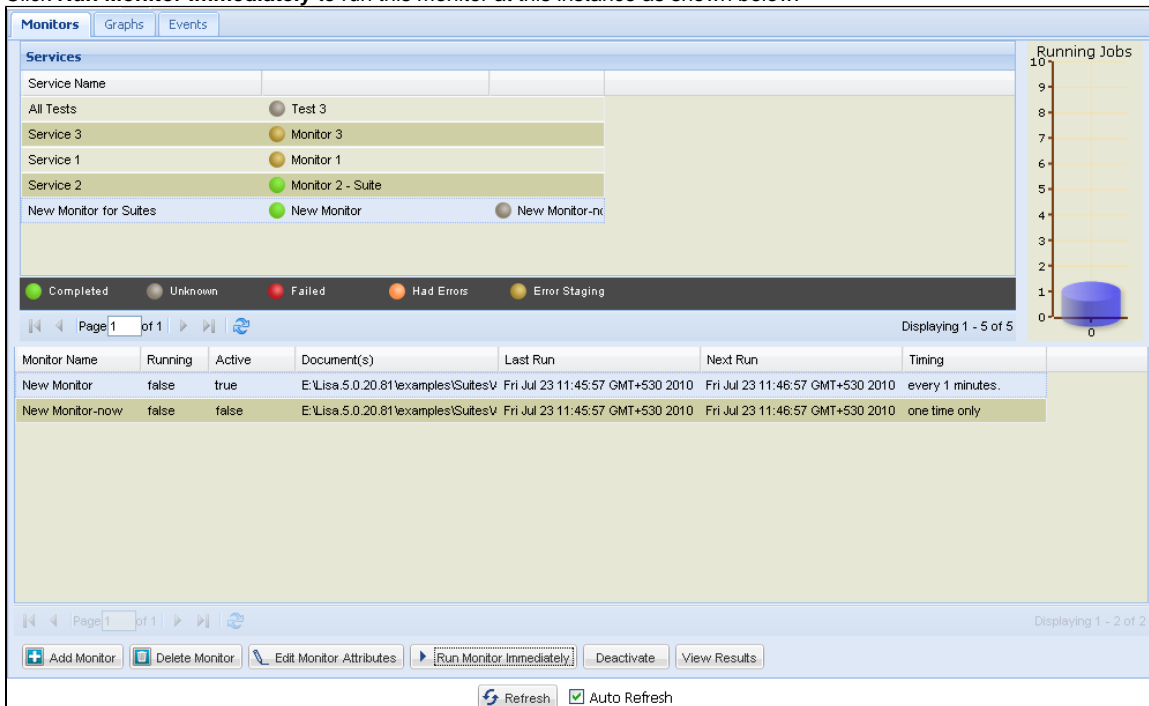
Click the **New Monitor** to get it added to the list below:



As seen above, the New Monitor has run its cycle, as it has reported its last run date and time.

Now, we will run this monitor again at this moment,

Click **Run Monitor Immediately** to run this monitor at this instance as shown below:



Note - The monitor run at this instance is suffixed as -now, to indicate that this has been run now.

To know the further details of the run completion events, double click in the top panel on the monitor to open window as shown below:

Monitors

Graphs

Events

Services

Service Name

All Tests

Test 3

Service 3

Monitor 3

Service 1

Monitor 1

Service 2

Monitor 1

New Monitor for Suites

Completed

Unknown

Page 1 of 1

Monitor Name

Running

New Monitor

false

New Monitor for Suites-New Monitor-now

Time Stamp	Monitor Name	Document(s)	Action	Message
Fri Jul 23 11:55:40 GMT+530 2010	New Monitor-now	E:\Lisa.5.0.20.81\ex	Completed	No error events fou
Fri Jul 23 11:55:39 GMT+530 2010	New Monitor-now	E:\Lisa.5.0.20.81\ex	Started	
Fri Jul 23 11:46:56 GMT+530 2010	New Monitor-now	E:\Lisa.5.0.20.81\ex	Completed	No error events fou
Fri Jul 23 11:46:55 GMT+530 2010	New Monitor-now	E:\Lisa.5.0.20.81\ex	Started	

Page 1 of 1

Displaying 1 - 4 of 4

Add Monitor

Delete Monitor

Edit Monitor Attributes

Run Monitor Immediately

Deactivate

View Results

In case of any errors, if you want to know further details of a particular monitor's column, click on it to open the monitor specific window:

New Monitor for Suites-New Monitor-now

Time Stamp	Monitor Name	Document(s)	Action	Message
Fri Jul 23 11:55:40 GMT+530 2010	New Monitor-now	E:\Lisa.5.0.20.81\ex	Completed	No error events fou
Fri Jul 23 11:55:39 GMT+530 2010	New Monitor-now	E:\Lisa.5.0.20.81\ex	Started	
Fri Jul 23 11:46:56 GMT+530 2010	New Monitor-now	E:\Lisa.5.0.20.81\ex	Completed	No error events fou
Fri Jul 23 11:46:55 GMT+530 2010	New Monitor-now	E:\Lisa.5.0.20.81\ex	Started	

Page 1 of 1

Displaying 1 - 4 of 4

Monitor: New Monitor-now

No error events found.

Add Monitor

Delete Monitor

Edit Monitor Attributes

Run Monitor Immediately

Deactivate

View Results

Example 2

We have taken an example of a Monitor that has failed.

The Service names for these tests are - All Tests, Suite and MTC.

The "Test 3 - All Tests" Monitor has run and has a green icon to denote successful completion.

Double clicking on that Test 3 Monitor, will open the Summary window as shown below:

The screenshot shows the 'Monitors' window with the 'All Tests' summary for Test 3. The summary table lists 20 events, all with the status 'Completed' and the message 'No error events fou'. The events are sorted by Time Stamp, showing a sequence of 'Started' and 'Completed' actions for Test 3. The interface includes a 'Services' panel on the left, a 'Running Jobs' graph on the right, and a bottom toolbar with buttons for 'Add Monitor', 'Delete Monitor', 'Edit Monitor Attributes', 'Run Monitor Immediately', 'Deactivate', and 'View Results'.

Time Stamp	Monitor Name	Document(s)	Action	Message
Tue Jul 20 13:19:06 GMT+530 2010	Test 3	E:\Lisa.5.0.20.81\ex	Completed	No error events fou
Tue Jul 20 13:19:05 GMT+530 2010	Test 3	E:\Lisa.5.0.20.81\ex	Started	
Tue Jul 20 13:18:06 GMT+530 2010	Test 3	E:\Lisa.5.0.20.81\ex	Completed	No error events fou
Tue Jul 20 13:18:05 GMT+530 2010	Test 3	E:\Lisa.5.0.20.81\ex	Started	
Tue Jul 20 13:17:06 GMT+530 2010	Test 3	E:\Lisa.5.0.20.81\ex	Completed	No error events fou
Tue Jul 20 13:17:05 GMT+530 2010	Test 3	E:\Lisa.5.0.20.81\ex	Started	
Tue Jul 20 13:16:06 GMT+530 2010	Test 3	E:\Lisa.5.0.20.81\ex	Completed	No error events fou
Tue Jul 20 13:16:05 GMT+530 2010	Test 3	E:\Lisa.5.0.20.81\ex	Started	
Tue Jul 20 13:15:06 GMT+530 2010	Test 3	E:\Lisa.5.0.20.81\ex	Completed	No error events fou
Tue Jul 20 13:15:05 GMT+530 2010	Test 3	E:\Lisa.5.0.20.81\ex	Started	
Tue Jul 20 13:14:06 GMT+530 2010	Test 3	E:\Lisa.5.0.20.81\ex	Completed	No error events fou

The "Test 2 - MTC" Monitor has run, but generated some errors.

When we double click on the Test 2, we get to see the error details as shown below:

The screenshot shows the 'Monitors' window with the 'MTC' summary for Test 2. The summary table lists 10 events, showing a sequence of 'Error Staging' and 'Started' actions for Test 2. The events are sorted by Time Stamp, showing a sequence of 'Error Staging' and 'Started' actions for Test 2. The interface includes a 'Services' panel on the left, a 'Running Jobs' graph on the right, and a bottom toolbar with buttons for 'Add Monitor', 'Delete Monitor', 'Edit Monitor Attributes', 'Run Monitor Immediately', 'Deactivate', and 'View Results'.

Time Stamp	Monitor Name	Document(s)	Action	Message
Tue Jul 20 13:13:17 GMT+530 2010	Test 2	E:\Lisa.5.0.20.81\ex	Error Staging	=====
Tue Jul 20 13:13:17 GMT+530 2010	Test 2	E:\Lisa.5.0.20.81\ex	Started	
Tue Jul 20 13:12:17 GMT+530 2010	Test 2	E:\Lisa.5.0.20.81\ex	Error Staging	=====
Tue Jul 20 13:12:17 GMT+530 2010	Test 2	E:\Lisa.5.0.20.81\ex	Started	
Tue Jul 20 13:11:17 GMT+530 2010	Test 2	E:\Lisa.5.0.20.81\ex	Error Staging	=====
Tue Jul 20 13:11:17 GMT+530 2010	Test 2	E:\Lisa.5.0.20.81\ex	Started	
Tue Jul 20 13:11:02 GMT+530 2010	Test 2-now	E:\Lisa.5.0.20.81\ex	Error Staging	=====
Tue Jul 20 13:11:02 GMT+530 2010	Test 2-now	E:\Lisa.5.0.20.81\ex	Started	
Tue Jul 20 13:10:17 GMT+530 2010	Test 2	E:\Lisa.5.0.20.81\ex	Error Staging	=====
Tue Jul 20 13:10:17 GMT+530 2010	Test 2	E:\Lisa.5.0.20.81\ex	Started	

23.7 CVS Notes

23.7 CVS Notes

Some points to note while using CVS:

- Closing the CVS dashboard or Test Manager, will not interfere with the scheduled tasks.
- The tests are managed by a Coordinator and run on a Simulator. State is maintained in a database on the LISA Registry.
- When you reconnect to the LISA Registry, you will see the dashboard with the current data, and status information.
- Your CVS dashboard will list all the monitors running on an attached LISA Registry, not just the ones initiated by you.
- Any reports generated during scheduled test runs are available and can be seen in the Report Viewer.

24. Reports

24. Reports

LISA has a wealth of features related to the Generation and Capture of data for the purpose of reporting results.

Metric collection is LISA's own metric calculation, which is an extensible reporting mechanism. This has the ability to generate a variety of reports to a variety of outputs.

LISA has a report viewer which contains "Event" and "Metric" information, and information derived from data that was captured during the running of tests. You can set the Events and Metrics that you wish to capture for reporting purposes, in the staging document.

Details on how to specify Events and Metrics can be found in [Staging documents](#) and [Building Test suites](#).

Reports can be selected in the following LISA modules and can be seen in the Report Viewer:

- [Quick Tests](#)
- [Staging Documents](#)
- [Test Suites](#)

Once they have been requested they can be viewed and managed at a later date.

The following topics are available in this section...

24.1 Reports 5.0

This is a new reporting portal for the 5.0 version of LISA. From LISA 5.0, LISA Reports can also be opened via a web portal. To open the Portal, click <http://localhost:1505/reporting> in a web browser. **Note** - Registry needs to be running to start the Portal.

24.2 Legacy Reports 4.0

This is the original reporting portal for the 4.x versions of LISA. Legacy Reports Types. From LISA 5.0, the legacy reports can be invoked separately.

To open the Legacy reports,

Go to **Start Menu > LISA 5.0 > ReportViewer**

This will open the legacy report viewer.

LISA reporting portal supports the following databases for generating reports:

- Derby (our default)
- MS SQL Server
- MySQL (Sun/Oracle)
- Oracle
- DB2

Reports General Info

24.1 Reports 5.0

24.1 LISA 5.0 Reports

LISA has a new reporting portal incorporated in version 5.0, which launches a Jetty instance via the test registry.

Report generation requires Events and Metrics to be selected.

You can generate reports within LISA from:

- [Staging document](#)
- [Quick Test Runs](#)
- [Test Case/Suite Executions](#)

You can view the Reports in a [Report Viewer](#).

From LISA 5.0, LISA Reports can also be opened via a web portal.

LISA 5.0 reports now can also be seen on the Web as it incorporates the new HTML based reporting console. You can view individual test case results and can also narrow down to each step results and response.

To open the Portal, click <http://localhost:1505/reporting> in a web browser.

The new reporting console incorporates new Search capabilities which can narrow search results.

24.1.1 Report Types

24.1.1 Report Types

Reports can be selected in the [Reports Tab](#) of the Staging Document Editor.

Following is a list of the 5.0 reports available in LISA:

5.0 Reports

- [Append Events to Comma-Delimited File](#)
- [Lisa Default Report Generator](#)
- [Print Events to Screen \(standard out\)](#)
- [Report Performance Info to a Database](#)
- [Write Events to JDBC Database Table](#)

A brief summary of the reports is given below:

Append Events to Comma-Delimited File

This report writes out all non-filtered events to a **comma-delimited file** in the following format:
TestName, RunName, Instance, Robot, EventID, Time, ShortDesc, LongDesc.

All repeated tests append data to this same file.

LISA Default Report Generator

This will generate the default LISA report.

Print Events to Screen (standard out)

This report contains all non-Filtered events, as they occur, to the window where the Simulator is running.

Report Performance Info to a Database

A table in a specified database detailing performance data for the test, including average response time by step.

Write Events to JDBC Data table:

This report will open a JDBC specific dialog box, where in you can specify the parameters.

24.1.1.1 Append Events to Comma Delimited File Report

24.1.1.1 Append Events to Comma Delimited File Report

Reports can be selected in the [Reports Tab](#) of the Staging Document Editor.

This report writes events to the given file. If the file does not exist it will attempt to create it.

Events are the information items that LISA generates as models are run; an example is the Step Response Time event or the Step Response event.

The data in the this table can be used for various custom reporting needs. CVS (comma-separated-values) is a popular format for importing into spreadsheets.

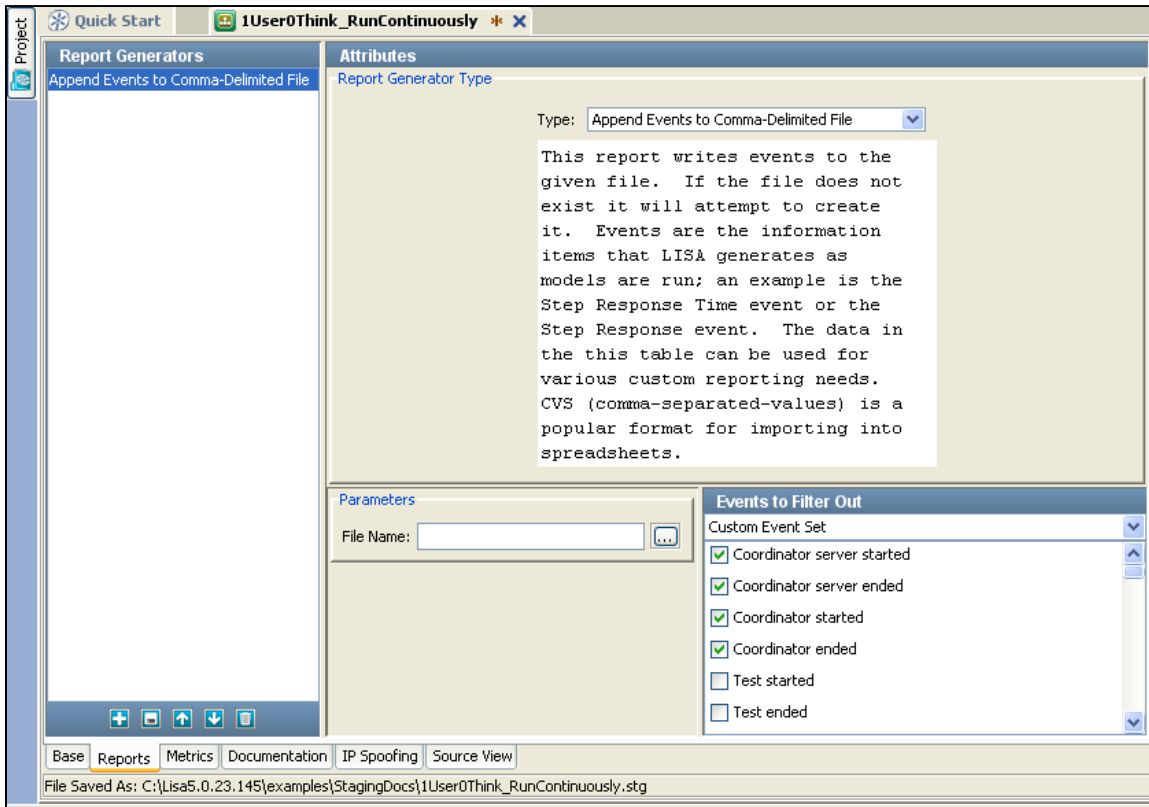
This report writes out all non-filtered events to a **comma-delimited file** in the following format:

TestName, RunName, Instance, Robot, EventID, Time, ShortDesc, LongDesc.

All repeated tests append data to this same file.

Set Report for Viewing

- Open a Staging Document and click the Reports tab.
- Select the "Append Events to Comma-Delimited File" in the Report Generator Type section as shown below:



- Enter the **Name of the file** to append the events to, in the file Name field or Browse to overwrite the previous file.
- Check the **Events to be filtered out**, from the Events to Filter Out section. Only the unchecked Events will be seen in the Generated Report.

Example -

We have used the **1UserThink_RunContinuously.stg** staging document and run the **LISA Config Info.tst** test case.

We are taking the output to comma delimited file called **samplereport.csv**.

Open the csv file in Excel. The generated report can be seen as below:

	A	B	C	D	E	F	G	H	I	J	K	L
10611	LISA Config Info	""	LISA Config Info [1User0Think_RunContinuously]	""	0,0	Info message,"2010.10.28.10.42.49.695"	""	assert bu				
10612	LISA Config Info	""	LISA Config Info [1User0Think_RunContinuously]	""	0,0	Info message,"2010.10.28.10.42.49.695"	""	assert bu				
10613	LISA Home:	E:\Lisa.Workstation5.0.23.181Oct21\										
10614	OS:	Windows XP 5.1										
10615	LISA Config Info	""	LISA Config Info [1User0Think_RunContinuously]	""	0,0	Step response time,"2010.10.28.10.42.49.960"	""	Envi				
10616	LISA Config Info	""	LISA Config Info [1User0Think_RunContinuously]	""	0,0	Info message,"2010.10.28.10.42.49.991"	""	JavaConf				
10617	JVM Version:	1.6.0_21										
10618	Classpath:	E:\Lisa.Workstation5.0.23.181Oct21\install4j\j4runtime.jar;E:\Lisa.Workstation5.0.23.181Oct21\bin\.\bin\lisa-acl.jar										
10619	Free Mem	108302184 bytes										
10620	Total Mem	290127872 bytes										
10621	Max Mem	477233152 bytes										
10622	LISA Config Info	""	LISA Config Info [1User0Think_RunContinuously]	""	0,0	Step response time,"2010.10.28.10.42.50.116"	""	Jave				
10623	LISA Config Info	""	LISA Config Info [1User0Think_RunContinuously]	""	0,0	Step response time,"2010.10.28.10.42.50.538"	""	Bin				
10624	LISA Config Info	""	LISA Config Info [1User0Think_RunContinuously]	""	0,0	Step response time,"2010.10.28.10.42.50.569"	""	LibF				
10625	LISA Config Info	""	LISA Config Info [1User0Think_RunContinuously]	""	0,0	Step response time,"2010.10.28.10.42.50.600"	""	Hot				
10626	LISA Config Info	""	LISA Config Info [1User0Think_RunContinuously]	""	0,0	Step response time,"2010.10.28.10.42.50.663"	""	Writ				
10627	LISA Config Info	""	LISA Config Info [1User0Think_RunContinuously]	""	0,0	Cycle ended normally,"2010.10.28.10.42.50.663"	""	36				
10628	LISA Config Info	""	LISA Config Info [1User0Think_RunContinuously]	""	0,0	Cycle ending,"2010.10.28.10.42.50.663"	""	36663334				
10629	LISA Config Info	""	LISA Config Info [1User0Think_RunContinuously]	""	0,0	Test ended,"2010.10.28.10.42.51.334"	""	3434633737				

24.1.1.2 Default Report Generator Report

24.1.1.2 Default Report Generator Report

Reports can be selected in the [Reports Tab](#) of the Staging Document Editor.

This is the default report by LISA and the Reports tab opens in this Report.

This report captures functional, performance, and metric information and publishes that data to the reporting database referenced by the Registry. The Reporting Portal uses the Reporting Database.

Set Report for Viewing

- Open a Staging Document and click the Reports tab.
- Select the "Default Report Generator" in the Report Generator Type section as shown below:

Functional Test Report

gourik@algiers

Customize



Name : LISA Config Info

Start Time: 28 Oct 2010 10:42:49 AM

Path: E:\Lisa.Workstation5.0.23.18\Oct21\examples\Tests\LISA Config Info.tst

Config: project.config

End Time: 28 Oct 2010 10:42:51 AM

Path: project.config

Staging: 1User0Think_RunContinuously Duration: 1.640 s

Path: E:\Lisa.Workstation5.0.23.18\Oct21\examples\StagingDocs\1User0Think_RunContinuously.stg

Run Name: 1User0Think_RunContinuously

Load Pattern: Run N Times

Distribution Pattern: Percent Distribution

Cycles:	1	Instances:	1	Avg Time(ms):	1640
Think Time:	0%	Plan Duration:	N/A	Plan VUsers:	1
Test Pacing:	N/A				

Tests Summary

Test Attempted	1
Test Started	1
Test Passed	1
Pass Percent	100%
Test Failed	0
Fail Percent	0%

- For load testing, check the Record Performance Metrics in the "Parameters" section:

- Record All Events - If checked, will record all Events.
- Record Properties Set/Referenced - If checked, will record the set or referenced properties.
- Record Performance Metrics - If checked, will record the performance Metrics.
- Record Request/Response - If checked, will record the Request and Response.

- Click the **Add** button  to Add the Report.

Example

We have used the 1UserThink_RunContinuously.stg staging document and run the LISA Config Info.tst test case.

The default report for the same is as shown below:

Functional Test Report

Customize



gourik@algiers

Name : LISA Config Info

Start Time: 28 Oct 2010 10:42:49 AM

Path: E:\Lisa.Workstation5.0.23.18\Oct21\examples\Tests\LISA Config Info.tst

Config: project.config

End Time: 28 Oct 2010 10:42:51 AM

Path: project.config

Staging: 1User0Think_RunContinuously **Duration:** 1.640 s

Path: E:\Lisa.Workstation5.0.23.18\Oct21\examples\StagingDocs\1User0Think_RunContinuously.stg

Run Name: 1User0Think_RunContinuously

Load Pattern: Run N Times

Distribution Pattern: Percent Distribution

Cycles: 1 **Instances:** 1 **Avg Time(ms):** 1640

Think Time: 0% **Plan Duration:** N/A **Plan VUsers:** 1

Test Pacing: N/A

Tests Summary

Test Attempted	1
Test Started	1
Test Passed	1
Pass Percent	100%
Test Failed	0
Fail Percent	0%

24.1.1.3 Print Events to Screen Report

24.1.1.3 Print Events to Screen Report

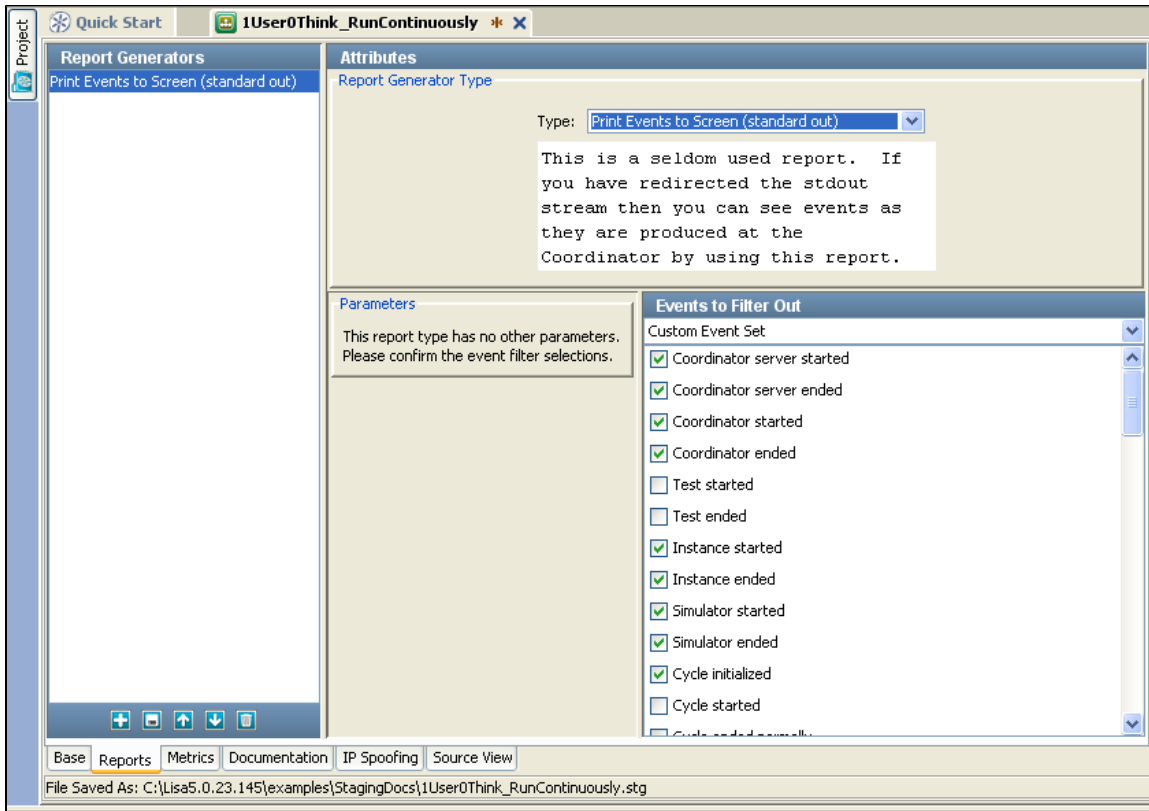
Reports can be selected in the [Reports Tab](#) of the Staging Document Editor.

This is a seldom used report and used to print the Events to screen.


If you have redirected the stdout stream then you can see events as they are produced at the Coordinator by using this report.

Set Report for Viewing

- Open a Staging Document and click the Reports tab.
- Select the "Print Events to Screen" in the Report Generator Type section as shown below:



There are no parameters to be selected for this report.

- Check the events to be filtered out in the "Events to Filter Out" section. Checked out events will not be seen in the Report.
- Click the **Add** button  to Add the Report.

24.1.2.4 Report Performance Info to a Database Report

24.1.2.4 Report Performance Info to a Database Report

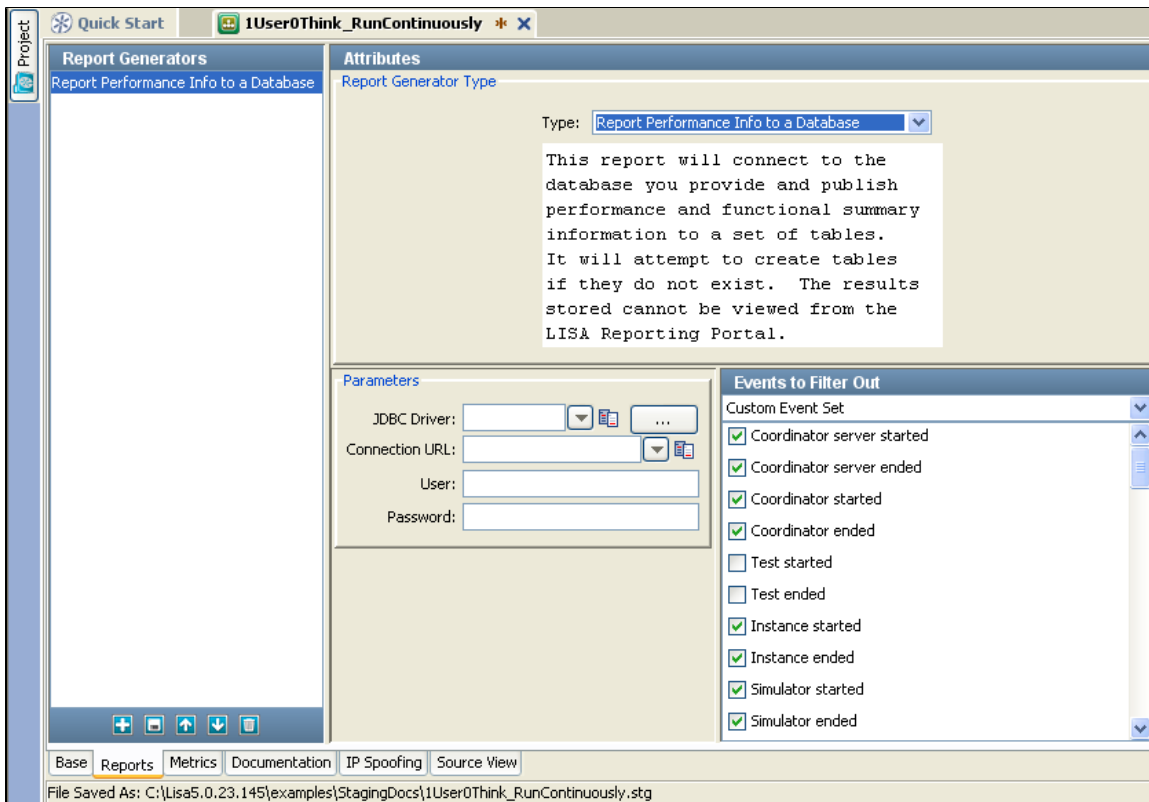
Reports can be selected in the [Reports Tab](#) of the Staging Document Editor.

This report will connect to the database you provide and publish performance and functional summary information to a set of tables. It will attempt to create tables if they do not exist.

The results stored cannot be viewed from the LISA Reporting Portal.

Set Report for Viewing

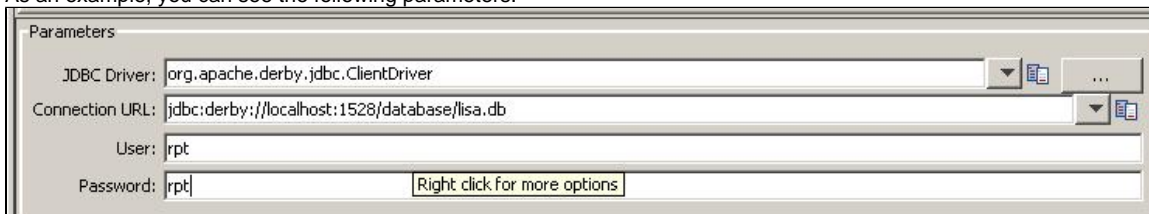
- Open a Staging Document and click the Reports tab.
- Select the "Report Performance Info to a Database Report" in the Report Generator Type section as shown below:



The following parameters need to be set for this report:

1. **JDBC Driver** - Enter the JDBC driver name
2. **Connection URL** - Enter the Connection URL for the same
3. **User** - Enter the User name
4. **Password** - Enter the password

As an example, you can see the following parameters:



- Check the events to be filtered out in the "Events to Filter Out" section. The checked out events will not be seen in the Report.

- Click the **Add** button  to Add the Report.

24.1.2.5 Write Events to JDBC Database Table Report

24.1.2.5 Write Events to JDBC Database Table Report

Reports can be selected in the [Reports Tab](#) of the Staging Document Editor.

This report will connect to the database that you specify and publish events to that database connection. If the table it expects does not exist it will attempt to create it.

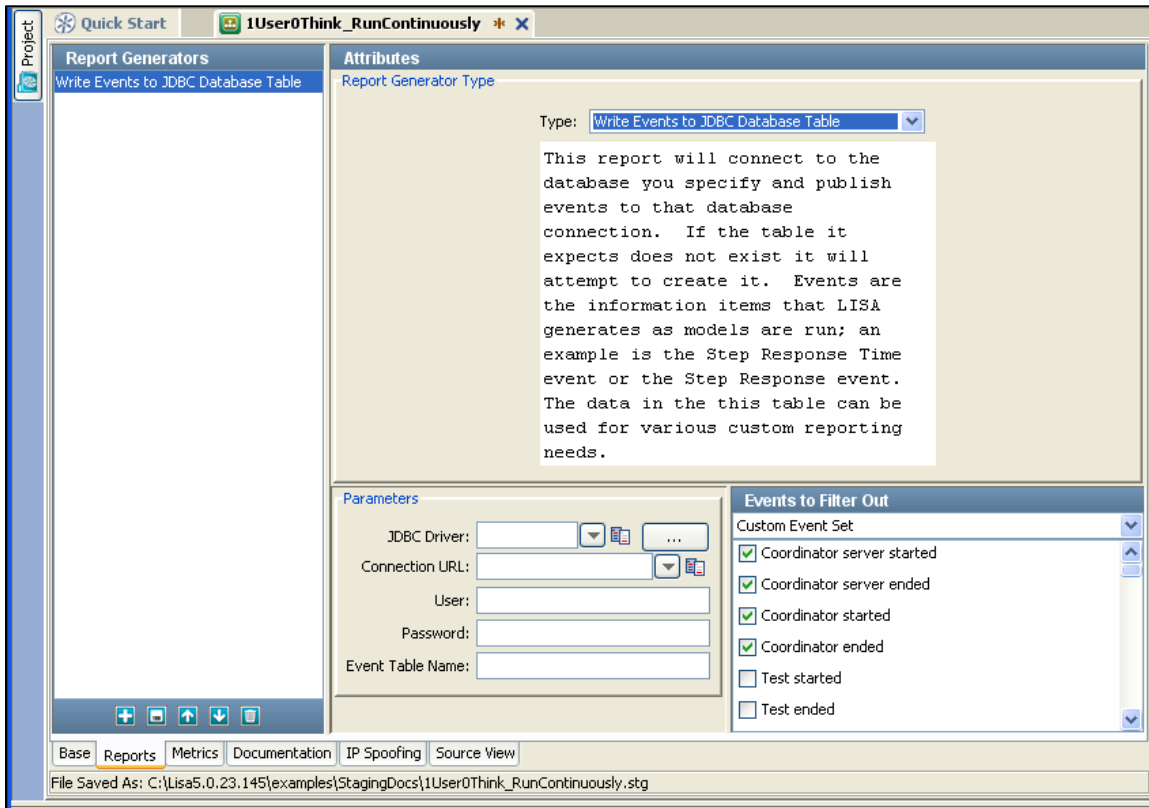
Events are the information items that LISA generates as models are run; an example is the Step Response Time event or the Step Response event.

The data in the this table can be used for various custom reporting needs.

Set Report for Viewing

- Open a Staging Document and click the Reports tab.

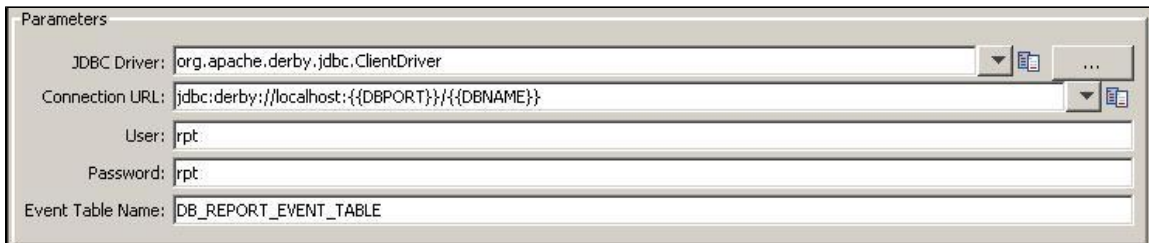
- Select the **"Write Events to JDBC Database Table"** in the Report Generator Type section as shown below:



The following parameters need to be set for this report:

1. **JDBC Driver** - Enter the JDBC driver name
2. **Connection URL** - Enter the Connection URL for the same
3. **User** - Enter its User name
4. **Password** - Enter its password
5. **Event Table Name** - Enter the name of the event table in which the data will be stored.

As an example, see the following:



- Check the events to be filtered out in the **"Events to Filter Out"** section. The checked out events will not be seen in the Report.

- Click the **Add** button  to Add the Report.

Run the test as shown below:

directory (multi-tier-combo.tst).

- [Quick stage](#) the **multi-tier-combo** test case.

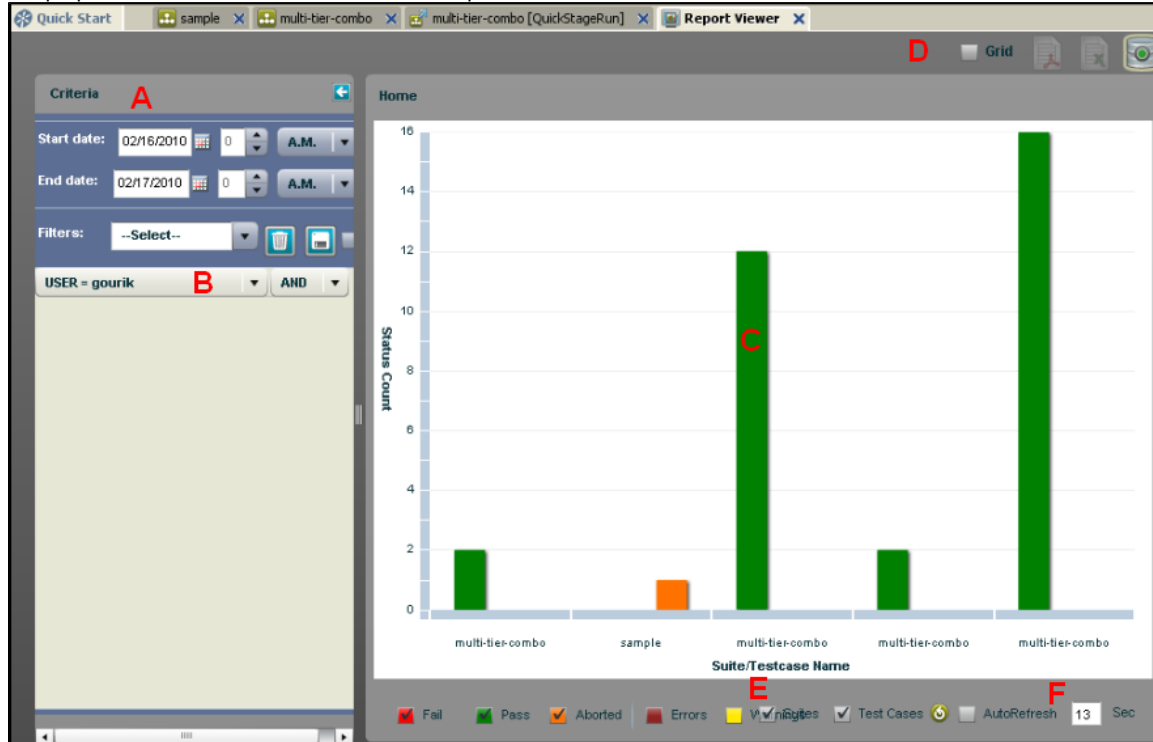


- Once it is complete, click on the Reports icon on the toolbar.

This will open the Report Viewer as shown below:

All the data that is in the database, will be seen in the Report Viewer.

For purpose of illustration, we have shown the report viewer with A,B,C,D,E and F as markers -



Section A

In the left panel, you can select the Date and Time criteria and select the Filters for use.

Start Date/End Date - Select the Start/End date by clicking on the Calendar icon.

By default, the Start and End dates will be in the range of last 24 hours.
In the above example, by default the Start Date is from 02/16/2010 and End Date is upto 02/17/2010.

Time - Select the Start and End time by clicking on the AM drop down.

Section B

Filters - You can create Filters of your choice here.

Enter the name of the filter and click on the **Save** button.

You can also delete a Filter by clicking on the **Delete** button.

Select **Show All Filters** to show the available list of Filters .

For example, we have created a Pass and Fail filter which can be seen in the list as shown below:



You can see the name of the user here -

Select from the **AND/OR** operators for the criteria and click to + sign to Add or - sign to delete.

Section C

The right panel consists of the **Graphs** charted depending on the selected criteria.

For more information, refer to Reports [Graphical View](#)

Section D

You can **export the report data** to a .PDF file or Excel file to view further details.

For more information, refer to Exporting the report to a [PDF File](#) and [Excel File](#) .

Section E

You can filter the reports depending on the test cases which have passed or failed.

For more information, refer to [Filtering of Reports](#).

Auto Expire Reports

You can set an Report Expire Timer, which will allow you to keep reports for scheduled time.

For more information on Report Timer, see [Auto Expire Report](#) .

24.1.3 Generating Reports - Example

24.1.3 Generating Reports - Example

Available LISA 5.0 Reports

Following is a list of 5.0 reports available in LISA:

- Append Events to Comma-Delimited File
- Lisa Default Report Generator
- Print Events to Screen (standard out)
- Report Performance Info to a Database
- Write Events to JDBC Database Table

Reports can be selected in the [Staging Document Editor](#).

Example

24.1.4 Filtering of Reports

24.1.4 Filtering of Reports for Viewing

The right panel of the reporting portal displays the reporting graphs.

You can view reports by filtering reports using certain criteria.

Once you select the criteria, the report viewer will show graphs only for selected criteria.

Following are the filtering criteria:



Pass

-If clicked will show all the test cases/suites that have passed.



Fail

- If clicked will show all the test cases/suites that have failed.



Aborted

- If clicked will show all the test cases/suites that have been Aborted.



Error

- If clicked will show all the test cases/suites that have generated errors.



Warning

- If clicked will show all the test cases/suites that have generated warnings.



Suites

- If clicked will show all the test suites results.



Test Cases

- If clicked will show all the test cases results.

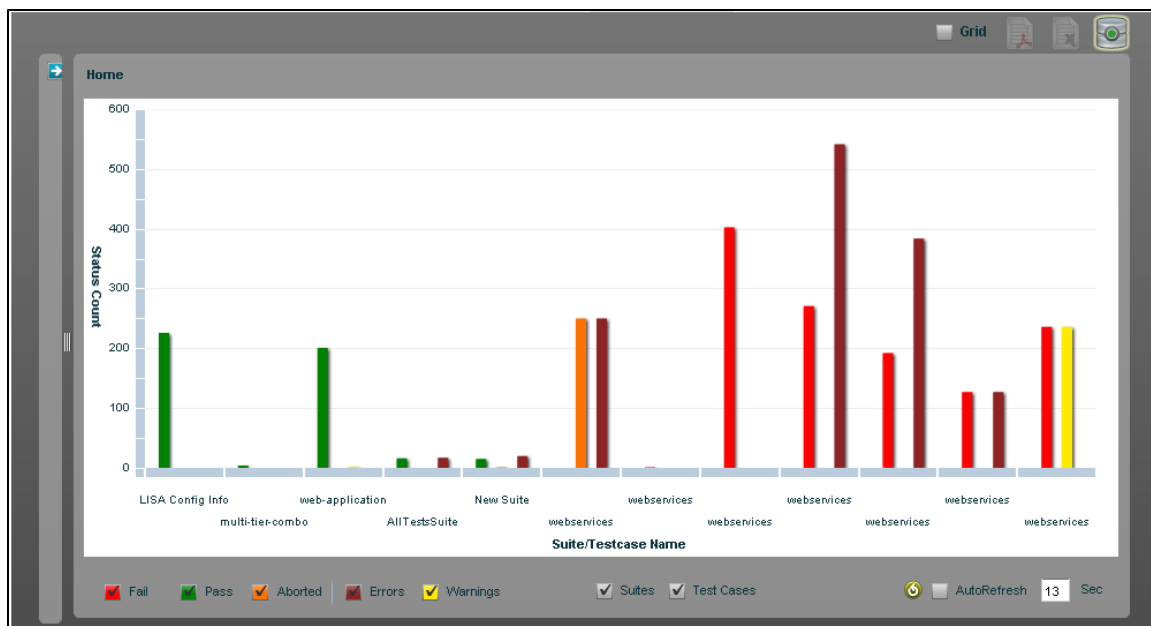
For the purpose of illustration, we have executed and run a few test cases and a test suite from the %LISA_HOME% examples directory.

The test cases are -

- multi-tier-combo.tst
- LisaConfig Info.tst
- Web Services.tst
- Web application.tst
- AllSuite.ste

All the filtering creiterias are applied and shown as example in the next section.


Note - The report viewer will show the reports for all the testcases, test suites that have been staged and that are currently in its database. So we need to open all the test cases and stage them so that the data for reports is ready.



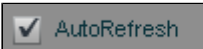
In the above report, there is no search criteria specified as all are clicked, hence we will see the report for all the test cases that have passed,

failed, aborted, had errors and warnings.

To refresh the reports,

Click on the Refresh  icon.

To AutoRefresh the reports,

Check the AutoRefresh icon  and enter the number of seconds after which you want the reports to be refreshed. For example, in the above image, we have set the reports to be auto refreshed after every 13 seconds.

24.1.4.1 Pass Tests Reports


24.1.4.1 Pass Tests Report

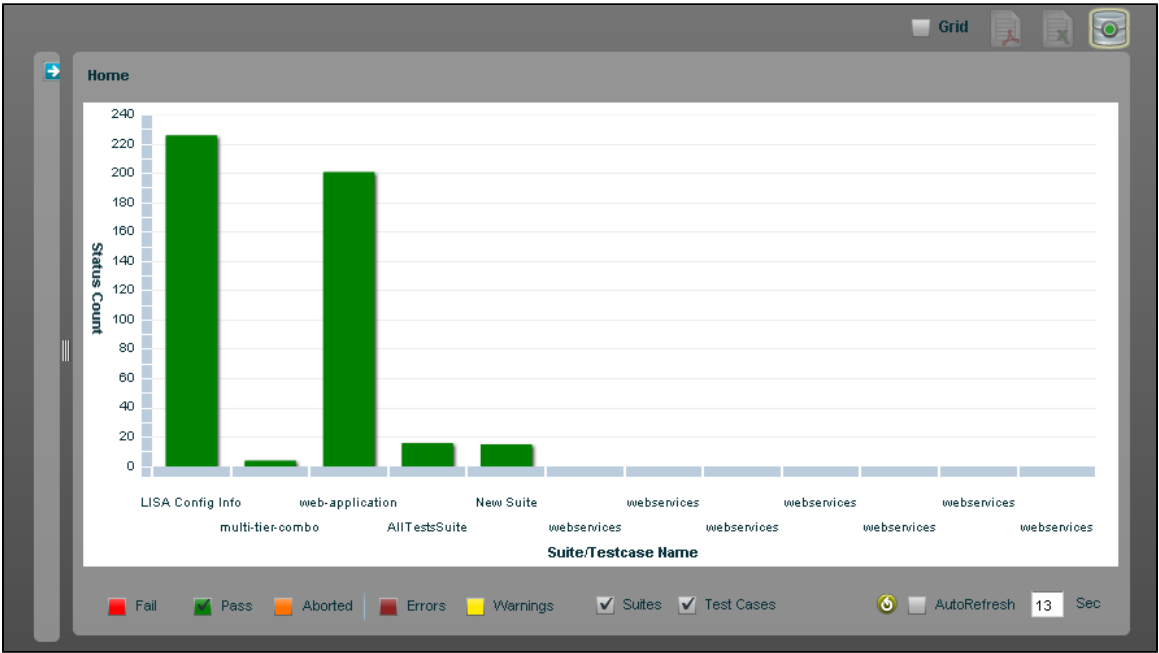
The right panel of the reporting portal displays the reporting graphs.

At the bottom of the graph, there is the search criteria from where you can select the reports to be seen.

For the purpose of illustration, we have executed and run a few test cases and a test suite from the %LISA_HOME% examples directory.

The example test cases are: multi-tier-combo.tst, LisaConfig Info.tst, Web Services.tst, Web application.tst, AllSuite.ste

 **Pass** - Click to show all the test cases/suites that have passed as shown below:



24.1.4.2 Fail Tests Reports


24.1.4.2 Fail Tests Report

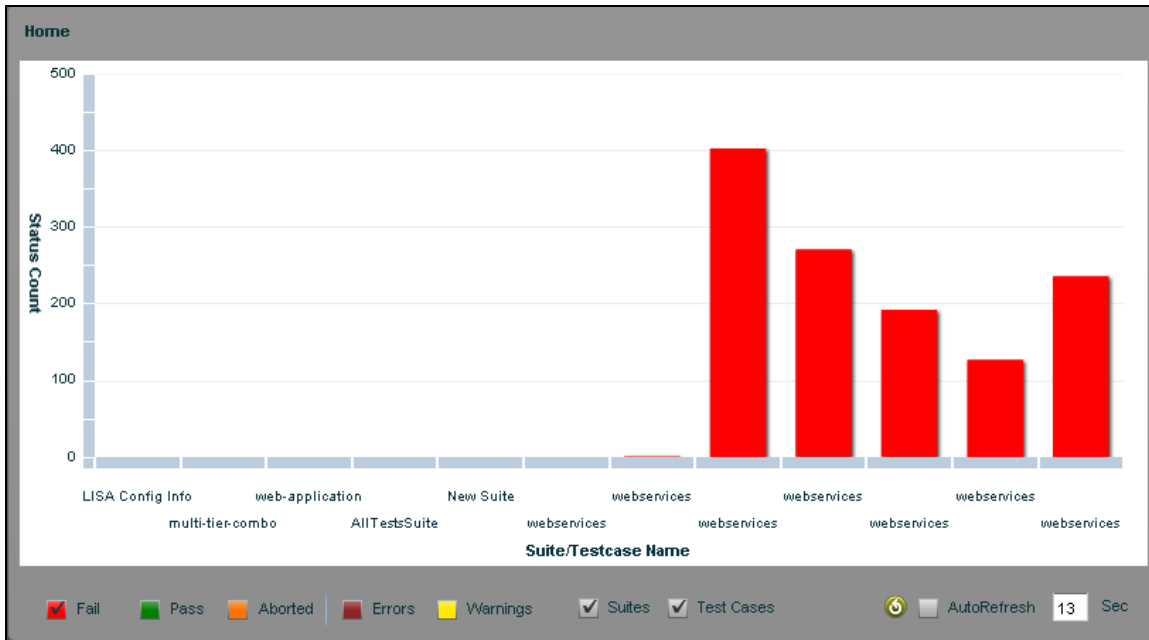
The right panel of the reporting portal displays the reporting graphs.

At the bottom of the graph, there is the search criteria from where you can select the reports to be seen.

For the purpose of illustration, we have executed and run a few test cases and a test suite from the %LISA_HOME% examples directory.

The example test cases are: multi-tier-combo.tst, LisaConfig Info.tst, Web Services.tst, Web application.tst, AllSuite.ste

 **Fail** - Click to show all the test cases/suites that have Failed as shown below:



24.1.4.3 Aborted Tests Reports

24.1.2.3 Aborted Tests Report

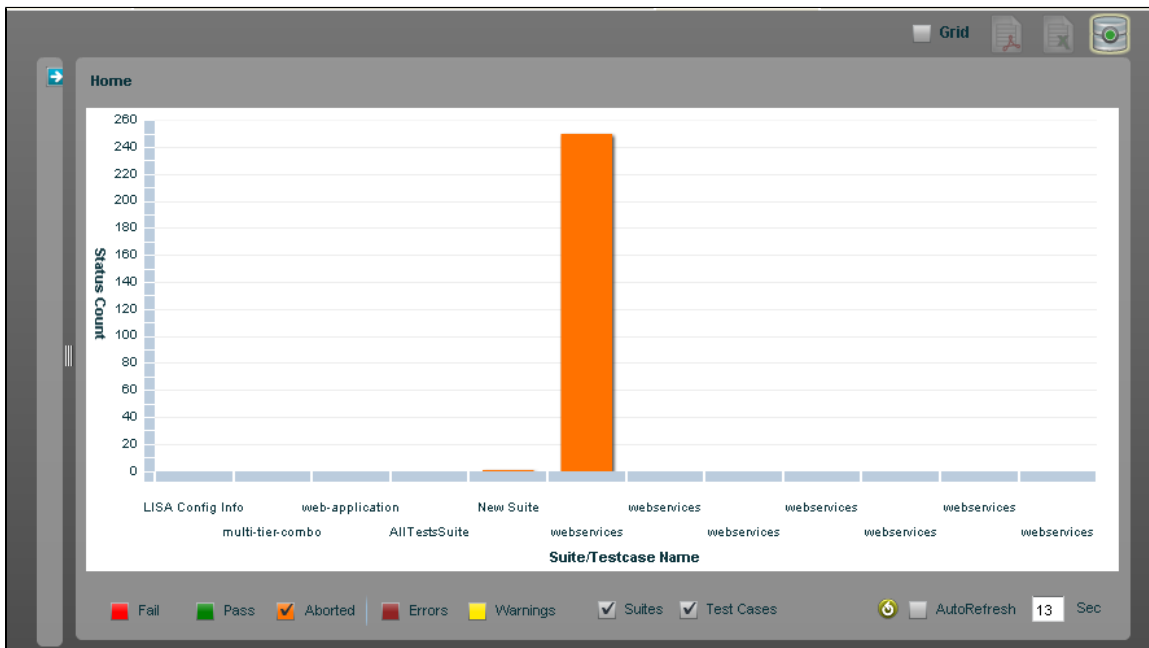
The right panel of the reporting portal displays the reporting graphs.

At the bottom of the graph, there is the search criteria from where you can select the reports to be seen.

For the purpose of illustration, we have executed and run a few test cases and a test suite from the %LISA_HOME% examples directory.

The example test cases are: multi-tier-combo.tst, LisaConfig Info.tst, Web Services.tst, Web application.tst, AllSuite.ste

☒ **Abort** - Click to show all the test cases/suites that have **Aborted** as shown below:



24.1.4.4 Error Related Tests Reports


24.1.4.4 Error Related Tests Report

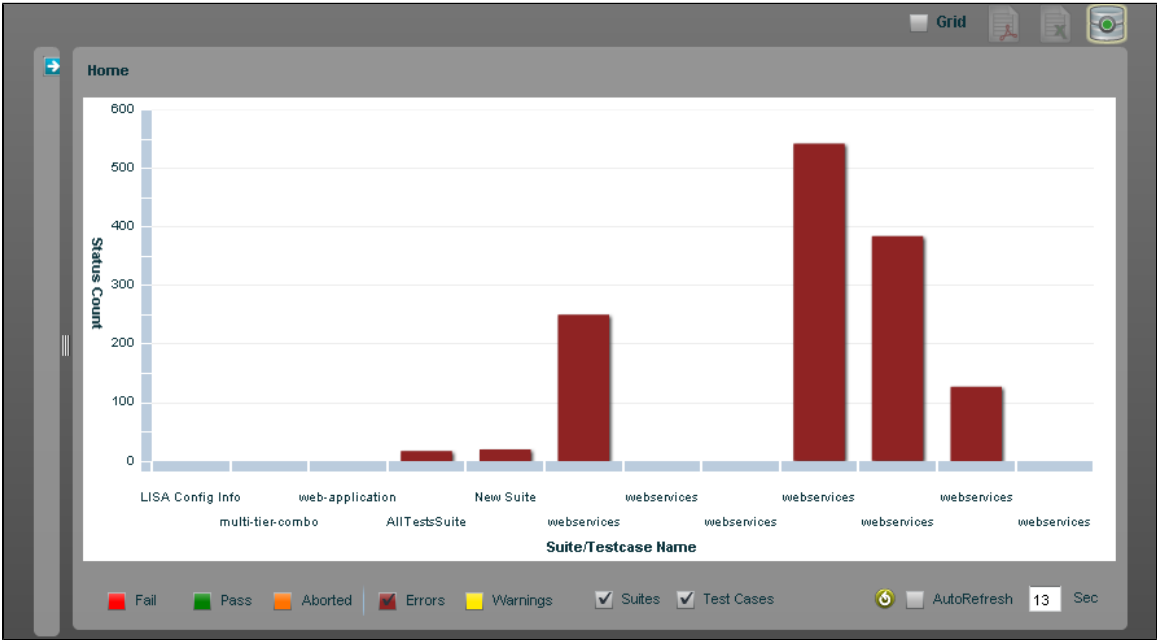
The right panel of the reporting portal displays the reporting graphs.

At the bottom of the graph, there is the search criteria from where you can select the reports to be seen.

For the purpose of illustration, we have executed and run a few test cases and a test suite from the %LISA_HOME% examples directory.

The example test cases are: multi-tier-combo.tst, LisaConfig Info.tst, Web Services.tst, Web application.tst, AllSuite.ste

 **Error** - Click to show all the test cases/suites that gave **Errors** as shown below:



24.1.4.5 Warnings Related Test Reports


24.1.4.5 Warnings Related Tests Report

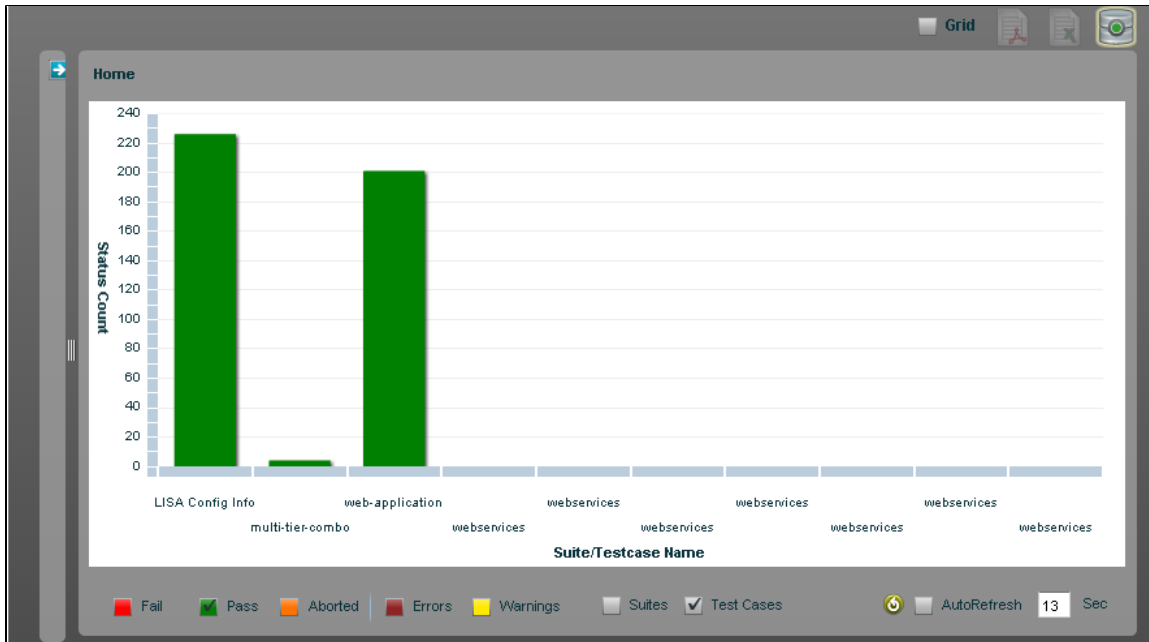
The right panel of the reporting portal displays the reporting graphs.

At the bottom of the graph, there is the search criteria from where you can select the reports to be seen.

For the purpose of illustration, we have executed and run a few test cases and a test suite from the %LISA_HOME% examples directory.

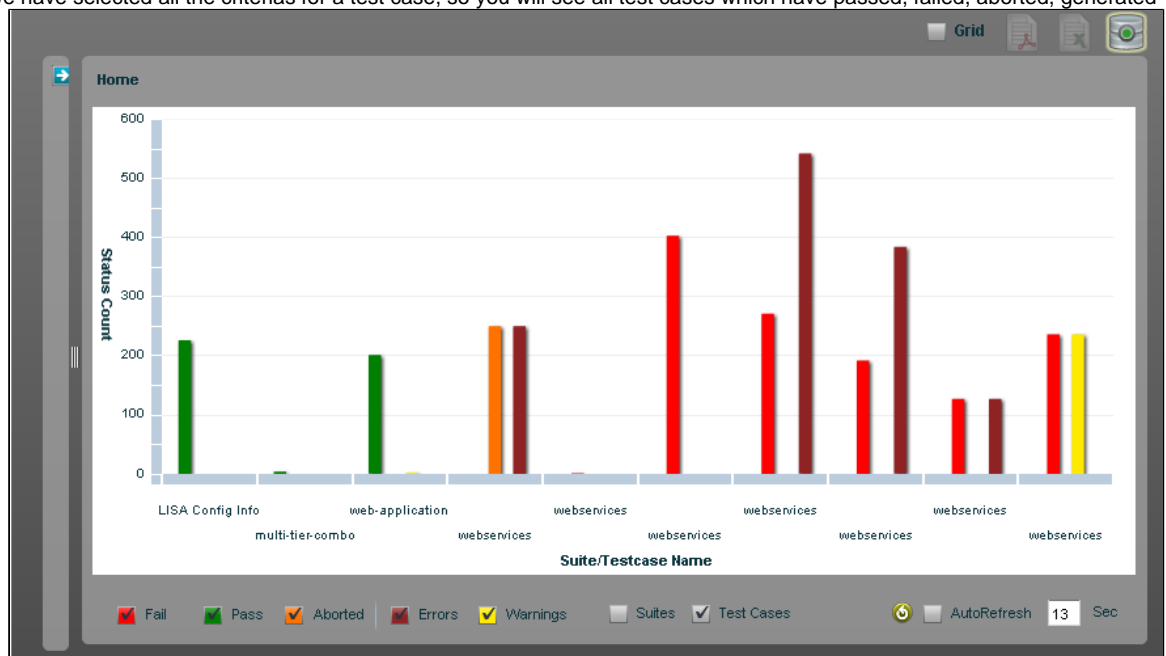
The example test cases are: multi-tier-combo.tst, LisaConfig Info.tst, Web Services.tst, Web application.tst, AllSuite.ste

 **Warning** - Click to show all the test cases/suites that gave **Warnings** as shown below:



Similarly, you can define the search criteria for test cases that have failed and/or have generated errors and/or warnings.

For example, below we have selected all the criterias for a test case, so you will see all test cases which have passed, failed, aborted, generated



errors and warnings...

24.1.4.7 Test Suites Related Test Reports

24.1.4.7 Test Suites Related Tests Report

The right panel of the reporting portal displays the reporting graphs.

At the bottom of the graph, there is the search criteria from where you can select the reports to be seen.

For the purpose of illustration, we have executed and run a few test cases and a test suite from the %LISA_HOME% examples directory.

The example test cases are: multi-tier-combo.tst, LisaConfig Info.tst, Web Services.tst, Web application.tst, AllSuite.ste

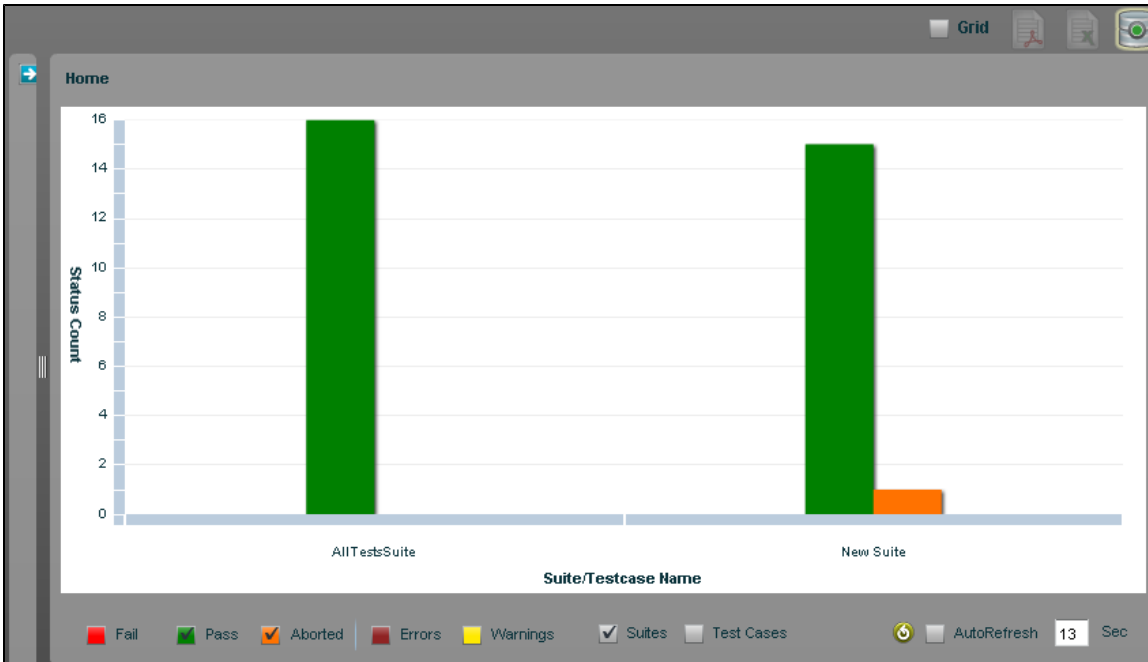
When you select these two icons -

☒ - Test Suite



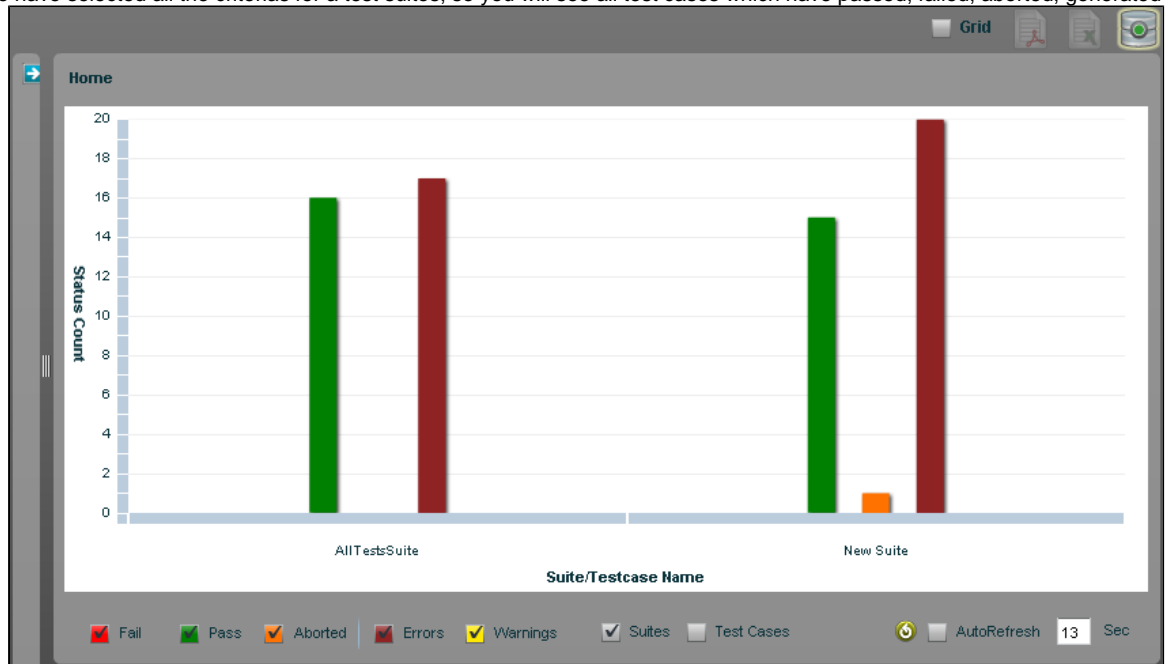
- Abort

You can refine your search for **Aborted Test Suites** as shown below:



Similarly, you can define the search criteria for test suites that have failed and/or have generated errors and/or warnings.

For example, below we have selected all the criterias for a test suites, so you will see all test cases which have passed, failed, aborted, generated



errors and warnings...

24.1.5 Viewing Reports

24.1.5 Viewing Reports

In the Reports viewer, the reports can be viewed in two formats:

- [Graph View](#) - This is the default view of the reporting portal. Can see all the reports in the graphical format.
- [Grid View](#) - You can select the grid view to have the data arranged in a grid format.

Both of these views are explained in detail in the next section.

24.1.5.1 Reports - Graphical View

24.1.5.1 Reports - Graphical View

By default, all the reports can be seen as Graphs.

To apply Test Case Criterias

You can sort/filter out the reports data on certain **search criteria**. Once you select the criteria, the report viewer will show data for selected criteria.

Following filters can be applied to the data (Test cases/Test Suites): ☒ Pass ☒ Fail ☒ Aborted ☒ Error ☒ Warning ☒ Test Suites
☒ Test Cases

To refresh the reports,

Click on the Refresh  icon.

To AutoRefresh the reports after setting time,

Check the AutoRefresh icon ☒ AutoRefresh 13 Sec and enter the number of seconds after which you want the reports to be refreshed.

For example, in the above image, we have set the reports to be auto refreshed after every 13 seconds.

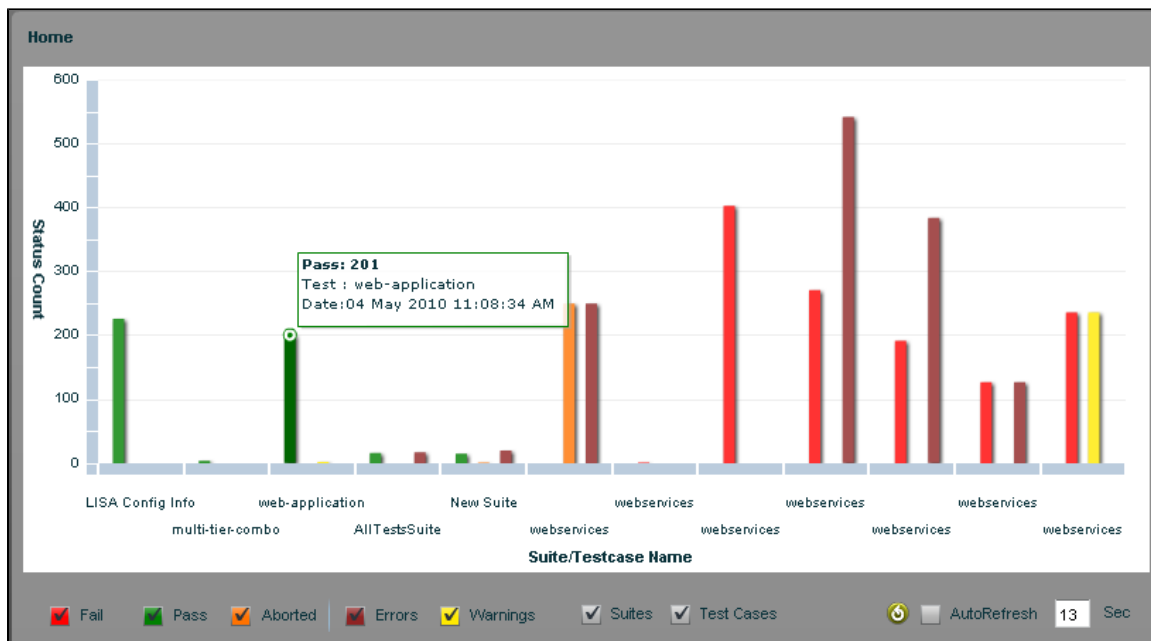
Example - Graphical View

For the purpose of illustration, we have the following reports graph, which contains the following test cases: multi-tier-combo.tst, LisaConfig Info.tst, Web Services.tst, Web application.tst, AllSuite.ste

The graphical report of all the above test cases is as shown below. As there is no search criteria specified, we will see the report for all the test cases that have passed, failed, aborted, had errors and warnings.

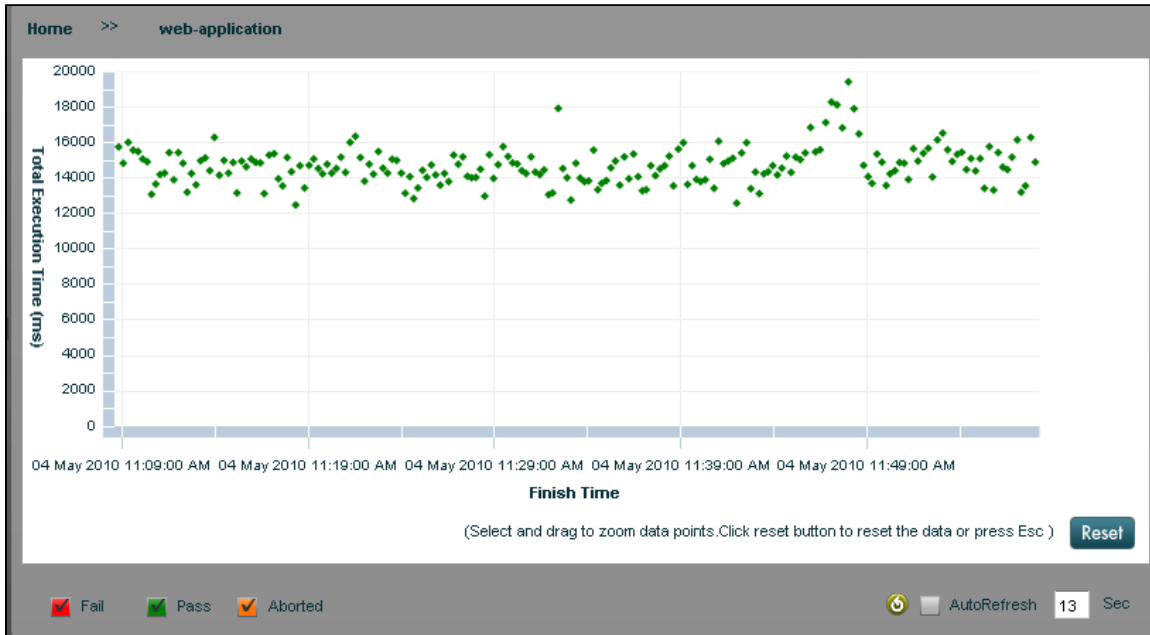
For the purpose of illustration, we have selected the "Web-application" test case.

Clicking on the test case, will pop up a information box which informs about the test case name, test execution details like the number of pass/fail tests and the date of execution as shown below:



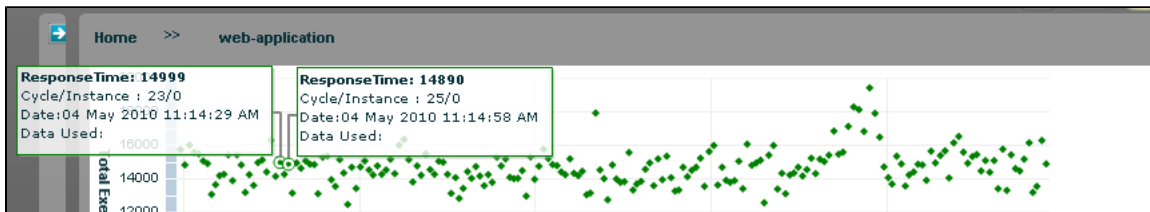
Click on any Graphical bar of any of the Test Case to view more information regarding the same.

For example, we have now clicked on the "web-application" test case to know its details. Another graph with dots at a more detailed level will be seen as below:



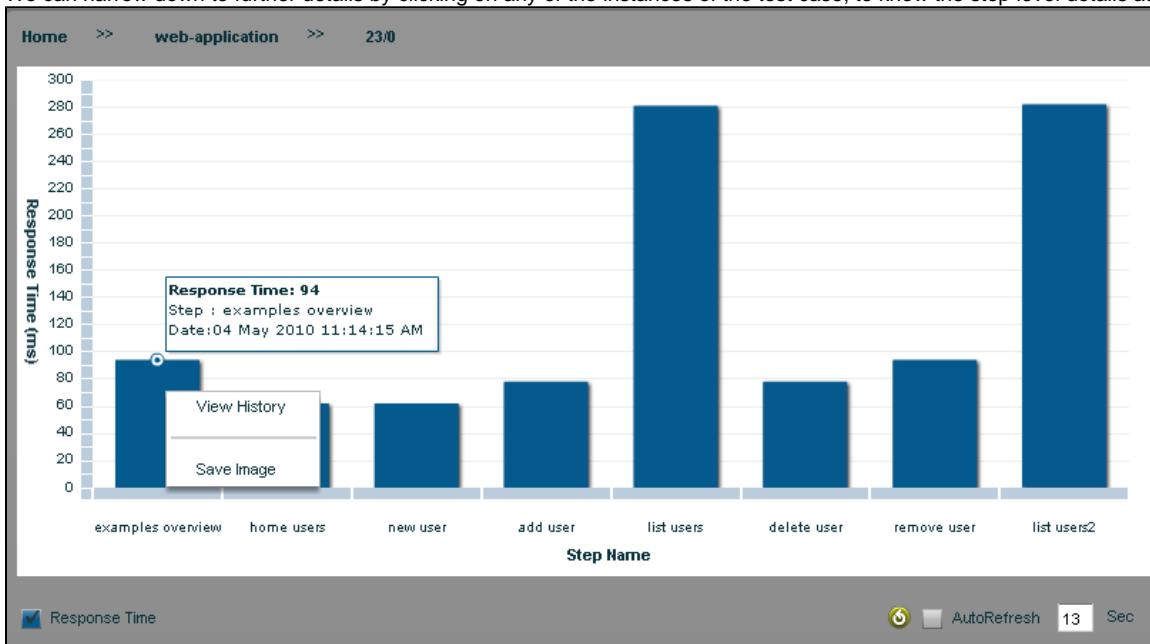
When you move your mouse over these dots, you will know that each dot represents a Cycle of the test case.

The details seen will be - The Response time in milliseconds, the Cycle/Instance number, the date of the cycle and the Data used as shown below:



Click on these dots to know more details regarding each test step as shown below.

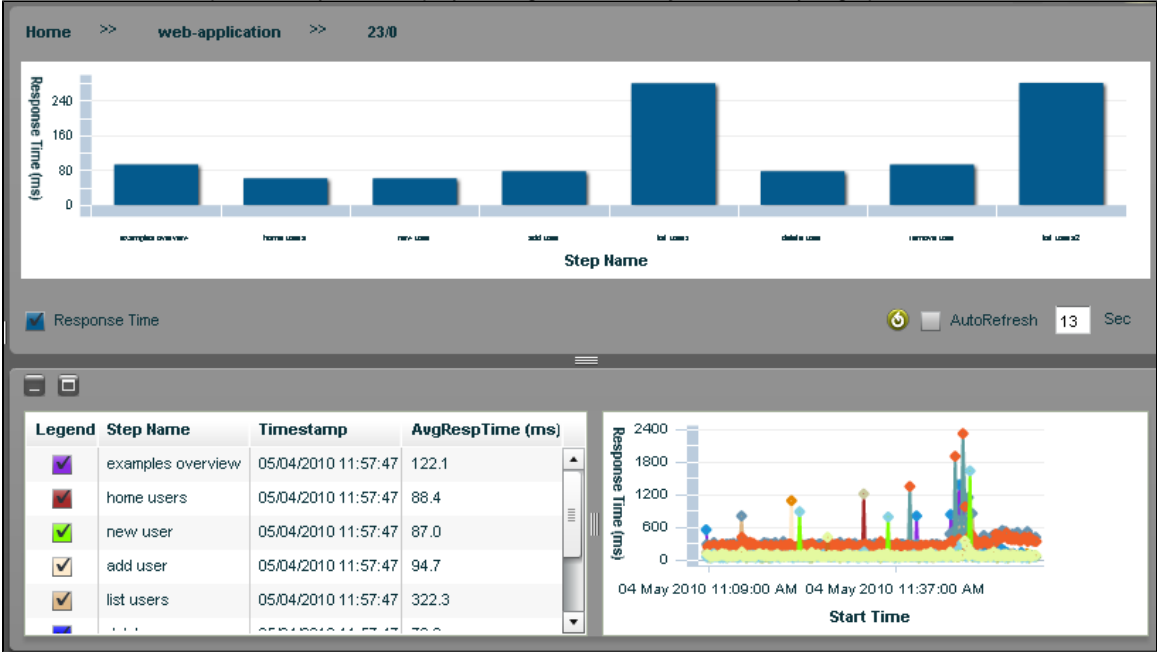
We can narrow down to further details by clicking on any of the instances of the test case, to know the step level details as shown below:



As you can see, this report gives information regarding the Response Time, the step name and the Date of execution.

You can right click the step bar and "Save the image".

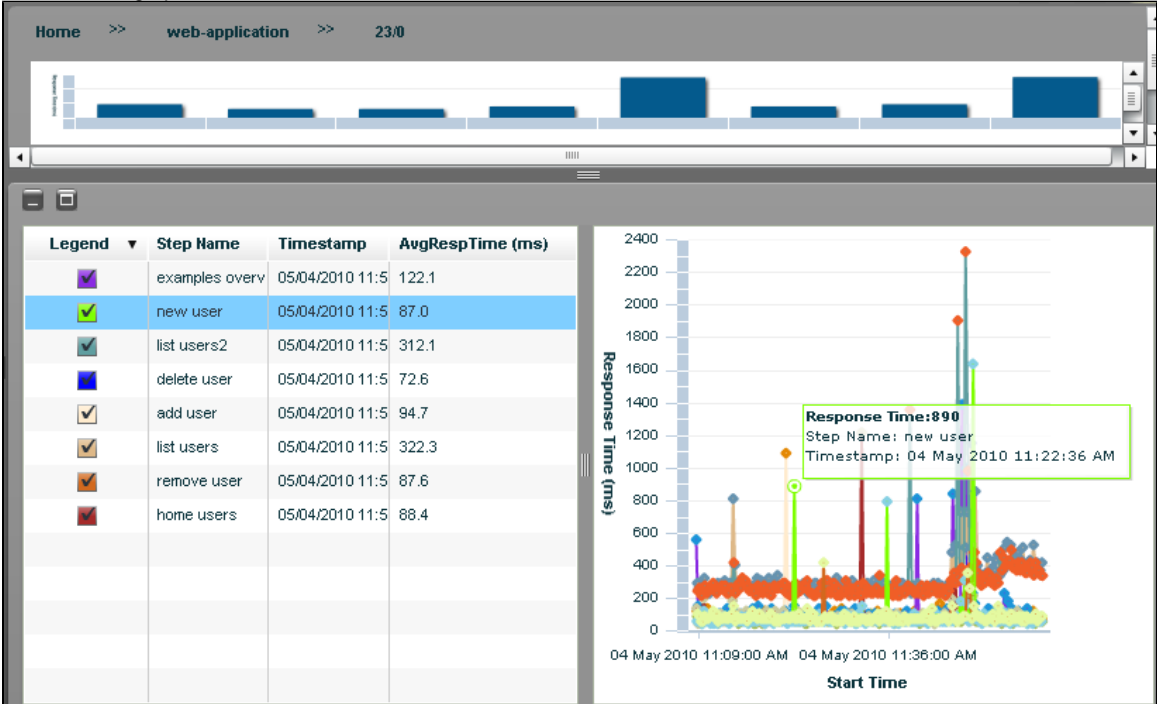
You can view the complete History of the step by clicking "**View History**". A summary of graphs can be seen as below:



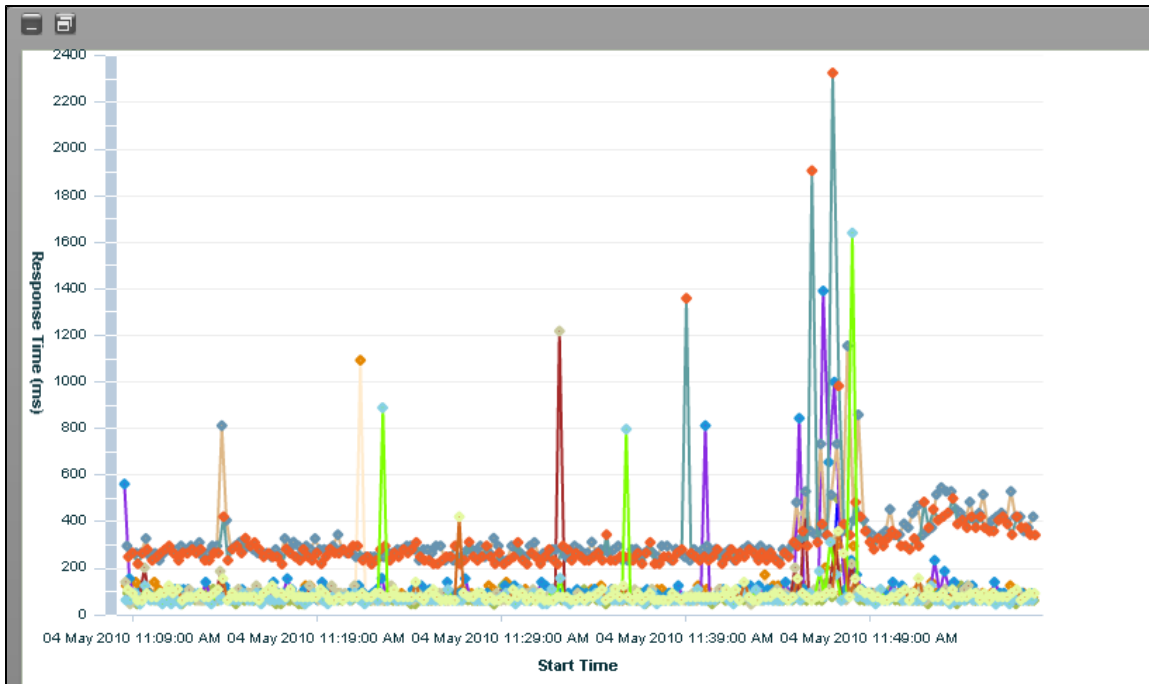
You can minimize the upper graph to view the **collective summary** of all the steps in one graph as shown below:

A **Legend** denotes the color of each individual step and the color in the graph represents all information about that step.

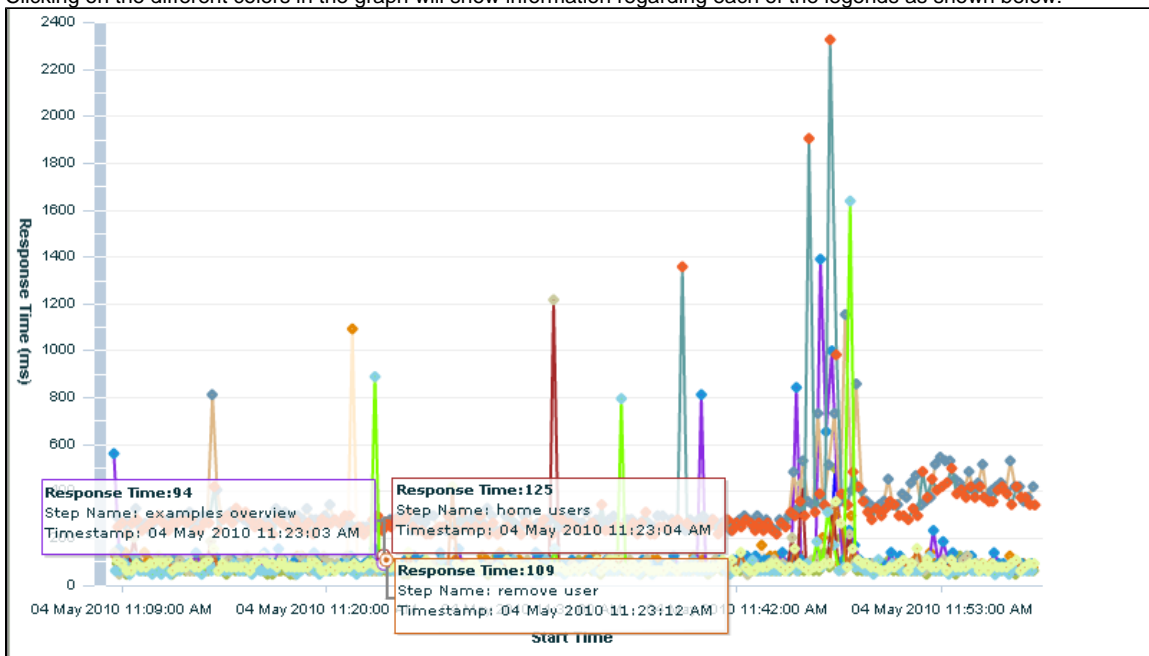
For example, we have selected "**Lime green**" color, the legend for the "**new user**" step, corresponding information regarding the "new user" step is seen in the graph in the same color as shown below:



You can analyze the graphs at a detailed level by clicking on the full screen button :



Clicking on the different colors in the graph will show information regarding each of the legends as shown below:



24.1.5.2 Reports - Grid View

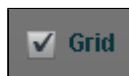
24.1.5.2 Reports - Grid View

The right panel of the reporting portal displays the reporting graphs.

By default, all the reports can be seen as graphs.

You can also see the report results in a **Grid view**.

To view the results in a grid,

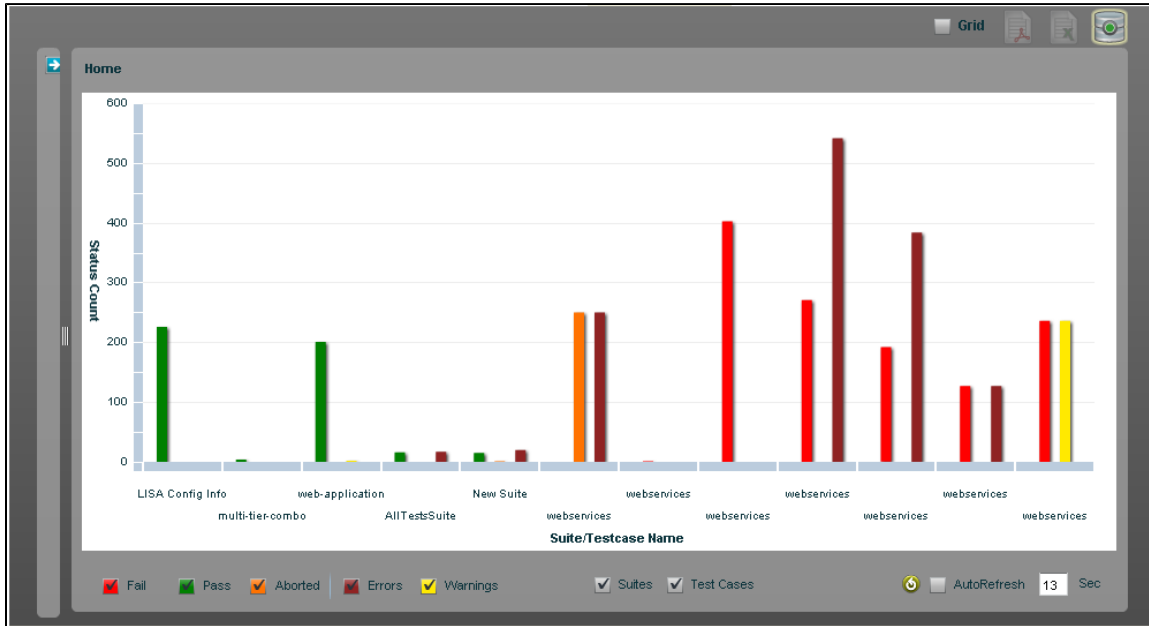


Check the Grid icon on the top right corner.

Now the reports graph will change to a grid view as shown below.

For the purpose of illustration, we have the following reports graph, which contains the following test cases: multi-tier-combo.tst, LisaConfig Info.tst, Web Services.tst, Web application.tst, AllSuite.ste

The Reports graph is as shown below:



In the above report, there is no search criteria specified as all are clicked, hence we will see the report for all the test cases that have passed, failed, aborted, had errors and warnings.

The same report in the grid format can be seen by checking the Grid icon as shown below:

Timestamp	Suite Name	Pass	Fail	Could not	Errors	Warning	Total Execution Time (hr min sec ms)
04 May 2010 10:33:09 AM	LISA Config Info	226	0	0	0	0	33 min 37 sec 402 ms
04 May 2010 10:33:57 AM	multi-tier-combo	4	0	0	0	0	03 min 58 sec 266 ms
04 May 2010 11:08:34 AM	web-application	201	0	0	0	2	49 min 28 sec 294 ms
04 May 2010 11:44:58 AM	AllTestsSuite	16	0	0	17	0	03 min 19 sec 847 ms
04 May 2010 11:45:19 AM	New Suite	15	0	1	20	0	09 min 12 sec 824 ms
04 May 2010 11:55:05 AM	webservices	0	0	250	250	0	12 sec 483 ms
04 May 2010 11:59:35 AM	webservices	0	1	0	0	0	03 sec 172 ms
04 May 2010 12:00:17 PM	webservices	0	403	0	0	0	34 sec 671 ms
04 May 2010 12:02:52 PM	webservices	0	271	0	542	0	12 sec 796 ms
04 May 2010 12:04:54 PM	webservices	0	192	0	384	0	23 sec 342 ms
04 May 2010 12:08:12 PM	webservices	0	127	0	127	0	16 sec 697 ms
04 May 2010 12:10:34 PM	webservices	0	236	0	0	236	26 sec 168 ms

You can then select each of the test case within the report to find out more details regarding the same. Clicking on any of the test case, will open further details regarding the same.

For example, we have now clicked on the second test case in the list -**multi-tier-combo** test case to know its details.

Home >> multi-tier-combo						
Timestamp	Instance	Cycle	Runs	Status	Exec Time (t	Data Used
04 May 2010 10:35:09 AM	0	0	multi-tier-combo [QuickStageRun]	PASS	01 min 11 sec	
04 May 2010 10:36:10 AM	0	1	multi-tier-combo [QuickStageRun]	PASS	01 min 730 ms	
04 May 2010 10:36:46 AM	0	2	multi-tier-combo [QuickStageRun]	PASS	35 sec 591 ms	
04 May 2010 10:37:32 AM	0	3	multi-tier-combo [QuickStageRun]	PASS	45 sec 997 ms	
04 May 2010 10:37:55 AM	0	4	multi-tier-combo [QuickStageRun]	PASS	23 sec 467 ms	

The details seen will be: The time when the test case was Run, its Instances, its Cycles, its Status, its Execution time and the Data used as shown above.

We can narrow down to further details by clicking on any of the instances of the test case, to know the step level details as shown below:

We have selected the first instance of the multi-tier-combo test case. So all the information related to this instance of the test case will be seen:

Home >> multi-tier-combo >> 0/0							
Timestamp	StepName	Status	Warnings	Response Time(m	Bandwidth (b	Request	Response
04 May 2010 10:33:57 AM	Add User	PASS	0	1797	2443	Click for Detail	Click for Detail
04 May 2010 10:34:05 AM	Verify User Added	PASS	0	15	0	Click for Detail	Click for Detail
04 May 2010 10:34:06 AM	Deposit Money	PASS	0	2718	0	Click for Detail	Click for Detail
04 May 2010 10:34:14 AM	Get User	PASS	0	500	0	Click for Detail	Click for Detail
04 May 2010 10:34:16 AM	Login	PASS	0	3453	18905	Click for Detail	Click for Detail
04 May 2010 10:34:25 AM	Account Activity	PASS	0	11218	19001	Click for Detail	Click for Detail
04 May 2010 10:34:46 AM	Pay Bill Online	PASS	0	765	16631	Click for Detail	Click for Detail
04 May 2010 10:34:54 AM	Logout	PASS	0	250	14480	Click for Detail	Click for Detail
04 May 2010 10:35:01 AM	Get Transactions	PASS	0	94	0	Click for Detail	Click for Detail
04 May 2010 10:35:01 AM	Delete User	PASS	0	172	1122	Click for Detail	Click for Detail
04 May 2010 10:35:09 AM	Verify User Deleted	PASS	0	16	0	Click for Detail	Click for Detail

As you can see, this report gives a further detailed view of the Test Case, by showing test step level information.

The information seen at this level is the test step execution time stamp, step name, step status, number of warnings for this step, Response time in milliseconds, Bandwidth in Bytes, and step Request/Response related details.

For details regarding the Request,

Click on the Request of the selected test step, ex - Add User. You will see the following information -

The screenshot shows a window titled "Step Name : Add User (Request)". The main content area displays XML request details for a test step. The XML is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ejb3="http://ejb3.examples.itko.com/">
  <soapenv:Body>
    <addUserObject xmlns="http://ejb3.examples.itko.com/">
      <userObject xmlns="">
        <accounts>
          <balance>101.01</balance>
          <name>Basic Checking</name>
          <type>CHECKING</type>
        </accounts>
      </userObject>
    </addUserObject>
  </soapenv:Body>
</soapenv:Envelope>
```

On the right side of the window, there is a table with the following columns: "bandwidth (bytes)", "Request", and "Response". The table contains 10 rows of data, each with a bandwidth value and two "Click for Detail" links.

bandwidth (bytes)	Request	Response
443	Click for Detail	Click for Detail
	Click for Detail	Click for Detail
	Click for Detail	Click for Detail
	Click for Detail	Click for Detail
8905	Click for Detail	Click for Detail
9001	Click for Detail	Click for Detail
6631	Click for Detail	Click for Detail
4480	Click for Detail	Click for Detail
	Click for Detail	Click for Detail
122	Click for Detail	Click for Detail

For details regarding the Response,

Click on the Response of the selected test step, ex - Add User. You will see the following information -

The screenshot shows a window titled "Step Name : Add User (Response)". The main content area displays XML response details for a test step. The XML is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header />
  <env:Body>
    <ns2:addUserObjectResponse xmlns:ns2="http://ejb3.examples.itko.com/">
      <return>
        <accounts>
          <balance>103.22</balance>
          <id>87189496470</id>
          <name>Orange Savings</name>
          <type>SAVINGS</type>
        </accounts>
      </return>
    </ns2:addUserObjectResponse>
  </env:Body>
</env:Envelope>
```

On the right side of the window, there is a table with the following columns: "bandwidth (bytes)", "Request", and "Response". The table contains 10 rows of data, each with a bandwidth value and two "Click for Detail" links.

bandwidth (bytes)	Request	Response
443	Click for Detail	Click for Detail
	Click for Detail	Click for Detail
	Click for Detail	Click for Detail
	Click for Detail	Click for Detail
8905	Click for Detail	Click for Detail
9001	Click for Detail	Click for Detail
6631	Click for Detail	Click for Detail
4480	Click for Detail	Click for Detail
	Click for Detail	Click for Detail
122	Click for Detail	Click for Detail

24.1.6 Exporting Reports

24.1.6 Exporting Reports

In the Reports viewer, the reports can be exported in two formats:

- PDF -
- Excel -

Both of these views are explained in detail in the next section.

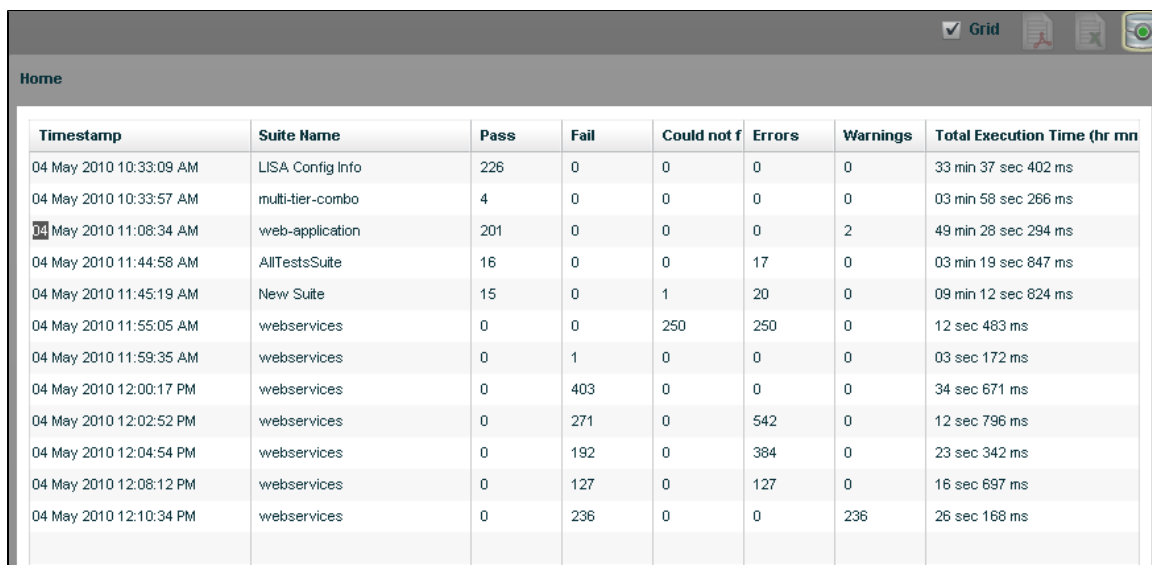
24.1.6.1 Exporting Reports to Excel

24.1.6.1 Exporting to Excel

You can export all the LISA Reports related data to: An Excel file or a PDF file.

To Export data to Excel file,

- Open the report which you want to export to an Excel file.

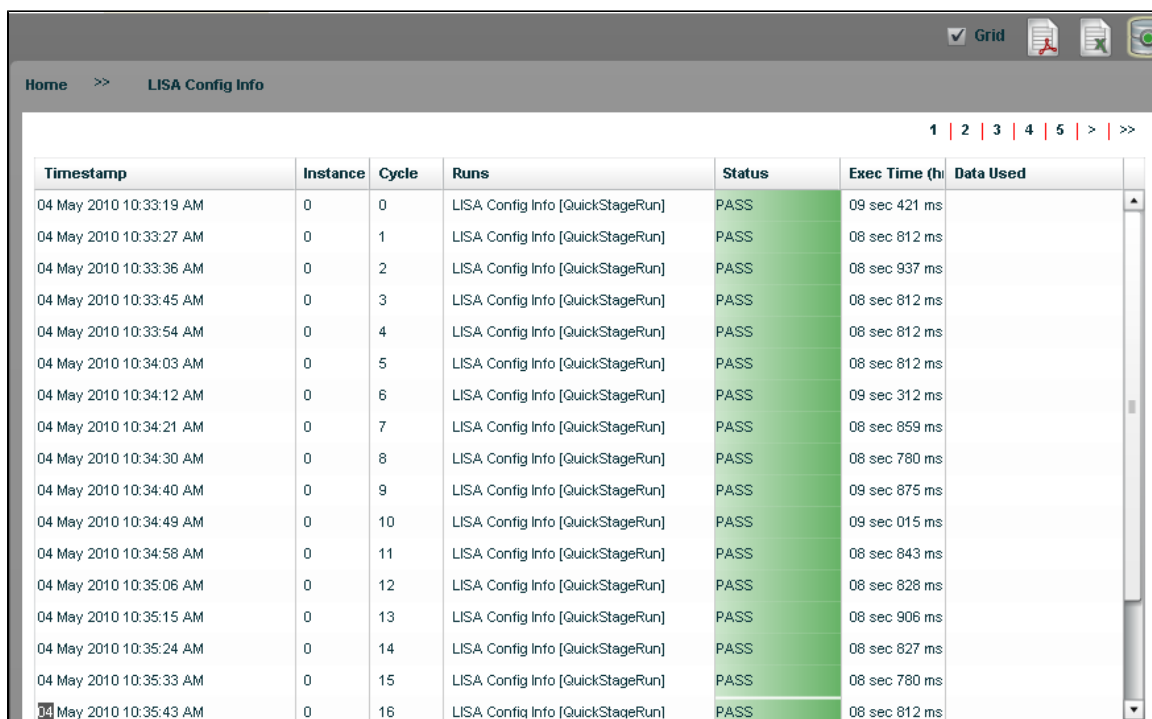


Timestamp	Suite Name	Pass	Fail	Could not f	Errors	Warnings	Total Execution Time (hr mn)
04 May 2010 10:33:09 AM	LISA Config Info	226	0	0	0	0	33 min 37 sec 402 ms
04 May 2010 10:33:57 AM	multi-tier-combo	4	0	0	0	0	03 min 58 sec 266 ms
04 May 2010 11:08:34 AM	web-application	201	0	0	0	2	49 min 28 sec 294 ms
04 May 2010 11:44:58 AM	AllTestsSuite	16	0	0	17	0	03 min 19 sec 847 ms
04 May 2010 11:45:19 AM	New Suite	15	0	1	20	0	09 min 12 sec 824 ms
04 May 2010 11:55:05 AM	webservices	0	0	250	250	0	12 sec 483 ms
04 May 2010 11:59:35 AM	webservices	0	1	0	0	0	03 sec 172 ms
04 May 2010 12:00:17 PM	webservices	0	403	0	0	0	34 sec 671 ms
04 May 2010 12:02:52 PM	webservices	0	271	0	542	0	12 sec 796 ms
04 May 2010 12:04:54 PM	webservices	0	192	0	384	0	23 sec 342 ms
04 May 2010 12:08:12 PM	webservices	0	127	0	127	0	16 sec 697 ms
04 May 2010 12:10:34 PM	webservices	0	236	0	0	236	26 sec 168 ms

As you can see here, the **Export to Excel** icon is disabled.

So we need to view this report in a **Grid view** only, so that it can be exported to excel.

- Click on any of the Tests to get further details:



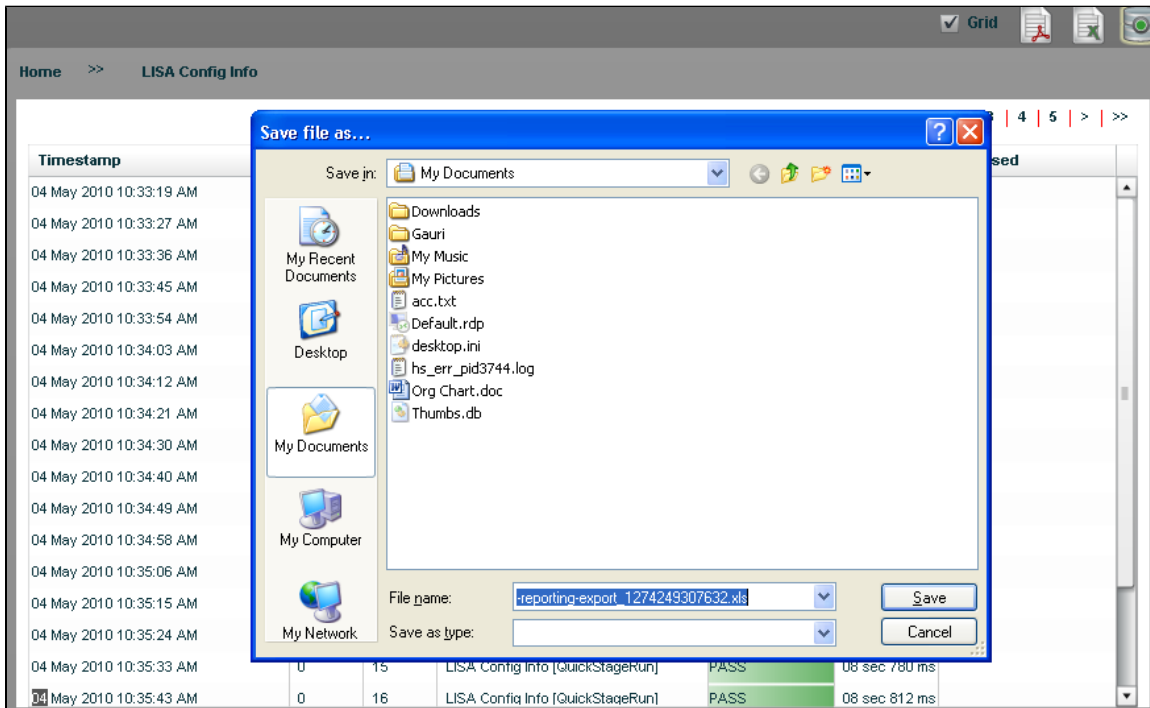
Timestamp	Instance	Cycle	Runs	Status	Exec Time (h)	Data Used
04 May 2010 10:33:19 AM	0	0	LISA Config Info [QuickStageRun]	PASS	09 sec 421 ms	
04 May 2010 10:33:27 AM	0	1	LISA Config Info [QuickStageRun]	PASS	08 sec 812 ms	
04 May 2010 10:33:36 AM	0	2	LISA Config Info [QuickStageRun]	PASS	08 sec 937 ms	
04 May 2010 10:33:45 AM	0	3	LISA Config Info [QuickStageRun]	PASS	08 sec 812 ms	
04 May 2010 10:33:54 AM	0	4	LISA Config Info [QuickStageRun]	PASS	08 sec 812 ms	
04 May 2010 10:34:03 AM	0	5	LISA Config Info [QuickStageRun]	PASS	08 sec 812 ms	
04 May 2010 10:34:12 AM	0	6	LISA Config Info [QuickStageRun]	PASS	09 sec 312 ms	
04 May 2010 10:34:21 AM	0	7	LISA Config Info [QuickStageRun]	PASS	08 sec 859 ms	
04 May 2010 10:34:30 AM	0	8	LISA Config Info [QuickStageRun]	PASS	08 sec 780 ms	
04 May 2010 10:34:40 AM	0	9	LISA Config Info [QuickStageRun]	PASS	09 sec 875 ms	
04 May 2010 10:34:49 AM	0	10	LISA Config Info [QuickStageRun]	PASS	09 sec 015 ms	
04 May 2010 10:34:58 AM	0	11	LISA Config Info [QuickStageRun]	PASS	08 sec 843 ms	
04 May 2010 10:35:06 AM	0	12	LISA Config Info [QuickStageRun]	PASS	08 sec 828 ms	
04 May 2010 10:35:15 AM	0	13	LISA Config Info [QuickStageRun]	PASS	08 sec 906 ms	
04 May 2010 10:35:24 AM	0	14	LISA Config Info [QuickStageRun]	PASS	08 sec 827 ms	
04 May 2010 10:35:33 AM	0	15	LISA Config Info [QuickStageRun]	PASS	08 sec 780 ms	
04 May 2010 10:35:43 AM	0	16	LISA Config Info [QuickStageRun]	PASS	08 sec 812 ms	

Now as you can see, the Export to Excel icon is enabled.



- Click on the **Export to Excel file** icon in the top bar.

It will open the Export to file dialog box, where in you can specify the name of the Excel file as shown below:



The reporting data will be stored in the saved excel file as shown below:

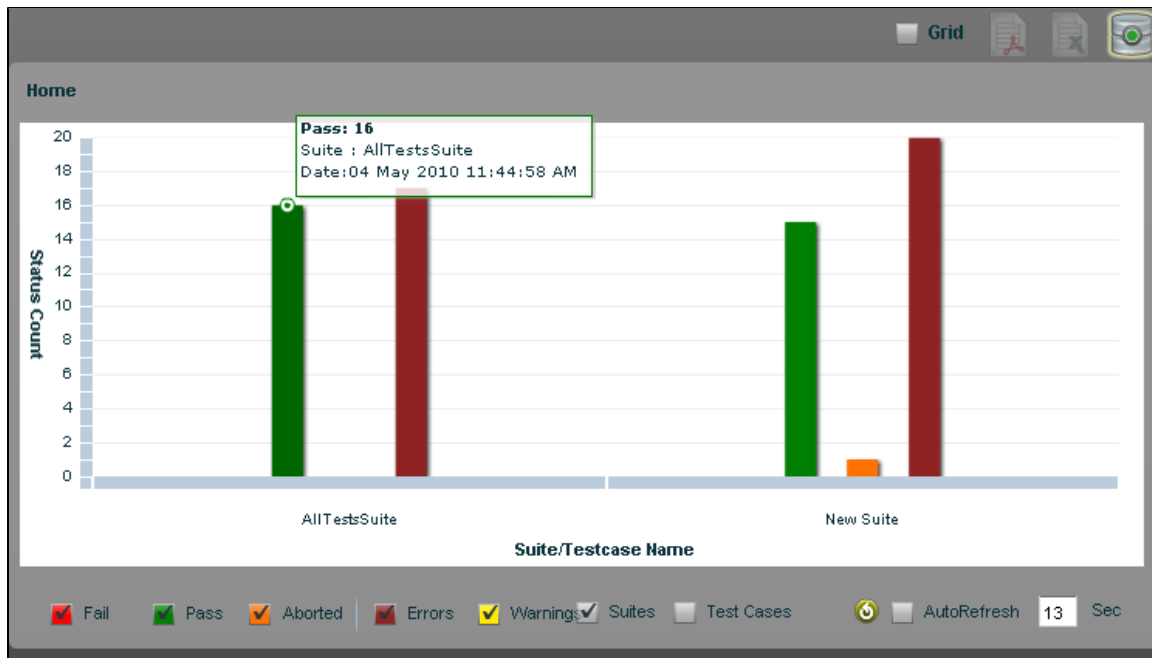
	A	B	C	D	E	F	G
	Timestamp	Instance	Cycle	Runs	Status	Exec Time (hr mm s)	Data Used
1	Tue May 04 10:33:19 ISO	0		LISA Config Info [QuickStageRun]	PASS	09 sec 421 ms	
2	Tue May 04 10:33:27 ISO	1		LISA Config Info [QuickStageRun]	PASS	08 sec 812 ms	
3	Tue May 04 10:33:36 ISO	2		LISA Config Info [QuickStageRun]	PASS	08 sec 937 ms	
4	Tue May 04 10:33:45 ISO	3		LISA Config Info [QuickStageRun]	PASS	08 sec 812 ms	
5	Tue May 04 10:33:54 ISO	4		LISA Config Info [QuickStageRun]	PASS	08 sec 812 ms	
6	Tue May 04 10:34:03 ISO	5		LISA Config Info [QuickStageRun]	PASS	08 sec 812 ms	
7	Tue May 04 10:34:12 ISO	6		LISA Config Info [QuickStageRun]	PASS	09 sec 312 ms	
8	Tue May 04 10:34:21 ISO	7		LISA Config Info [QuickStageRun]	PASS	08 sec 859 ms	
9	Tue May 04 10:34:30 ISO	8		LISA Config Info [QuickStageRun]	PASS	08 sec 780 ms	
10	Tue May 04 10:34:40 ISO	9		LISA Config Info [QuickStageRun]	PASS	09 sec 875 ms	
11	Tue May 04 10:34:49 ISO	10		LISA Config Info [QuickStageRun]	PASS	09 sec 015 ms	
12	Tue May 04 10:34:58 ISO	11		LISA Config Info [QuickStageRun]	PASS	08 sec 843 ms	
13	Tue May 04 10:35:06 ISO	12		LISA Config Info [QuickStageRun]	PASS	08 sec 828 ms	
14	Tue May 04 10:35:15 ISO	13		LISA Config Info [QuickStageRun]	PASS	08 sec 906 ms	
15	Tue May 04 10:35:24 ISO	14		LISA Config Info [QuickStageRun]	PASS	08 sec 827 ms	
16	Tue May 04 10:35:33 ISO	15		LISA Config Info [QuickStageRun]	PASS	08 sec 780 ms	
17	Tue May 04 10:35:43 ISO	16		LISA Config Info [QuickStageRun]	PASS	08 sec 812 ms	
18	Tue May 04 10:35:52 ISO	17		LISA Config Info [QuickStageRun]	PASS	09 sec 358 ms	

24.1.6.2 Exporting Reports to PDF

24.1.6.2 Exporting Reports to PDF

You can export the report data to PDF.

Open the Graphical view of the Report as shown below:



We have selected to view the "AllTestsSuite" as shown above. The Export to PDF/Excel icons are disabled currently in this view.

Clicking on this Test Suite bar, you can go to the next level of detail. Now the Export to PDF/Excel icons are enabled.

To export to PDF,

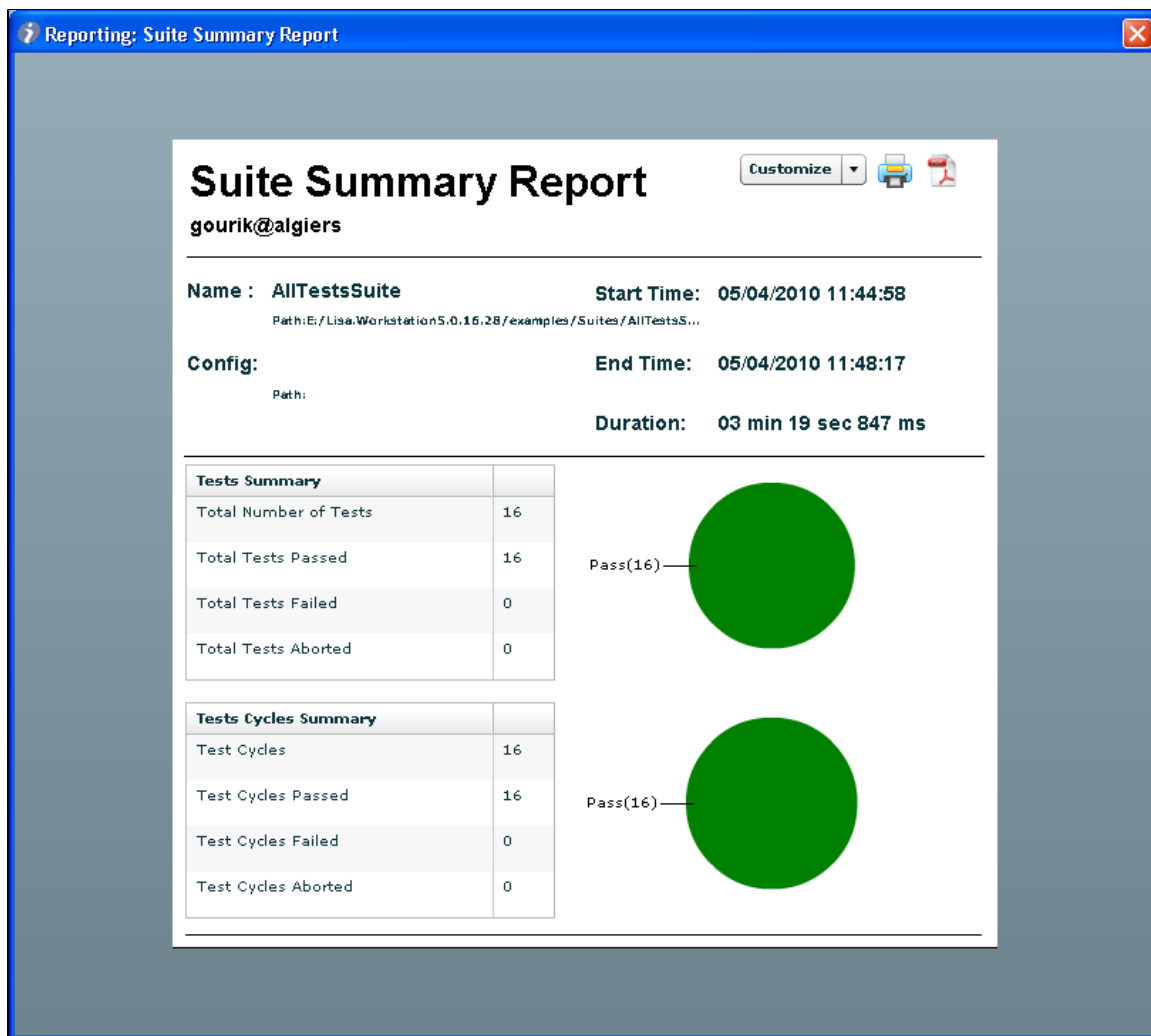
- Click on the **Export to PDF**



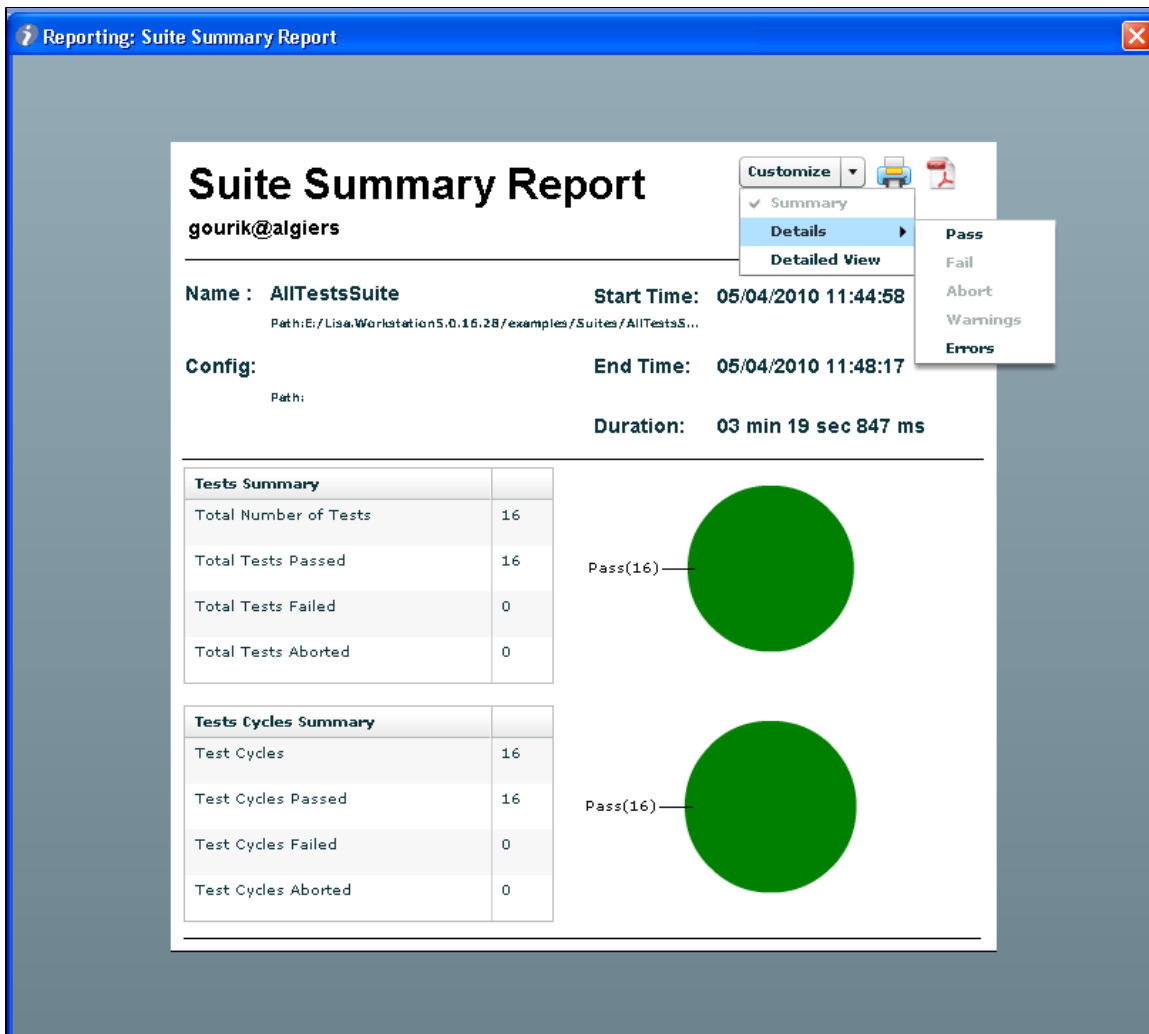
icon.

The report data is seen with the following format: As you can see below, it displays information regarding the Suite, its start time and end time and the configuration if set.

It also lists the test case summary: number of tests passed and failed - in a tabular as well as graphical format.



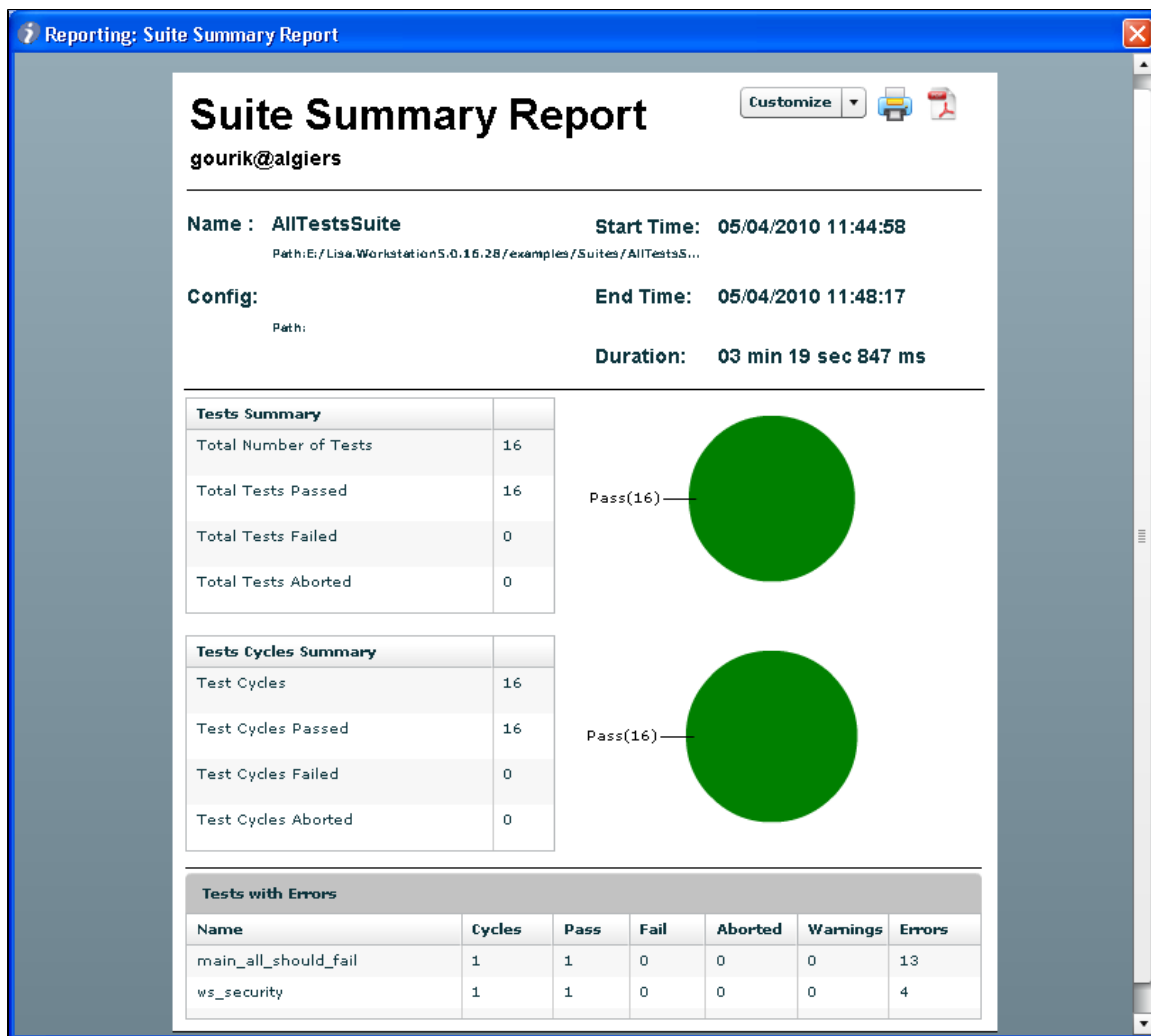
You can also choose to Customize the report further, by clicking on the **Customize** drop down on the top right corner:



You can click on any Criteria (Pass/Fail/Abort/Warnings/Errors).

Here as you can see above "Fail, Abort, Warnings" have been disabled, as there are no cases with this criteria within the suite.

So Click on **Errors** to see the report of Error giving tests. It will show you two tests which have generated Errors as shown below:



You can also select the "Detailed view" of the report, so that all the level of details will be captured as shown below:

Reporting: Suite Summary Report						
Tests with Errors						
Name	Cycles	Pass	Fail	Aborted	Warnings	Errors
main_all_should_fail	1	1	0	0	0	13
ws_security	1	1	0	0	0	4

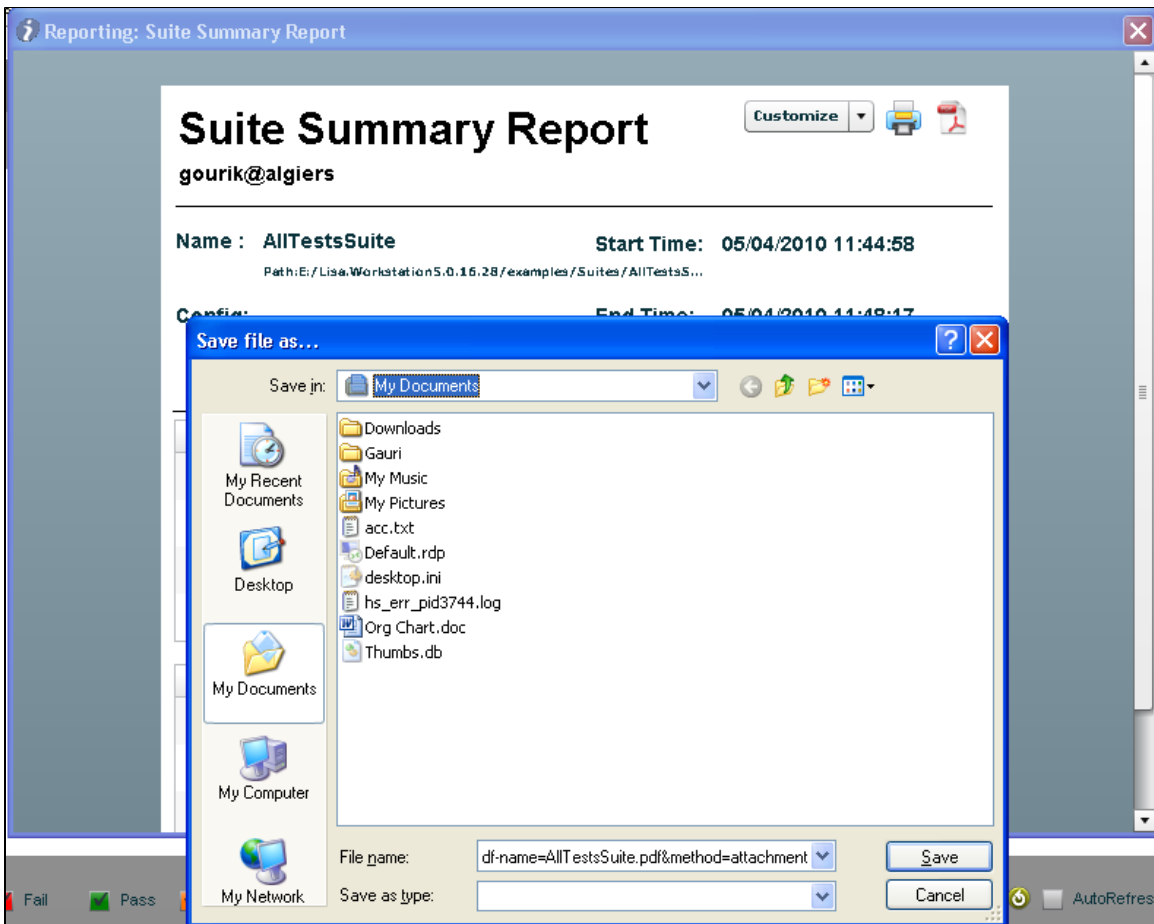
Test Case	TotalTime(ms)	Cycles	Instances	Avg Time	Result
E:\Lisa.Workstation5.0.16.28\exa	1000	1	1	1000	PASS
E:\Lisa.Workstation5.0.16.28\exa	2475	1	1	2475	PASS
E:\Lisa.Workstation5.0.16.28\exa	3147	1	1	3147	PASS
E:\Lisa.Workstation5.0.16.28\exa	69168	1	1	69168	PASS
E:\Lisa.Workstation5.0.16.28\exa	4703	1	1	4703	PASS
E:\Lisa.Workstation5.0.16.28\exa	2203	1	1	2203	PASS
E:\Lisa.Workstation5.0.16.28\exa	6015	1	1	6015	PASS
E:\Lisa.Workstation5.0.16.28\exa	1781	1	1	1781	PASS
E:\Lisa.Workstation5.0.16.28\exa	16656	1	1	16656	PASS
E:\Lisa.Workstation5.0.16.28\exa	4313	1	1	4313	PASS
E:\Lisa.Workstation5.0.16.28\exa	20233	1	1	20233	PASS
E:\Lisa.Workstation5.0.16.28\exa	781	1	1	781	PASS
E:\Lisa.Workstation5.0.16.28\exa	6201	1	1	6201	PASS
E:\Lisa.Workstation5.0.16.28\exa	2844	1	1	2844	PASS
E:\Lisa.Workstation5.0.16.28\exa	828	1	1	828	PASS
E:\Lisa.Workstation5.0.16.28\exa	16967	1	1	16967	PASS

Test Case	VUsers	Duration	Think Time	Transaction	Duration
E:\Lisa.Workstation5.0.16.28\exa	1	-1	0	0	0
E:\Lisa.Workstation5.0.16.28\exa	1	-1	0	0	0
E:\Lisa.Workstation5.0.16.28\exa	1	-1	0	0	0
E:\Lisa.Workstation5.0.16.28\exa	1	-1	0	0	0
E:\Lisa.Workstation5.0.16.28\exa	1	-1	0	0	0
E:\Lisa.Workstation5.0.16.28\exa	1	-1	0	0	0
E:\Lisa.Workstation5.0.16.28\exa	1	-1	0	0	0

You can either **Print** the report or convert to **PDF** by clicking the respective icons as shown below:

- Click on the  to print the report.

- Click on the  to convert the same to PDF.



- Enter the **Name** of the PDF file to be saved as
- Click **Save** to save the PDF file.

24.1.7 Auto Expire Reports

24.1.7 Auto Expire Reports Timer

The Registry will automatically remove/delete reports older than a **set time** from the reports database.

This is done with the help of a report expire timer which is set in the local.properties file.

To change the cut off date,

- Open the **local.properties** file.

Report related data is set here -

```
# In order to check for expired reports
# set autoExpire = true
# set the expiration period -- an integer followed by (m=month,w=week,d=day,h=hour)
# the default expiration period is 30d (30 days)
perfmgr.rvwiz.whatrpt.autoExpire=true
perfmgr.rvwiz.whatrpt.expireTimer=30d
```

Edit the following portion, to include one of the following key/values:

```
perfmgr.rvwiz.whatrpt.autoExpire=true
perfmgr.rvwiz.whatrpt.expireTimer=30d
```

As seen above, by default the reports are set to Auto expire, with the expire time set as 30d (30 days).

You can change/modify this expire timer. The expire Timer can be set to any reasonable unit of time.

For example - 360d, 1m, 5d, 1h etc.

24.2 Legacy Reports 4.0

24.2 Legacy Reports 4.0

LISA 4.x provides an extensive reporting framework that contains **two reporting engines**.

The first generation report generator is **XML** based. The XML report generator requires that you select the reports that you want when the test(s) are being staged. When viewing reports you can only see the reports that were previously generated as XML documents. The XML report generated is as shown at the end of this section.

The second generation report generator stores the raw data collected during a test in a database, giving you more flexibility in report generation at a later date. The **database report generator** is recommended for future report generation, and consequently it is referred to as the **Default Report Generator**.

The XML report generator has been retained, however, for backward compatibility. The advantages of the default report generator will become apparent in the following pages.

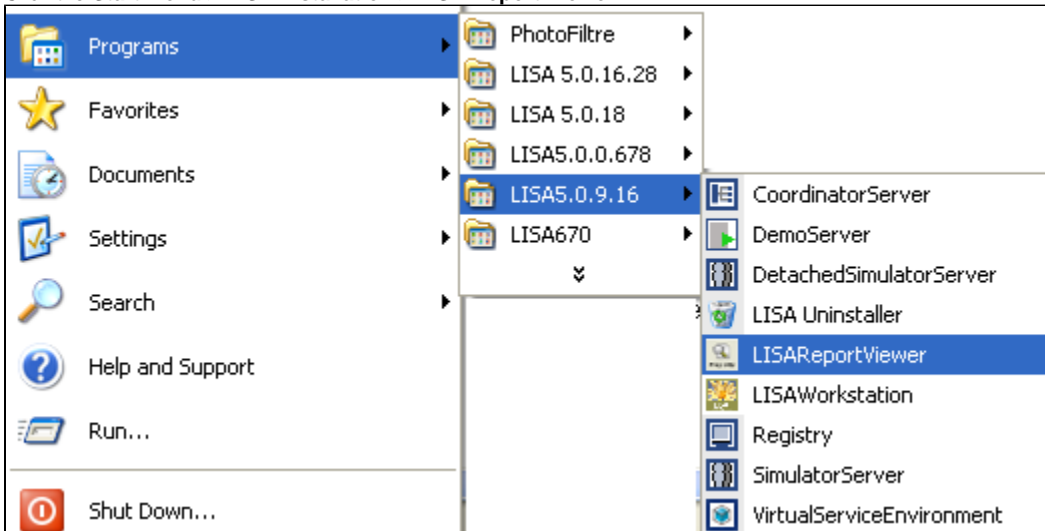
Pre-requisites

1. Coordinator server should be running.
2. Simulator server should be running.

Starting the Report Viewer for 4.x versions

To open the Legacy reports,

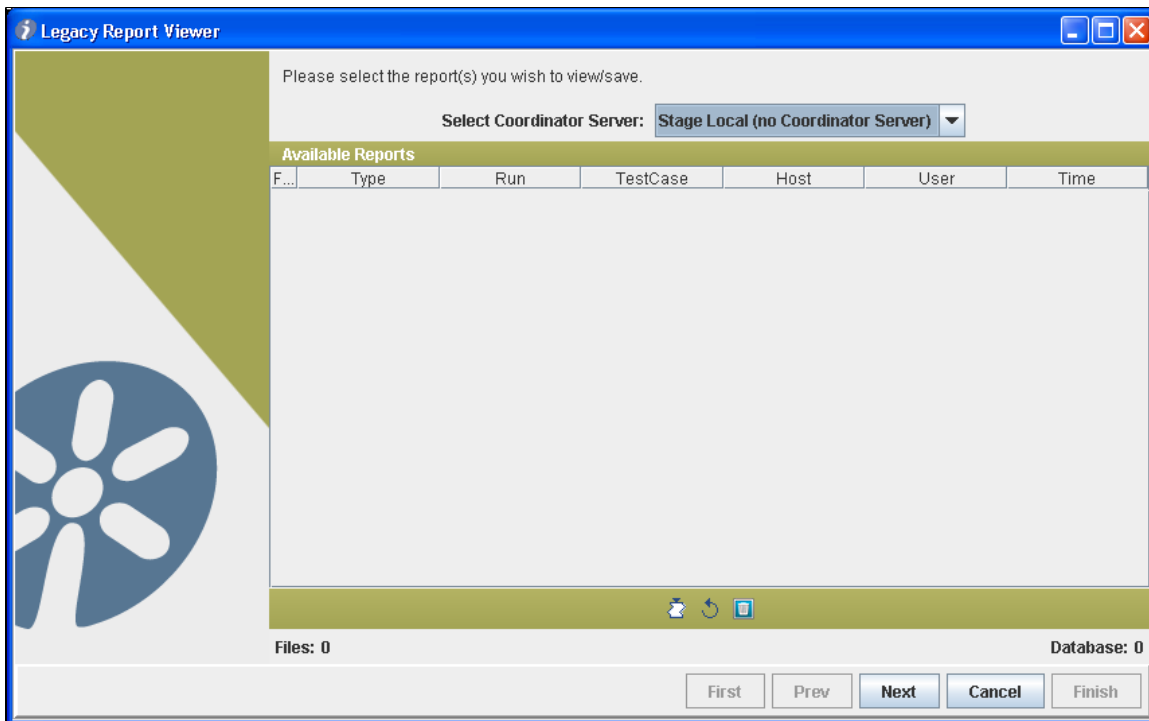
- Click the **Start Menu > LISA Installation > LISA Report Viewer**



This will start the legacy report viewer as a Standalone viewer.

- Enter the License credentials.

The legacy report viewer will open as shown below:



You first need to generate these reports and only then can you view them in the report list.

Following kind of **XML** file is generated under %LISA_HOME%/Reports:

```
<?xml version="1.0" ?>
- <ITKOReport>
  <type>FunctionalSummary</type>
  - <Key>
    <type>multi-tier-combo</type>
    <tc>multi-tier-combo</tc>
    <run>multi-tier-combo [1User0Think_RunContinuously]</run>
    <host>algiers</host>
    <user>gourik</user>
    <time>2010.05.24.12.34.28.180</time>
  </Key>
  - <Functional>
    <TestCase>multi-tier-combo</TestCase>
    <NodeCount>28</NodeCount>
    <Start>1274684668180</Start>
    <Complete>1274684724818</Complete>
    <Registry>local</Registry>
    <Coordinator>local</Coordinator>
    <Simulator />
    <Result>PASS</Result>
  - <NodeList>
    - <Node>
      <Name>Add User</Name>
      <Type>Web Service Execution (Legacy)</Type>
      <ExecTime>3359</ExecTime>
      <ExecCount>15</ExecCount>
      <Bandwidth>0</Bandwidth>
      <Ordinal>0</Ordinal>
      <AssertList />
    - <ItemList>
      <Item>com.itko.lisa.wsgen.EJB3UserControlService.tns.EJB3UserControlBeanServiceLocator</Item>
      <Item>com.itko.lisa.wsgen.EJB3UserControlService.tns.EJB3UserControlBeanServiceLocator</Item>
      <Item>com.itko.lisa.wsgen.EJB3UserControlService.tns.EJB3UserControlBeanServiceLocator</Item>
      <Item>com.itko.lisa.wsgen.EJB3UserControlService.tns.EJB3UserControlBeanServiceLocator</Item>
      <Item>com.itko.lisa.wsgen.EJB3UserControlService.tns.EJB3UserControlBeanServiceLocator</Item>
```

24.2.1 Report Types

24.2.1 Report Types

The Legacy Report Viewer has the following types of standard reports:

Functional Report:

This report shows the Performance data and the request and response for every step invocation.

Performance Report:

This report shows the Total, average, min, max and standard deviation performance data (response times) for the whole test, and each step in the test.

Report Metric Data Info:

This report shows the details of each of the metrics monitored during the test. The items reported are listed by the interval in which they were captured.

Report Metric Data Info (Grouped):

This report shows the details of each of the metrics monitored during the test. The items reported are grouped by metric type.

Suite Report:

This report shows the Test suite summary information.

24.2.1.1 Full Run Report

24.2.1.1 Full Run Report

This report generates XML files into the LISA_HOME/reports directory.

The report creates a xml file with all the possible data that can be captured.

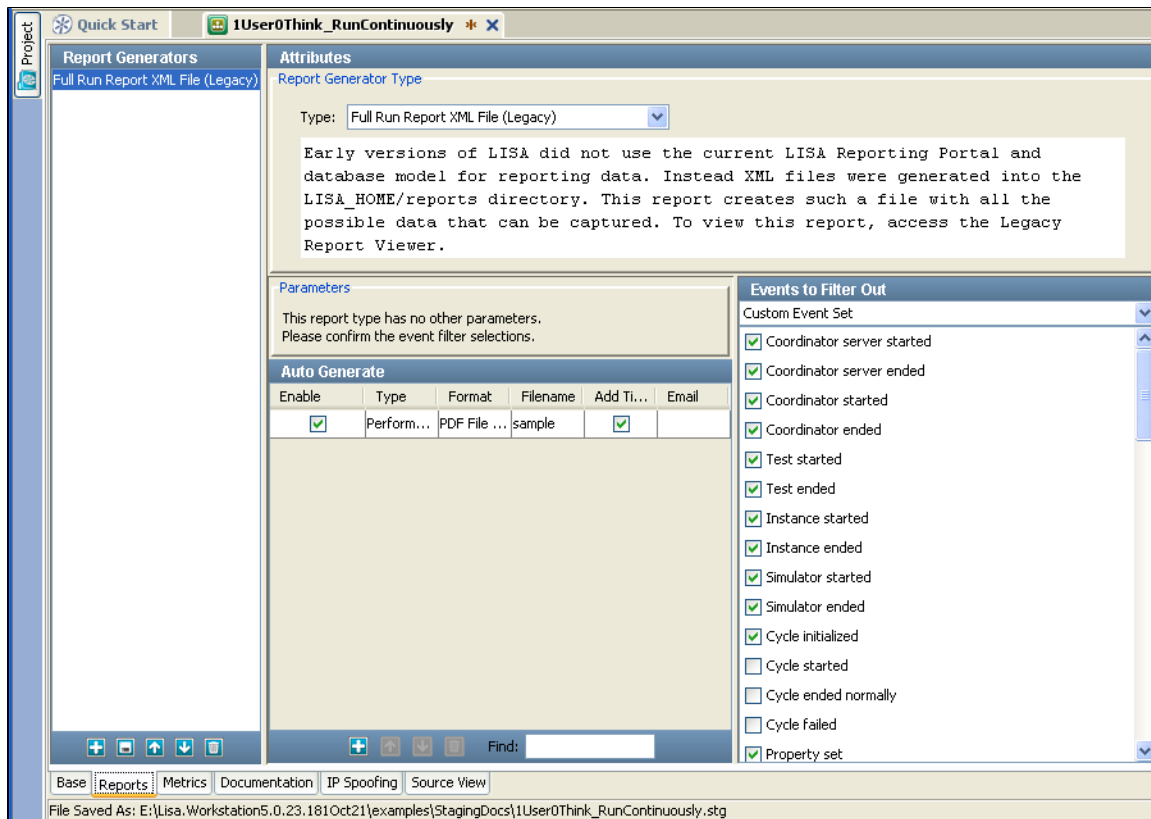
This XML document contains as much information about the test as is known.

Information captured includes non-Filtered test event broadcast during the test, the metric data collected, and performance data.

You can view this information in the following output views:

- **Performance Summary:** Total, average, min, max and standard deviation performance data (response times) for the whole test, and each step in the test
- **Report Metric Data Info:** Details of each of the metrics monitored during the test. The items reported are listed by the interval in which they were captured.
- **Report Metric Data Info (Grouped):** Details of each of the metrics monitored during the test. The items reported are grouped by metric type.
- **Report all events in received order:** Details of each non-Filtered event generated by the test, in the order in which they were received.

Open the Staging document and set the report as shown in the screen below:



This report has no parameters to set.

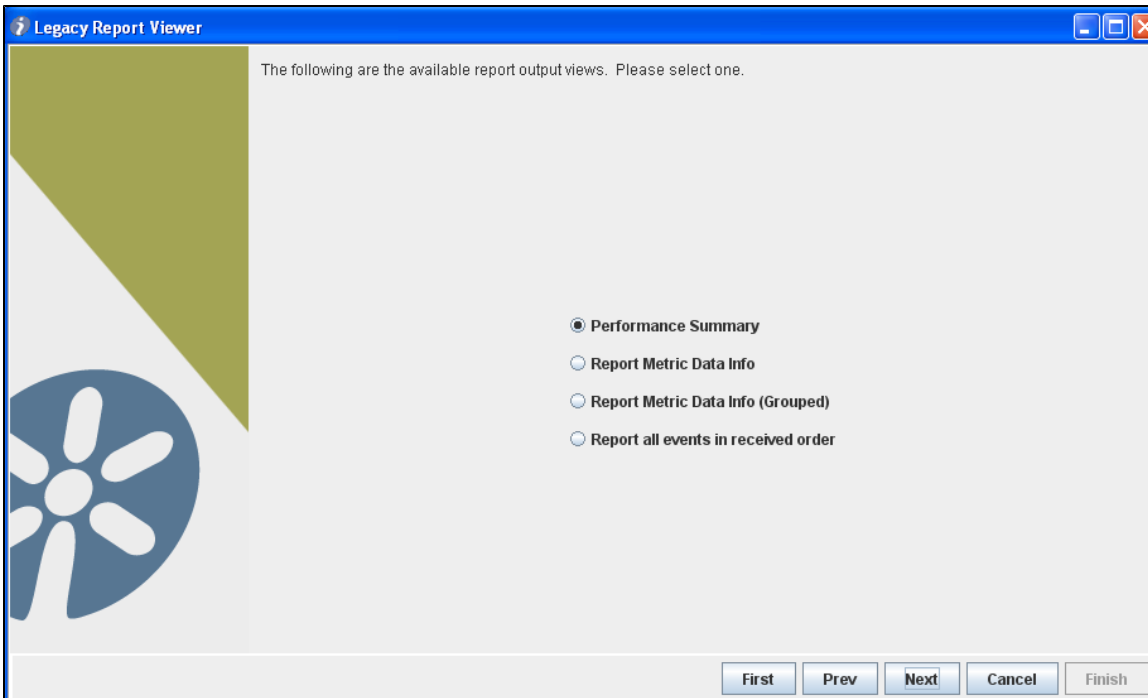
Filter out the events in the "Events to filter Out" section.
All the checked events will be filtered out and will not be seen in the report.

For Auto Generation of Reports, see [Auto Generation of Reports](#).

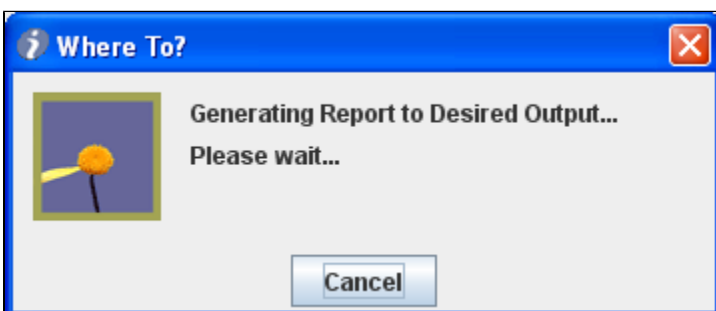
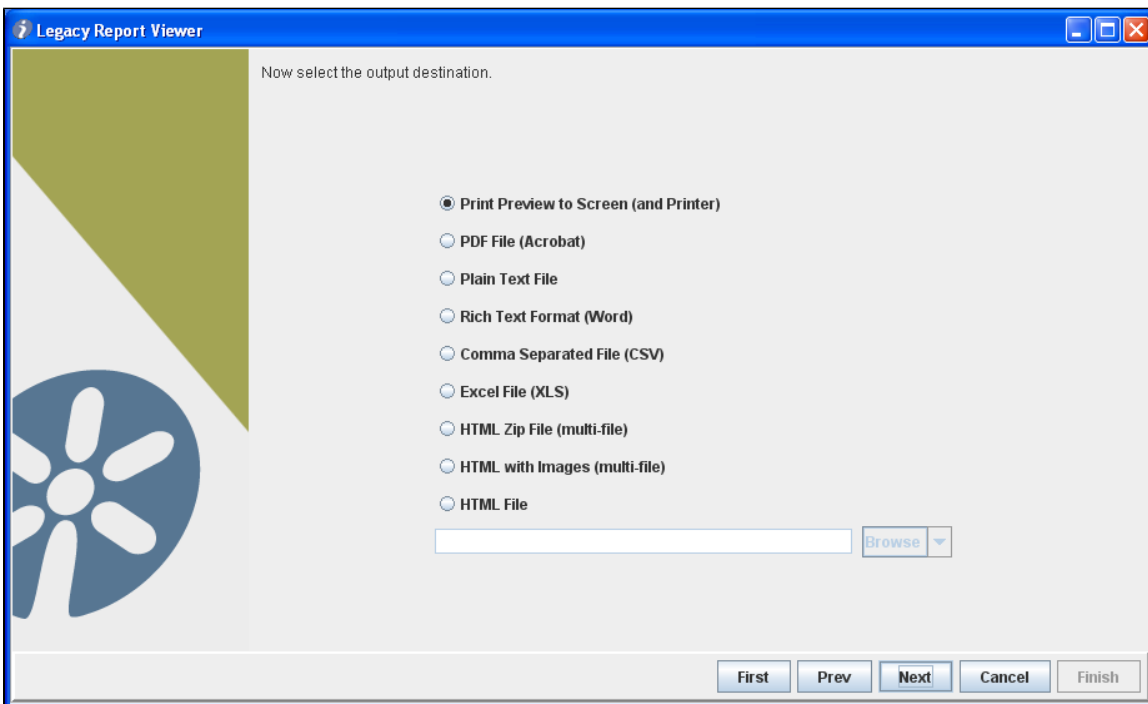
Open the Legacy report viewer from Start menu > LISA Installation> Legacy Report Viewer, to see the reports as follows:

This is the first screen of the legacy report viewer.

Here we choose the performance summary of the report.



We choose the **Print preview to screen** option to display the output destination of the report:



The generated reports will be seen as follows:

The screenshot shows a web application window titled "Metric Data Info - Print Preview". The window has a blue header bar with the title and standard window controls (minimize, maximize, close). Below the header is a navigation bar with links for "File", "Navigation", "Zoom", and "Help". The main content area displays a "Performance Summary" table. The table has seven columns: "Name", "Count", "Total (ms)", "Avg (ms)", "Min (ms)", "Max (ms)", and "StdDev (ms)". The rows list various system operations and their performance metrics. At the bottom of the window, there is a summary section showing "Start Time", "Finish Time", "Elapsed Time", "Tests Failed", and "EVENT_RESPTIME count".

Name	Count	Total (ms)	Avg (ms)	Min (ms)	Max (ms)	StdDev (ms)
All Test Runs	16	137,032	8,564.5	0	107,891	26,520.6
Add User	15	15,577	1,038.5	31	14,531	3,732.8
Get User	15	2,047	136.5	16	1,453	364.7
Verify User Added	15	473	31.5	1	312	77.9
Deposit Money	15	12,704	846.9	78	10,360	2,632.9
Login	15	28,532	1,902.1	47	26,969	6,934.8
Account Activity	15	42,405	2,827.0	93	35,297	9,022.6
Pay Bill Online	15	7,856	523.7	234	1,640	370.2
Logout	15	1,217	81.1	31	610	147.2
Get Transactions	15	686	45.7	15	171	37.4
Delete User	15	8,622	574.8	46	7,312	1,864.4
Verify User Deleted	15	43	2.9	1	15	4.9
All steps	165	120,162	728.3	1	35,297	-

Start Time: Nov 01, 2010 10:30:00 A.
Finish Time: Nov 01, 2010 10:32:18 A.
Elapsed Time: 2m 18.375s
Tests Failed: 0
EVENT_RESPTIME count: 165



100 %

**Metric Data Information**

Interval	Name	Count	Avg	Min	Max	Sum	First	Last
0	Event: Abort/*	10	0	0	0	0	0	0
1	Event: Abort/*	10	0	0	0	0	0	0
2	Event: Abort/*	10	0	0	0	0	0	0
3	Event: Abort/*	10	0	0	0	0	0	0
4	Event: Abort/*	10	0	0	0	0	0	0
5	Event: Abort/*	10	0	0	0	0	0	0
6	Event: Abort/*	10	0	0	0	0	0	0
7	Event: Abort/*	10	0	0	0	0	0	0
8	Event: Abort/*	10	0	0	0	0	0	0
9	Event: Abort/*	10	0	0	0	0	0	0
10	Event: Abort/*	10	0	0	0	0	0	0
11	Event: Abort/*	10	0	0	0	0	0	0
12	Event: Abort/*	10	0	0	0	0	0	0
13	Event: Abort/*	7	0	0	0	0	0	0

Interval	Name	Count	Avg	Min	Max	Sum	First	Last
0	Event: Cycle failed/*	10	0	0	0	0	0	0
1	Event: Cycle failed/*	10	0	0	0	0	0	0
2	Event: Cycle failed/*	10	0	0	0	0	0	0
3	Event: Cycle failed/*	10	0	0	0	0	0	0
4	Event: Cycle failed/*	10	0	0	0	0	0	0
5	Event: Cycle failed/*	10	0	0	0	0	0	0
6	Event: Cycle failed/*	10	0	0	0	0	0	0
7	Event: Cycle failed/*	10	0	0	0	0	0	0
8	Event: Cycle failed/*	10	0	0	0	0	0	0
9	Event: Cycle failed/*	10	0	0	0	0	0	0
10	Event: Cycle failed/*	10	0	0	0	0	0	0
11	Event: Cycle failed/*	10	0	0	0	0	0	0
12	Event: Cycle failed/*	10	0	0	0	0	0	0
13	Event: Cycle failed/*	7	0	0	0	0	0	0

Metric Data Info - Print Preview									
File Navigation Zoom Help									
<div> 100 % </div>									
Metric Data Information									
Interval	Name	Count	Avg	Min	Max	Sum	First	Last	
0	Event: Cycle failed/*	10	0	0	0	0	0	0	
0	Event: Step error/*	10	0	0	0	0	0	0	
0	Event: Step started/*	10	1	0	1	7	0	1	
0	Event: Abort/*	10	0	0	0	0	0	0	
0	LISA: Instances	10	1	1	1	10	1	1	
0	LISA: Avg Resp Time (ms)	10	0	0	0	0	0	0	
0	LISA: Steps per second	10	0	0	0	0	0	0	
Interval	Name	Count	Avg	Min	Max	Sum	First	Last	
1	Event: Cycle failed/*	10	0	0	0	0	0	0	
1	Event: Step error/*	10	0	0	0	0	0	0	
1	Event: Step started/*	10	1	1	2	11	1	2	
1	Event: Abort/*	10	0	0	0	0	0	0	
1	LISA: Instances	10	1	1	1	10	1	1	
1	LISA: Avg Resp Time (ms)	10	2,906	0	14,531	29,062	0	14,531	
1	LISA: Steps per second	10	0	0	0	0	0	0	
Interval	Name	Count	Avg	Min	Max	Sum	First	Last	
2	Event: Cycle failed/*	10	0	0	0	0	0	0	
2	Event: Step error/*	10	0	0	0	0	0	0	
2	Event: Step started/*	10	4	2	4	36	2	4	
2	Event: Abort/*	10	0	0	0	0	0	0	
2	LISA: Instances	10	1	1	1	10	1	1	
2	LISA: Avg Resp Time (ms)	10	6,598	5,432	14,531	65,979	14,531	5,432	
2	LISA: Steps per second	10	0	0	0	1	0	0	
Interval	Name	Count	Avg	Min	Max	Sum	First	Last	
3	Event: Cycle failed/*	10	0	0	0	0	0	0	
3	Event: Step error/*	10	0	0	0	0	0	0	
3	Event: Step started/*	10	5	4	5	46	4	5	
<div> 11/01/2010 <div>Lisa © iTKO, 2009</div> <div>Page 1 of 5</div> </div>									

24.2.1.2 Functional Report

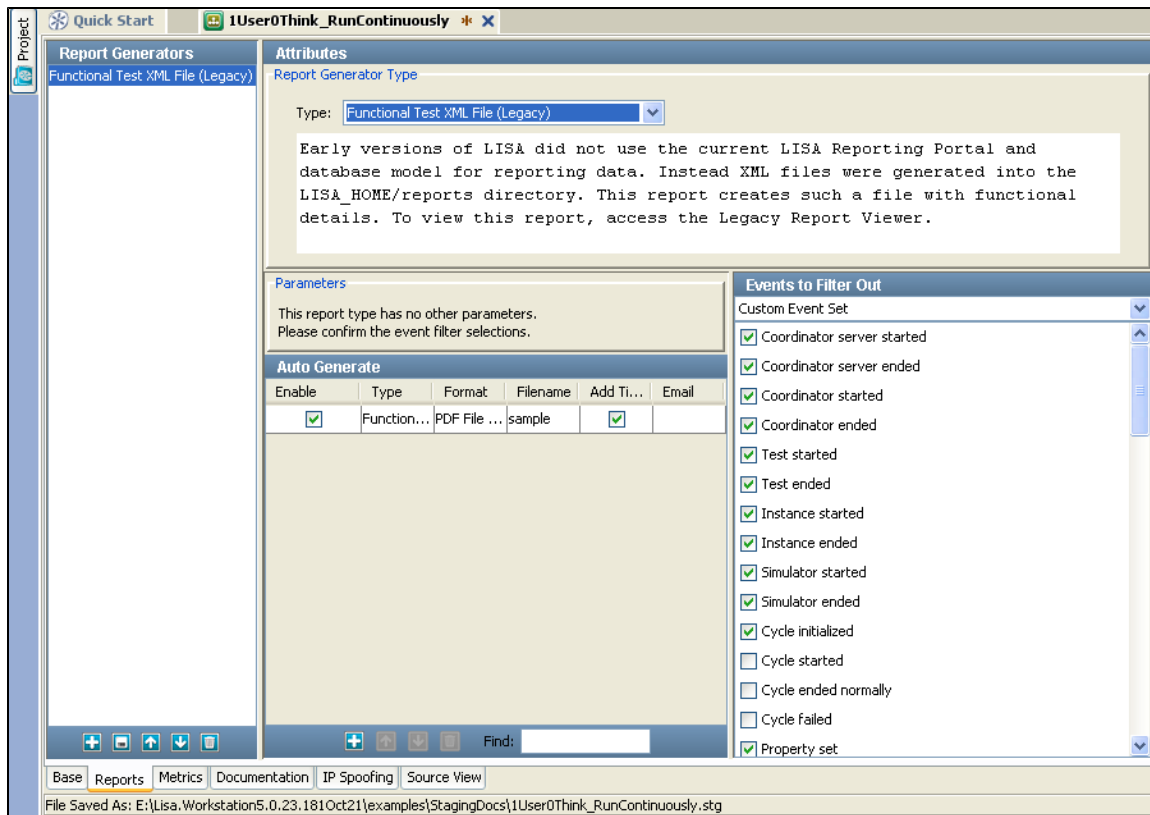
24.2.1.2 Functional Report

This report generates XML files into the LISA_HOME/reports directory.

The report creates a XML file with all the **Functional** details.

The XML document contains, the total, average, min, max and standard deviation performance data (response times) for the whole test, and each step in the test.

To view this report, access the Legacy Report Viewer.

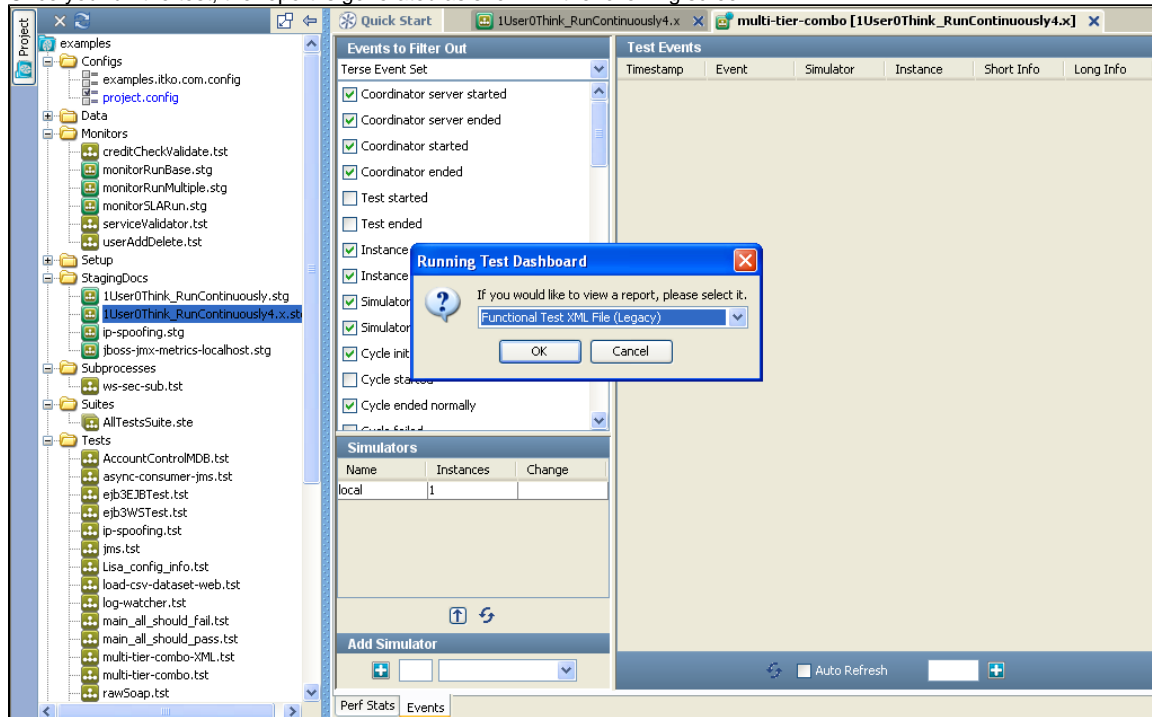


This report has no parameters to set.

Filter out the events in the "Events to filter Out" section.
All the checked events will be filtered out and will not be seen in the report.

For Auto Generation of Reports, see Auto Generation of Reports

Once you run the test, the report is generated as shown in the following screen:



- Go to the **Reports** folder in the **LISA installation directory**.

As you can see below, this "Reports" folder contains all the generated reports.

Folders	Name	Size	Type	Date Modified
Local Disk (C:)	img		File Folder	10/25/2010 10:05 AM
5.0.22.138_GA	lisa-reports.db		File Folder	11/1/2010 11:28 AM
5.0.23.145 POC	Derby-create.ddl	7 KB	DDL File	9/22/2010 12:43 AM
10092010	Derby-drop.ddl	1 KB	DDL File	9/22/2010 12:43 AM
Documents and Settings	lisa-report-db-convert-3.6e	2 KB	RSP File	9/22/2010 12:43 AM
Intel	lisa-report-db-convert-3.6e	33 KB	LISA Test Case	9/22/2010 12:43 AM
Jagjit songs	MySQL-create.ddl	7 KB	DDL File	9/22/2010 12:43 AM
Lisa5.0.23.145	MySQL-drop.ddl	1 KB	DDL File	9/22/2010 12:43 AM
.install4j	Oracle10g-create.ddl	8 KB	DDL File	9/22/2010 12:43 AM
addons	Oracle10g-drop.ddl	1 KB	DDL File	9/22/2010 12:43 AM
agent	SQLServer-create.ddl	7 KB	DDL File	9/22/2010 12:43 AM
bin	SQLServer-drop.ddl	1 KB	DDL File	9/22/2010 12:43 AM
database	FunctionalReport\$\$--\$\$multi-t...	17 KB	XML Document	11/1/2010 10:32 AM
defaults	Perfsummary2010.11.01.10....	11 KB	Adobe Acrobat Doc...	11/1/2010 10:32 AM
DemoServer	report	14 KB	Text Document	11/1/2010 10:32 AM
doc	FullReport\$\$--\$\$multi-tier-co...	604 KB	XML Document	11/1/2010 10:32 AM
embedded_server	MetricsReport\$\$--\$\$multi-tier...	31 KB	XML Document	11/1/2010 10:32 AM
examples	ReportStats\$\$--\$\$multi-tier-c...	8 KB	XML Document	11/1/2010 10:32 AM
examples_src				
hotDeploy				
incontainer				
jre				
legacy_reports				
lib				
licenses				
My Tutorials				
reports				
img				
lisa-reports.d				
snmp				
tmp				
umetrics				
webserver				

Open the **Legacy Report viewer** to view these reports.

24.2.1.3 LISA 4.x Report

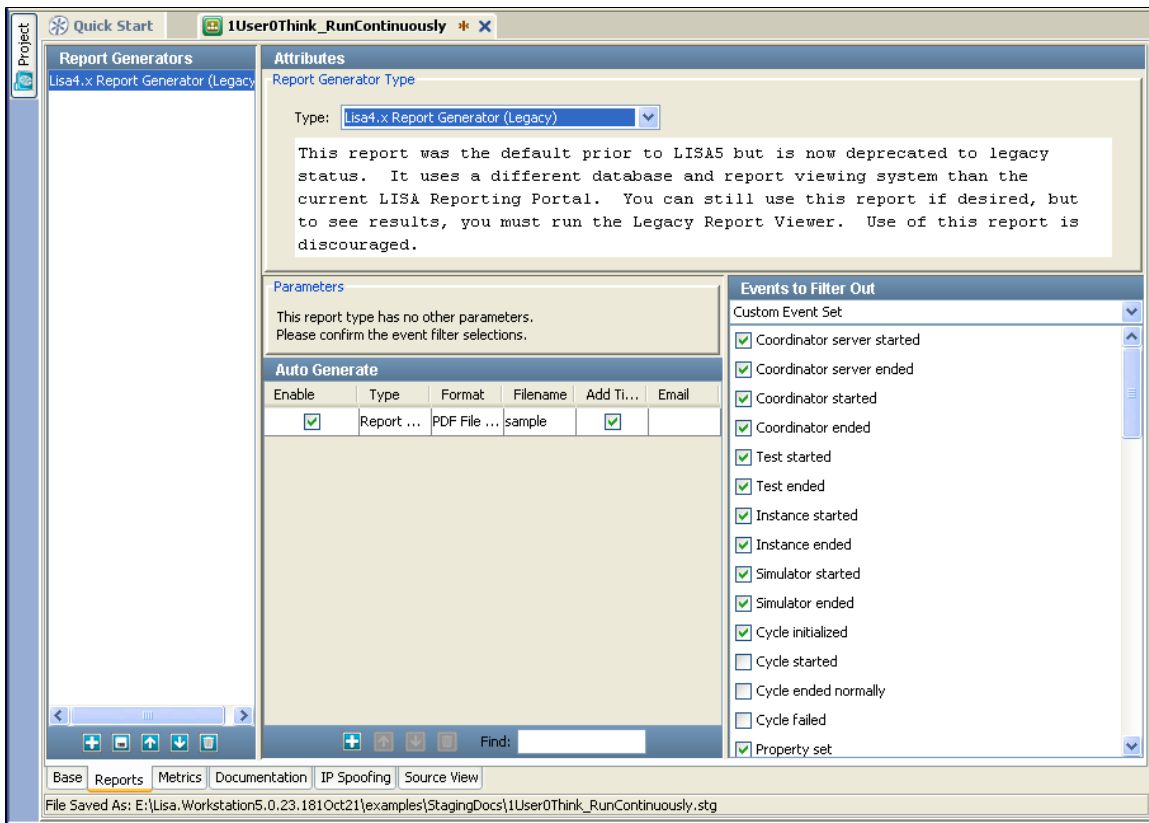
24.2.1.3 LISA 4.x Report

This report was the default prior to LISA5 but is now deprecated to legacy status.

It uses a different database and report viewing system than the current LISA Reporting Portal.

You can still use this report if desired, but to see results, you must run the Legacy Report Viewer.

Note - Use of this report is discouraged.



This report has no parameters to set.

Filter out the events in the "Events to filter Out" section.
All the checked events will be filtered out and will not be seen in the report.

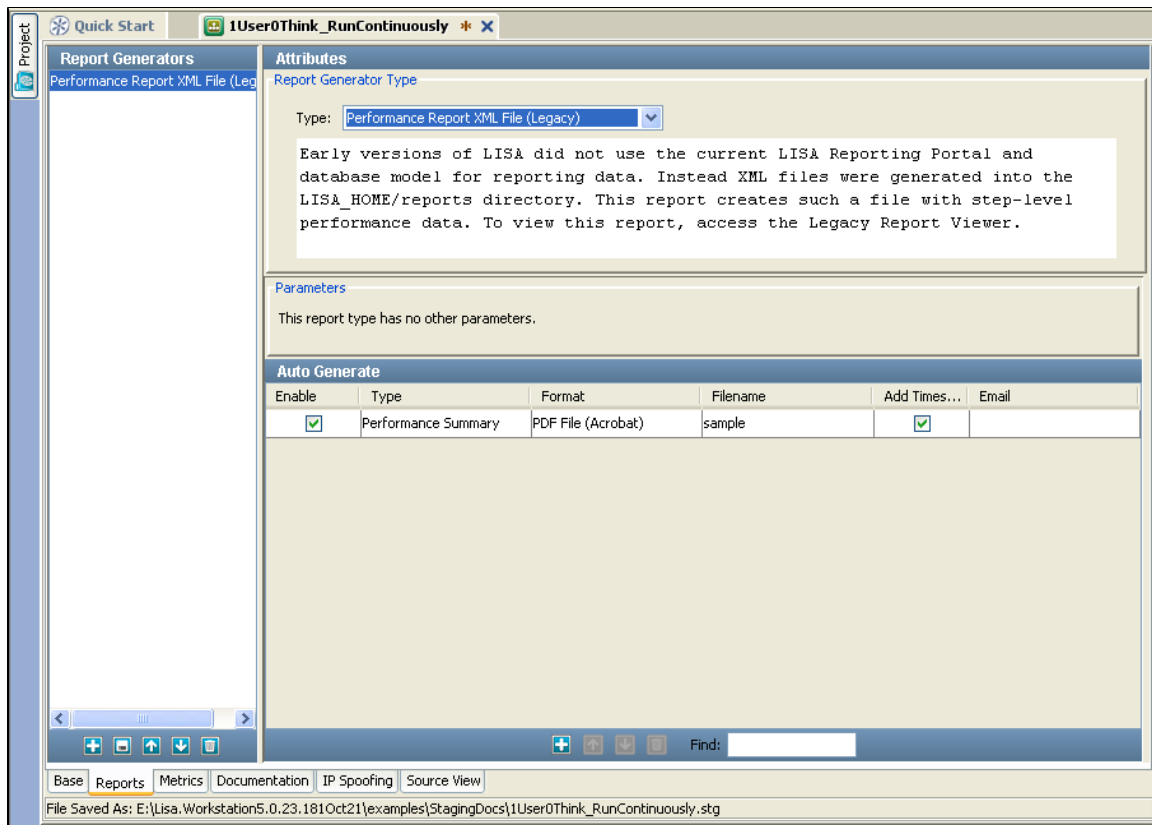
For Automatic report Generation, see [Auto Generation of Reports](#) .

24.2.1.4 Performance Report

24.2.1.4 Performance Report

This report creates a XML file with **step-level performance** data.

To view this report, access the Legacy Report Viewer.



This report has no parameters to set.

Filter out the events in the "Events to filter Out" section.
All the checked events will be filtered out and will not be seen in the report.

For Auto Generation of Reports, see [Auto Generate Reports](#) .

The generated report is seen in the Reports folder as follows:

Perfsummary2010.11.01.10.32.18.640.pdf - Adobe Reader

File Edit View Document Tools Window Help

1 / 8 95.1% Find

Functional Test Report

Test Summary:

Test Case: multi-tier-combo
Node Count: 30
Test Start: Mon Nov 01 10:30:00 GMT+05:30 2010
Test Complete: Mon Nov 01 10:32:18 GMT+05:30 2010
Duration: 2m 18.375s
LISA Registry: local
Coordinator Server: local
Simulator Server(s): -
Test Result: PASS

Test Node: Add User	
Type:	Web Service Execution (Legacy)
Exec Count:	15
Avg Resp Time (ms):	1,038
Avg Bandwidth (KB):	-

Test Items

1. com.itko.lisa.wsgen.EJB3UserControlService.ms.EJB3UserControlBeanServiceLocator
2. com.itko.lisa.wsgen.EJB3UserControlService.ms.EJB3UserControlBeanServiceLocator
3. com.itko.lisa.wsgen.EJB3UserControlService.ms.EJB3UserControlBeanServiceLocator
4. com.itko.lisa.wsgen.EJB3UserControlService.ms.EJB3UserControlBeanServiceLocator
5. com.itko.lisa.wsgen.EJB3UserControlService.ms.EJB3UserControlBeanServiceLocator
6. com.itko.lisa.wsgen.EJB3UserControlService.ms.EJB3UserControlBeanServiceLocator
7. com.itko.lisa.wsgen.EJB3UserControlService.ms.EJB3UserControlBeanServiceLocator
8. com.itko.lisa.wsgen.EJB3UserControlService.ms.EJB3UserControlBeanServiceLocator
9. com.itko.lisa.wsgen.EJB3UserControlService.ms.EJB3UserControlBeanServiceLocator
10. com.itko.lisa.wsgen.EJB3UserControlService.ms.EJB3UserControlBeanServiceLocator
11. com.itko.lisa.wsgen.EJB3UserControlService.ms.EJB3UserControlBeanServiceLocator
12. com.itko.lisa.wsgen.EJB3UserControlService.ms.EJB3UserControlBeanServiceLocator
13. com.itko.lisa.wsgen.EJB3UserControlService.ms.EJB3UserControlBeanServiceLocator
14. com.itko.lisa.wsgen.EJB3UserControlService.ms.EJB3UserControlBeanServiceLocator
15. com.itko.lisa.wsgen.EJB3UserControlService.ms.EJB3UserControlBeanServiceLocator
16. Data set User causing the workflow to execute end

24.2.1.5 Save Metric Report

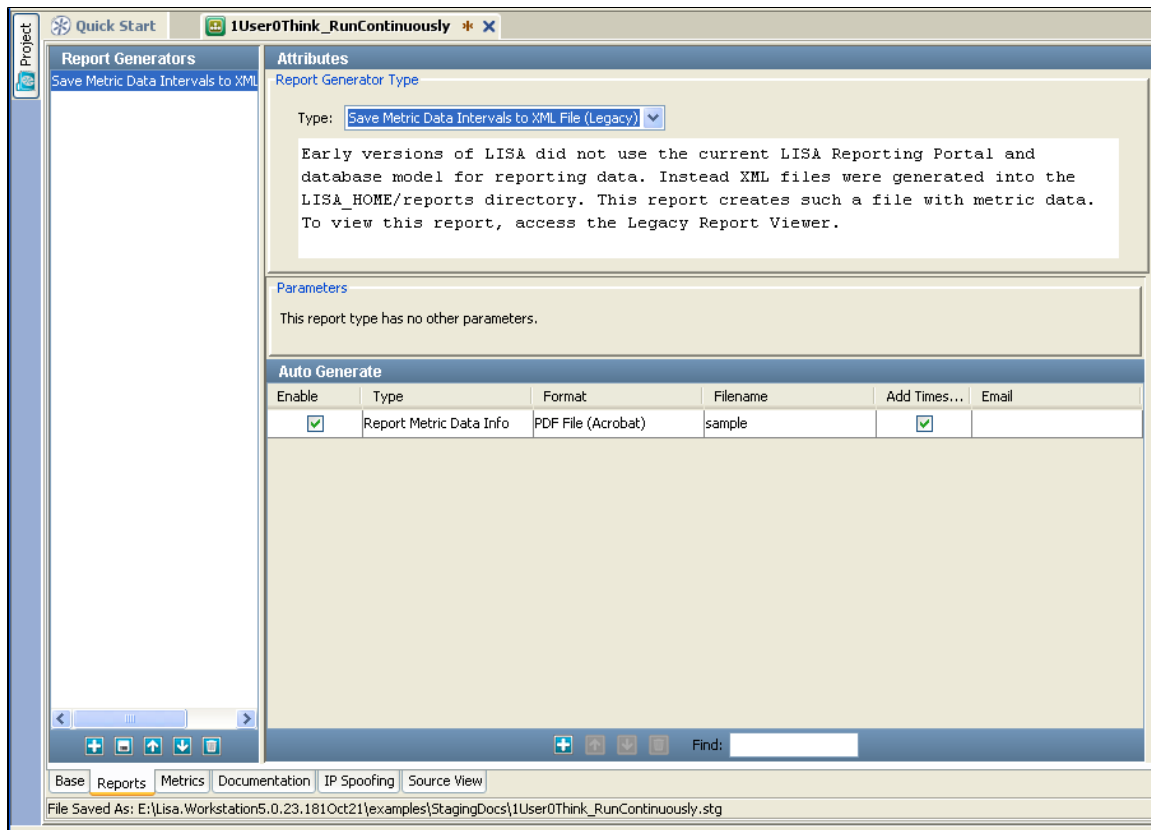
24.2.1.5 Save Metric Report

This report creates a XML file with **Metric** data.

The XML files are generated into the LISA_HOME/reports directory.

The XML document details all metric data collected during the test.

To view this report, access the Legacy Report Viewer.



This report has no parameters to set.

For Automatic report generation, see [Auto Generation of Reports](#) .

24.2.2 Report Formats

24.2.2 Report Formats

The Report Generator stores the event and metric data that was specified in the staging document in a database.

When reports are selected for viewing at a later date, they are generated on the fly in an output format that you can specify.

Report Formats

The following output formats are supported:

- Print Preview to Screen (and Printer)
- PDF File (Acrobat)
- Plain Text File
- Rich Text Format (Word)
- Comma Separated File (CSV)
- Excel File (XLS)
- HTML File

In addition to the standard reports and output formats, the default report generator can be extended to suite your specific needs.

The default database manager (Derby), which is included with LISA can be replaced with a database manger of your choice.

Details on configuring your database are given in [configuring a Different Report Database](#)

Third party report generators can be integrated with the report generator database to allow custom reports to be produced. The existing data model can be extended. Contact support@itko.com for further details.

24.2.3 Adding a Report

24.2.3 Adding a Report

The Default Report Generator stores the event and metric data that was specified in the staging document in a database.

When reports are selected for viewing at a later date, they are generated on the fly in an output format that you can specify.

Adding a Report

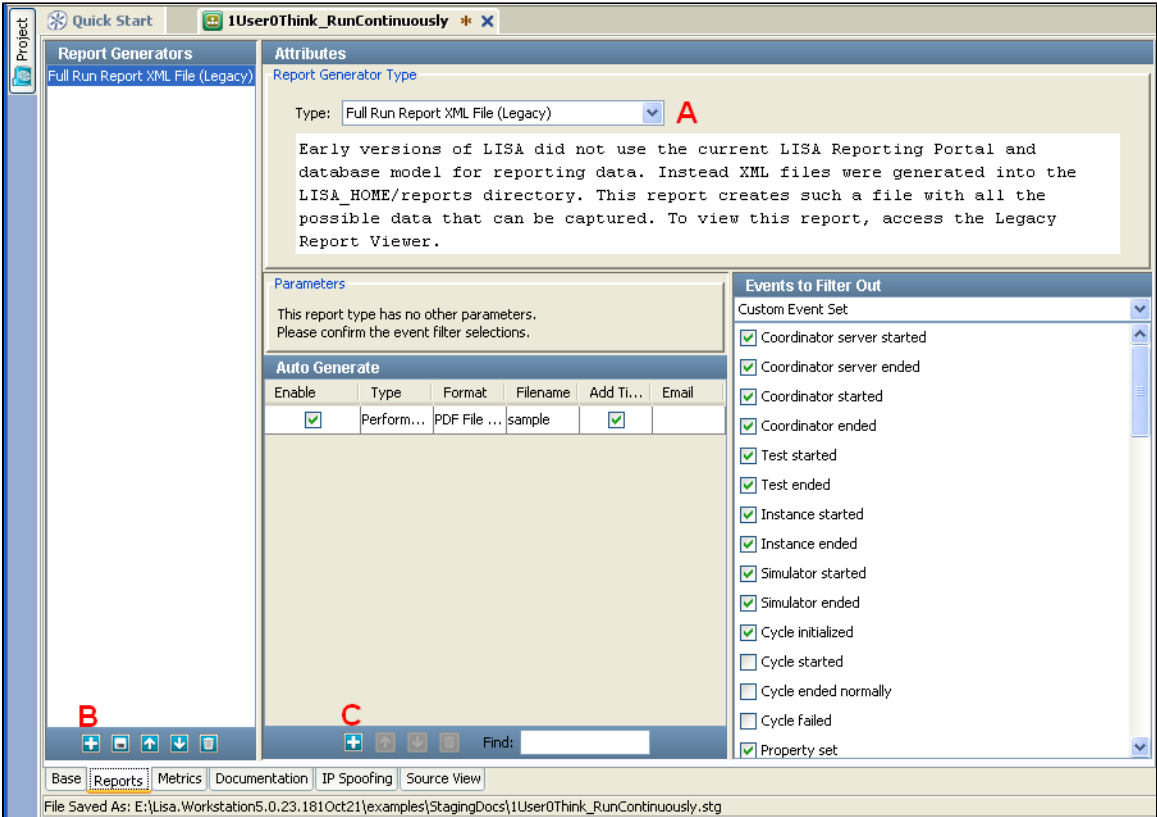
You can add a Report in one of the following documents:

- Quick Tests
- Staging Documents
- Test Suites

For the purpose of illustration, we will look at Adding a Report in the **Reports** tab of a **Staging Document**.

To Add a Report,

- Select the desired Report in the Type drop down in the **Attributes** panel (A)as shown below:

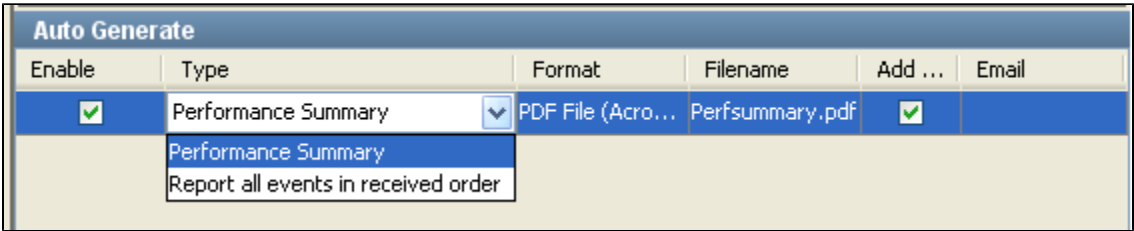


- Click the Add button (B) in the Report Generator panel to add the selected Report to this list.

For example, we have now added "Full Run Report XML" type of Report.

Within this Report, we can add Reports for **Auto Generation** as shown below:

- Click on the Add button (C) in the Auto Generate panel to add multiple Reports from the pull down menu as shown below:



Note- This option of report type will change with selection of different report in the top panel.

As you can see, here we have added the **Performance Summary** report Category.

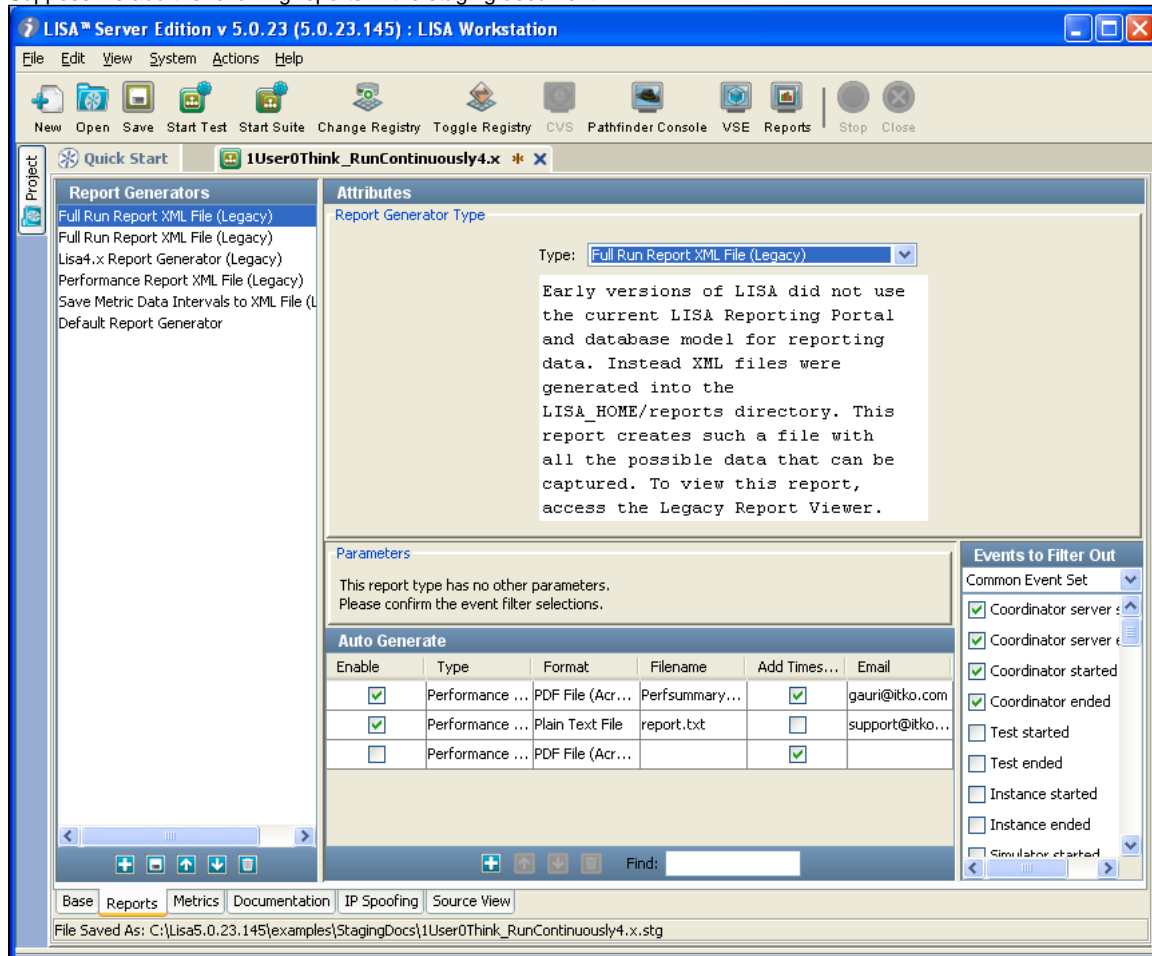
You can specify the File name and Email Address, incase you need a report file to be generated or the Report to be sent to someone by email. To

enable the Email feature in LISA, you need to set up email settings.

Note - The filenames of the auto generated files can now be appended with the timestamp so that the user does not have to give different file names each time. This is a checkbox besides the filename and if unchecked the file name will not be appended with the timestamp.

Multiple Reports Example -

Suppose we add the following reports in the staging document -



All the added reports are seen in the left panel.

- Save the Staging document.
- Stage a Quick Test using this staging document.

Once the run is over, you will see all the generated reports are stored in the reports folder of the LISA Installation home directory as shown below:

Folders	Name	Size	Type	Date Modified
Local Disk (C:)	img		File Folder	10/25/2010 10:05 AM
5.0.22.138_GA	lisa-reports.db		File Folder	11/1/2010 11:28 AM
5.0.23.145 POC	Derby-create.ddl	7 KB	DDL File	9/22/2010 12:43 AM
10092010	Derby-drop.ddl	1 KB	DDL File	9/22/2010 12:43 AM
Documents and Settings	lisa-report-db-convert-3.6e	2 KB	RSP File	9/22/2010 12:43 AM
Intel	lisa-report-db-convert-3.6e	33 KB	LISA Test Case	9/22/2010 12:43 AM
Jagjit songs	MySQL-create.ddl	7 KB	DDL File	9/22/2010 12:43 AM
Lisa5.0.23.145	MySQL-drop.ddl	1 KB	DDL File	9/22/2010 12:43 AM
.install4j	Oracle10g-create.ddl	8 KB	DDL File	9/22/2010 12:43 AM
addons	Oracle10g-drop.ddl	1 KB	DDL File	9/22/2010 12:43 AM
agent	SQLServer-create.ddl	7 KB	DDL File	9/22/2010 12:43 AM
bin	SQLServer-drop.ddl	1 KB	DDL File	9/22/2010 12:43 AM
database	FunctionalReport\$\$\$multi-tier-co...	17 KB	XML Document	11/1/2010 10:32 AM
defaults	Perfsummary2010.11.01.10....	11 KB	Adobe Acrobat Doc...	11/1/2010 10:32 AM
DemoServer	report	14 KB	Text Document	11/1/2010 10:32 AM
doc	FullReport\$\$\$multi-tier-co...	604 KB	XML Document	11/1/2010 10:32 AM
embedded_server	MetricsReport\$\$\$multi-tier-co...	31 KB	XML Document	11/1/2010 10:32 AM
examples	ReportStats\$\$\$multi-tier-co...	8 KB	XML Document	11/1/2010 10:32 AM
examples_src				
hotDeploy				
incontainer				
jre				
legacy_reports				
lib				
licenses				
My Tutorials				
reports				
img				
lisa-reports.d				
snmp				
tmp				
umetrics				
webserver				

To view these reports, open the Legacy Report Viewer.

Deleting a Report

To delete a report,

- Highlight the report in the Report list
- Click the **Delete** icon from the toolbar or Right click the Report to Delete.

24.2.4 Auto Generate Reports

24.2.4 Auto Generate Reports

The Auto generate reports feature, allows you to add a Report automatically and in the specified format.

If you specify an email, it will also mail you the generated report.

Auto Generate					
Enable	Type	Format	Filename	Add Times...	Email
<input checked="" type="checkbox"/>	Performance Summary	PDF File (Acro...	Perfsummary.pdf	<input checked="" type="checkbox"/>	gauri@it...
<input checked="" type="checkbox"/>	Report all events in received order	Plain Text File	report.txt	<input type="checkbox"/>	support...

- Click on the Add button  at the bottom to add a report.

Enable - Check to enable the report generation.

Auto Generate					
Enable	Type	Format	Filename	Add ...	Email
<input checked="" type="checkbox"/>	Performance Summary	PDF File (Acro...)	Perfsummary.pdf	<input checked="" type="checkbox"/>	
	Performance Summary				
	Report all events in received order				

*Type- Choose between the available type of reports as shown above.

Format - Choose the report format, from the available options:

Format
PDF File (...)
PDF File (Acrobat)
Plain Text File
Rich Text Format
Comma Separate
Excel File (XLS)
HTML Zip File (mu)

Filename - Enter the Filename for the report.

Add Times - Check to add time line to the generated reports.

Email - Enter the email address, so that the generated report will be sent to that email.

For more information about email settings, see [Auto Generate Email](#).

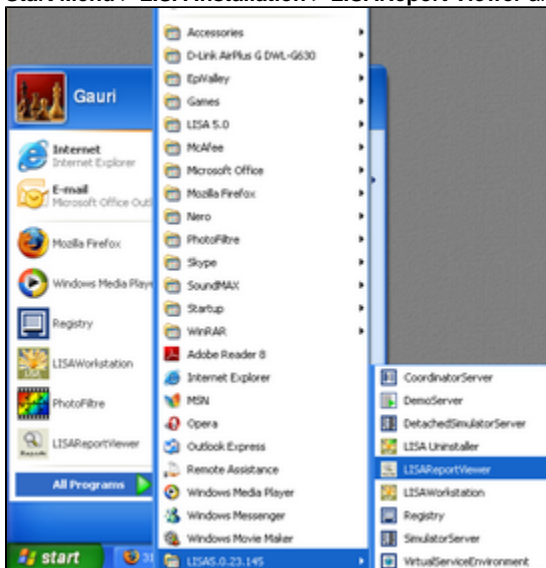
24.2.5 Legacy Report Viewer

24.2.5 Legacy Report Viewer

All the Legacy reports can be viewed in the Legacy Report Viewer.

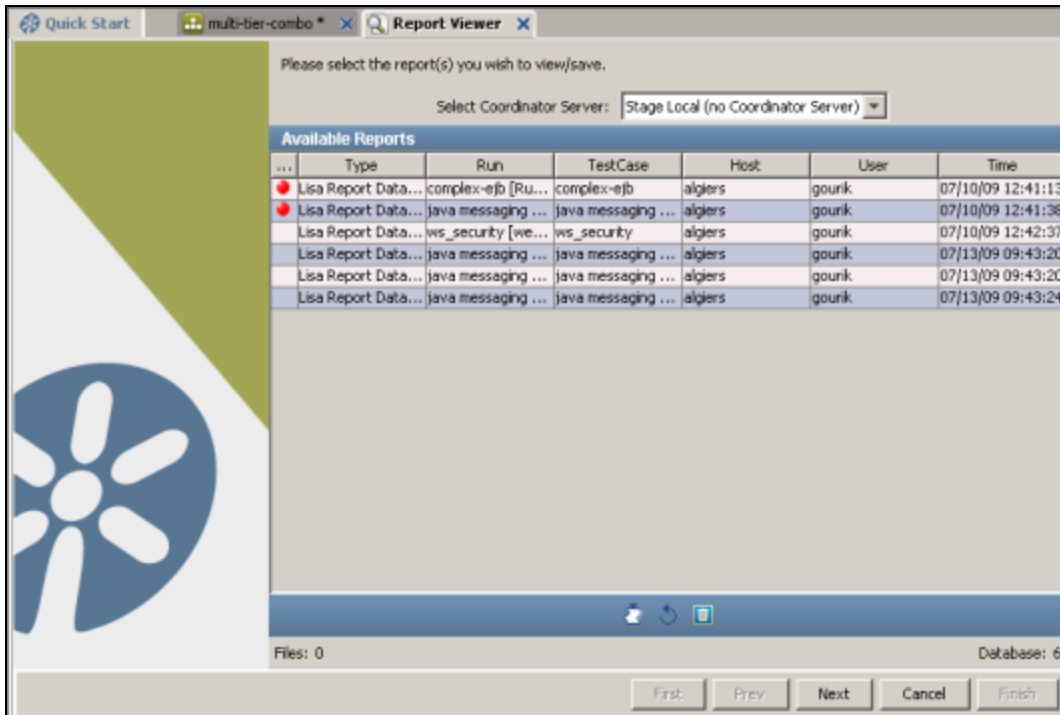
You can open the legacy report viewer from:

Start Menu > LISA Installation > LISAReport Viewer and access the available reports.



For purpose of illustration, we will run the **multi-tier-combo** Test Case and view its reports.

The report viewer will open and LISA will display a list of all your available reports as shown below:



Reports are stored on the Coordinator Server that staged the test.

For LISA Workstation, this will always be Stage Local.

For LISA Server, you will have to select the appropriate Coordinator Server from the pull-down menu at the top of the Available Reports panel.

Each row in the "Available Reports" table represents a report. Each report entry displays:

- **Type**: The type of report. For the default database this will always be Lisa Report Database.
- **Run**: The name of the run. This is entered when the run was staged, or when the test suite run was initiated.
- **Test Case**: The name of the Test Case (or the word 'Suite', if this is a suite run report)
- **Host**: Machine name
- **User**: User name of the user that was logged on when the test was run.
- **Time**: The time the test or suite was run.

Report Toolbar



At the bottom of the Available Reports Screen is a toolbar that can:

Compress, **Refresh** or **Delete** a selected database or report.

To compress a internal database, highlight the report and click **Compress Internal Database** icon to compress the database.

To delete a report, highlight the report and click the **Delete** icon or right click and delete.

To refresh the Reports list, click the **Refresh** icon.

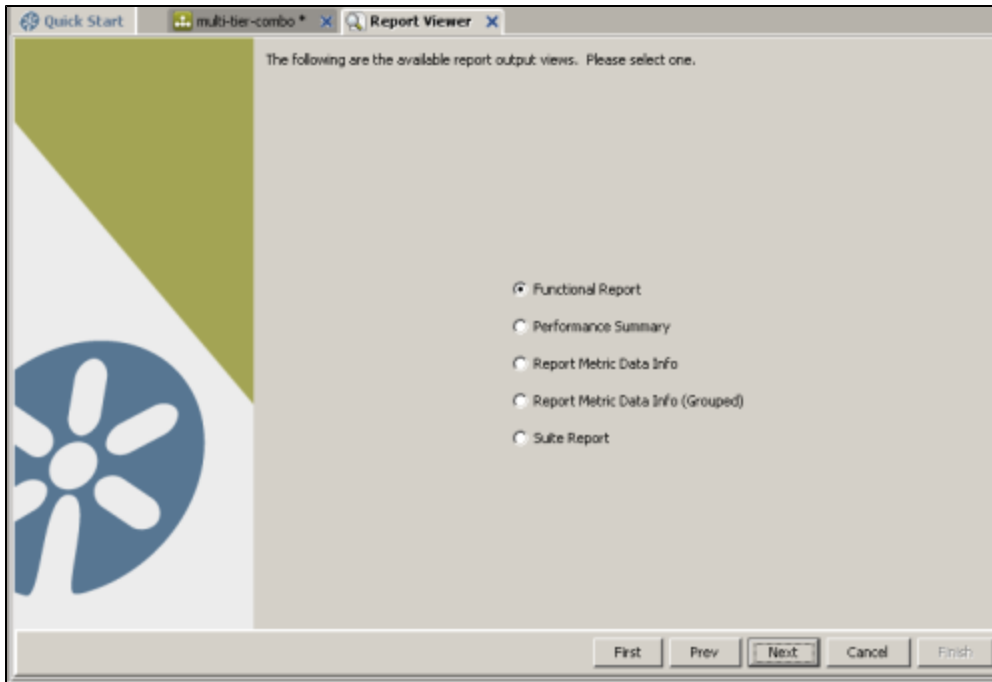
Report Selection

Select the desired report for your Test Case run.

The type will be LISA Report Database. In the table above all the reports are of this type, but in general, the XML reports will also be listed, and XML reports are listed by type, and there are several different types (as you will see later in this chapter).

Click **Next**.

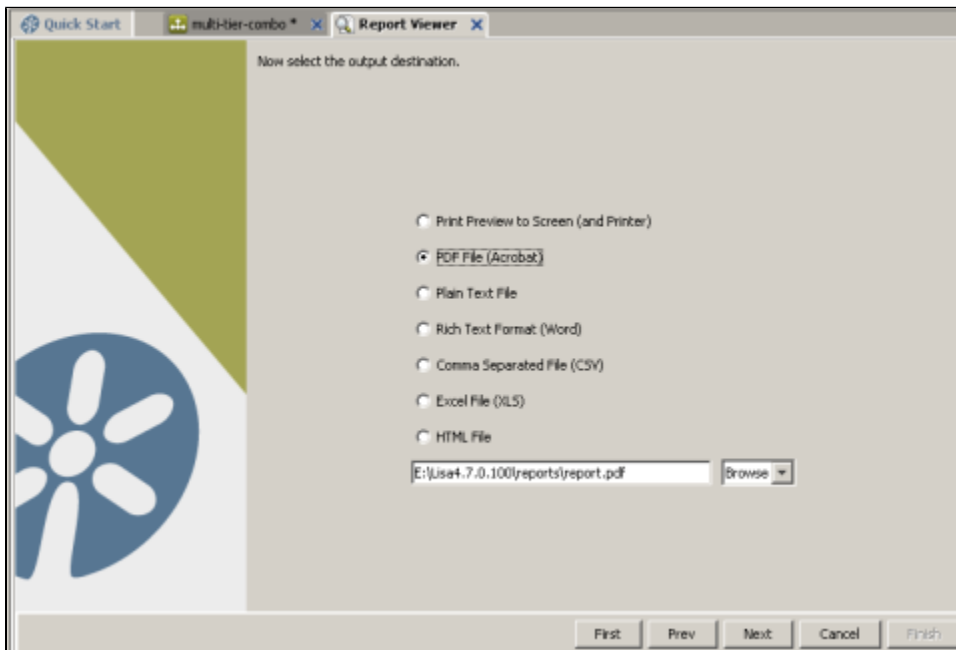
LISA will display a list of the available report types (views).



Select a report type (view) and click **Next**.

Note: Sample Reports are shown in a later section (Sample Reports).

LISA will display a list of output options:



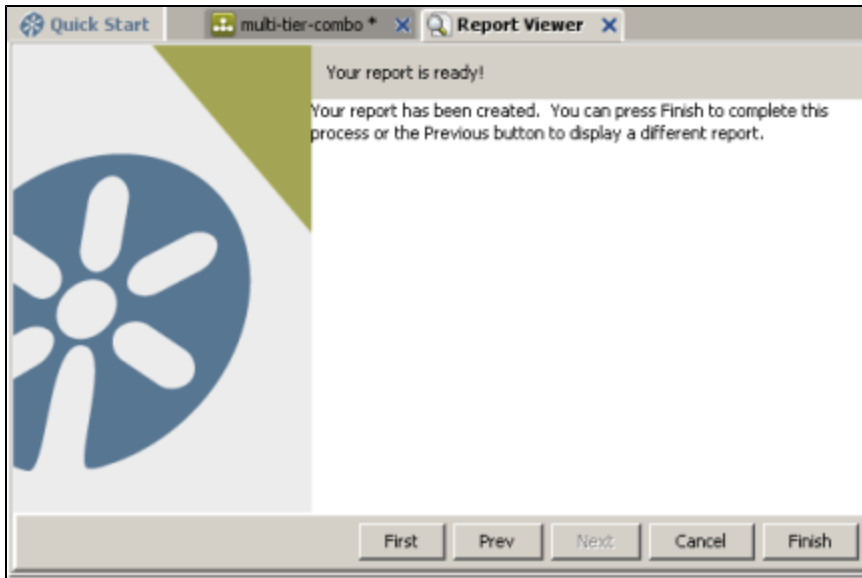
Select an **output** destination and enter a **file name** for the report.

Click **Next**.

At this point you could click **Prev** to return to the **Available report types** listing (back screen), or **First** to return to the **Available Reports listing** (first screen).

LISA will generate the report and will be made available for you to view.

LISA will show a report confirmation screen:



At this point you can:

- **Click Finish** to leave the Report viewer
- **Click Prev** to choose a different output destination
- **Click First** to return to the Available Reports listing

Generated Report

The generated reports are listed in the Reports folder of the LISA installation directory.

Some of the generated reports are shown below:



100 %

**Metric Data Information**

Interval	Name	Count	Avg	Min	Max	Sum	First	Last
0	Event: Cycle failed/*	10	0	0	0	0	0	0
0	Event: Step error/*	10	0	0	0	0	0	0
0	Event: Step started/*	10	1	0	1	7	0	1
0	Event: Abort/*	10	0	0	0	0	0	0
0	LISA: Instances	10	1	1	1	10	1	1
0	LISA: Avg Resp Time (ms)	10	0	0	0	0	0	0
0	LISA: Steps per second	10	0	0	0	0	0	0

Interval	Name	Count	Avg	Min	Max	Sum	First	Last
1	Event: Cycle failed/*	10	0	0	0	0	0	0
1	Event: Step error/*	10	0	0	0	0	0	0
1	Event: Step started/*	10	1	1	2	11	1	2
1	Event: Abort/*	10	0	0	0	0	0	0
1	LISA: Instances	10	1	1	1	10	1	1
1	LISA: Avg Resp Time (ms)	10	2,906	0	14,531	29,062	0	14,531
1	LISA: Steps per second	10	0	0	0	0	0	0

Interval	Name	Count	Avg	Min	Max	Sum	First	Last
2	Event: Cycle failed/*	10	0	0	0	0	0	0
2	Event: Step error/*	10	0	0	0	0	0	0
2	Event: Step started/*	10	4	2	4	36	2	4
2	Event: Abort/*	10	0	0	0	0	0	0
2	LISA: Instances	10	1	1	1	10	1	1
2	LISA: Avg Resp Time (ms)	10	6,598	5,432	14,531	65,979	14,531	5,432
2	LISA: Steps per second	10	0	0	0	1	0	0

Interval	Name	Count	Avg	Min	Max	Sum	First	Last
3	Event: Cycle failed/*	10	0	0	0	0	0	0
3	Event: Step error/*	10	0	0	0	0	0	0
3	Event: Step started/*	10	5	4	5	46	4	5

[illegible]

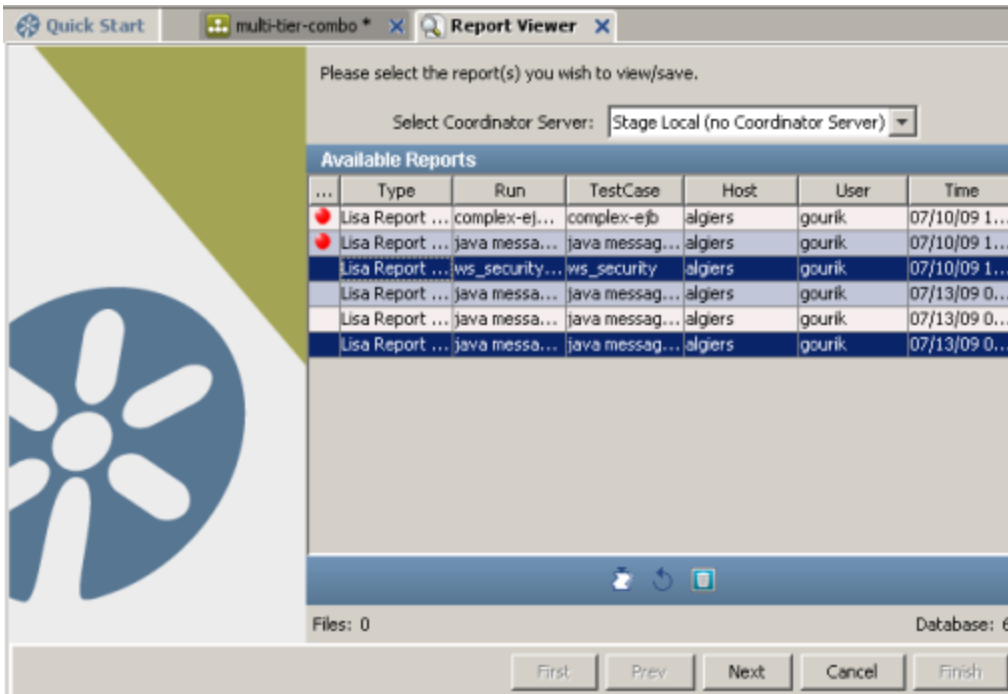
Metric Data Info - Print Preview								
File Navigation Zoom Help								
100 %								
Metric Data Information								
Interval	Name	Count	Avg	Min	Max	Sum	First	Last
0	Event: Abort/*	10	0	0	0	0	0	0
1	Event: Abort/*	10	0	0	0	0	0	0
2	Event: Abort/*	10	0	0	0	0	0	0
3	Event: Abort/*	10	0	0	0	0	0	0
4	Event: Abort/*	10	0	0	0	0	0	0
5	Event: Abort/*	10	0	0	0	0	0	0
6	Event: Abort/*	10	0	0	0	0	0	0
7	Event: Abort/*	10	0	0	0	0	0	0
8	Event: Abort/*	10	0	0	0	0	0	0
9	Event: Abort/*	10	0	0	0	0	0	0
10	Event: Abort/*	10	0	0	0	0	0	0
11	Event: Abort/*	10	0	0	0	0	0	0
12	Event: Abort/*	10	0	0	0	0	0	0
13	Event: Abort/*	7	0	0	0	0	0	0
Interval	Name	Count	Avg	Min	Max	Sum	First	Last
0	Event: Cycle failed/*	10	0	0	0	0	0	0
1	Event: Cycle failed/*	10	0	0	0	0	0	0
2	Event: Cycle failed/*	10	0	0	0	0	0	0
3	Event: Cycle failed/*	10	0	0	0	0	0	0
4	Event: Cycle failed/*	10	0	0	0	0	0	0
5	Event: Cycle failed/*	10	0	0	0	0	0	0
6	Event: Cycle failed/*	10	0	0	0	0	0	0
7	Event: Cycle failed/*	10	0	0	0	0	0	0
8	Event: Cycle failed/*	10	0	0	0	0	0	0
9	Event: Cycle failed/*	10	0	0	0	0	0	0
10	Event: Cycle failed/*	10	0	0	0	0	0	0
11	Event: Cycle failed/*	10	0	0	0	0	0	0
12	Event: Cycle failed/*	10	0	0	0	0	0	0
13	Event: Cycle failed/*	7	0	0	0	0	0	0
11/01/2010		Lisa © ITKO, 2009					Page 1 of 5	

24.2.6 Comparing two Baseline Reports

24.2.6 Comparing two Baseline Reports

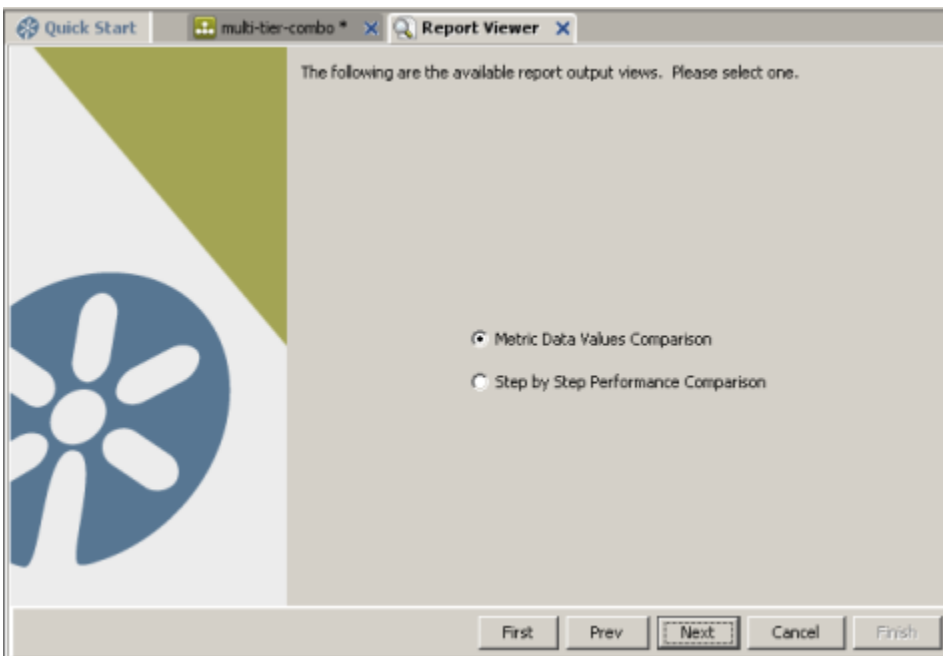
To compare two reports at one time, use the **Ctrl** key to select the two reports.

The two reports should be reports from the same test configuration performed at different times:



Click **Next**.

You can now choose between a baseline comparison of metrics or performance data:



Click **Next**.

LISA will now show both sets of results in a single report.

You can see an example of a baseline report below in the Sample Reports section.

24.2.7 Sample Reports

24.2.7 Sample Reports

For the purpose of illustration, we will look at the reports created using the **multi-tier-combo** Test Case in the examples directory (multi-tier-combo.tst).

The following sample reports are available -

Functional Test Report

Performance Summary Report,

Report Metric Data Info Report,

Baseline Performance Report.

The figures show the reports using the 'Print Preview to Screen' output option, and the full Report Viewer was captured in the screenshots.

The 'standard' icons at the top of the viewer allow you to manipulate the reports as you view them:



Functional Test Report

The sample Functional Test report shows a fragment of the first page. For each step in the test, the report shows a response time summary, and the request and response for each invocation of the step.

Functional Test Report

ws_security [weblogic-jmx-localhost-run]
gounik@algiers

Start: Jul 10, 2009 12:42:37 PM
Finish: Jul 10, 2009 12:49:59 PM
Duration: 7m 22.738s

Tests Attempted: 124
Tests Started: 124
Tests Passed: 124 100.0 %
Tests Failed: 0 0.0 %

Test Step: Step 1: Test XML Signature			
Type :	Web Service Execution		
Exec Count:	124	Min Resp (ms): 703	Std Dev Resp (ms): 215.1
Exec Time (ms):	96013	Max Resp (ms): 2484	Avg Resp (ms): 774.30
Total Bandwidth:	537.1 K	Margin of Error:	8.98 %

1. Info message

com.iko.lisa.ws.gen.SignedCalculatorService.CalculatorServiceLocator

2. Info message

TransInfo for http://webservice.samples.iko.com:add Build status is S has 0 child components.

3. Info message

com.iko.lisa.ws.gen.SignedCalculatorService.CalculatorServiceLocator

4. Info message

TransInfo for http://webservice.samples.iko.com:add Build status is S has 0 child components.

5. Info message

com.iko.lisa.ws.gen.SignedCalculatorService.CalculatorServiceLocator

6. Info message

TransInfo for http://webservice.samples.iko.com:add Build status is S has 0 child components.

7. Info message

com.iko.lisa.ws.gen.SignedCalculatorService.CalculatorServiceLocator

8. Info message

TransInfo for http://webservice.samples.iko.com:add Build status is S has 0 child components.

9. Info message

com.iko.lisa.ws.gen.SignedCalculatorService.CalculatorServiceLocator

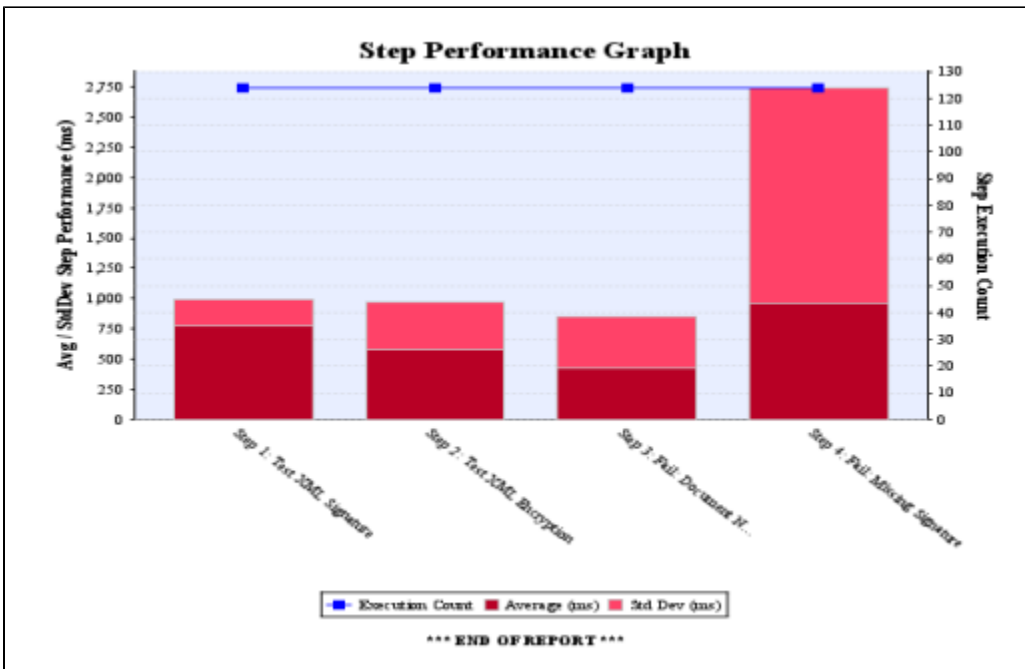
Performance Summary Report

The sample Performance Summary report shows both pages of the report.

The first page shows the performance numbers in tabular form.

Step Performance Summary						
ws_security [weblogic-jmx4localhost-run] gourik@algiers						
Step Name	Count	Total(ms)	Avg(ms)	Min(ms)	Max(ms)	Std Dev(ms)
All Test Runs	124	441,568	3,561.0	2,922	20,953	1,902.0
Step 1: Test XML Signature	124	96,013	774.3	703	2,484	215.1
Step 2: Test XML Encryption	124	71,673	578.0	453	2,406	393.4
Step 3: Fail: Document Not Encrypted	124	52,002	419.4	343	4,891	427.5
Step 4: Fail: Missing Signature	124	119,352	962.5	656	18,313	1,781.4
All Steps	496	339,040	683.5	343	18,313	962.1
Start Time: Jul 10, 2009 12:42:37 PM	Tests Passed:	124	EVENT_RESPTIME count:			496
Finish Time: Jul 10, 2009 12:49:59 PM	Tests Failed:	0				
Duration: 7m 22.738s						
Steps / sec: 1.12						

The second page shows the same information graphically.



Report Metric Data Info Report

The sample of Report Metric Data Info is as shown below. The report groups the metrics by **Interval** and by **Name** of the Metrics.

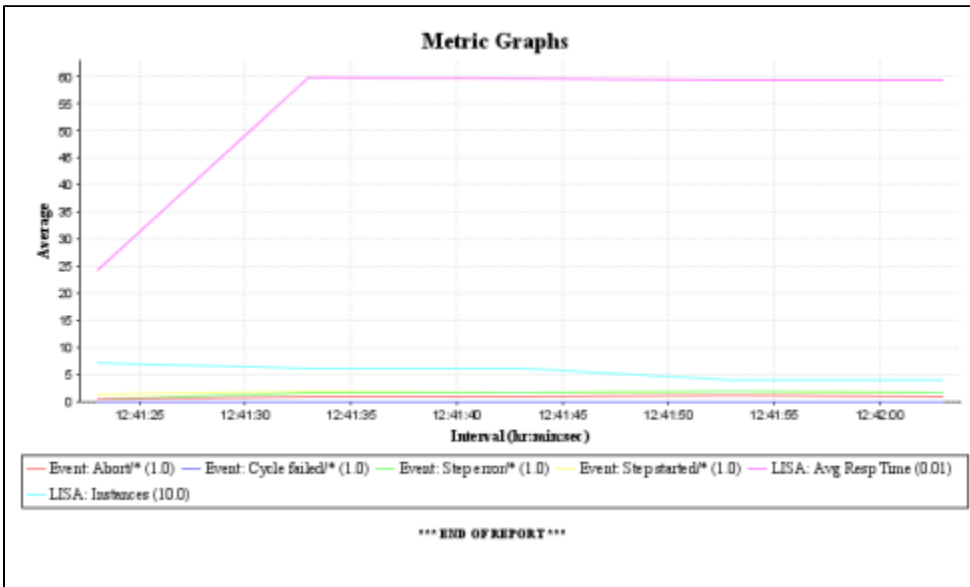
Interval Data										
complex-eyb [Run User] Min]										
gounik@edgiers										
Start Time: Jul 10, 2009 12:41:13 PM			Finish Time: Jul 10, 2009 12:42:14 PM					Duration: 1m 1.000s		
Interval	Name	Samples	Avg	Std Dev	Min	Max	First	Last	Sum	
0	Event: Abort*	10	0.3	0.5	0	1	0	1	3	
0	Event: Cycle failed*	10	0.0	0.0	0	0	0	0	0	
0	Event: Step error*	10	0.4	0.5	0	1	0	1	4	
0	Event: Step started*	10	1.3	0.5	1	2	1	2	13	
0	LISA: Avg Resp Time	10	2,418.4	3,322.1	0	6,046	0	6,046	24,184	
0	LISA: Instances	10	0.7	0.5	0	1	1	0	7	
Interval	Name	Samples	Avg	Std Dev	Min	Max	First	Last	Sum	
1	Event: Abort*	10	0.9	0.4	1	2	1	2	12	
1	Event: Cycle failed*	10	0.0	0.0	0	0	0	0	0	
1	Event: Step error*	10	1.6	0.7	1	3	1	3	20	
1	Event: Step started*	10	1.7	0.7	2	4	2	4	30	
1	LISA: Avg Resp Time	10	5,092.2	20.6	5,063	6,046	6,046	5,063	50,922	
1	LISA: Instances	10	0.6	0.5	0	1	0	0	6	
Interval	Name	Samples	Avg	Std Dev	Min	Max	First	Last	Sum	
2	Event: Abort*	10	0.0	0.0	2	2	2	2	20	
2	Event: Cycle failed*	10	0.0	0.0	0	0	0	0	0	
2	Event: Step error*	10	1.6	0.5	3	4	3	4	36	
2	Event: Step started*	10	1.6	0.5	4	5	4	5	46	
2	LISA: Avg Resp Time	10	5,019.4	3.1	5,057	5,063	5,063	5,057	50,194	
2	LISA: Instances	10	0.6	0.5	0	1	0	1	6	
Interval	Name	Samples	Avg	Std Dev	Min	Max	First	Last	Sum	
3	Event: Abort*	10	1.0	0.0	3	3	3	3	30	

10-Jul-2009

Lisa ITFCO, 2008

Page 1 of 3

Both reports show the metric data graphically at the end of the report, as shown below:



Descriptions of all metrics and details on how to configure them for inclusion in your reports, can be found in next chapter [Metrics](#).

Baseline Performance Report

The sample Baseline Performance report shows both pages of the report.

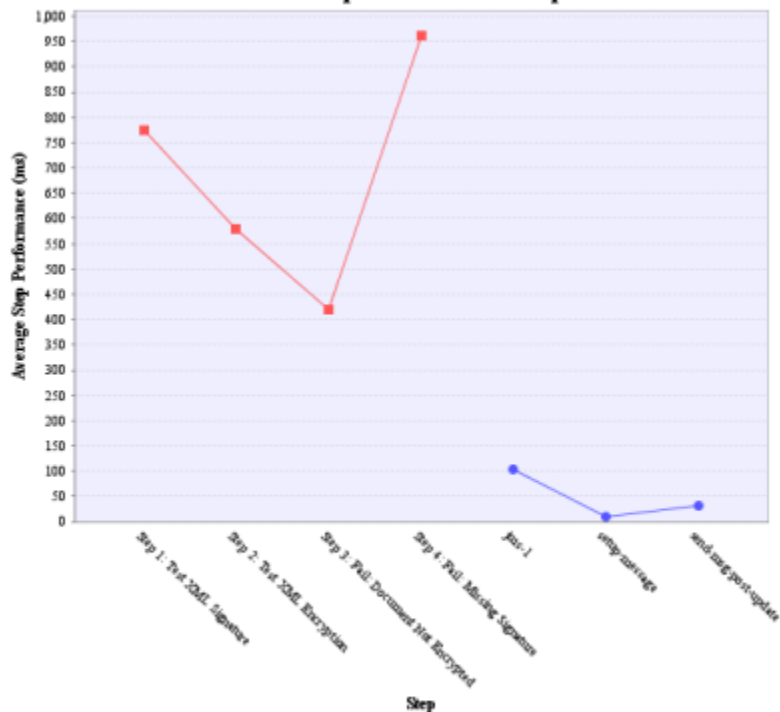
The first page shows a comparison of performance numbers for the two runs in tabular form.

Combined Step Performance Summary

Step: Step 1: Test XML Signature	Count	Total(ms)	Avg(ms)	Min(ms)	Max(ms)	Std Dev (ms)
ws_security [weblogic-jmx-localhost-run]7/10/09 12:42	124	96,013	774.3	703	2,484	215.1
Step: Step 2: Test XML Encryption	Count	Total(ms)	Avg(ms)	Min(ms)	Max(ms)	Std Dev (ms)
ws_security [weblogic-jmx-localhost-run]7/10/09 12:42	124	71,673	578.0	453	2,406	393.4
Step: Step 3: Fail: Document Not Encrypted	Count	Total(ms)	Avg(ms)	Min(ms)	Max(ms)	Std Dev (ms)
ws_security [weblogic-jmx-localhost-run]7/10/09 12:42	124	52,002	419.4	343	4,891	427.5
Step: Step 4: Fail: Missing Signature	Count	Total(ms)	Avg(ms)	Min(ms)	Max(ms)	Std Dev (ms)
ws_security [weblogic-jmx-localhost-run]7/10/09 12:42	124	119,352	962.5	656	18,313	1,781.4
Step: check-order	Count	Total(ms)	Avg(ms)	Min(ms)	Max(ms)	Std Dev (ms)
java messaging service [jboss-metric-run]7/13/09 9:43	0	0	0.0	0	0	0.0
Step: end	Count	Total(ms)	Avg(ms)	Min(ms)	Max(ms)	Std Dev (ms)
ws_security [weblogic-jmx-localhost-run]7/10/09 12:42	0	0	0.0	0	0	0.0
java messaging service [jboss-metric-run]7/13/09 9:43	0	0	0.0	0	0	0.0
Step: jms-1	Count	Total(ms)	Avg(ms)	Min(ms)	Max(ms)	Std Dev (ms)
java messaging service [jboss-metric-run]7/13/09 9:43	3	313	104.3	47	219	99.3
Step: send-msg-post-update	Count	Total(ms)	Avg(ms)	Min(ms)	Max(ms)	Std Dev (ms)
java messaging service [jboss-metric-run]7/13/09 9:43	3	93	31.0	16	46	15.0
Step: setup-message	Count	Total(ms)	Avg(ms)	Min(ms)	Max(ms)	Std Dev (ms)
java messaging service [jboss-metric-run]7/13/09 9:43	3	31	10.3	0	16	9.0
Step: start	Count	Total(ms)	Avg(ms)	Min(ms)	Max(ms)	Std Dev (ms)

The second page shows the same information graphically.

Combined Step Performance Graph



ws_security [weblogic-jmx-localhost-run] 2009 Jul 10 12:42:37(037)
 java messaging service [jboss-metric-run] 2009 Jul 13 09:43:24(360)

*** END OF REPORT ***

24.3 General Information

24.3 General Information

The following topics are available in this chapter.

[24.3.1 Using a Different Reporting Database](#)
[24.3.2 Auto Generate Report Email Settings](#)

24.3.1 Using a Different Reporting Database

24.3.1 Using a Different Reporting Database

A user defined report database can be specified in one of the LISA property files instead of the default internal reporting database. You can modify **site.properties** for a Server installation, or **local.properties** for a local installation. These property files include example properties for MySQL, Oracle, and SQLServer. Uncomment the appropriate lines and specify the required information.

By default the report database writes to the 'internal' connection. If you would like all Lisa applications to write to the same database (other than the default) then you can change the properties of the 'internal' connection:

1. rpt.db.internal.url =org.hibernate.dialect.MySQLDialect
2. rpt.db.internal.user =com.mysql.jdbc.Driver
3. rpt.db.internal.password =jdbc:mysql:my-db-server.com://lisa-reports

However, if you need the Workstation to write to one database and Coordinator to write to a different database, then you will need to include the **application_id** in the property name. For example if you would like Workstation to write to internal database and Coordinator to write to MySQL database then you can set the properties for Coordinator as follows:

```
#set Coordinator to use MySQL
#rpt.2.hibernate.dialect=org.hibernate.dialect.MySQLDialect
#rpt.2.hibernate.connection.driver_class=com.mysql.jdbc.Driver
#rpt.2.hibernate.connection.url=jdbc:mysql:my-db-server.com://lisa-reports
#rpt.2.hibernate.connection.username=my_username
#rpt.2.hibernate.connection.password=my_password
The fragment below shows the samples in the property files:
#LISA Reporting Database example
#use the internal database by default – see rpt.db.internal settings
rpt.hibernate.connection.url=internal
rpt.db.internal.user=rpt
rpt.db.internal.password_enc=76f271db3661fd50082e68d4b953fbee
rpt.db.internal.url=jdbc:derby://localhost:1527/reports/lisa-reports.db:create=true
#
```

1. In order to check for expired reports
2. set autoExpire = true
3. set the expiration – an integer followed by (m=month,w=week,d=day,h=hour)
4. the default expiration period is 30d (30 days)
perfmgr.rvwiz.whatrpt.autoExpire=true
perfmgr.rvwiz.whatrpt.expireTimer=30d

```
#properties may have an index for the proper application
#if you wish to run testmanager and coordinator on the same box, then you
#must have entries for each one
#0=testmanager, 2=coordinator, 5=testrunner
#
#rpt.0.hibernate.dialect=org.hibernate.dialect.MySQLDialect
#rpt.0.hibernate.connection.driver_class=com.mysql.jdbc.Driver
#rpt.0.hibernate.connection.url=jdbc:mysql://localhost:3306/lisareports
#rpt.0.hibernate.connection.username=
#rpt.0.hibernate.connection.password=
#
#rpt.0.hibernate.dialect=org.hibernate.dialect.OracleDialect
#rpt.0.hibernate.connection.driver_class=oracle.jdbc.driver.OracleDriver
#rpt.0.hibernate.connection.url=jdbc:oracle:thin:@localhost:1526:lisareports
#rpt.0.hibernate.connection.username=
#rpt.0.hibernate.connection.password=
#
#rpt.0.hibernate.dialect=org.hibernate.dialect.SQLServerDialect
#rpt.0.hibernate.connection.driver_class=net.sourceforge.jtds.jdbc.Driver
```

```
#rpt.0.hibernate.connection.url=jdbc:jtds:sqlserver://localhost:1433/lisareports
#rpt.0.hibernate.connection.username=
#rpt.0.hibernate.connection.password=
```

The DDL for the LISA reporting database is located in **LISA_HOME/sql/lisa-report-ddl.sql**. This script must be run before LISA can record reporting information to the database specified above.

Current Hibernate Dialects

The current Hibernate dialects are as follows:

RDBMS	Dialect
DB2	org.hibernate.dialect.DB2Dialect
DB2 AS/400	org.hibernate.dialect.DB2400Dialect
DB2 OS390	org.hibernate.dialect.DB2390Dialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect
MySQL	org.hibernate.dialect.MySQLDialect
MySQL with InnoDB	org.hibernate.dialect.MySQLInnoDBDialect
MySQL with MyISAM	org.hibernate.dialect.MySQLMyISAMDialect
Oracle (any version)	org.hibernate.dialect.OracleDialect
Oracle 9i/10g	org.hibernate.dialect.Oracle9Dialect
Sybase	org.hibernate.dialect.SybaseDialect
Sybase Anywhere	org.hibernate.dialect.SybaseAnywhereDialect
Microsoft SQL Server	org.hibernate.dialect.SQLServerDialect
SAP DB	org.hibernate.dialect.SAPDBDialect
Informix	org.hibernate.dialect.InformixDialect
HypersonicSQL	org.hibernate.dialect.HSQLDialect
Ingres	org.hibernate.dialect.IngresDialect
Progress	org.hibernate.dialect.ProgressDialect
Mckoi SQL	org.hibernate.dialect.MckoiDialect
Interbase	org.hibernate.dialect.InterbaseDialect
Pointbase	org.hibernate.dialect.PointbaseDialect
FrontBase	org.hibernate.dialect.FrontbaseDialect
Firebird	org.hibernate.dialect.FirebirdDialect

24.3.2 Auto Generate Report Email Settings

24.3.2 Auto Generator Report Email Settings

In a Staging document, you can specify the filename under the **Autogenerate** panel in the Reports tab.

In the last column we can specify the email address to send the report file to.

The email field takes a comma separated list of email addresses.

By default the report generator will use the same SMTP server as the alert email feature. This host name is set in property **lisa.alert.email.defHosts** which defaults to localhost.

If you wish to have a separate server for reports you may set it using the **rpt.autogen.email.host** property.

PART 6 - Recorders & Test Generators

PART 6 - Recorders and Test Generators

LISA provides a variety of methods to record the test cases and re-run them.

LISA Test's no-code testing environment allows QA, Development and others to rapidly design and execute functional, unit, regression and load tests against dynamic web sites (RIAs).

LISA can be used to test rich browser and web user interfaces, as well as the many building blocks and data residing below the UI. With LISA, all of the data and implementation layers the team needs to functionally test can be analyzed, invoked and verified to ensure requirements are met.

- You can use Web Recorder (HTTP Proxy) when you want to track the path through the website. Steps are created for each HTTP request.
- You can use Web Recorder (DOM Events) when you wish to capture mouse clicks, mouse movements, keys being typed, etc. and play those events back during test execution.

In this section, the following topics are covered:

25. Recording a Website

26. Generating a Web Service

25. Recording a Website

25. Recording a Website

The recording of a website can be done in two ways:

1. Via HTTP proxy -

LISA Workstation has a HTTP recorder to test a Web Site Test Case via a Proxy recorder.

This helps to track your path through the web site and automatically create Test steps for each HTTP request that is generated while recording.

2. Via Web 2.0 Browser -

This will record the web site events as well as all the physical and logical events.

The following sections are available.

25.1 Recording Via HTTP Proxy

25.2 Recording Via DOM Events

25.1 Recording a Web Site via HTTP Proxy

25.1 Recording a Web Site via HTTP Proxy

LISA Workstation provides a HTTP recorder to test a Web Site Test Case via a Proxy recorder.

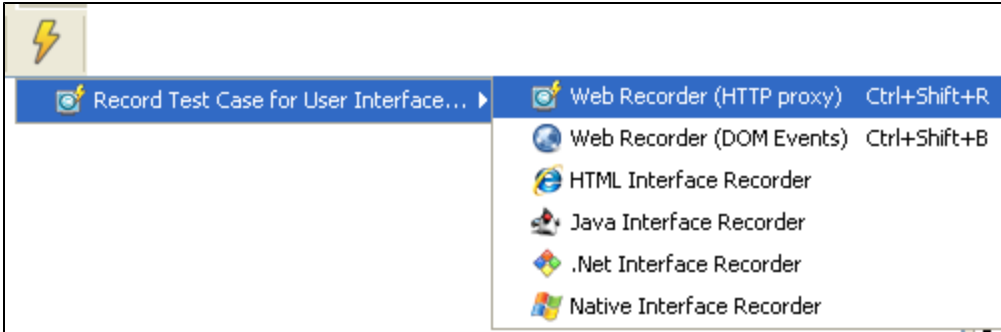
This helps to track your path through the web site and automatically create Test steps for each HTTP request that is generated while recording.

To start the recording,



Click the icon to open the following menu.

Click **Record Test Case for User Interface > Web Recorder (HTTP Proxy)**...

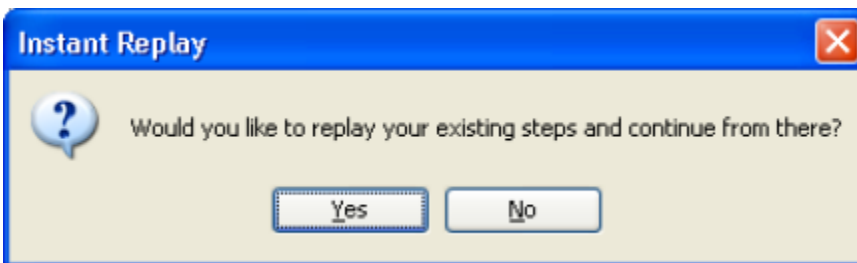


Or from the main menu,

Select **Actions > Record Test Case for User Interface > Web Recorder (HTTP Proxy)...**

This allows you to launch the browser that LISA uses to record and playback HTTP tests.

If there is a test case already open in the active tab, it will ask whether you want to replay the tests in the browser as below:



If there is no test case open in LISA Workstation,

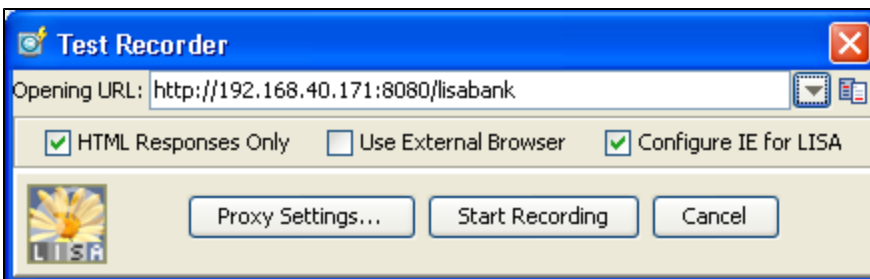
The Test Recorder window opens, where you need to enter the name of the web site to record.

For purpose of illustration, we will test the LISA Bank web application.

<http://localhost:8080/lisabank>.

Note - Replace localhost in the above path, with your machine IP address

- Enter the URL for the web page to test Ex: <http://192.168.40.171:8080/lisabank>.



Tick your preferences from the following:

- **HTML Responses Only** - will capture only the HTML responses
- **Use External Browser** - will open an external browser window
- **Configure IE for LISA** - will configure Internet Explorer for LISA

Port Usage

By default the LISA Proxy recorder uses port 8010 for recording.

If you do not want to use this port, you can override the setting in your **lisa.properties** file with the following property:

`lisa.editor.http.recorderPort=8010`

Pl change 8010 to the port you want to use.

To start the recording -

- Click **Start Recording** to start the Web recorder
- Or Click **Proxy Settings...** to Configure the Proxy.

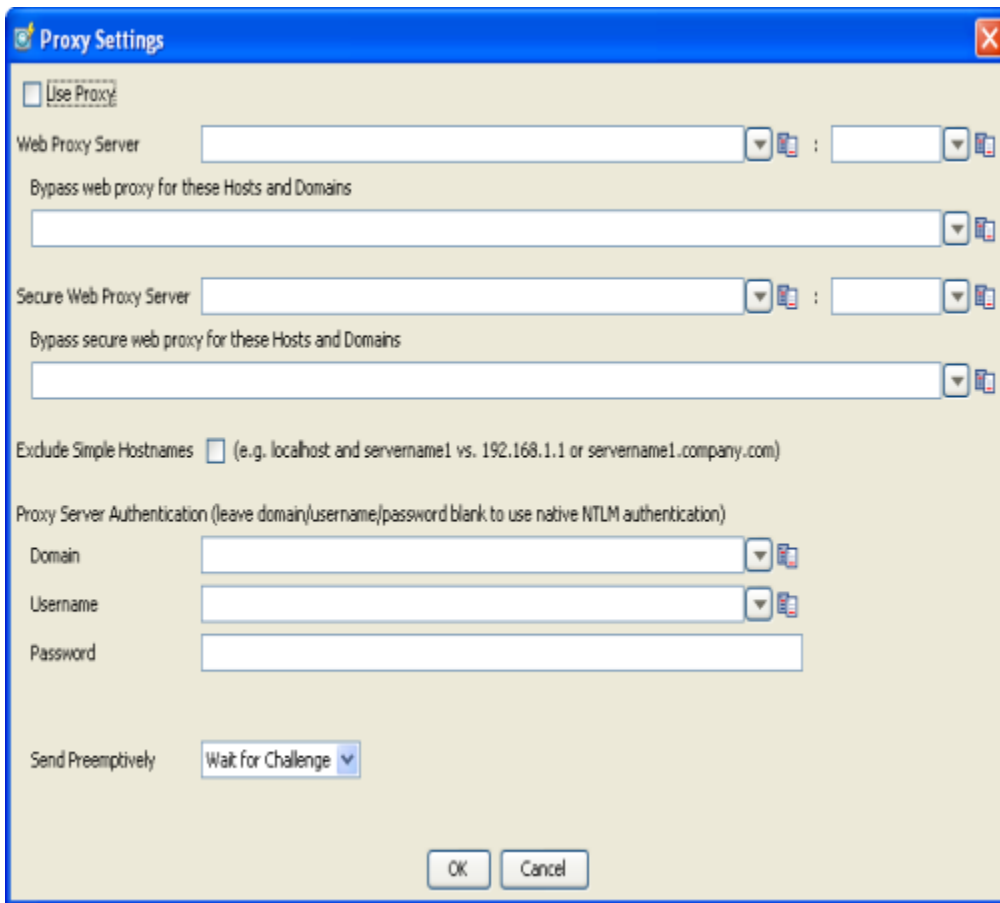
The following topics are available.

[25.1.1 Configure Proxy](#)
[25.1.2 Start Recording](#)
[25.1.3 Transactions Recorded](#)
[25.1.4 View in ITR](#)

25.1.1 Configure Proxy

25.1.1 Configure Proxy

If a Proxy is required, click the **Proxy Settings...** button to open the Proxy Setting dialog box as shown below:



The image shows a 'Proxy Settings' dialog box with a blue title bar and a close button. It contains several sections for configuring proxy settings. At the top, there is a checkbox labeled 'Use Proxy'. Below this, there are two main sections: 'Web Proxy Server' and 'Secure Web Proxy Server'. Each section has a text input field for the server address, a port input field, and a 'Bypass' section with a text input field for hosts and domains. There is also an 'Exclude Simple Hostnames' checkbox with a note: '(e.g. localhost and servername1 vs. 192.168.1.1 or servername1.company.com)'. The 'Proxy Server Authentication' section includes fields for 'Domain', 'Username', and 'Password', with a note: '(leave domain/username/password blank to use native NTLM authentication)'. At the bottom, there is a 'Send Preemptively' dropdown menu set to 'Wait for Challenge'. The dialog box has 'OK' and 'Cancel' buttons at the bottom right.

- Use Proxy - check to use Proxy settings
- Web Proxy Server - Enter the proxy server settings
 - Bypass Web Proxy for these hosts and domains - Enter the host name and domains of those, which you need to bypass proxy.
- Secure Web Proxy Server - Enter the secure proxy settings
 - Bypass Web Proxy Server for these hosts and domains
- Exclude simple host names - check if you want to exclude simple host names
- Proxy server Authentication - Enter authentication details
 - Domain - Enter domain name
 - User name - Enter user name
 - Password - Enter password
- Send Preemptively - Choose from Wait for Challenge/Send Basic or Send NTLM

Enter the Proxy configuration information in the above dialog box and click **OK**.

25.1.2 Start Recording

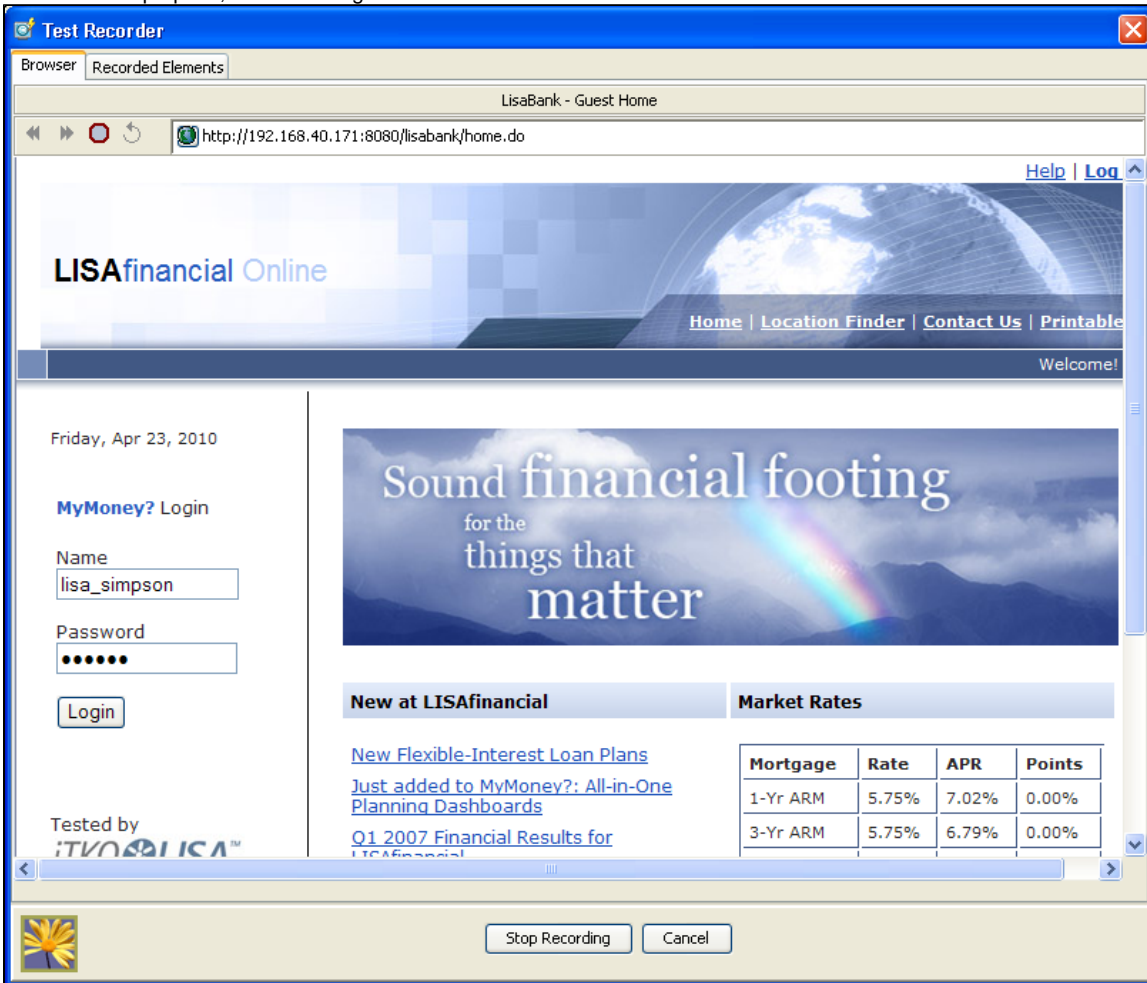
25.1.2 Start Recording

Click **Start Recording** in the Test Generator to begin recording the test.

The **Test Recorder** window opens up and displays the web page URL loaded.

You can test the web page by entering information as a user would.

For illustration purpose, we are testing the LISA Bank at the local demo server as shown below:



You can do many transactions in this site, like Add new Account, Deposit money into account, Add/Delete address etc.

Once you are done, you can stop recording your browser activity by clicking **Stop Recording** at the bottom of the Test Recorder window as shown above.

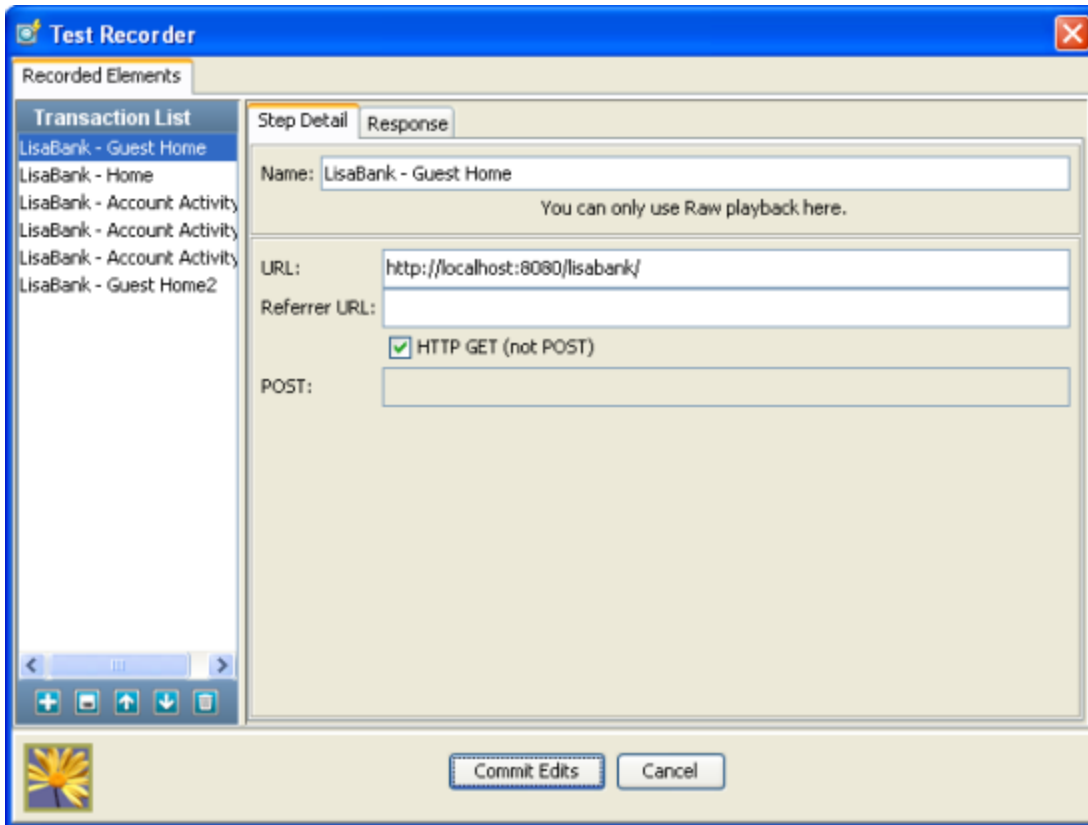
25.1.3 Transactions Recorded

25.1.3 Transactions Recorded

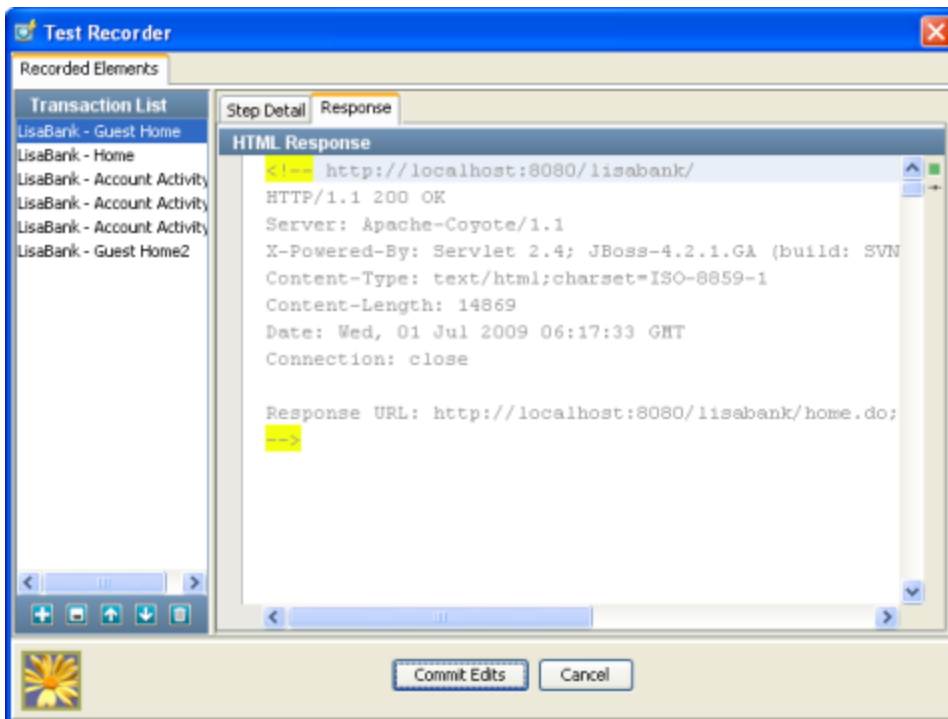
Once you stop recording, all the recorded transactions are seen in the **Recorded Elements** tab.

All the transactions done are listed in the left panel. The **Step wise details** and the **Response** are seen in the right tab.

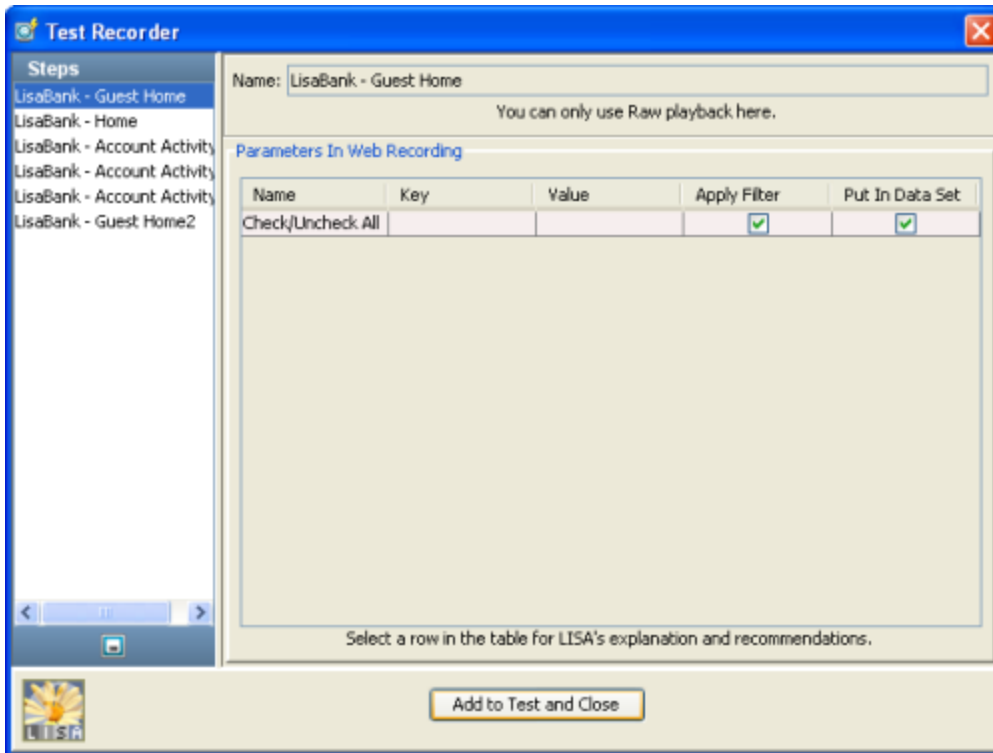
The Step Details of a particular transaction (LisaBank – Guest Home) are as shown below:



Select the **Response** tab to see the HTML response recorded.



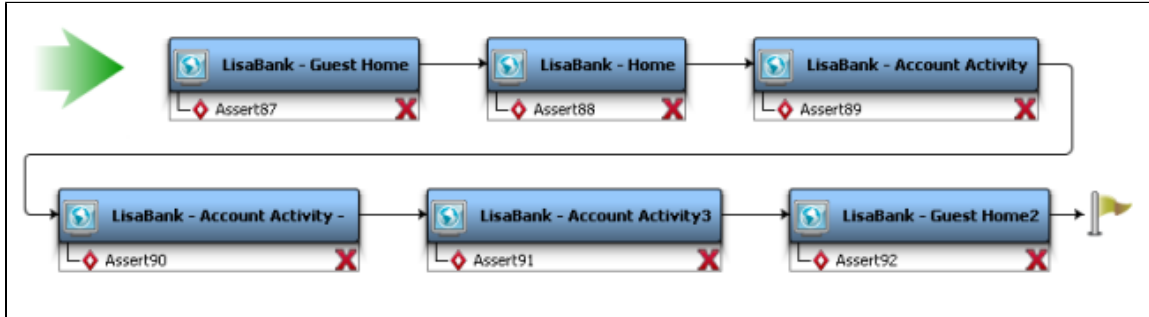
Click **Commit Edits** to commit these transactions in the Test Recorder as shown below:



In the **Parameters In Web Recording** you can enter any parameters required by your test.

Press **Add to Test and Close** at the bottom of the Test Recorder window to add transactions as test steps.

LISA then creates a Test Case based on your HTTP requests in LISA workflow. Each Step in the Test Case represents a recorded HTTP request.

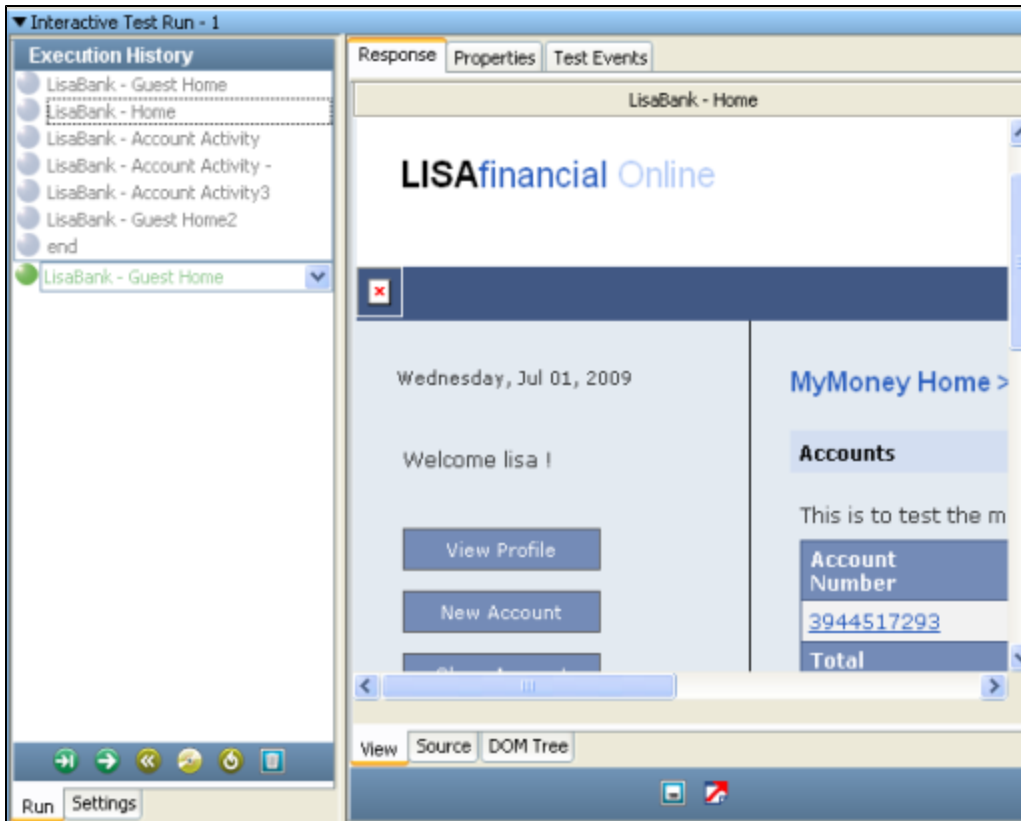


25.1.4 View in ITR

25.1.4 View in ITR

You can view all these transactions again, when you run this Test Case in the ITR.

Refer to the **View, Source and DOM Tree** tabs to see more information on the recorded step as shown below:



25.2 Recording a Web Site via DOM Events

25.2 Recording a Web Site via DOM Events

This feature is covered in a separate user document. See the [LISA Web 2.0 Guide](#) for additional information.

26. Generating a Web Service

18. Generating a Web Service

The Generate Web Service facility creates a **web service Test Case** automatically using just the information in a Web Service Definition Language (WSDL) file.

Since 5.0, there is a new step: **Web Services > Web Service Execution (XML)**

Though the legacy method - **Web Service Execution (Legacy)** still exists, users are encouraged to use the new one.

LISA reads the WSDL and generates a WSDL Validation step, along with a Web Service Execution step for each operation specified in the WSDL. This is equivalent to producing a Test Case manually using a combination of these steps, but it is more convenient if you want to produce steps for several operations in the WSDL. Once the Test Case has been created by LISA you can go in and delete any unwanted steps, and customize the remaining steps.

For more information on the WSDL Validation and the Web Service Execution steps see the [LISA Reference Guide](#).

The following topics are available.

[26.1 XML Web Service](#)

[26.2 Legacy Web Service](#)

[26.3 Web Service Customization](#)

26.1 XML Web Service

26.1 XML Web Service

You can create a Web service (XML) test case.

For this, you will use the **Web Service Execution** step to call Web service operations in a test case and test the response and request. These Web service operations provide the same functionality as the equivalent method calls in the EJB used in Tutorial 7.

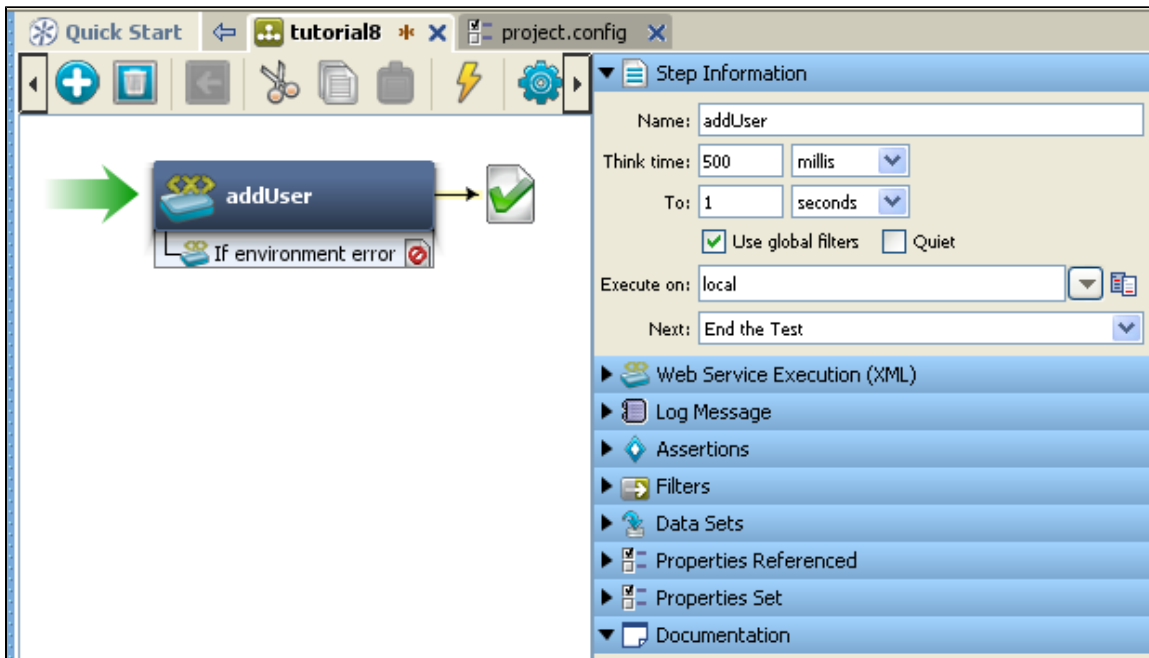
Make sure you are running the Demo server (either the local Demo Server, or the iTKO demo server) to use this step.

To Create a Web Service XML step,

Open the LISA Workstation and Right-click in the model editor

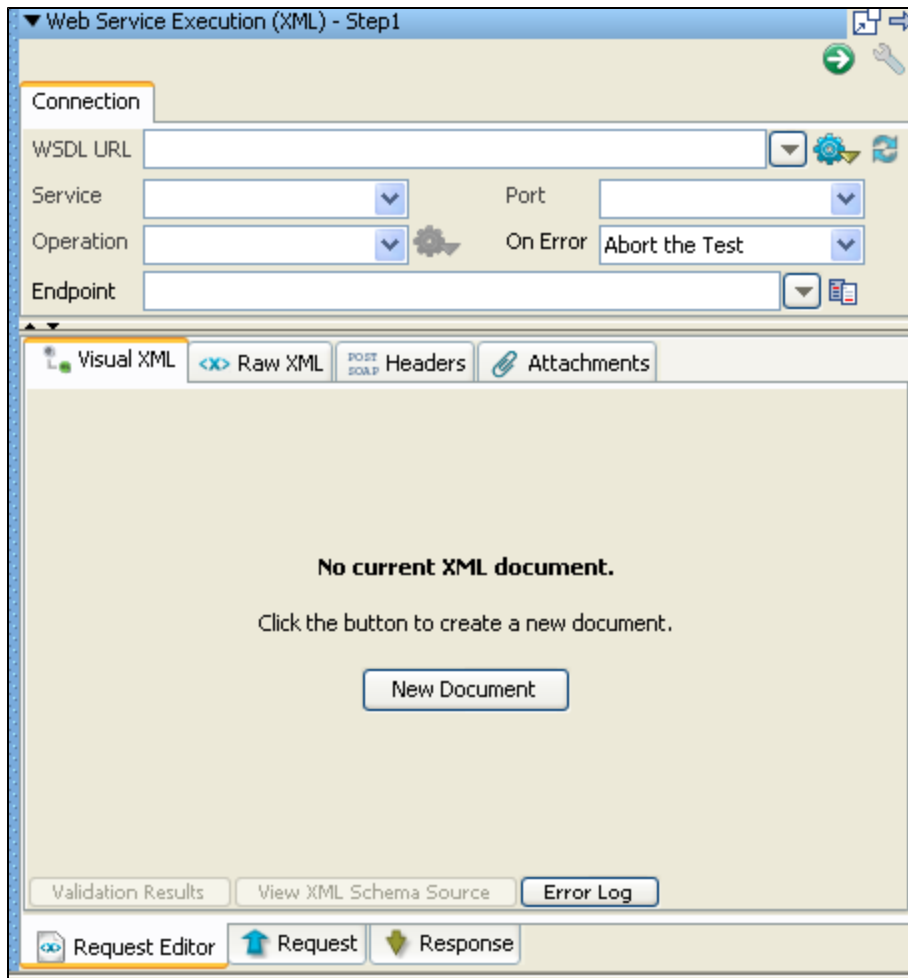
Click **Add Steps** or from the test case toolbar or you can click the Add Steps icon.

Click **Web/Web Services> Web Service Execution (XML)**, to add the step in the Model editor as shown below:



As you can see, a Web Services step is added. Rename the step as **AddUser** in the Step Information area.

4. Double click the Add User step to open the **Web Service Execution** editor as shown below:



6. Click on the **New Document** button to create a new XML document.

▼ Web Service Execution (XML) - Step1

Connection

WSDL URL

Service Port

Operation On Error

Endpoint

Visual XML Raw XML POST SOAP Headers Attachments

Node	Occurs	Nil	Nilable	Value

Validation Results View XML Schema Source Error Log

Request Editor Request Response

Creating the Web Service Client,

1. In the WSDL field, enter the location of the WSDL.

`http://WSSERVER:WSPORT/itko-examples/services/UserControlService?wsdl`

2. In the **Service Name** field, enter **UserControlServiceService**.

Note - Do not use spaces within the name of the web service.

3. In the **Port** field, enter **UserControlServiceService**.
4. In the Operation field, select the operation to be tested. Here we have selected "**addUser**"
5. In the On Error field, select the action to be taken on test Error - **Abort the Test**.

Connection

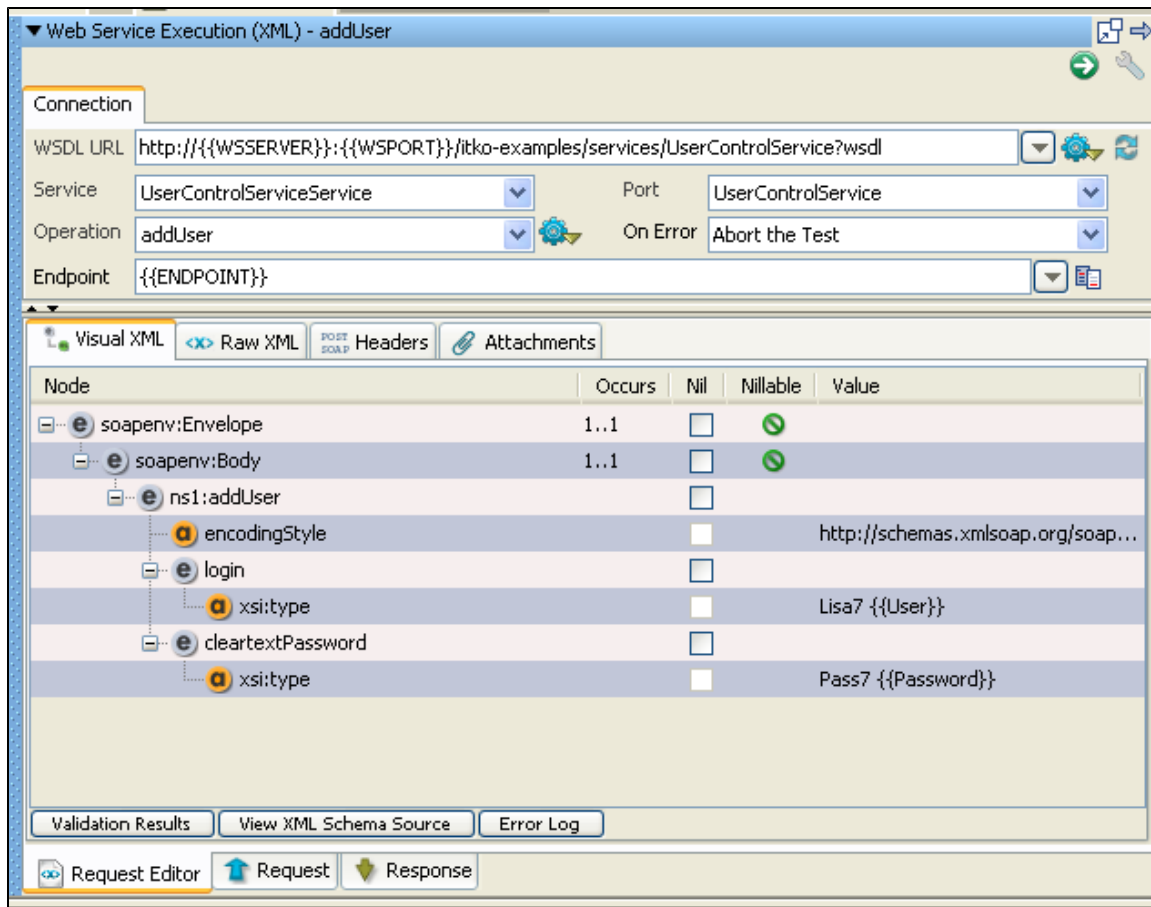
WSDL URL

Service Port

Operation On Error

Endpoint

LISA builds the Web Service client on this criteria as shown below:



Executing the Test Case

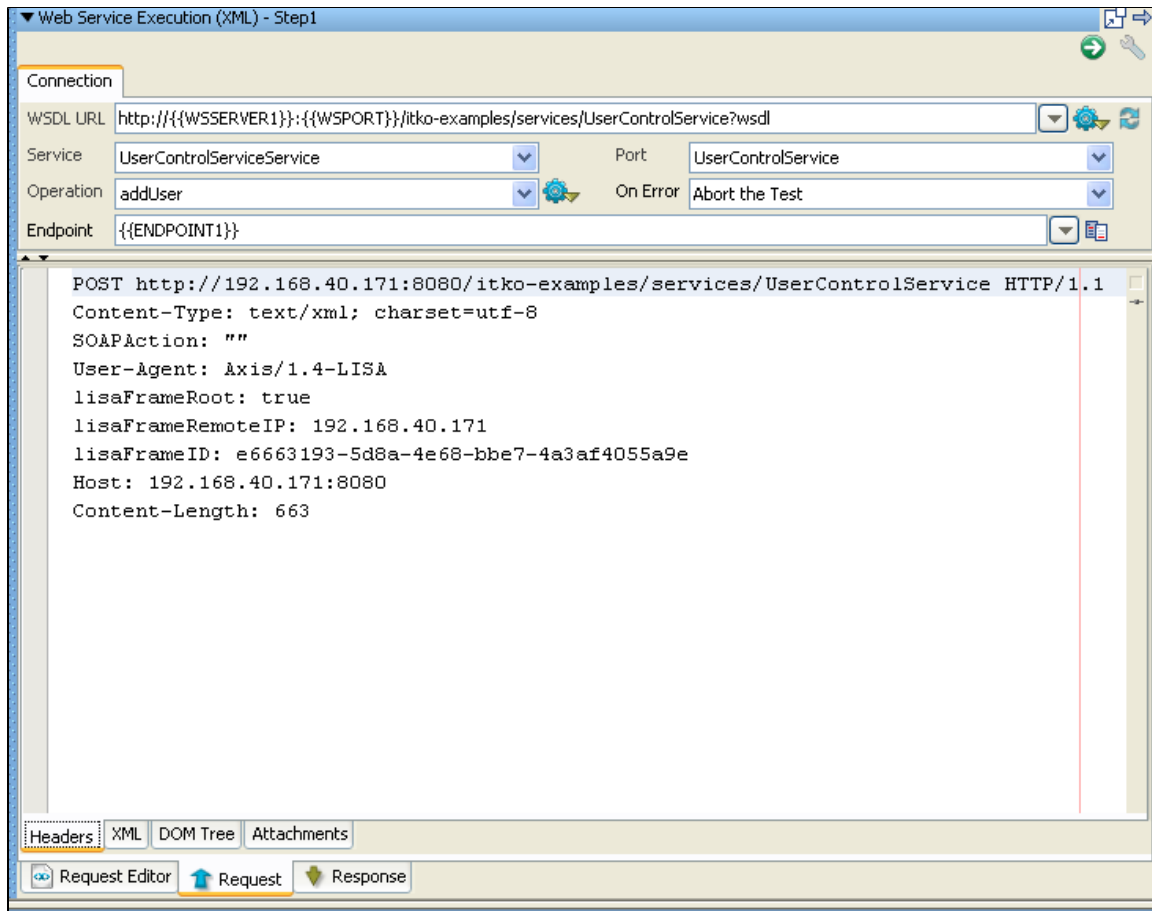
To execute the test case,

1. Click on the  button on top right.

This will Execute the test and show us the Request and Response as shown below:

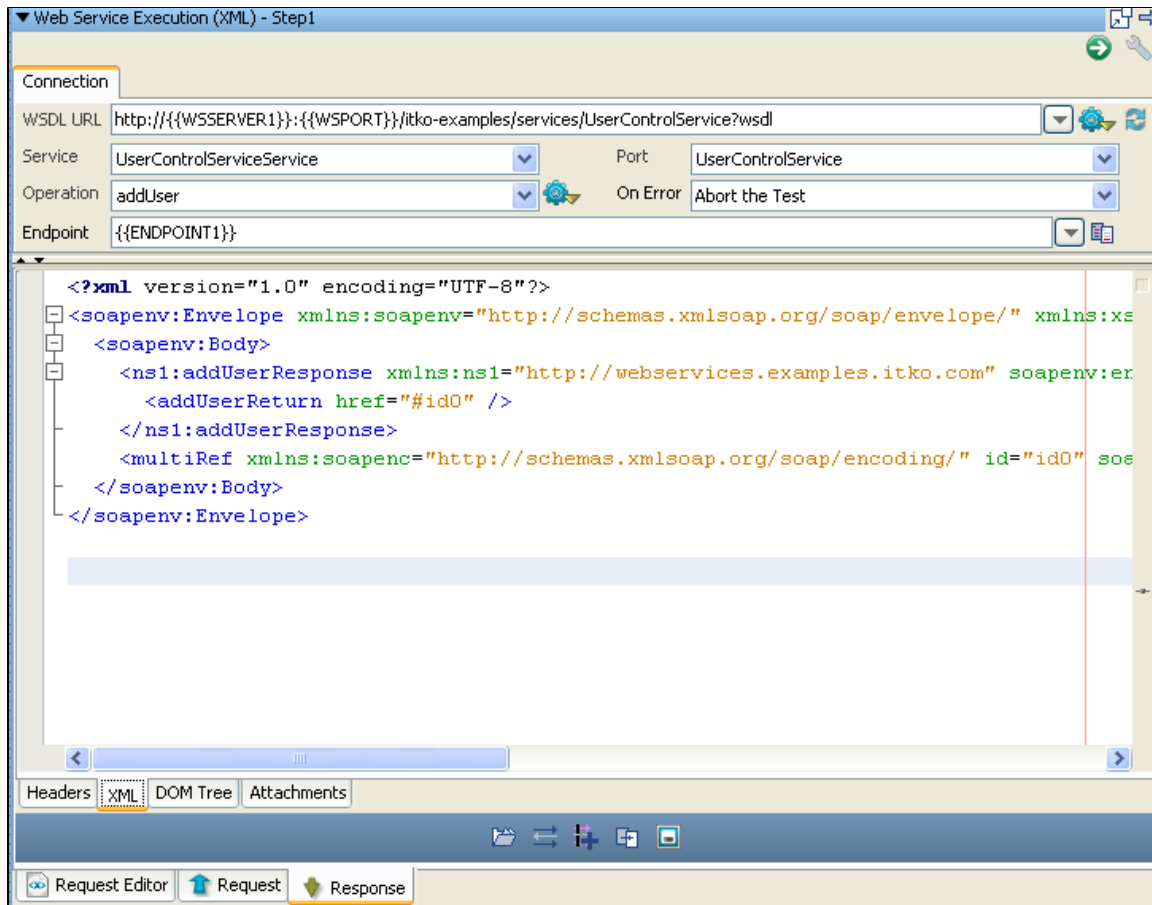
To view the request upon execution,

Click on the **Request** tab



To view the request upon execution,

Click on the **Response** tab



26.2 Legacy Web Service

26.2 Legacy Web Service

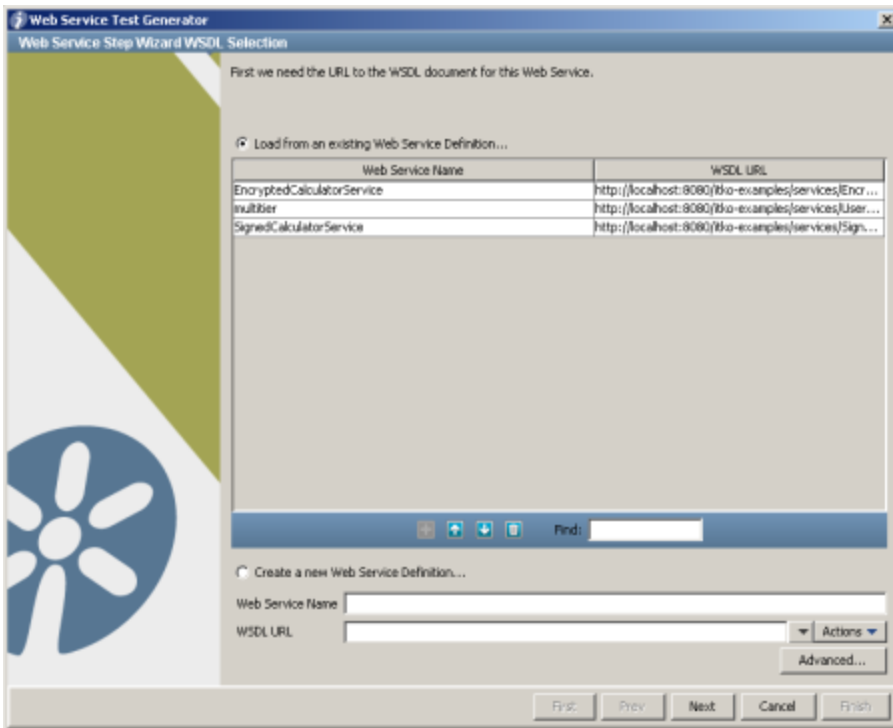
The first step is to create a new Test Case in Model Editor.

For information on creating a new Test Case see [7. Creating Test Cases](#).

To start the Generate Web Service facility,

Click Add step button > Web Services > Web Service Execution (Legacy)

This opens the **WSDL selection page** as shown below:



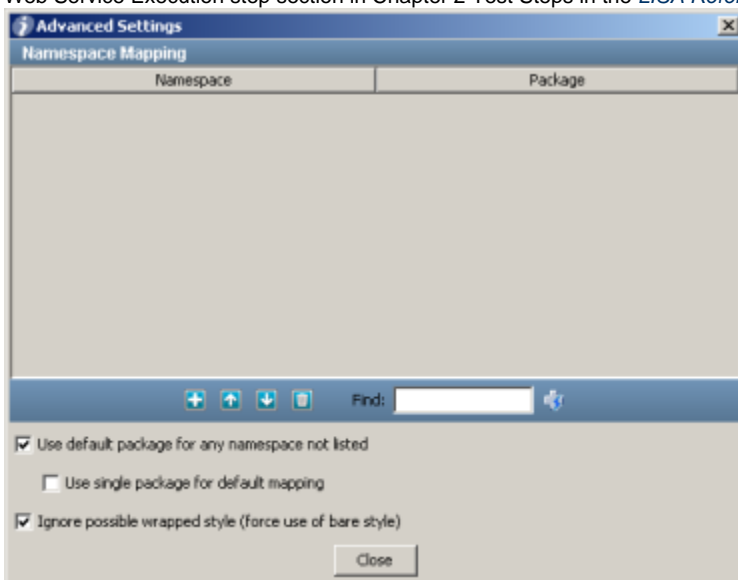
There are two sections on this screen selectable by radio button.

The **upper section** allows you to choose a **WSDL** that LISA already knows about, and has already constructed a client that is available to you.

- Select the radio button **Load from an existing Web Service definition...** (the radio button is automatically selected if you select a Web Service name from the list)
- Click and select the **Web Service** name

The **lower section** allows you to enter a new WSDL. LISA will build a client corresponding to the contents of this WSDL.

- Select the radio button **Create a new Web Service Definition...** (the radio button will automatically be selected if you begin entering information in the Web Service Name or WSDL URL fields)
- **Web Service Name:** Enter the desired name for the new service. This name must be unique. If a group of people plan on sharing Test Cases it's recommended to use a naming scheme that will ensure unique names for unique web services.
- **WSDL URL:** Enter the URL of the new WSDL, or select it from the pull-down list. You can also click the Actions pull-down to browse the file system, or search a UDDI Server.
- **Advanced...** button: This brings up a pop-up window where you can manage your namespaces. Namespace mapping is described in the Web Service Execution step section in Chapter 2 Test Steps in the [LISA Reference Guide](#).

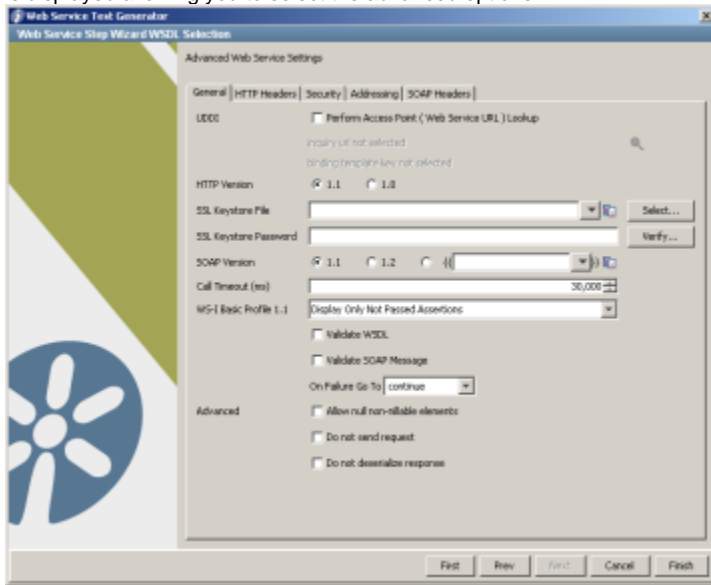


We will use an example that is available on the Demo Server.

The WSDL location is: <http://examples.itko.com:80/itko-examples/services/UserControlService?wsdl>, or <http://localhost:8080/itko-examples/services/UserControlService?wsdl>

We have chosen the latter in the figure above, and we have called our web service '**UserGuideWS**'. LISA will now generate the web service client based on the above input.

Once the web service client has been built, you will have the opportunity to add any advanced options you may wish to use. The following screen is displayed allowing you to select the advanced options:



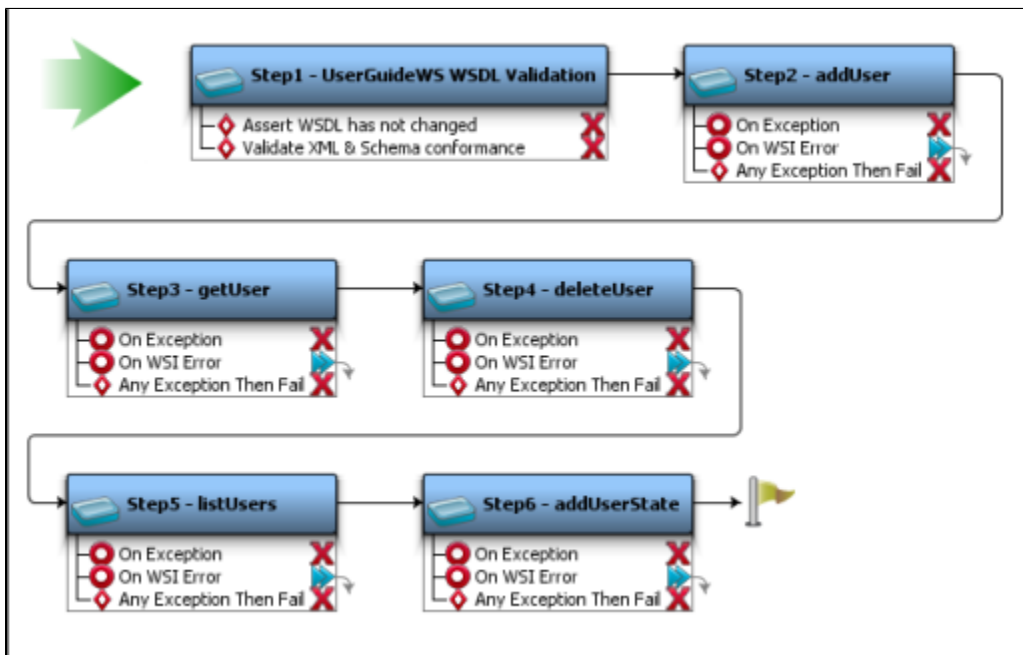
There are many options available to you here – **HTTP Headers, Security, Addressing, SOAP Headers**. These are explained in detail in the [LISA Reference Guide](#).

This is the same window as the one available in the **Web Service Execution** Step. However, any options chosen here are applied to each web service execution step in your Test Case.

For detailed information about the options on this screen, and in the five tabs that are present on this screen, we refer you to the Web Service Execution step section in Chapter 2 Test Steps in the [LISA Reference Guide](#).

Click **Finish**.

LISA will display the new Test Case in the Model Editor, as shown in the figure below.



In the Test Case workflow you can see new steps that were created for you.

Notice that the **WSDL Validation step**, Step 1, already has two Assertions attached to it.

The remaining steps are all of type **Web Service Execution**.

You are now ready to customize your Test Case.

26.3 Web Service Customization

26.3 Web Service Customization

At this point you have a basic Test Case, ready to be customized.

You would typically configure each step in a Test Case as you create it. Here all the steps were created at once. Never the less the procedure is very similar.

You can **delete** any steps that are not wanted. Then, starting with the first step, you would **modify** it, usually by inputting the necessary data into the web service operation parameters, and adding Filters and Assertions. You would then **add** Companions, configuration and Data Sets, etc.

For more details on configuring the WSDL Validation step and the Web Service Execution step see the [LISA Reference Guide](#).

26.4 Virtual Web Service (VSE)

26.4 Virtual Web Service (VSE)

IMPORTANT NOTE: *As of LISA 5.0, the Create a Virtual Web Service Step has been deprecated in favour of VSE and Virtual Service from WSDL.

For more information, please refer to the [LISA VSE Guide](#).

For 4.x Users

You can create a **Virtual Web Service** in LISA using just a Web Service Definition Language (**WSDL**) file.

The virtual web service simulates the responses of a real web service. This is useful where a WSDL is available, but access to the web service is not available. LISA can create a virtual instance of the web service that can be used to develop Test Cases.

The LISA installation contains an **embedded Tomcat Server** that is used to host the virtual web service. Using just the WSDL, LISA constructs and deploys the virtual web service to this Server.

Creating the virtual web service requires the use of just two existing LISA steps:

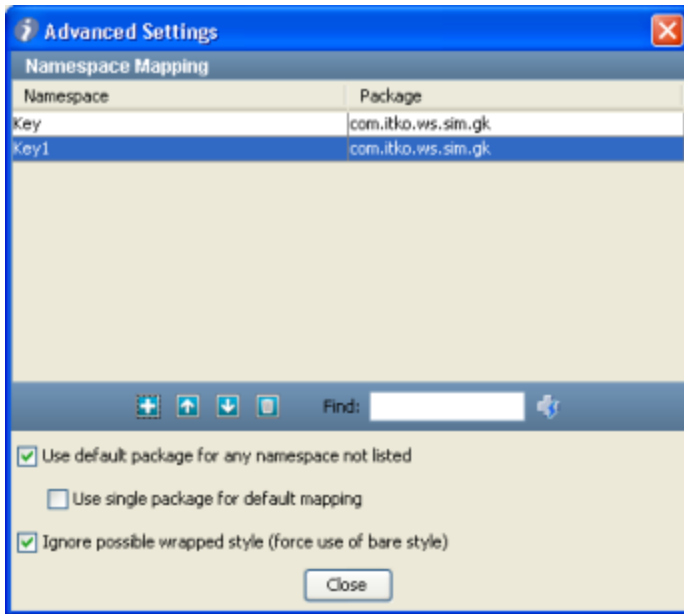
1. **Create a Virtual Web Service**
2. **Start or Stop Web Server**



- Click the **Add Step**
- Click on the **Custom Extensions > Create a Virtual Web Service**.

This opens the step editor as below:

- Enter the Web service details like the **WSDL URL** and the **Web Service Name**.
- Click the **Advanced** button in case you wish to enter advanced settings as below.
- Click the **Load** button to load the WSDL.



The Start or Stop Web Server step is configured in the Test Case before the **Create a Virtual Web Service** step.



- Click the **Add Step**
- Click on the **Web Services** step > **Start or Stop Web Server**.
This opens the step editor as below:

▼ Start or Stop Web Server - Step15

Start or Stop Web Server

Web Server: Tomcat 5.0.28

Server State

☒ Start Web Server ☐ Stop Web Server

Port Settings

HTTP Port Number: 9080

HTTPS Port Number: 9443

AJP Port Number: 9009

Proxy Port Number: 9082

JMX Port Number: 9085

Execute

Editor Results

- Click on the **Start Web Server** radio button.

By default the window opens in the Editor pane.

- Click **Execute** to execute this step.

Another **Stop Web Server** step is configured in the Test Case after the virtual web service step.

- Click on the Stop Web Server radio button and again click Execute to execute this step.

The responses from the virtual web service can be simple constants, or they can be calculated using a Server-side Test Case, where a step in the Test Case provides the response for one operation in the web service.

For more information on configuring the Create a Virtual Web Service and the Start or Stop Web Server steps see the [LISA Reference Guide](#).

PART 7 - LISA Advanced Features

PART 7 - LISA Advanced Features

The following chapters are available in this section.

35. Using BeanShell in LISA
36. Running LISA from the Command Line
37. Running LISA with Ant and JUnit
38. Class Loader Sandbox Example
39. In-Container Testing (ICT)

27. Using BeanShell in LISA

27. Using BeanShell in LISA

BeanShell (www.beanshell.org) is a free, open-source, lightweight Java scripting language. It is a Java application that uses the Reflection API to execute Java statements and expressions dynamically. By using BeanShell there is no need to compile class files.

BeanShell allows you to type standard Java syntax (statements and expressions) on a command line and see the results immediately. A Swing GUI is also available. BeanShell can also be called from within a Java class; this is how it is used by LISA.

LISA uses BeanShell in several places:

- To interpret property expressions (User Guide – Properties).
- As the interpreter framework for the 'Java Script Execution' step [LISA Reference Guide](#).
- As the interpreter framework for the 'Assert by Script Execution' Assertion [LISA Reference Guide](#).

For more information, see the [LISA Reference Guide](#).

The following topics are available in this Chapter.

[27.1 Using BeanShell Scripting Language](#)
[27.2 Using LISA Date Utilities](#)

27.1 Using BeanShell Scripting Language

27.1 Using BeanShell Scripting Language

The major different between BeanShell Java and compiled Java is in the type system. Java is very strongly typed whereas BeanShell has the capability to loosen the typing in its scripting environment. You can however impose strict typing in BeanShell if you so desire.

BeanShell relaxes typing in a natural way so that you can write BeanShell scripts that look like standard Java method code, while on the other hand you can write scripts that look more like a traditional scripting language, such as Perl or JavaScript, while still maintaining the framework of the Java syntax.

If a variable has been typed, then BeanShell will honor and check the type. If a variable is not typed, BeanShell will only signal an error if you attempt to misuse the actual type of the variable.

The following Java fragments are all valid in BeanShell:

```
foo = "Foo";
four = (2+2) * 2 / 2.0;
print(foo + " = " + four);
.
.
hash = new Hashtable();
date = new Date();
hash.put("today", date);
.
.
```

BeanShell allows you to declare and then use methods. Arguments and return types can also be loosely typed:

1. Typed

```
int addTwoNumbers(int a, int b){
    return a + b;
}
```

B) Loosely Typed

```
add(a,b){
    return a + b;
}
```

In case B) the following would work correctly:

```
sumI = add (5,7);
sumS = add("LISA ", "Rocks");
sumM = add ("version ", 2);
BeanShell also provides a library of commands that facilitate its use.
```

A few examples of these commands are:

- `source()` – Read a BeanShell (bsh) script.
- `run()` – Run a bsh script.
- `exec()` – Run a native application.
- `cd()`, `copy()` etc – Unix-like shell commands.
- `print()` – Print argument as a string.
- `eval()` – Evaluate string argument as code.

We have only scratched the surface of BeanShell here.

For more information see the BeanShell User Guide at www.beanshell.org. You can also get BeanShell, the source code, and the complete JavaDoc at the same place.

Using BeanShell as Standalone

BeanShell is available as a standalone interpreter should you want to try it outside of LISA. You can download BeanShell from www.beanshell.org. It is a single small jar file called **bsh-xx.jar** (xx is the version number; currently 2.0). Add the jar file to your classpath.

You can use BeanShell in the following configurations:

- **From a command line** – `java bsh.Interpreter [script name] [args]`
- **From BeanShell Gui** – `java bsh.Console`
- **From within a Java class** as below:

```
Import bsh.Interpreter;
.
.
Interpreter I = new Interpreter();
i.set ("x",5);
i.set("today", new Date());
Date d = (Date)i.get("date");
i.eval("myX = x * 10");
System.out.println(i.get("myX"));
.
```

Using BeanShell in LISA

LISA uses the BeanShell interpreter in both the **Java Script Execution** test step, and the **Assert by Script Execution** Assertion. Both of these elements also expose LISA Java objects and the current LISA state (properties). This provides a powerful environment for you to use to add custom functionality easily. The exposed Java objects can be used to both interrogate and modify the current state of the test. For example, you can read, modify and create LISA properties in your scripts.

As a starting point you should become familiar with the **TestExec** class in LISA. Information on TestExec and many other LISA classes can be found in the **LISA Extension Kit**. The Kit includes both a Developer Guide and Javadoc for the LISA classes that you have access to in these scripts.

LISA also uses BeanShell inside property notation when an equal sign is present, i.e. `= new Date()`. This property expression is interpreted using BeanShell. You can also set a new property using a property expression and BeanShell: `myDate=new Date()` would create a new LISA property called `myDate`.

27.2 Using LISA Date Utilities

27.2 Using LISA Date Utilities

LISA provides a number of date utility functions as static methods of the **com.itko.util.DateUtils** class. They all return the formatted date as a string. These functions can be used in parameter expressions or the Java Script Execution step.

```
com.itko.util.DateUtils.formatDate(Date date, String format)
com.itko.util.DateUtils.formatCurrentDate(String format)
com.itko.util.DateUtils.formatCurrentDate(int offsetInSec, String format)
com.itko.util.DateUtils.rfc3339(Date date)
com.itko.util.DateUtils.rfc3339()
com.itko.util.DateUtils.rfc3339(int offsetInSec)
```

```
com.itko.util.DateUtils.samlDate(Date date)
com.itko.util.DateUtils.samlDate()
com.itko.util.DateUtils.samlDate(int offsetInSec)
```

For example, if you have a web service call that takes a formatted date string, and the Server is two minutes slow, you can use
`=com.itko.util.DateUtils.formatCurrentDate(-120, "yyyy-MM-dd'T'HH:mm:ss.SSSZ")`

This will generate the string "2007-11-22T13:30:37.545-0500", the current time minus 120 seconds formatted according to these guidelines: RFC 3339 is slightly different from what the default java date formatter will generate. If you need a strict RFC 3339 date you can use the rfc3339 functions:

```
=com.itko.util.DateUtils.rfc3339()
```

This will generate the string "2007-11-22T13:30:37.545-05:00".

SAML Dates are formatted using the format "**yyyy-MM-dd'T'HH:mm:ss'Z'**". The saml Date functions are just helpers so you don't need to remember that format string when using the formatDate APIs.

For more information see: <http://java.sun.com/j2se/1.5.0/docs/api/java/text/SimpleDateFormat.html>

<http://tools.ietf.org/html/rfc3339#section-5.6>

28. Running LISA with Ant and JUnit

28. Running LISA with Ant and JUnit

You can execute LISA tests as **JUnit tests** within the execution path of an **Ant build**.

This provides real automated build and test integration opportunities, and allows you to leverage the flexibility of Ant and the simplicity of JUnit with the power of LISA to automate integration, load, stress, and production monitoring needs.

JUnit is a Java-based, open-source unit-testing framework widely supported by third party testing tools, including LISA. For more information on JUnit, see <http://www.junit.org>.

Ant is a Java-based, open-source automated software building tool extended with support for many third party tools, including JUnit. For more information on Ant, see <http://ant.apache.org/>

Additional topics available in this chapter are as follows...

- [37.1 Running LISA Tests as JUnit Tests](#)
- [37.2 Usage Examples](#)
- [37.3 Usage Examples II](#)
- [37.4 Integration with Cruise Control](#)

28.1 Running LISA Tests as JUnit Tests

28.1 Running LISA Tests as JUnit Tests

The JUnit Step will support JUnit3 and JUnit4 test cases and test suites.

To execute LISA tests as JUnit tests and have them report as native JUnit tests, complete the following steps:

1. Make sure both Ant and JUnit are available on your PC and ANT_HOME /bin is set in your PATH.
2. Copy junit.jar from your JUnit installation into ANT_HOME/lib.
3. Define the system property **LISA_HOME** and set its value to the LISA install directory.
4. Use the **junitlisatest** task to execute LISA tests as JUnit tests.
5. Optionally use the **junitlisareport** task to create HTML reports from the JUnit XML output.

Note: Logging output is written to a file 'junitlisa_log.log' in the LISA_HOME/tmp directory.

The logging level used is the same as that set in **LISA_HOME/logging.properties**.

To change the logging level,

Edit the first line of this file, from:

```
log4j.rootCategory=INFO,A1
```

to

```
log4j.rootCategory=DEBUG,A1
```

The Standard JUnit output is available at: `LISA_HOME/tmp/junit/index.html`

28.2 Usage Examples

28.2 Usage Examples

Example 1

The following Ant build script provides a complete example of a Test Suite of LISA tests:

This script is shipped with LISA Installation and is available at: `LISA_HOME/examples/build.xml`

```
<project name="LISA Example Ant/JUnit" default="lisaTests" basedir=".">
<property name="LISA_HOME" value=".." />
<property name="testReportDir" value="${LISA_HOME}/reports/junit" />
<taskdef resource="AntTasks.properties" classpath="${LISA_HOME}/bin/lisa-core.jar" />
<target name="lisaTests" description="Executes a LISA Test Suite as JUnit tests.">
<delete dir="${testReportDir}" />
<mkdir dir="${testReportDir}" />
<junitlisa suite="${basedir}/Suites/AllTestsSuite.ste" toDir="${testReportDir}" />
<junitlisareport toDir="${testReportDir}" />
<echo message="The JUnit report is available at ${testReportDir}/index.html" />
</target>
</project>
```

Example 2

```
<junitlisa test="MyTest.tst"
  config="dev"
  testRegistry="rmi://qaserver.mycomp.com/lisa.TestRegistry" ?toDir="${testReportDir}"
  haltOnError="no"
  errorProperty="test.failure">
  <jvmarg value="-DmySystemProp=someValue" />
</junitlisa>
```

The **junitlisa** task is a 'drop in' replacement for JUnit, but designed to execute LISA tests instead of JUnit tests. Most continuous build systems recognize the xml output files and will integrate the build dashboard with the test results.

You can also produce HTML reports with the **junitlisareport** task or the regular **junitreport** task.

The **junitlisa** is an Ant task that directly subclasses the regular JUnit task, so any of the attributes and nested elements that JUnit has, **junitlisa** also has. For more information, see <http://ant.apache.org/manual/OptionalTasks/junit.html>

Though there are several differences:

- We cannot set the 'fork' attribute to 'false' (we **always** fork)
- We cannot add nested <test> elements; use a test="testcase.tst" attribute instead
- We cannot add nested <batchtest> elements; use a suite="suiteFile.ste" attribute instead
- We add an implied classpath consisting of LISA_HOME/bin/.jar, LISA_HOME/lib/.jar, zip and LISA_HOME/lib/endorsed/.jar
- We add an implied java.endorsed.dir property pointing to LISA_HOME/lib/endorsed
- We add a default formatter of type "xml"
- We default the 'showoutput' attribute to 'true'
- We default the 'printsummary' attribute to 'true'
- We default the 'haltonfailure' attribute to 'true'
- We default the 'haltonerror' attribute to 'true'
- We default the 'maxmemory' attribute to '512m'

junitlisa supports the following attributes in addition to the attributes inherited from JUnit.

- suite="suiteName.ste" - a LISA suite. This is the most common use case
- test="filename.tst" - a single LISA test to run
- config="configName" which is either a named internal configuration set or a filename (.config or .properties file)
- testRegistry="registryName" e.g "rmi://testbox/lisa.TestRegistry" - a pointer to the registry you want to use if you want to stage the tests remotely. If you don't specify this the tests will be staged locally.
- All the attribute values can use curlybraces and LISA will resolve them in the usual fashion.
- You can specify either suite or test but not both.
- You **must** specify either a 'test' or 'suite' attribute. 'config' and 'testRegistry' attributes are optional.
- You will almost always want to specify the inherited 'toDir' attribute; although it will default to the current working directory **junitlisareport** is a subclass of the regular **junitreport** element except that we specify sensible defaults.

It is exactly equivalent to the following:

```
<junitreport todir="$\{testReportDir\}">
<fileset dir="<toDir specified in the junitreport tag>">
<include name="TEST-*.xml"/>
</fileset>
<report format="frames"
toDir="<toDir specified in the junitreport tag>">/>
</junitreport>
```

You can specify your own file set and report if you want. Because it's a direct subclass of **junitreport**, LISA also supports all the attributes and nested elements that **junitreport** does.

You will almost always want to specify the 'toDir' attribute of the **junitlisareport** element; however it will default to the current working directory.

28.3 Usage Examples II

28.3 Usage Examples - JUnit3 and JUnit4

JUnit 3

JUnit3 Test Case

For JUnit3 Test Cases, follow JUnit's conventions for writing your test case. In particular:

1. Your test class needs to extend junit.framework.TestCase
2. Your method names start with "test"

For example,

```
import junit.framework.TestCase;

public class JUnit3TestCase extends TestCase
{public void testOnesOne()

    Unknown macro: { assertEquals (1, 1);}

    {public void testTwosThree()

    Unknown macro: {assertEquals (2, 3); }

    }
```

JUnit3 Test Suite

For JUnit3 Test Suites, your suite is not required to extend the junit.framework.TestSuite, but it must implement the suite() method, with test cases wrapped by the JUnit4TestAdapter.

For example;

```
import junit.framework.JUnit4TestAdapter;
import junit.framework.TestSuite;

public class JUnit3VanillaTestSuite
{public static TestSuite suite()

Unknown macro: { TestSuite suite = new TestSuite(); suite.addTest ( new JUnit4TestAdapter ( MyJUnit3TestCase.class) ) ; return suite; }

}
```

JUnit 4

JUnit 4 Test Cases

For JUnit4 Test Cases, the test methods need to have the "@org.junit.Test" annotation on the methods, as already required by JUnit4.

For example:

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class JUnit4TestCase
{
    @Test
    public void oneIsOne()

Unknown macro: { assertEquals (1, 1); }

    @Test
    public void twoIsThree()

Unknown macro: { assertEquals (2, 3); }

}
```

JUnit4 Test Suites

To implement a JUnit4 Test Suite, add the @RunWith and @Suite.SuiteClasses annotations to flag the class as a test suite.

For example:

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses (

Unknown macro: { JUnit4TestCase.class }

)

public class JUnit4VanillaTestSuite

Unknown macro: { // empty }
```

Note - On the JUnit Step, when you click the "Load" button to load your JUnit test, if you see an "IllegalArgumentException", pl check if you have forgotten to add ".class" to the end of the classname. Simply check the spelling of the classname and make sure it ends with ".class" or use the classpath browser to locate the class.

28.4 Integration with Cruise Control

28.4 Integration with Cruise Control

If you are using **Cruise Control** to provide continuous integration, you can configure it to include test failures in its Control Panel.

In the JUnit Ant task shown in example 1, we defined \${testReportDir} to be 'LISA_HOME/tmp/junit'.

A typical Cruise config.xml file would then include something like the following:

```
<project name="myproj" buildafterfailed="false">
<log>
<merge dir="/home/cruise/build"/>
<merge dir="/path/to/lisajunit/output"/>
</log>
.
.
.
</project>
```

29. Class Loader Sandbox Example

29. Class Loader Sandbox Example

The following Java class is a simple example of a class that cannot be run in multi-threaded or multi-user fashion, because it accesses and modifies a static variable:

```
public class NeedsASandbox \{
static \{
System.out.println("This is my static initializer. You will see this many times.");
 
static String s;
 
public NeedsASandbox() \{\};
 
public void setS(String s)\{
this.s = s;
public String getS() \{
return s;
}
```

This class must run in a class loader sandbox.

Assume, for example, that you were to run this class within LISA and created a load test of ten users. If you do not use the class loader sandbox, you would see the `System.out.println` phrases only one time, and the value of `s` would be incorrect. This is because all users will be running within one class loader. This particular class will fail in those circumstances. Presuming that this is the proper function of the application, you'll need to use LISA's support for the class loader sandbox to make this work properly.

When you create the Class Loader Sandbox Companion, and LISA stages your ten users, you'll actually see the phrase in the code above ten times. LISA will construct ten separate class loaders and instantiate this class ten times; there will be ten separate instances of the class variable "static string s." This will allow your application logic, which is not thread safe, to be run in concurrent user tests.

The Class Loader Sandbox Companion is useful only if the following three conditions are present:

- You are testing a POJO with LISA.
- The POJO has static members.
- You are testing with multiple virtual users.

30. In-Container Testing (ICT)

30. In-Container Testing (ICT)

In-Container Testing (ICT) via remote proxies in LISA is now available in the Enterprise Java Execution and Dynamic Java Execution test nodes.

This feature allows in-container testing of local EJBs and arbitrary Java objects - any objects that are available in the container's classpath for the application being tested.

RMI and EJB are both supported as remote object protocols.

There are two connection modes used by in-container testing (ICT): EJB and RMI.

The following topics are available in this chapter.

- 39.1 Access via EJB (J2EE Container Environments)
- 39.2 Access via RMI (Custom Java Server and Application Environments)
- 39.3 Testing your ICT Installation from LISA Workstation

30.1 Access via EJB (J2EE Container Environments)

30.1 Access via EJB (J2EE Container Environments)

To test in-container objects in a standard J2EE container LISA uses a stateful session Enterprise JavaBean (EJB). This EJB is bundled as an exploded EAR directory named **lisa-remote-object-manager.ear** and a standalone jarfile **LISARemoteObjectManagerEJB.jar** with the LISA installer, in `LISA_HOME/incontainer/ejb`, where `LISA_HOME` is the directory where LISA is installed.

This EJB must be deployed along with your J2EE application in the J2EE container and be accessible via JNDI using the name "**LISARemoteObjectManagerEJB**". Deploying an EJB varies depending on the J2EE container being used, and vendor-provided documentation may help in the event a problem deploying the ICT EJB occurs.

If your J2EE application uses an isolated classloader, you must incorporate the ICT EJB into your application by modifying the XML deployment descriptors to include the ICT EJB and its dependencies.

a) JBoss

First test that you can successfully deploy the exploded ICT EAR in JBoss by copying the directory:

\$LISA/incontainer/ejb/lisa-remote-object-manager.ear to **\$JBoss/server/default/deploy** or other appropriate deployment directory.

JBoss should recognize the new EAR and deploy it without any errors. Check that you can connect to the EJB from LISA Workstation by following the steps in section 3 below.

Once the EAR is successfully deployed, in a standalone configuration, then attempt to integrate the ICT EJB with your J2EE application. This may be as simple as copying the contents of **\$LISA/incontainer/ejb/lisa-remote-object-manager.ear** and including them in your existing application EAR. Or create a new EAR that includes your J2EE application combined with these files.

See the application XML deployment descriptor **application.xml** for an idea of how to modify your own application deployment descriptors to include ICT and its dependencies.

Note: The use of the `<module>` XML elements to indicate the presence of the ICT EJB and Java jarfile dependencies.

b) WebLogic

The first test on WebLogic is to deploy the exploded ICT EAR in WebLogic, for example by using the WebLogic Administrator Console GUI. If your WebLogic Server is in development mode, you can also copy the directory **\$LISA/incontainer/ejb/lisa-remote-object-manager.ear** to your Server's **autodeploy** directory and WebLogic will automatically deploy the EAR. You should observe that the EAR is deployed without any errors. Check that you can connect to the EJB from LISA Workstation by following the steps in section 3 below.

Once the EAR is successfully deployed in a standalone configuration, then attempt to integrate the ICT EJB with your J2EE application. This may be as simple as copying the contents of **\$LISA/incontainer/ejb/lisa-remote-object-manager.ear** and including them in your own application EAR. Or create a new EAR that includes your J2EE application combined with these files.

30.2 Access via RMI (Custom Java Server and Application Environments)

30.2 Access via RMI (Custom Java Server and Application Environments)

For custom Java applications, you can modify your application's source code to integrate with ICT. ICT testing can be performed by binding **the LISARemoteObjectManagerRMIServer** remote Server object to the RMI registry. This object will accept connections from LISA Workstation to perform ICT.

For example, the following code can be included in your custom application in order to bind the ICT remote object Server via RMI:

```
LISARemoteObjectManagerRMIServer remoteObjectManagerServer = new LISARemoteObjectManagerRMIServer();Registry registry = LocateRegistry.createRegistry(port);registry.bind("LISARemoteObjectManager", remoteObjectManagerServer);
```

Note: The RMI name "**LISARemoteObjectManager**" must be entered exactly as-is in order for ICT to work correctly.

Workstation will attempt to connect to your application via the RMI URL **rmi://<hostname>:<port>/LISARemoteObjectManager**.

The hostname and port are variables and can be changed in your test application and configured in Workstation.

An example Server application can be found in `$LISA/incontainer/rmi/example`. Using a console window, you can run the example Server by

executing the command "java -jar ExampleServer.jar".

Note: For this command to run successfully, you must be running it from the \$LISA/incontainer/rmi/example directory. Source code has been provided for your reference in the \$LISA/incontainer/rmi/example/src directory.

30.3 Testing your ICT Installation from LISA Workstation

30.3 Testing your ICT Installation from LISA Workstation

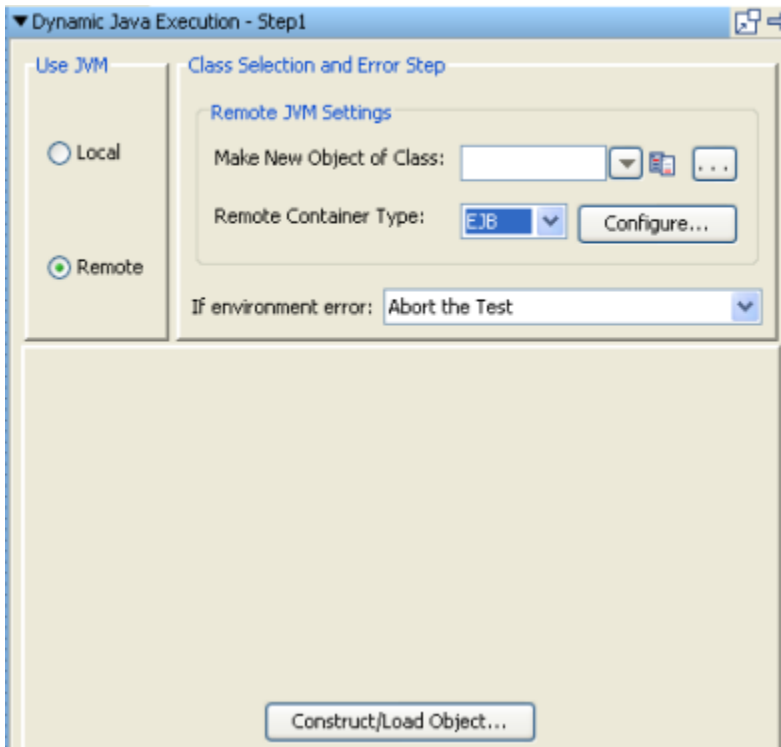
Once the Server-side portion of ICT is up and running, you should test that you can connect from LISA Workstation.

To accomplish this, open LISA Workstation and create a new Test Case called "ICT Connection Test". Inside of this Test Case, create a new **Dynamic Java Execution (DynExec)** Test Step.

In the **DynExec** editor for the new test step, select the **"Remote"** radio button.

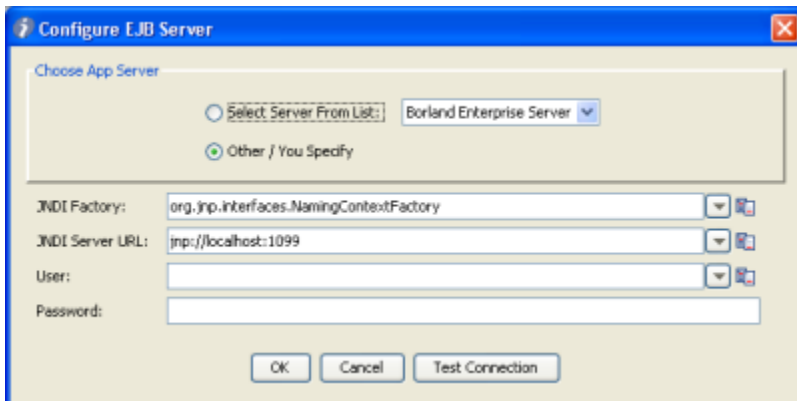
Make sure the **RemoteContainer Type** combo box displays your correct connection protocol, either EJB or RMI.

If you select Remote Container type as EJB -



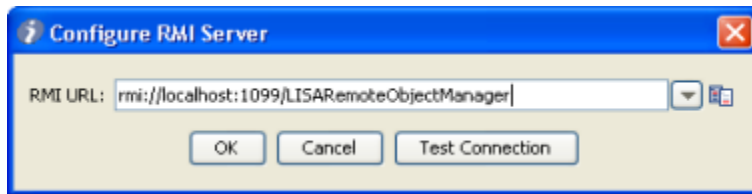
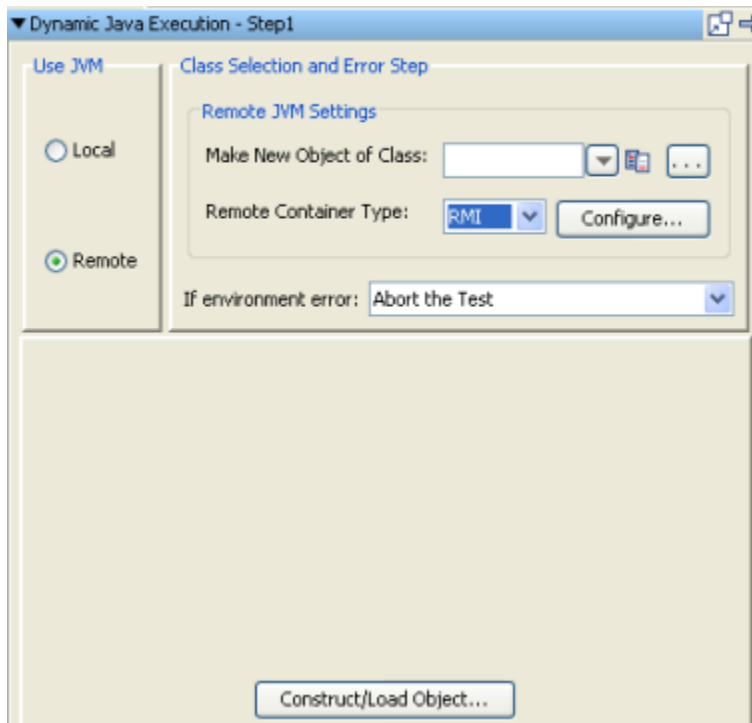
Then click the **"Configure..."** button to enter the configuration settings for your protocol.

For example, if you are using EJB as the protocol, fill in the Configure EJB Server dialog with host name or IP address and port number of the J2EE container running ICT.



If you select Remote Container type as RMI -

If you're using RMI as the connection protocol, here is an example of the Configure RMI Server dialog that you can use for RMI settings.



Once you've successfully tested the connection to the ICT Server, enter **java.util.Date** in the Make New Object of Class text field and click the **Construct/Load Object...** button to create a new in-container instance of the java.util.Date class.

If you've made it this far, your installation is complete and you're ready to use ICT!

Dependencies

The ICT Server-side classes depend on the following third-party libraries:

- BeanShell 2.0b4 (bsh-2.0b4-lisa-remote-object-manager.jar).

NOTE: The original jarfile has been modified so that.bsh scripts are removed. These scripts are known to cause problems with J2EE containers that inspect jarfiles and automatically execute them.

- XStream 1.1.2 (xstream-1.1.2.jar)
- Log4j 1.2.13 (log4j-1.2.13.jar)
- Jakarta Commons Logging 1.0.2 (commons-logging.jar)

These dependencies are intentionally distributed as separate files along with LISA ICT jarfiles to make ICT integration easier. Because the application you're testing may already include some or all of these libraries in its classpath, adding the above dependencies to the classpath may not be necessary.

31. LISA Command Line Utilities

31. LISA Command Line Utilities

Test Runner

TestRunner is a 'headless' version of LISA Workstation with the same functionality but no user interface.

For more information refer to [TestRunner](#).

CVSManager

CVS Manager is an application within LISA, that has the ability to add or remove Monitors to CVS Dashboard through a command line option [TestRunner](#) .

For more information, refer to [CVSManager](#) .

VSE Manager

This command line tool is used for managing virtual service environments.

For more information, refer to [VSE Command line Options](#).

Service Manager

This command line tool is used for managing service images.

For more information, refer to [VSE Command line Options](#) .

31.1 Test Runner

31.1 Test Runner: Running LISA from the Command Line

You can run LISA silently without the Workstation UI, on command line.

TestRunner is a 'headless' version of LISA Workstation with the same functionality but no user interface i.e it can be run as a stand-alone application.

Test Runner allows you to run tests as batch applications. You give up the opportunity to monitor tests in real time, but you still have the capability to request reports for later viewing.

On Windows platforms, Test Runner is available in the LISA bin directory as a Windows executable, **TestRunner.exe**.

On Unix platforms, Test Runner is available as a Unix executable, TestRunner, and a Unix script, **TestRunner.sh**.

For further details on implementation, see *LISA for Java Developers*.

This allows you to incorporate LISA tests into a continuous build workflow, or use it with JUnit to run standard JUnit tests in Ant or some other build tool.

Test Runner can be found in the LISA bin directory.

- On Windows you run it as an executable (TestRunner.exe)
- On Unix you run a Unix executable (TestRunner).

You can use the following options while using the TestRunner:

```
TestRunner [-h] [[-r StagingDocument] [-t TestCaseDocument] [-cs CoordinatorServerName]] [-s TestSuiteDocument] [-m TestRegistryName] [-a]
```

- Run **TestRunner.exe** on command line to get all the help related to TestRunner.

Tip: As a Java class you can run it as follows:

```
java com.itko.lisa.coordinator.TestRunner [-h] [[-r StagingDocument] [-t TestCaseDocument] [-cs CoordinatorServerName]] [[-s TestSuiteDocument] [-m TestRegistryName] [-a]
```

As Part of an Automated Build

LISA Test Cases can be incorporated into an automatic build and test process. LISA provides the additional software required to use Java ANT, and Java JUnit, and an example ANT build script.

LISA Test Cases will be run and reported as native JUnit tests.

For further details on implementation, see [LISA Agent guide]../display/DOC/LISA+Agent+Guide]

Launching Multiple Instances

You are able to stage multiple instances of TestRunner from a single workstation to a LISA Server.

Following things are needed -

- You will need 512 MB for each instance
- Your license will need to be able to support launching multiple instances of TestRunner.

Some additional arguments are given in next section to specify other needed documents:

36.1 Single Test
36.2 Test Suite
36.3 Other Options
36.4 Usage Examples

31.1.1 Runing Single Test Case

31.1.1 Running a Single Test Case

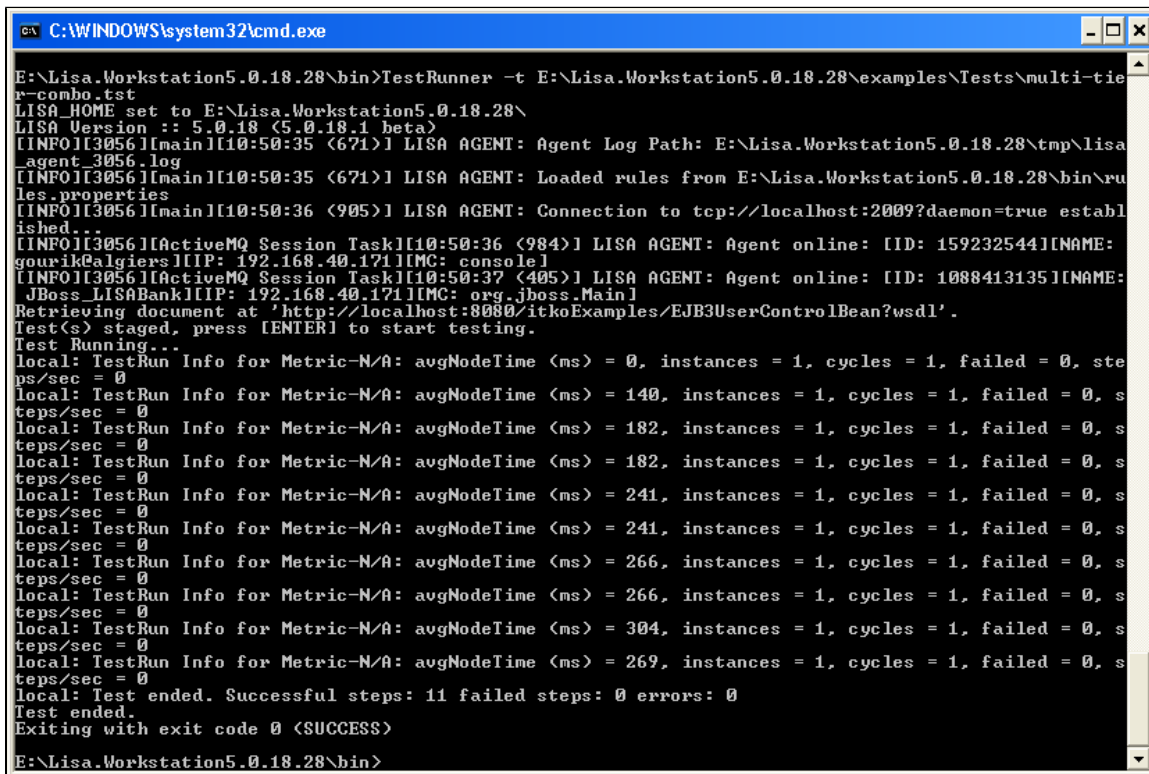
To run a single Test Case within the TestRunner*,* you can specify the following options:

- **-t** name of the Test Case document
- **-r** name of the staging document
- **-cs** name of the Coordinator Server to stage the test
- **-m** name of the LISA Registry to use.

To stage remotely, provide **-m** and **--cs**.

To stage locally, do not provide **-m** and **--cs**.

For the purpose of illustration, we have run the **multi-tier-combo** test case, as shown below:



```
C:\WINDOWS\system32\cmd.exe

E:\Lisa.Workstation5.0.18.28\bin>TestRunner -t E:\Lisa.Workstation5.0.18.28\examples\Tests\multi-tier-combo.tst
LISA_HOME set to E:\Lisa.Workstation5.0.18.28\
LISA Version :: 5.0.18 <5.0.18.1 beta>
[INFO][3056][main][10:50:35 <671>] LISA AGENT: Agent Log Path: E:\Lisa.Workstation5.0.18.28\tmp\lisa_agent_3056.log
[INFO][3056][main][10:50:35 <671>] LISA AGENT: Loaded rules from E:\Lisa.Workstation5.0.18.28\bin\rules.properties
[INFO][3056][main][10:50:36 <905>] LISA AGENT: Connection to tcp://localhost:2009?daemon=true established...
[INFO][3056][ActiveMQ Session Task][10:50:36 <984>] LISA AGENT: Agent online: [ID: 1592325441][NAME: gourik@algiars][IP: 192.168.40.171][MC: console]
[INFO][3056][ActiveMQ Session Task][10:50:37 <405>] LISA AGENT: Agent online: [ID: 10884131351][NAME: JBoss.LISABank][IP: 192.168.40.171][MC: org.jboss.Main]
Retrieving document at 'http://localhost:8080/itkoExamples/EJB3UserControlBean?wsdl'.
Test(s) staged, press [ENTER] to start testing.
Test Running...
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 0, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 140, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 182, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 182, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 241, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 241, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 266, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 266, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 304, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 269, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: Test ended. Successful steps: 11 failed steps: 0 errors: 0
Test ended.
Exiting with exit code 0 <SUCCESS>

E:\Lisa.Workstation5.0.18.28\bin>
```

31.1.2 Running a Test Suite

31.1.2 Running a Test Suite

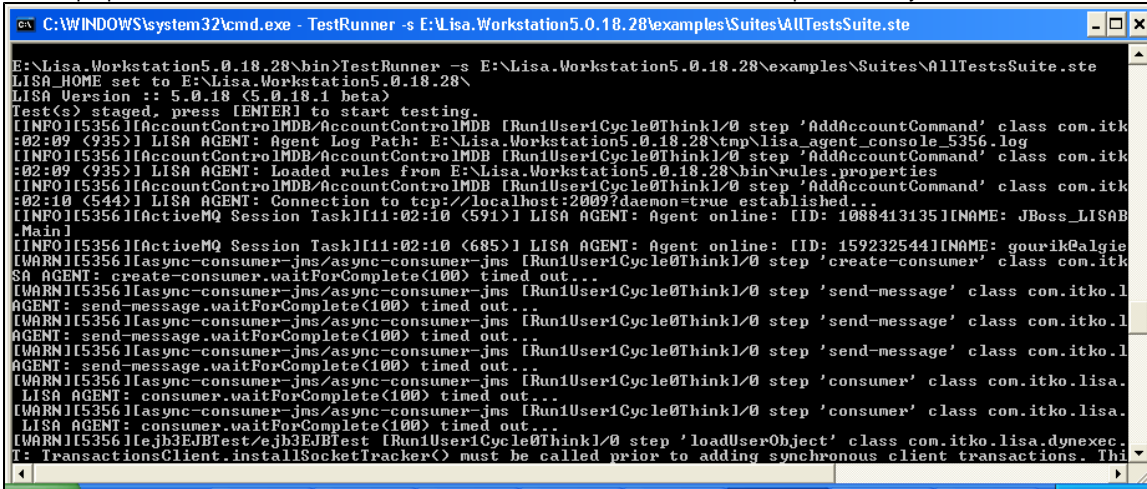
To stage/run a test suite,

- -s name of the test suite document.
- -m name of the LISA Registry to use.

Note - No Auditing is performed.

To stage locally, do not provide --m.

For the purpose of illustration, we have run "AllTestsSuite.ste" suite from the examples directory as shown below:



```
C:\WINDOWS\system32\cmd.exe - TestRunner -s E:\Lisa.Workstation5.0.18.28\examples\Suites\AllTestsSuite.ste

E:\Lisa.Workstation5.0.18.28\bin>TestRunner -s E:\Lisa.Workstation5.0.18.28\examples\Suites\AllTestsSuite.ste
LISA_HOME set to E:\Lisa.Workstation5.0.18.28\
LISA Version :: 5.0.18 (5.0.18.1 beta)
Test(s) staged, press [ENTER] to start testing.
[INFO][5356][AccountControlMDB/AccountControlMDB [Run1User1Cycle0Think1/0 step 'AddAccountCommand' class com.itk
:02:09 (935)] LISA AGENT: Agent Log Path: E:\Lisa.Workstation5.0.18.28\temp\lisa_agent_console_5356.log
[INFO][5356][AccountControlMDB/AccountControlMDB [Run1User1Cycle0Think1/0 step 'AddAccountCommand' class com.itk
:02:09 (935)] LISA AGENT: Loaded rules from E:\Lisa.Workstation5.0.18.28\bin\rules.properties
[INFO][5356][AccountControlMDB/AccountControlMDB [Run1User1Cycle0Think1/0 step 'AddAccountCommand' class com.itk
:02:10 (544)] LISA AGENT: Connection to tcp://localhost:2009?daemon=true established...
[INFO][5356][ActiveMQ Session Task][11:02:10 (591)] LISA AGENT: Agent online: [ID: 1088413135][NAME: JBoss_LISAB
.Main]
[INFO][5356][ActiveMQ Session Task][11:02:10 (685)] LISA AGENT: Agent online: [ID: 1592325441][NAME: gourik@algie
[WARN][5356][async-consumer-jms/async-consumer-jms [Run1User1Cycle0Think1/0 step 'create-consumer' class com.itk
SA AGENT: create-consumer.waitForComplete(100) timed out...
[WARN][5356][async-consumer-jms/async-consumer-jms [Run1User1Cycle0Think1/0 step 'send-message' class com.itko.l
AGENT: send-message.waitForComplete(100) timed out...
[WARN][5356][async-consumer-jms/async-consumer-jms [Run1User1Cycle0Think1/0 step 'send-message' class com.itko.l
AGENT: send-message.waitForComplete(100) timed out...
[WARN][5356][async-consumer-jms/async-consumer-jms [Run1User1Cycle0Think1/0 step 'send-message' class com.itko.l
AGENT: send-message.waitForComplete(100) timed out...
[WARN][5356][async-consumer-jms/async-consumer-jms [Run1User1Cycle0Think1/0 step 'consumer' class com.itko.lisa.
LISA AGENT: consumer.waitForComplete(100) timed out...
[WARN][5356][async-consumer-jms/async-consumer-jms [Run1User1Cycle0Think1/0 step 'consumer' class com.itko.lisa.
LISA AGENT: consumer.waitForComplete(100) timed out...
[WARN][5356][ejb3EJBTest/ejb3EJBTest [Run1User1Cycle0Think1/0 step 'loadUserObject' class com.itko.lisa.dynexec.
T: TransactionsClient.installSocketTracker() must be called prior to adding synchronous client transactions. Thi
```

31.1.3 Other Options

31.1.3 Other Options

Other command line arguments are:

- h displays help
- a auto starts the test (so you don't have to hit [ENTER] after the test has been staged).
- config Name of an alternate configuration document to be used.

Once the test(s) have been completed you can view your reports in the standard way using LISA Workstation.

Note: You can use LISA properties when specifying file names. However in this context only system properties and properties defined in your property files (lisa.properties, local.properties and site.properties) will work.

To Add Monitors in CVS Dashboard

Test Runner can also be used for adding or removing a monitor in the [CVS Dashboard](#).

Following options are available:

```

C:\Lisa.Workstation5.0.20.13AlphaRel\bin\CVSManager.exe
LISA_HOME set to C:\Lisa.Workstation5.0.20.13AlphaRel\
LISA Version :: 5.0.20 <5.0.20.45>

Please provide all the mandatory arguments. See below for the Usage help.
USAGE:

To stage a monitor to the Continuous Validation Service, use
  CUSManager [options] [addSpec|deleteJobSpec|deleteAllSpec] ...
      addSpec      : -a                | --add
      deleteJobSpec : -d                | --delete
      deleteAllSpec : -da serviceNameSpec | --deleteAll serviceNameSpec

  registrySpec    : -m RegistryName    | --registry RegistryName
  (will attempt to acquire default)
  jobNameSpec     : -j jobName          | --jobName jobName
  serviceNameSpec : -sn serviceName    | --serviceName serviceName
  fromSpec        : -from date in format MM/dd/yy HH:mm:ss
  toSpec          : -to date in format MM/dd/yy HH:mm:ss
  intervalSpec    : -i intervalInMinutes <default is 5>
  testCaseSpec    : -t testCaseName    | --testCase testCaseName
  stagingDocSpec   : -r stagingDocName   | --stagingDoc stagingDocName

  suiteDocSpec    : -s suiteDocName     | --suiteDoc suiteDocName
  -config configurationNameOrFileName | --configFile configurationNameOrFileName

  -cs CoordinatorServerName           | --coordinatorService CoordinatorServerName

  prioritySpec    : -p 1-5 priority <1 is highest, default is 3>
  emailSpec       : -e emailAddress

  To print this usage message
  -h | --help

Examples:
Stage a Monitor to CVS:
CUSManager -a -sn OrderManager -j CheckOrderToCash -t ../Tests/orderchecker.tst
CUSManager -a -sn OrderManager -j CheckOrdersE2E -s ../Suites/orderend2end.ste

Delete that same monitor:
CUSManager -d -j CheckOrders
CUSManager -da -sn OrderManager

```

To Generate an HTML report

To have the TestRunner produce an HTML report,

- Use the **-html** or **--htmlReport** flags.

The **-html** output flag will write a html summary report.

The parameter after this flag needs to be a fully-qualified filename.

For example:

```
TestRunner -html /some/directory/MyReport.html
```

To Set the Configuration

To define which configuration to use for a test run,

- Use the **-config** or **--configFile** option.

Note - TestRunner does not understand LISA projects, so the fully-qualified path to the configuration file must be given.

For example:

```
TestRunner -config /path/to/lisa/home/examples/Configs/examples.itko.com.config
```

Logging Output

Logging output is written to a file `trunner_log.log` in the `LISA_HOME/tmp` directory. The logging level used is the same as that set in `LISA_HOME/logging.properties`.

To change the logging level edit the first line of this file from:

```
log4j.rootCategory=INFO,A1
```

to

```
log4j.rootCategory=DEBUG,A1
```

Test Runner can be used in standard JUnit tests using a custom LISA Java class.

For more details see the next section [Running LISA Tests with ANT and JUnit].

`-html` output flag will write a test summary report to disk.

Primary Development Assignee:	[Rich Denis]../jira/secure/ViewProfile.jspa?name=rdenis]
Primary QE Assignee:	[Jon Caulfield]../jira/secure/ViewProfile.jspa?name=jcaulfield]

Date Closed:	02/Jun/10
Code Review?:	Bypassed
How to Test?:	someone needs to make an enhancement to integrate that with the new 5.0 reporting...see CORE-
QA Return to Dev?:	Yes - Not QA Ready
Added Test Plan to TMT?:	No

Mgmt Only: Release Plan:	5.0.0 Post GA Maintenance
Track with GA Version:	5.0.10
Fix Build #:	5.0.10.15

Doc case created / linked?:	No
Release Note Needed?:	Done
Rank - Development:	N/A
Rank - Account:	N/A

31.1.4 Usage Examples

31.1.4 Usage Examples

Test Case

Running a single test case "**multi-tier-combo.tst**", in a examples directory called "Examples\Tests" using a Staging document "Stage1.stg" at your project location:

Note - `PROJECT_HOME` must be defined in `local.properties`.

```
bin\TestRunner -a --t PROJECT_HOME/examples/tests/multi-tier-combo.tst --r PROJECT_HOME/tests/stage1.stg
```

All these tests are Running from LISA_HOME

Or running simple test case as shown below:*

```
C:\WINDOWS\system32\cmd.exe
E:\Lisa.Workstation5.0.18.28\bin>TestRunner -t E:\Lisa.Workstation5.0.18.28\examples\Tests\multi-tier-combo.tst
LISA_HOME set to E:\Lisa.Workstation5.0.18.28\
LISA Version :: 5.0.18 (5.0.18.1 beta)
[INFO][13056][main][10:50:35 (671)] LISA AGENT: Agent Log Path: E:\Lisa.Workstation5.0.18.28\temp\lisa_agent_3056.log
[INFO][13056][main][10:50:35 (671)] LISA AGENT: Loaded rules from E:\Lisa.Workstation5.0.18.28\bin\rules.properties
[INFO][13056][main][10:50:36 (905)] LISA AGENT: Connection to tcp://localhost:2009?daemon=true established...
[INFO][13056][ActiveMQ Session Task][10:50:36 (984)] LISA AGENT: Agent online: [ID: 159232544][NAME: gourik@algiers][IP: 192.168.40.171][MC: console]
[INFO][13056][ActiveMQ Session Task][10:50:37 (405)] LISA AGENT: Agent online: [ID: 1088413135][NAME: JBoss_LISABank][IP: 192.168.40.171][MC: org.jboss.Main]
Retrieving document at 'http://localhost:8080/itkoExamples/EJB3UserControlBean?wsdl'.
Test(s) staged, press [ENTER] to start testing.
Test Running...
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 0, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 140, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 182, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 182, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 241, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 241, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 266, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 266, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 304, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: TestRun Info for Metric-N/A: avgNodeTime (ms) = 269, instances = 1, cycles = 1, failed = 0, steps/sec = 0
local: Test ended. Successful steps: 11 failed steps: 0 errors: 0
Test ended.
Exiting with exit code 0 (SUCCESS)
E:\Lisa.Workstation5.0.18.28\bin>
```

Test Suite

Running a Test Suite "AllTestsSuite", in the LISA "examples\suite" directory with a configuration "examples.itko.com"

```
bin\TestRunner -a -s examples\suite\AllTestsSuite.ste -config examples.itko.com
```

or running simply the Test Suite "AllTestsSuite" as shown below:

```
C:\WINDOWS\system32\cmd.exe - TestRunner -s E:\Lisa.Workstation5.0.18.28\examples\Suites\AllTestsSuite.ste
E:\Lisa.Workstation5.0.18.28\bin>TestRunner -s E:\Lisa.Workstation5.0.18.28\examples\Suites\AllTestsSuite.ste
LISA_HOME set to E:\Lisa.Workstation5.0.18.28\
LISA Version :: 5.0.18 (5.0.18.1 beta)
Test(s) staged, press [ENTER] to start testing.
[INFO][15356][AccountControlIMDB/AccountControlIMDB [Run1User1Cycle0Think1/0 step 'AddAccountCommand' class com.itk
:02:09 (935)] LISA AGENT: Agent Log Path: E:\Lisa.Workstation5.0.18.28\temp\lisa_agent_console_5356.log
[INFO][15356][AccountControlIMDB/AccountControlIMDB [Run1User1Cycle0Think1/0 step 'AddAccountCommand' class com.itk
:02:09 (935)] LISA AGENT: Loaded rules from E:\Lisa.Workstation5.0.18.28\bin\rules.properties
[INFO][15356][AccountControlIMDB/AccountControlIMDB [Run1User1Cycle0Think1/0 step 'AddAccountCommand' class com.itk
:02:10 (544)] LISA AGENT: Connection to tcp://localhost:2009?daemon=true established...
[INFO][15356][ActiveMQ Session Task][11:02:10 (591)] LISA AGENT: Agent online: [ID: 1088413135][NAME: JBoss_LISAB
.Main]
[INFO][15356][ActiveMQ Session Task][11:02:10 (685)] LISA AGENT: Agent online: [ID: 159232544][NAME: gourik@algie
[WARN][15356][lasync-consumer-jms/async-consumer-jms [Run1User1Cycle0Think1/0 step 'create-consumer' class com.itk
SA AGENT: create-consumer.waitForComplete(100) timed out...
[WARN][15356][lasync-consumer-jms/async-consumer-jms [Run1User1Cycle0Think1/0 step 'send-message' class com.itko.l
AGENT: send-message.waitForComplete(100) timed out...
[WARN][15356][lasync-consumer-jms/async-consumer-jms [Run1User1Cycle0Think1/0 step 'send-message' class com.itko.l
AGENT: send-message.waitForComplete(100) timed out...
[WARN][15356][lasync-consumer-jms/async-consumer-jms [Run1User1Cycle0Think1/0 step 'send-message' class com.itko.l
AGENT: send-message.waitForComplete(100) timed out...
[WARN][15356][lasync-consumer-jms/async-consumer-jms [Run1User1Cycle0Think1/0 step 'consumer' class com.itko.lisa.
LISA AGENT: consumer.waitForComplete(100) timed out...
[WARN][15356][lasync-consumer-jms/async-consumer-jms [Run1User1Cycle0Think1/0 step 'consumer' class com.itko.lisa.
LISA AGENT: consumer.waitForComplete(100) timed out...
[WARN][15356][lasync-consumer-jms/async-consumer-jms [Run1User1Cycle0Think1/0 step 'loadUserObject' class com.itko.lisa.dynexec.
T: TransactionsClient.installSocketTracker() must be called prior to adding synchronous client transactions. Thi
```

31.1.5 TestRunner Instances

31.1.5 TestRunner Instances

You are able to stage multiple instances of TestRunner from a single workstation, to a LISA Server.

Following things are mandatory to do so,

- You will need 512 MB for each instance
- Your license will need to be able to support launching multiple instances of TestRunner.

For more information, contact ITKO support.

31.2 CVS Manager

31.2 CVS Manager

The CVS Manager is an application within LISA, that has the ability to add or remove Monitors to CVS Dashboard through a command line option, i.e a Testrunner.

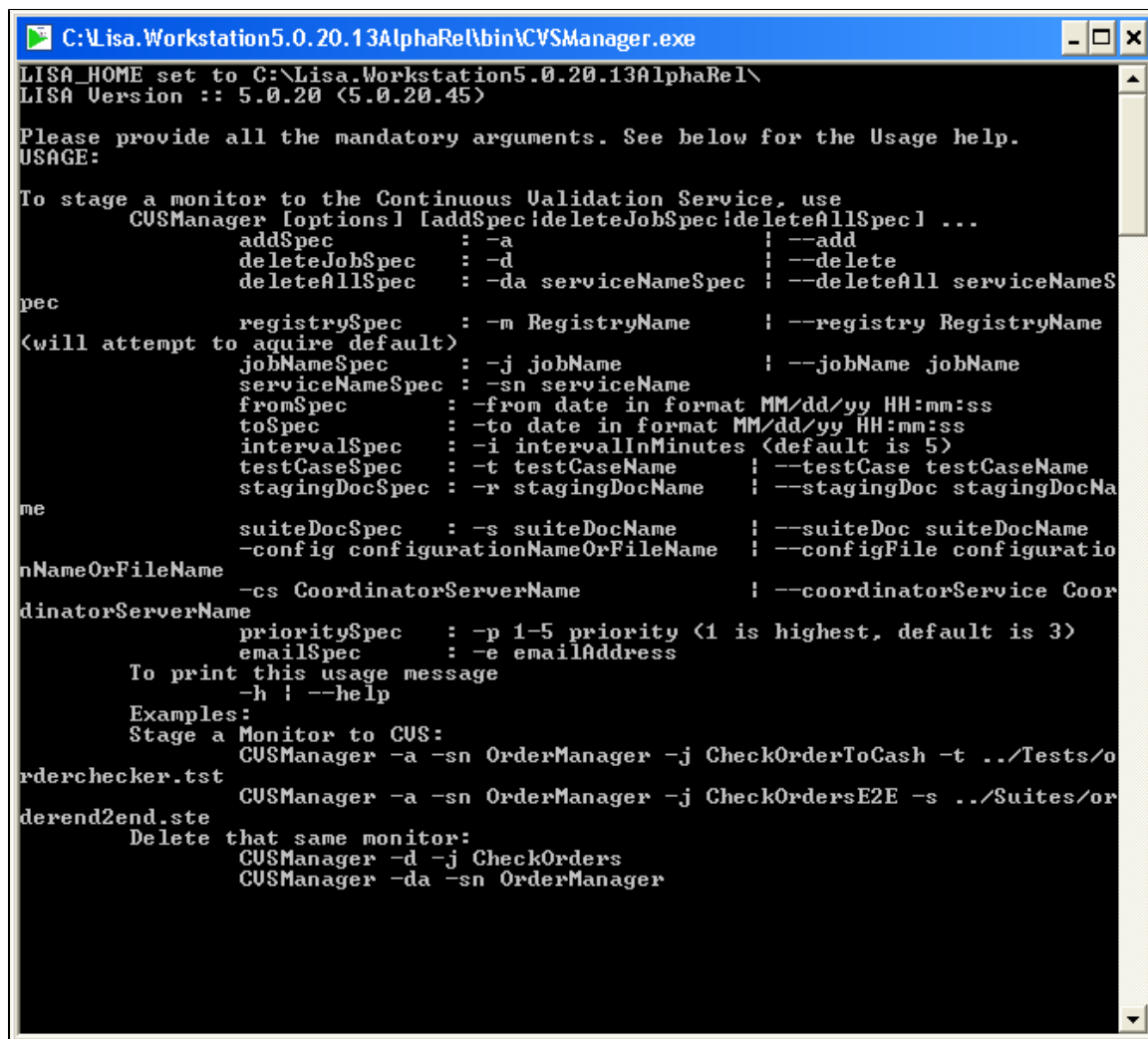
A Testrunner is the command line application for LISA Workstation.

The CVSManager.exe is located in the %LISA_HOME%/bin directory.

To view the CVS Manager,

- Run/execute the **CVSManager.exe** for a usage that includes how to perform this function.

The command line gives out the following usage:



```
C:\Lisa.Workstation5.0.20.13AlphaRel\bin\CVSManager.exe
LISA_HOME set to C:\Lisa.Workstation5.0.20.13AlphaRel\
LISA Version :: 5.0.20 <5.0.20.45>

Please provide all the mandatory arguments. See below for the Usage help.
USAGE:

To stage a monitor to the Continuous Validation Service, use
CUSManager [options] [addSpec deleteJobSpec deleteAllSpec] ...
    addSpec      : -a serviceNameSpec      ! --add serviceNameSpec
    deleteJobSpec : -d jobNameSpec          ! --delete jobNameSpec
    deleteAllSpec : -da serviceNameSpec     ! --deleteAll serviceNameSpec

    registrySpec : -m RegistryName         ! --registry RegistryName
    jobNameSpec  : -j jobName               ! --jobName jobName
    serviceNameSpec : -sn serviceName       ! --serviceName serviceName
    fromSpec     : -from date in format MM/dd/yy HH:mm:ss
    toSpec       : -to date in format MM/dd/yy HH:mm:ss
    intervalSpec : -i intervalInMinutes (default is 5)
    testCaseSpec : -t testCaseName        ! --testCase testCaseName
    stagingDocSpec : -r stagingDocName      ! --stagingDoc stagingDocName

    suiteDocSpec : -s suiteDocName         ! --suiteDoc suiteDocName
    configNameOrFileName : -config configurationNameOrFileName ! --configFile configurationNameOrFileName
    coordinatorServerName : -cs CoordinatorServerName          ! --coordinatorService CoordinatorServerName
    prioritySpec   : -p 1-5 priority (1 is highest, default is 3)
    emailSpec      : -e emailAddress

To print this usage message
-h ! --help

Examples:
Stage a Monitor to CVS:
CUSManager -a -sn OrderManager -j CheckOrderToCash -t ../Tests/orderchecker.tst
CUSManager -a -sn OrderManager -j CheckOrdersE2E -s ../Suites/orderend2end.ste

Delete that same monitor:
CUSManager -d -j CheckOrders
CUSManager -da -sn OrderManager
```

LISA VSE Guide

LISA VSE Guide

The VSE online documentation is broken out into the following chapters.

1. VSE Introduction
2. VSE Installation and Configuration
3. Understanding LISA Virtualize
4. Understanding Transactions
5. Recording a Service Image
6. Editing a Service Image
7. Editing a VSM
8. Doing the Virtualization

1. VSE Introduction

1. VSE Introduction

The word Virtualization usually refers to hardware virtualization, wherein the behavior of a physical asset – such as a server or application in a software emulator – is simulated and the emulator is hosted in a virtual environment. The virtual environment provides the same communication with the emulated asset as the physical environment.

Virtualization allows you to:

- Better manage physical assets, which eases change and configuration management
- Improve utilization of physical capacity, which better leverages the capacity of the physical assets
- Improve agility, which avoids the costly delays caused by waiting for IT to reconfigure or switch servers

LISA Virtualize provides Service virtualization. The process is in principle the same as that for hardware virtualization.

The following topics are available.

- 1.1 LISA Workstation Main Menu for VSE
- 1.2 LISA workstation Toolbar for VSE
- 1.3 Service Virtualization
- 1.4 High-level Virtualization Steps
- 1.5 Benefits of Virtualization

1.1 LISA Workstation Main Menu for VSE

1.1 LISA Workstation Main Menu for VSE

The LISA VSE main menu has items for all the major functions available in the LISA Workstation.

These are the global functions. There are many more menus, usually drop down menus and toolbars, throughout the Workstation in the individual editors.

- 1.1.1 File menu
- 1.1.2 Edit menu
- 1.1.3 View menu
- 1.1.4 System menu
- 1.1.5 Actions menu
- 1.1.6 Help menu

1.1.1 File Menu

1.1.1 File Menu

The File main menu has the standard items related to file manipulation.

For more information refer to File Menu in User Guide [2.2.1 File Menu](#)

1.1.2 Edit Menu

1.1.2 Edit Menu

The Edit menu has the normal set of editing options.

For more information refer to Edit Menu in User Guide [2.2.2 Edit Menu](#)

1.1.3 View Menu

1.1.3 View Menu

The View menu contains a list of menus which deal with the viewing of the Reporting portal, CVS Dashboard, VSE Dashboard, Pathfinder Console etc.

You can set the look and feel of LISA Workstation here and also visit the LISA portal for more information related to LISA.

For more information refer to View Menu in User Guide [2.2.3 View Menu](#)

1.1.4 System Menu

1.1.4 System Menu

The System menu has a menu to manage the system registries and system messages.

You can also edit the LISA properties, add JAR or zip files to Hot Deploy or reset Hot Deploy Class Loader and set the Tools.jar file here.

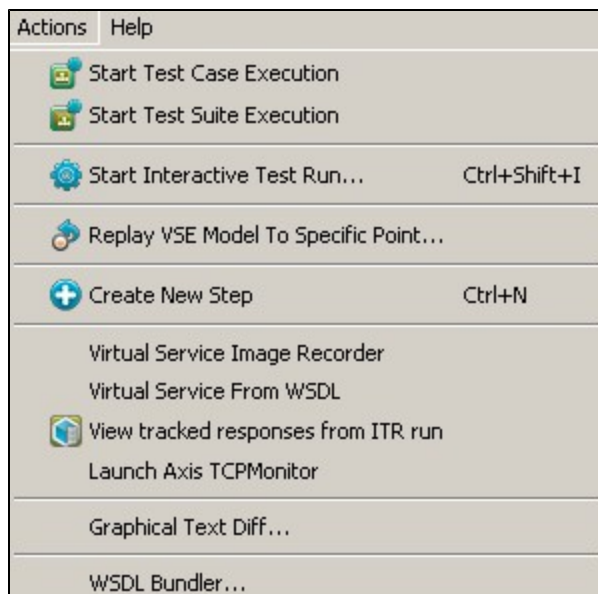
For more information refer to System Menu in User Guide [2.2.4 System Menu](#)

1.1.5 Actions Menu

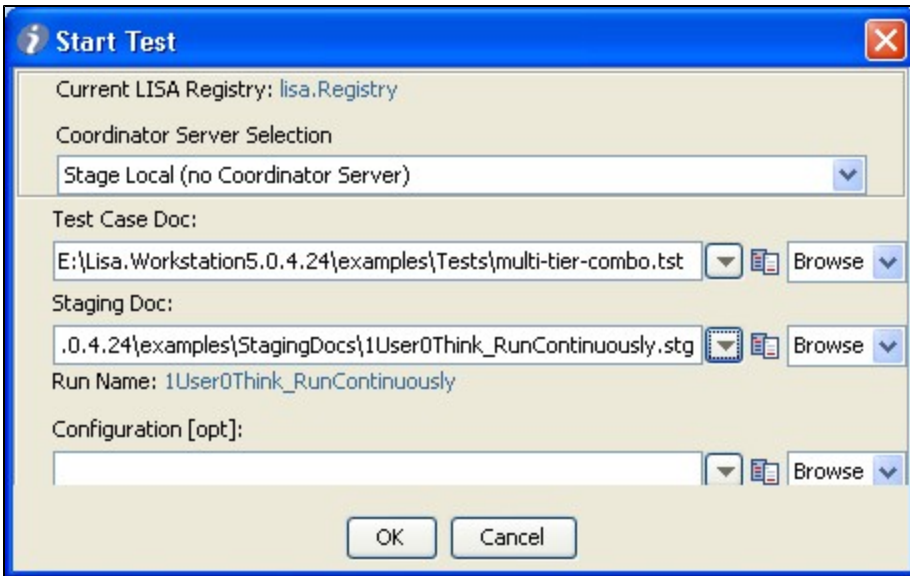
1.1.5 Actions Menu

The Actions menu has items related to Test Step or Test Suite Execution, Virtual Service Model Steps related, VSM execution tools in Workstation, and some external tools that are integrated into the LISA environment.

VSE Actions Menu

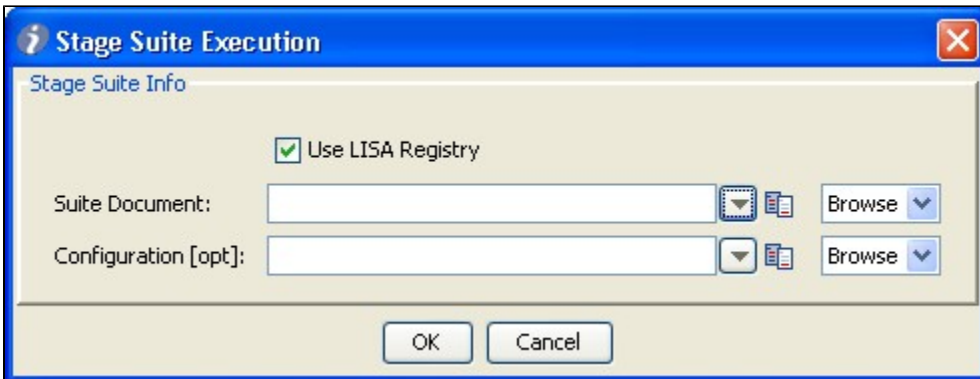


Start Test Case Execution - When selected, opens a dialog box wherein we can specify the test case to be executed.



For more information, refer to 18. Running a Single Test.

Start Test Suite Execution - When selected, opens a dialog box wherein we can specify the test suite to be executed.

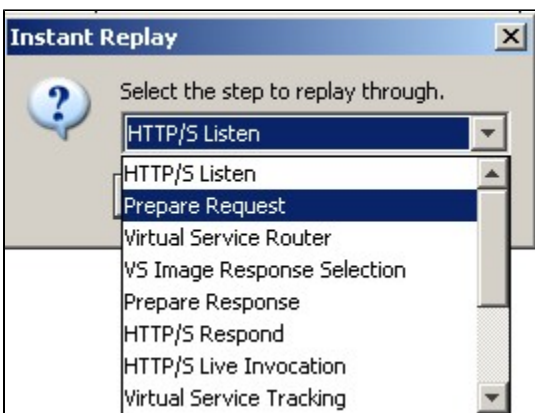


For more information, refer to 14. Building Test Suites

Start Interactive Test Run (ITR): Click to start the Interactive Test Run (ITR) utility.

For more information, refer to [PART 4 - Running Test Cases](#).

Replay VSE Model to Specific Point: Click to run a VSM to a specific point in VSE. A dialog box as shown below is displayed, to allow you to enter the last step of the replay. This will populate the known state up to that point.



Create New Step: When selected, opens a Steps panel with a list of all the available test steps in LISA. The selected step will be added to the

VSE Model.

For more information, refer to [PART 2 - Building Test Cases](#)

Virtual Service Image Recorder: When selected, opens **Virtual Service Image Recorder** dialog box to add the corresponding information depending on the protocol selected.

For more information, refer to [5.2 Service Image](#)

Virtual Service from WSDL: When selected, opens **Virtual Service from WSDL** dialog box to add the corresponding information depending on the WSDL.

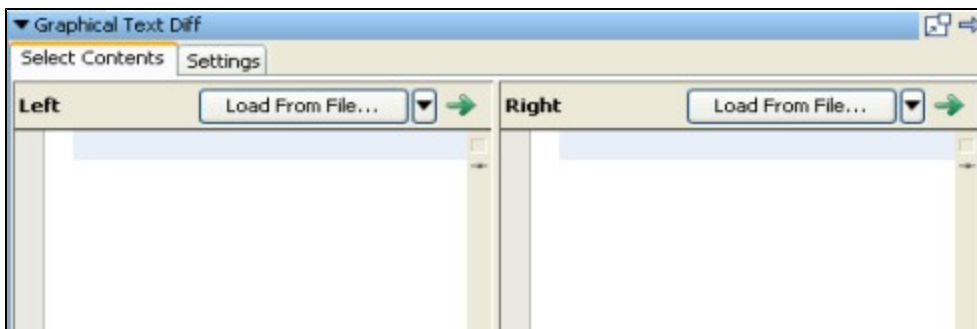
For more information, refer to [5.4 Virtual Service From WSDL](#)

View tracked Responses from ITR Run: Opens ITR dialog box which displays session information and corresponding request and response information generated from ITR runs.

For more information, refer to [8.5 Session Viewing and Model Healing](#)

Launch Axis TCPMonitor: When selected, launches an external application, TCPMonitor, which allows you to monitor client/Server communications. For more information see <http://ws.apache.org/axis/>.

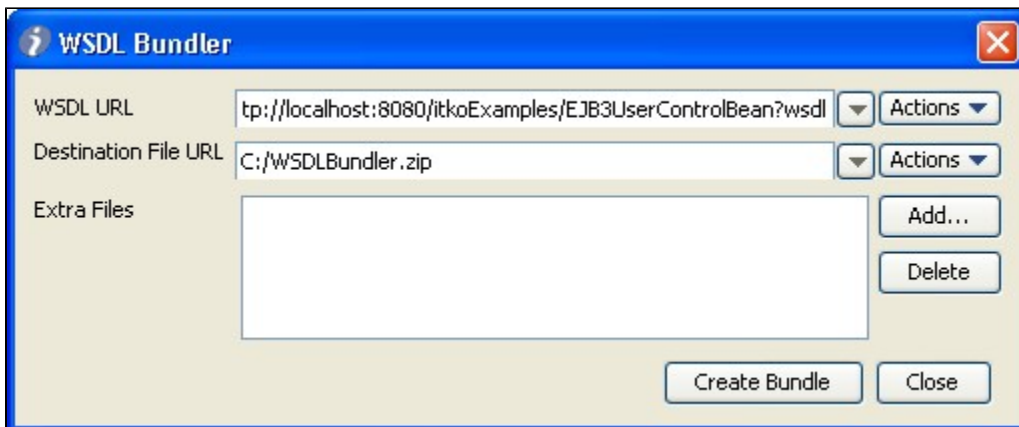
Graphical Text Diff...: When selected, opens a new graphical XML diff engine and visualizer which has been implemented in LISA 5.0. This menu is used to start a graphical text differentiators for XML comparison.



For more detailed information, please refer to the [22.3 Graphical Diff Facility](#) Graphical Diff Facility.

WSDL Bundler: When selected, allows you to bundle a WSDL and any supporting documents into a single zip file. The zip file can be sent to iTKO support if you are experiencing problems using the WSDL. The figure below shows an example.

Note: The Destination file is specified as a file URL.



1.1.6 Help Menu

1.1.6 Help Menu

LISA Help menu has the Help menu functions that talk about LISA, its Documentation, Runtime Information, LISA License settings, Debugger etc.

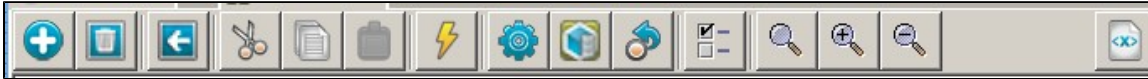
For more information refer to Help Menu in User Guide [2.2.6 Help Menu](#)


1.2 LISA workstation Toolbar for VSE




1.2 LISA Workstation Toolbar for VSE

The LISA Workstation toolbar reproduces the common functions present in the menus. Below is the toolbar for VSE.

VSE Toolbar



Icon	Description
	Create New Step (Actions -> Create New Step , or Ctrl+N)
	Delete a Test Step
	Set the currently selected step as the Starting step in the workflow
	Cut the selected text
	Copy the selected text
	Paste the selected text
	<div> Record Test Case for User Interface... ▶ Virtual Service Image Recorder Virtual Service From WSDL</div> <p>Record Test Case for User Interface : For recording Test Cases. Disabled under VSE environment.</p> <p>Virtual Service Image Recorder (Actions -> Virtual Service Image Recorder) Virtual Service From WSDL(Actions->Virtual Service From WSDL)</p>
	Start a New Interactive Test Run (Actions -> Start a New Interactive Test Run or Ctrl+Shift+I)
	View session/tracking information generated from ITR runs.
	Replay Model To a Specific Point (Actions -> Replay VSE Model To Specific Point)
	Show Model Properties (Help -> Property Window or Ctrl+Shift+P)
	Reset Zoom Scale to 1:1

	Zoom In
	Zoom Out
	View XML source

1.3 Service Virtualization

1.3 Service Virtualization

Service virtualization involves the imaging of software service behavior and the modeling of a virtual service to stand in for the actual service during development and testing. Service virtualization is complementary to hardware virtualization and addresses its limitations. The word virtualization is meant to refer to service virtualization for the rest of the document.

You may not want or be able to stay connected to the server for your quality assurance tasks. LISA Virtualize emulates the behavior of the server.

LISA can virtualize the following transport and data protocols:

- Transport protocols (interface protocol)
- Web Applications (HTTP/HTTPS)
- Databases (JDBC)
- Messaging Platforms (webMethods, TIBCO, MQ, Sonic, JCAPS)
- Data protocols
- Web Services (SOAP)
- Web Services Bridge

A data handler which can parse requests

Further, it is possible to add your own transport and data protocols to LISA Virtualize with [LEK \(Lisa Extension Kit\)](#).

Since this document concerns LISA's Virtualization capabilities, the terms LISA and LISA Virtualization are used interchangeably in this document.

While you can use LISA as the client to drive the service to be recorded, you will most likely exercise the service for recording with your own client (for example, a browser or an in-house application).

1.4 High-level Virtualization Steps

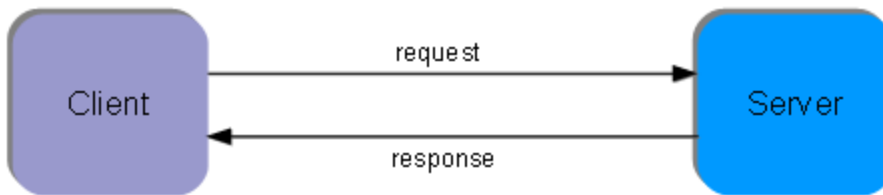
1.4 High-level Virtualization Steps

The high level steps are:

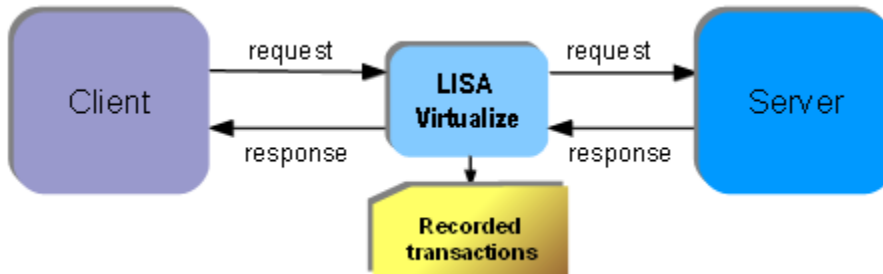
1. Take an image of the service behavior (service image): Record transactions that the server handles.
2. Construct the virtual service from the behavior (virtual service model or VSM).
3. Deploy the VSM to Virtual Service Environment (VSE). The VSM then looks at the captured service images to find the appropriate responses for requests coming in to the VSE.

The following images show that at the time of image recording, LISA Virtualize acts as the pass-thru mechanism between the client and server. While it passes the requests and responses along, it records the transactions.

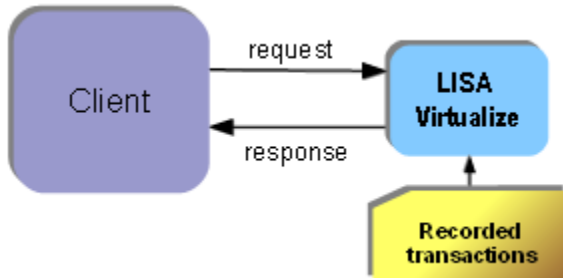
Normal Operation



Recording



At the time of virtualization, in absence of server, LISA Virtualize responds to the client requests by consulting the recorded transactions.



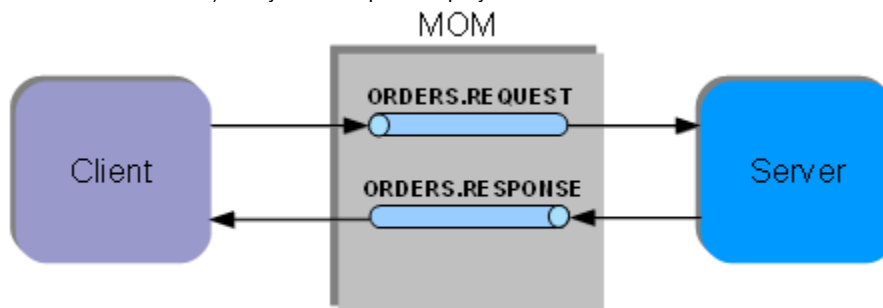
Virtualization of Messaging Systems

Message Oriented Middleware (MOM), or simply Messaging systems, are services that provide a means to enable asynchronous communication between two or more software applications. This communication always happens in the form of messages. The messages are posted to message destinations configured into the MOM.

The two types of message destinations are:

1. Queues: A publisher can add a message to the queue, and a subscriber pulls messages out of the queue, in a "first in, first out" fashion.
2. Topics: A publisher can publish a message to a topic, and all subscribers that are subscribed to the topic receive the message.

Below is a concept diagram of a simple message-based service. In the scenario shown, the client adds messages to a queue (ORDERS.REQUEST) which are picked up by the server. The response from the server is in the form of messages added on to another queue (ORDERS.RESPONSE). They are then picked up by the client.

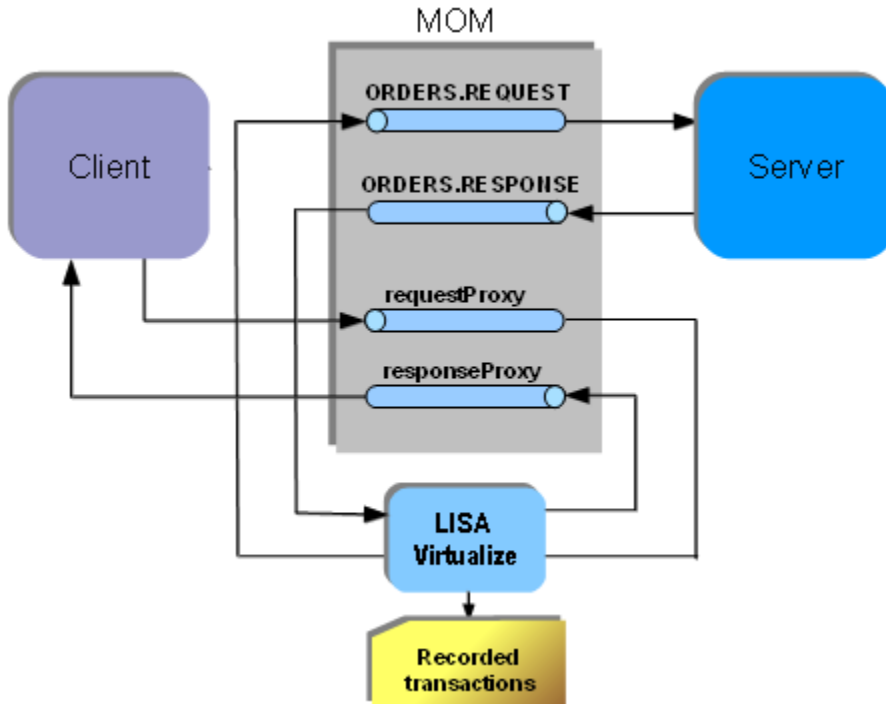


Some possible variations include:

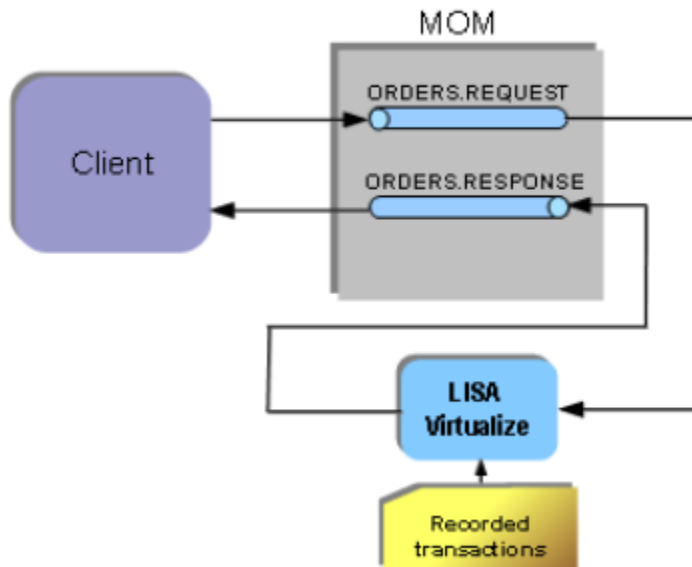
1. Use of topics instead of queues
2. Multiple responses to a single request, which can possibly target different destinations

As before, LISA Virtualize aims to virtualize the server. As shown in the following diagram, in the recording mode LISA Virtualize requires extra

proxy destinations (requestProxy and responseProxy queues in the diagram) which are used by the client in place of their counterparts. The server still listens and posts to the real destinations. LISA Virtualize acts as a pass-through between these proxy and real destinations, in turn recording the traffic to create the VSM and the service image that it needs for virtualization.



Later, when LISA virtualizes the server, it just works with the real destinations. It does not need the proxy destinations then.



1.5 Benefits of Virtualization

1.5 Benefits of Virtualization

The benefits of using LISA Virtualize include:

1. Providing round-the-clock access to service end points
2. Removing capacity constraints
3. Removing contention for shared resources
4. Providing an alternative to unavailable systems or systems still under development
5. Controlling complex data scenarios that are inherent during the SDLC
6. Reducing or eliminating the cost of invoking third-party systems for non-production use

7. Increasing agility and quality in complex and changing IT environments

2. VSE Installation and Configuration

2. VSE Installation and Configuration

LISA VSE is required for maintaining a Service Oriented Virtualization environment.

As LISA VSE is a server level service, it can co-exist with a TestRegistry that has a Coordinator and Simulator attached to it, but the Simulator and Coordinator are not mandatory to run VSE.

The following topics are available.

- [2.1 System Requirements](#)
- [2.2 Installing LISA Virtualize](#)
- [2.3 Configuring LISA Virtualize](#)
- [2.4 Installing the Database Simulator](#)
- [2.5 Installing a Messaging Simulator](#)
- [2.6 DDLs for Major DBs](#)

2.1 System Requirements

2.1 System Requirements

System Requirements

For VSE - below are baseline requirements only.

Description	Requirement
CPU	2GHz or faster
RAM	2GB or more
Disk Space	5GB free space
OS	Recommended: 64 Bit OS. Also supported: Windows 2000 *, 2003, XP, Linux, Solaris, AIX 6.1 (Only with LISA 5.0)
DB	Not required but recommended to use MySQL, Oracle, DB2, or SQL Server for production usage. DB can reside on a different system and should have at least 10 GB of storage.

The following requirements are recommended for larger scale VSE deployments.

256 Virtual Service Threads per VSE instance
1 Processor Core and 2GB RAM per VSE instance

Example: 1,000,000 transactions per day
1 thread/service to support functional tests ~6 threads/service to support virtualization for L&P Tests
8 cores = 2,048 concurrent virtual service threads
16 GB RAM (for LISA)

For more details on other system requirements, see [System Requirements and Prerequisites](#) in the *LISA Installation and Configuration Guide*.

Software Directory Structure and Files

The following is the LISA Virtualize directory structure. The directories are located in the LISA root installation folder.

Directory	Description
Bin	Contains LISA executables, such as TestRegistry.exe , LISAWorkstation.exe , VirtualServiceEnvironment.exe , VSEManager.exe .
DemoServer	Contains LISA Demo by default.
Doc	Contains LISA documentation.

examples\vse	Contains examples relating to LISA Virtualize.
Reports	Contains LISA's built-in database where it stores reports and where VSE stores service images.
Tmp	Contains LISA VSE's workspace where it temporarily stores conversations and stateless transactions.
vseDeploy	Contains deployed VSMs and related data.

2.2 Installing LISA Virtualize

2.2 Installing LISA Virtualize

To install LISA Virtualize:

1. Use the installation file provided to you. (You may alternatively download it from www.itko.com/download/ga after logging in.)
2. Double-click the installation file to launch the LISA Setup Wizard, which guides you through the installation process.
3. Click Next.
4. Read and accept the license agreement, and click Next.
5. Click Next to accept the default installation destination directory (for example, C:\LISA).
6. Click Next to accept the default Start Menu folder called LISA and create shortcuts for all users.
7. Click Next to accept the desktop icon creation.
8. The installation takes a few minutes.
9. Click Next to accept the four default file associations (.tst, .vsm, .ste, and .stg).
10. In the Windows Services window, click Next to accept the default settings to install all LISA components as services. If needed, check Start on bootup for any of the services you want to start automatically when the server is started.
11. Read the information window, which provides information about:
 - Locating the readme.html file.
 - Resolving difficulty starting LISA applications.
 - Authorizing LISA Workstation to work with firewalls in a Windows environment.
 - Completing the installation in a Unix environment.
12. Click Next.
13. Click Finish to complete the installation and start LISA Workstation.

2.3 Configuring LISA Virtualize

2.3 Configuring LISA Virtualize

Setting up the License

Virtual services are limited to the maximum number of virtual services you own. Each component must have access to the license server or have a file-based license installed in [LISA_HOME]. In addition, the **virtualService** item in the License Info tab must be set to **true**.

You must add your license information for LISA Server to the **local.properties** file.

You should receive your license information from iTKO Support after purchasing LISA.

To setup a server-based license:

1. In the root LISA installation folder, make a copy of the **_local.properties** file.
2. Name the copy **local.properties**.
3. Open **local.properties** in a text editor and add the PROJECT_ROOT definition:


```
# =====
# Project root definition
# =====
PROJECT_ROOT=C:/TestAssets
```
4. In the license properties section, uncomment the properties lines and add your specific licensing information:

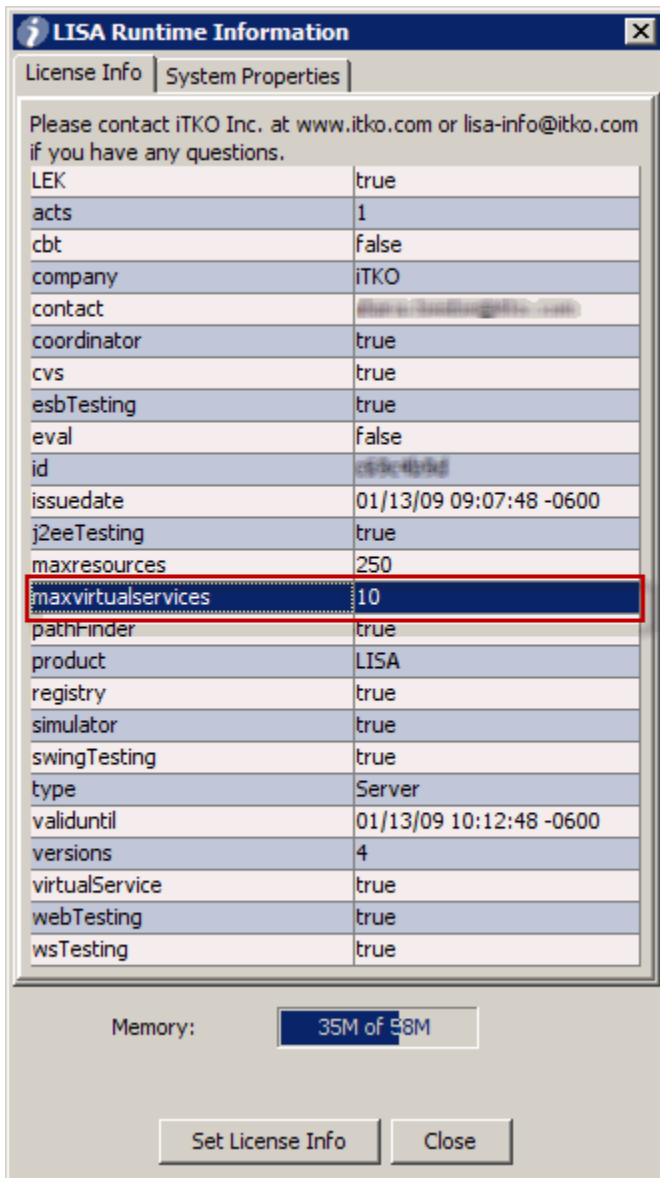

```
# =====
# license properties
# =====
laf.server.url={*}https://license.itko.com*
laf.domain=iTKO/LISA/YOURCO
laf.username=YOURUSERNAME
laf.password=YOURPASSWORD
```
5. Save and close the file.

Viewing Your Number of Licenses

You can view the number of virtual services for which you have licenses in the LISA Runtime Information window.

To view your license information:

1. From the menu bar, select Help > LISA Runtime Info.
2. In the License Info tab, locate the **maxvirtualservices** property. The number of licenses is shown in the second column.
3. Click Close.



You can install one Test Registry which runs as a server, a number of Virtual Service Environment servers and a number of LISA Workstations which run as desktop applications, depending on your LISA license.

Modifying the VSE Service Image Database Settings

LISA Virtualize uses the same Derby database instance as the reports database. If you need to change the database instance in which service images are stored, you must uncomment and modify the following settings in **local.properties**. Additionally, copy the relevant database driver .jar files to the **LISA_HOME/lib** folder.

The following example settings use Oracle:

```
#eclipselink.jdbc.driver=oracle.jdbc.driver.OracleDriver
#eclipselink.jdbc.url=jdbc:oracle:thin:@_HOST:1521:SID_
#eclipselink.jdbc.user=USER
#eclipselink.jdbc.password=PASS
#eclipselink.ddl-generation=create-tables
#eclipselink.ddl-generation.output-mode=database
```

Proxy for localhost

When LISA acts as the HTTP client for LISA Virtualize, the HTTP traffic from LISA needs to be passed to LISA Virtualize. A common way to do this is setting LISA Virtualize as the web proxy. However, the use of proxy is disabled by default for simple names like "localhost". To override this behavior, uncomment the following LISA property and set its value to false:

```
lisa.http.webProxy.nonProxyHosts.excludeSimple=false
```

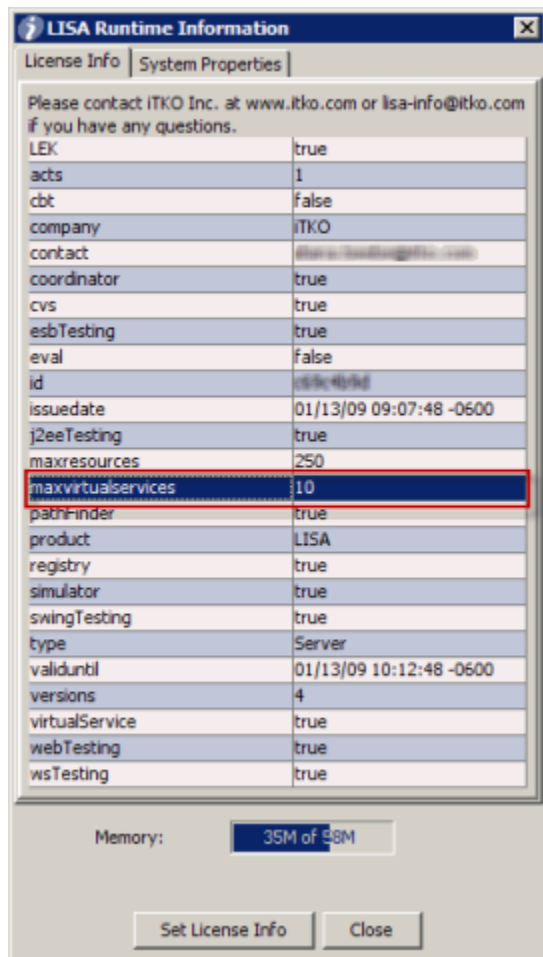
Other Settings

You can optionally set some other basic configurations in local.properties:

Rename the VSE Server— Uncomment the following LISA property and change the value where **VSENAME** is to the name of the VSE Server:
`lisa.vseName=VSENAME`

Connect to another test registry—uncomment the following LISA property and change the values of **SERVER**, **PORT**, and **TRNAME** to the server name (or IP address), port number, and name of the new test registry:

```
lisa.testRegistry=rmi://SERVER:PORT/TRNAME
```



2.4 Installing the Database Simulator

2.4 Installing the Database Simulator

Recording the JDBC traffic is done by installing the LISA Simulation JDBC driver on the database client, so that it uses that in place of the actual one.

The driver is packaged in a jar file called `lisajdbcsim.jar` located in the `LISA_HOME/lib` directory. You need to copy this file to be in your database client's classpath. For convenience, it is already copied into the Demo server in the `DEMO_HOME/jboss/server/default/lib` directory, to aid the use of Demo server as the database client.

The driver class needs to be set to `com.itko.lisa.vse.jdbc.driver.Driver`. This needs to be done differently based on the style of JDBC the database client uses:

DriverManager Style: If the database client uses Java's DriverManager to acquire connections (not likely in an application server), add the following to the startup command for the database client:

```
-Djdbc.drivers=com.itko.lisa.vse.jdbc.driver.Driver
```

If this property is already used, add the LISA driver to the front, separating it from the rest of the driver class names with a colon.

DataSource Style: If the database client uses the DataSource style for connection acquisition (as the Demo Server does), then update its configuration. In the place where it specifies a data source definition, we need to specify **com.itko.lisa.vse.jdbc.driver.Driver** as the JDBC driver to use and a connection URL, as discussed below.

The driver is packaged in a jar file called **lisajdbcsim.jar** located in the **LISA_HOME/lib** directory. For convenience, it is also copied into the Demo server (for virtualizing it) in the **DEMO_HOME/jboss/server/default/lib** directory.

The LISA Simulation JDBC driver needs to know the real driver information to achieve a pass-thru. It is achieved by modifying the connection url as below. Format the connection URL as:

```
name=value[:name=value...]
```

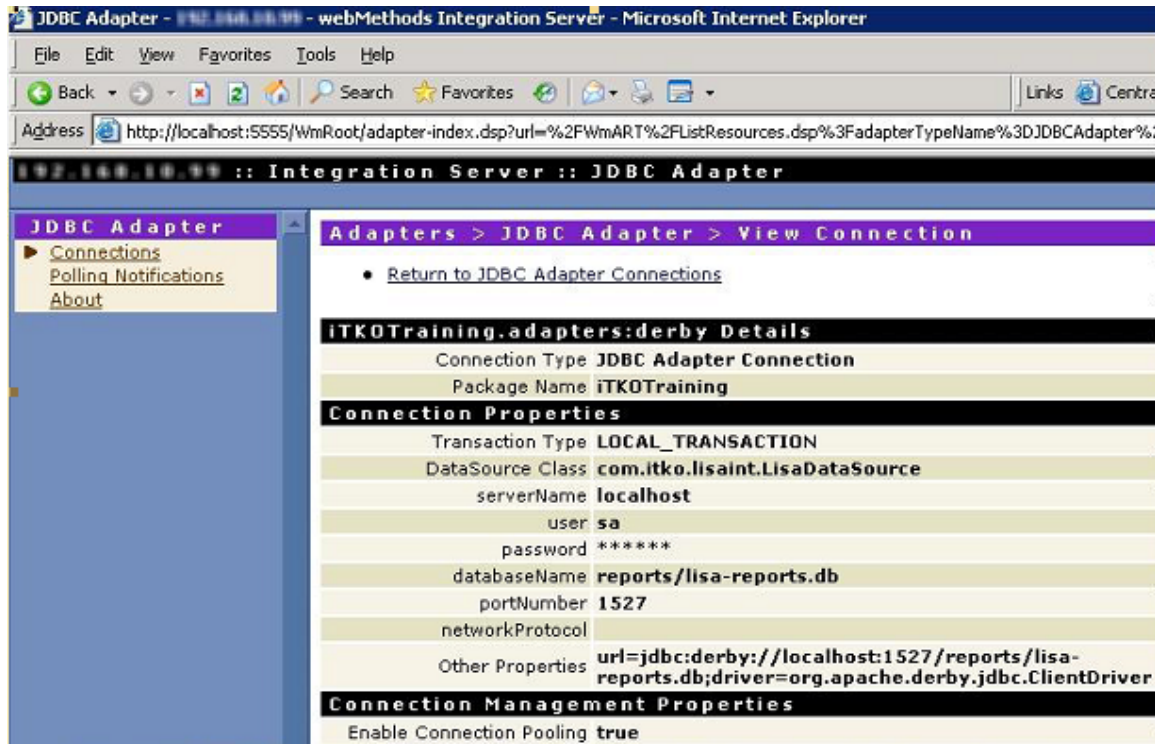
where **name** may be any of the following:

jdbc:lisasim:driver: The value must be the fully qualified class name of the real JDBC driver to use.

url: The value must be set to the connection url that the real driver is expecting. (It must be defined as the last property so that it can contain semi-colons.)

Note: LISA Virtualize does not support Oracle thin driver as a pass-thru driver, as it does not provide the full JDBC implementation. If you are using Oracle as a database, please use other JDBC drivers for virtualization.

As an example of setting the connection url, refer to **DEMO_HOME/jboss/server/default/deploy/itko-example-ds.xml**. As an example of virtualizing the database in WebMethods environment, see the following screenshot.



If you want to use both the simulation and Pathfinder drivers, make the simulation driver the "outside" driver and specify the Pathfinder class and URL as the real information described above. Please refer to the **itko-example-ds.xml** file in the **DEMO_HOME/jboss/server/default/deploy** folder for an example that defines LISA data source to JBoss for LISA Demo Server.

Other Startup Properties

Regardless of the database client's connection style, you can add these properties to the startup command for the database client to affect the simulation driver:

lisa.jdbc.sim.require.remote---Valid values for this property are **true** and **false** and the default value is **false**. If set to true the driver blocks until there is an active connection with a LISA Workstation or VSE Server instance. This is the preferred way to have a database client synchronize with LISA VSE when you want to record or play back any startup database activity performed by the server.

lisa.jdbc.sim.port---This property must specify a valid IP port. If it is not specified, it defaults to **2999**. This is the port the driver listens on for connections from a recorder or a running virtual service model.

2.5 Installing a Messaging Simulator

2.5 Installing a Messaging Simulator

LISA can virtualize a messaging-based SOA environment. For this, it needs to connect to the MOM. Please note the following:

1. LISA has special steps for connecting to IBM WebSphere MQ server.
2. It can connect to other messaging systems if they support JMS.
3. Support for any other messaging system may be created by creating custom steps through LEK.

To connect to MQ server, copy the following jar files from your MQ installation into **LISA_HOME/lib** folder: **com.ibm.mq.commonservices.jar**, **com.ibm.mq.defaultconfig.jar**, **com.ibm.mq.headers.jar**, **com.ibm.mq.jar**, **com.ibm.mq.jmqi.jar**, **com.ibm.mq.pcf.jar**, **com.ibm.mqjms.jar**, **dhbcore.jar**, and **connector.jar**. These are found under **java/lib** folder under your MQ installation. The actual names of the files may differ between different versions of MQ.

In case of a JMS system, LISA already has the generic jar files for connecting to a JMS system – in particular, to the demo server. However, any additional driver files that may be needed to connect to the messaging system may be needed to be copied into **LISA_HOME/lib** folder.

2.6 DDLs for Major DBs

2.6 DDLs for Major DBs

LISA can generate DDL to a FILE by adding the following three lines to my local.properties:

```
eclipselink.ddl-generation=create-tables
eclipselink.ddl-generation.output-mode=sql-script
eclipselink.target-database=Oracle
```

Start LISA with these three lines and it creates two files - **createDDL.jdbc** and **dropDDL.jdbc** - that contain the necessary DDL.

You can generate for a different DBMS by changing the value of "target-database".

DDL is generated by JPA persistence stuff

This table lists the EclipseLink JPA persistence unit properties that you can define in a `persistence.xml` file to configure EclipseLink extensions for session, as well as the target database and application server.

Property	Usage
<code>eclipselink.session-name</code>	Specify the name by which the EclipseLink session is stored in the static session shared session outside of the context of the JPA or to use a pre-existing Eclipse Valid values: a valid EclipseLink session name that is unique in a server deployment Example: <code>persistence.xml</code> file<property value="MySession"/> Example: <code>org.eclipse.persistence.config.PersistenceUnitProperties;propertiesMap.put(Pers</code>
<code>eclipselink.sessions-xml</code>	Specify persistence information loaded from the EclipseLink session configuration You can use this option as an alternative to annotations and deployment XML. If annotation and the object relational mapping from the <code>persistence.xml</code> , as well as Indicate the session by setting the <code>eclipselink.session-name</code> property. Note: If you do not specify the value for this property, <code>sessions.xml</code> file will not Valid values: the resource name of the sessions XML file. Example: <code>persistence.xml</code> file<property value="mysession.xml"/> Example: <code>org.eclipse.persistence.config.PersistenceUnitProperties;propertiesMap.put(Pers</code>
<code>eclipselink.session-event-listener</code>	Specify a descriptor event listener to be added during bootstrapping. Valid values: qualified class name for a class that implements the <code>org.eclipse</code> Example: <code>persistence.xml</code> file<property value="mypackage.MyClass.class"/> <code>org.eclipse.persistence.config.PersistenceUnitProperties;propertiesMap.put(Pers</code> "mypackage.MyClass.class");

eclipselink.ddl-generation	<p>Specify what Data Definition Language (DDL) generation action you want for your JPA entity eclipselink.ddl-generation.output-mode.</p> <p>The following are the valid values for the use in a persistence.xml file:</p> <ul style="list-style-type: none"> • none – EclipseLink does not generate DDL; no schema is generated. • create-tables – EclipseLink will attempt to execute a CREATE TABLE SQL for will follow the default behavior of your specific database and JDBC driver combination (an already existing table). In most cases an exception is thrown and the table is not next statement • drop-and-create-tables – EclipseLink will attempt to DROP all tables, then CREATE EclipseLink will follow the default behavior of your specific database and JDBC driver statement. <p>The following are the valid values for the org.eclipse.persistence.config.</p> <ul style="list-style-type: none"> • NONE – see none. • CREATE_ONLY – see create-tables. • DROP_AND_CREATE – see drop-and-create-tables. <p>If you are using persistence in a Java SE environment and would like to create the define a Java system property INTERACT_WITH_DB and set its value to false.</p> <p>Example: persistence.xml file<property value="create-tables"/>Example: property Mapimport org.eclipse.persistence.config.PersistenceUnitProperties;propertiesMap.put(PersistenceUnitProperties.CREATE_ONLY);</p>
eclipselink.application-location	<p>Specify where EclipseLink should write generated DDL files. Files are written if eclipselink application-location is not none.</p> <p>Valid values: a file specification to a directory in which you have write access. The file specification can be relative to the application location or absolute. If it does not end in a file separator, EclipseLink will append one valid file separator.</p> <p>Example: persistence.xml file<property value="C:\\ddl\\"/>Example: property Mapimport org.eclipse.persistence.config.PersistenceUnitProperties;propertiesMap.put(PersistenceUnitProperties.APPLICATION_LOCATION, "C:\\ddl\\");</p>
eclipselink.create-ddl-jdbc-file-name	<p>Specify the file name of the DDL file that EclipseLink generates containing SQL statements written to the location specified by eclipselink.application-location when eclipselink.create-ddl-jdbc-file-name is not none.</p> <p>Valid values: a file name valid for your operating system. Optionally, you may prefix the file name with the concatenation of eclipselink.application-location + eclipselink.create-ddl-jdbc-file-name for your operating system.</p> <p>Example: persistence.xml file<property value="create.sql"/>Example: property Mapimport org.eclipse.persistence.config.PersistenceUnitProperties;propertiesMap.put(PersistenceUnitProperties.CREATE_DDL_JDBC_FILE_NAME, "create.sql");</p>
eclipselink.drop-ddl-jdbc-file-name	<p>Specify the file name of the DDL file that EclipseLink generates containing the SQL statements written to the location specified by eclipselink.application-location when eclipselink.drop-ddl-jdbc-file-name is not none.</p> <p>Valid values: a file name valid for your operating system. Optionally, you may prefix the file name with the concatenation of eclipselink.application-location + eclipselink.drop-ddl-jdbc-file-name for your operating system.</p> <p>Example: persistence.xml file<property value="drop.sql"/>Example: property Mapimport org.eclipse.persistence.config.PersistenceUnitProperties;propertiesMap.put(PersistenceUnitProperties.DROP_DDL_JDBC_FILE_NAME, "drop.sql");</p>
eclipselink.ddl-generation.output-mode	<p>Use this property to specify the DDL generation target.</p> <p>The following are the valid values for the use in the persistence.xml file:</p> <ul style="list-style-type: none"> • both – generate SQL files and execute them on the database. If eclipselink.ddl-generation is set to create-tables, then eclipselink.create-ddl-jdbc-file-name and eclipselink.drop-ddl-jdbc-file-name are used to generate SQL files and executed on the database. If eclipselink.ddl-generation is set to drop-and-create-tables, then eclipselink.create-ddl-jdbc-file-name and eclipselink.drop-ddl-jdbc-file-name are used to generate SQL files and both SQL files are executed on the database. • database – execute SQL on the database only (do not generate SQL files). • sql-script – generate SQL files only (do not execute them on the database). If eclipselink.ddl-generation is set to create-tables, then eclipselink.create-ddl-jdbc-file-name is used to generate SQL files. It is not executed on the database. If eclipselink.ddl-generation is set to drop-and-create-tables, then eclipselink.drop-ddl-jdbc-file-name is used to generate SQL files. Neither is executed on the database. <p>org.eclipse.persistence.config.PersistenceUnitProperties:</p> <ul style="list-style-type: none"> • DDL_BOTH_GENERATION – see both. • DDL_DATABASE_GENERATION – see database. • DDL_SQL_SCRIPT_GENERATION – see sql-script. Example: persistence.xml file<property value="DDL_DATABASE_GENERATION"/>Example: property Mapimport org.eclipse.persistence.config.PersistenceUnitProperties;propertiesMap.put(PersistenceUnitProperties.DDL_DATABASE_GENERATION, "DDL_DATABASE_GENERATION");

3. Understanding LISA Virtualize

3. Understanding LISA Virtualize

The following topics are available.

- [3.1 Following the LISA Virtualize Process](#)
- [3.2 Concept Diagram](#)
- [3.3 Virtual Service Model \(VSM\)](#)
- [3.4 Services Image](#)
- [3.5 How Virtualization Works](#)
- [3.6 Magic Strings and Dates](#)

3.1 Following the LISA Virtualize Process

3.1 Following the LISA Virtualize Process

The general process for working with LISA Virtualize includes these steps:

1. Create a Virtual Service Model (VSM) in the LISA Workstation.
2. Launch the Virtual Service Image Recorder.
3. In the Virtual Service Image Recorder, provide basic information about what needs to be recorded, select the appropriate protocols, and specify the default navigations. Provide any information required by the selected protocols. Begin the recording.
4. Exercise the client to cause communication with the server routed through LISA Virtualize. LISA Virtualize records the traffic.
5. In the Virtual Service Image Recorder, finish the recording.
6. In LISA Workstation, save the VSM.
7. In VSE Dashboard, deploy the VSM, and start the virtual service.
8. Run live requests against LISA VSE (Virtual Service Environment).

3.2 Concept Diagram

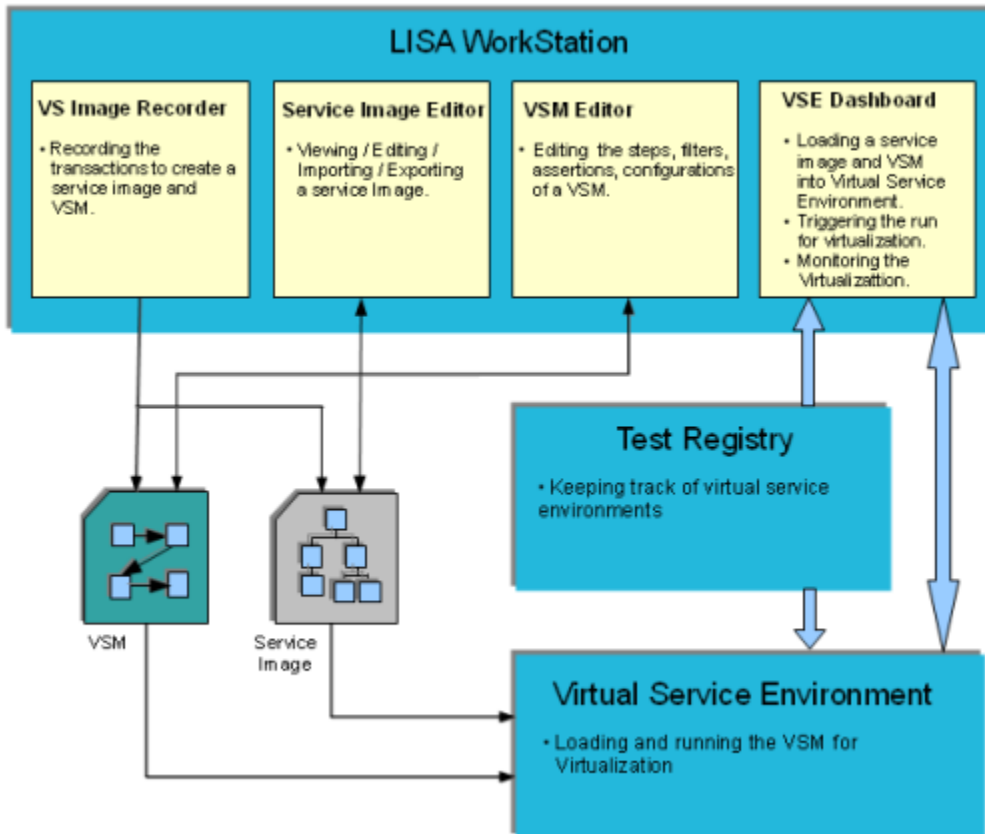
3.2 Concept Diagram

The following diagram illustrates how the various components in LISA Virtualize are related with each other.

At the time of recording, the Virtual Service Image Recorder is used to record a service image and a VSM. The Service Image Editor and VSM Editor help in editing and viewing them.

At the time of Virtualization, the VSM and service image are loaded and run on a VSE which runs as a service. The Test Registry service is used to keep track of one or more Virtual Service Environments running, and LISA Workstation uses it to connect to the VSE. The VSE Dashboard is

the UI provided in the LISA workstation, which helps in monitoring and controlling the various VSMs and service images loaded onto the Virtual Service Environments.



3.3 Virtual Service Model (VSM)

3.3 Virtual Service Model (VSM)

A virtual service model can be conceptualized as a series of steps to be executed once a request is received, to create and pass back a response to the request. It is stored in a **.vsm** file, and must contain at least one VSE step from the **Virtual Service Environment** step list in Workstation to be deployable to a VSE.

Once a service image is recorded, the protocol-specific steps are automatically generated in the VSM for all transports supported. Any of the generated steps can be modified. You can also:

1. Populate responses from an XLS or by cross referencing a database table and do math on inputs.
2. Add steps of different step types.
3. Manipulate the request and response steps before proceeding.
4. Specify the service image you would like to use from the Response Selection step

At the time of virtualization, a VSM needs to be deployed to the VSE. It defines how behavior patterns get used and queries the service image to determine how to respond. It knows how to traverse the service image. In general, VSMs are deployable to VSEs only while test cases can be staged to coordinators, but not vice versa.

Note: If you have a message-based virtual service and since the same messaging steps are used for both testing and virtualizing, add the Message Marker step, which lets you deploy to VSE.

3.4 Services Image

3.4 Services Image

A service image is a recording of the interaction between the client and server as created by LISA Virtualize and it is referenced in a VSM. Once recorded, the service image is used to deliver the appropriate response to the client in the absence of the server.

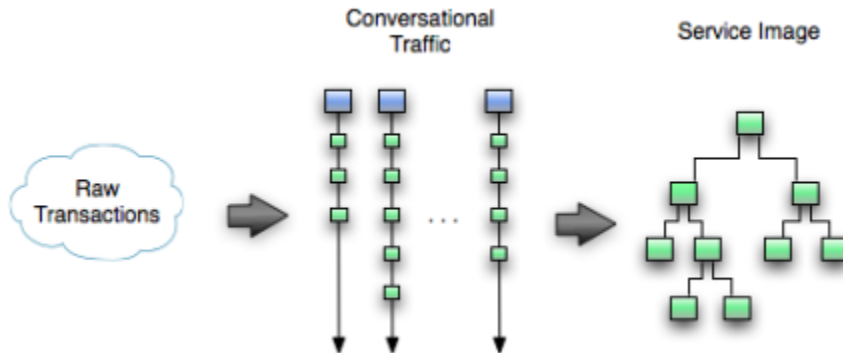
A service image contains three sets of information:

1. A list of conversations (requests and responses) recorded as conversation trees.

2. A list of stateless transactions (requests and responses).
3. The responses that should be sent when unknown conversational or stateless requests are encountered. (If the recorded service image is incomplete, error message will appear.)

Service images are stored in the LISA Internal Derby Reports database (**C:\Lisa\reports\lisa-reports.db**) by default. The destination may be changed by [modifying the appropriate settings](#). You can view, edit, or create a service image using the Service Image Editor.

A conversation may be visualized as a series of stateful transactions. However, multiple conversations (from multiple sessions) may be recorded in the same service image. Similar request structures are merged into a single transaction, thus creating a tree as shown in the diagram below.



As an example, if multiple users log into the system with login() transactions, all these transactions will be merged in a single transaction. But if one user logs in using login() and another user logs in with an acquireAuthToken(), then these transactions will not be merged.

Importing Transactions

The following xml documents may be imported into a service image being recorded:

Raw Transactions

The xml document represents raw traffic as if coming off the wire, characterized by a root element of <rawTraffic>.

A well-documented example can be found in [\[LISA_HOME\]/examples/vse/raw-traffic.xml](#).

Conversational Traffic

The xml document represents traffic that has been rearranged into conversations and/or stateless transaction sets. They are organized into linear lists of transactions, each of which represents a "real" conversation and are characterized by a root element of <traffic>.

A well-documented example can be found in [\[LISA_HOME\]/examples/vse/traffic.xml](#)

3.5 How Virtualization Works

3.5 How Virtualization Works

In the absence of a server, LISA Virtualize simulates the behavior of the server for its client. This process is known as the virtualization of server. It requires loading the service image referenced from a VSM and running it in the VSE dashboard.

When a request is received by the VSE, the request is examined and an attempt is made to match it to an existing conversational state (session) within the VSE. For example, a cookie ID or some other session identifier can 'tie' requests in stateless protocols such as HTTP. The conversational state is used to determine where in the conversation tree the 'current transaction' is, and any other 'state' such as a previously submitted authentication token, the user name, etc, which may not be part of the current request, but used in the subsequent response.

If an existing session cannot be found, the VSE attempts to match the request against the starter transactions of each conversation in the image. If a match is found, a new session is created and the relevant response is returned. The session is maintained until 20 minutes after it has last been 'seen' by the VSE. If no conversation starters match, no session is created and the list of stateless transactions is consulted in the order they are defined. If there is a match, the appropriate response is returned. If there is still no match, the 'unknown request' response is sent.

In the case where we do find a previous session and we are in the midst of a conversation, the next transaction match depends on many factors including navigation and match tolerances (described below).

If a matching transaction is not found in the conversational and stateless transactions, then the service image is consulted again for the kind of response that needs to be sent out for an unmatched conversational/stateless request.

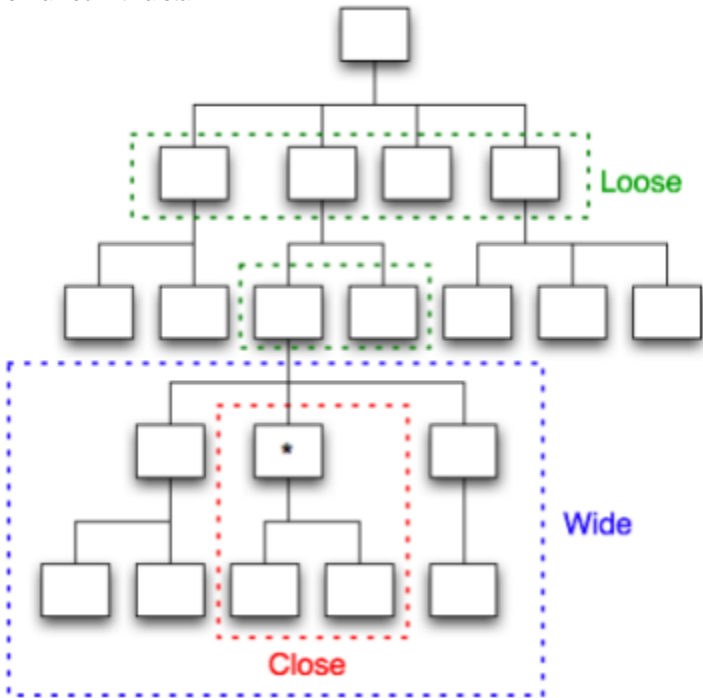
Handling Conversational Requests

The navigation tolerance that you can specify for every node in the tree plays an important role in how the VSM handles a conversation request. It is used to determine where in the conversation tree VSM should search for a transaction that should follow the given transaction.

Navigation Tolerance

Tolerance level	Description
Close	The current transaction's children are searched.
Wide	A close search, including the current transaction plus siblings and "nieces/nephews" of the current transactions.
Loose	A wide search, plus the current transaction's parent and its siblings followed by the children of the starting transaction. If both fail to find a match, then the starting transactions for all conversations are checked, resulting in searching the full conversation.

The following diagram illustrates how navigation tolerance impacts the transactions to be searched in a conversation tree. The current transaction is marked with a star ★.



At the time of recording, the VSE recorder allows for initializing the navigation tolerance on transactions with two separate settings, as described below:

Navigation concept	Description	Default
Default navigation	Refers to the default tolerance on all meta transactions that have child meta transactions.	Wide
Last	Refers to the default tolerance for meta transactions that are "leaf" transactions without any child meta transactions.	Loose

These may later be changed for each node through service image editor.

The defaults provide a wider hit on "right" behavior. VSE responds correctly more often in situations when current runtime sessions restart a conversation without the need to start a new conversation.

Handling Unknown Requests

There are two situations in which an unknown request may occur:

- 1. When there is an active conversation
- 2. When there is not

In the case where there is no active conversation, a request is identified as unknown if there are no stateless transactions that can satisfy the request. In this case, the service image's response for unknown stateless requests becomes the reply.

In the case where a request cannot be matched to a follow-on transaction

- 1. If the navigation tolerance is not CLOSE, then the conversation starter transactions are given the chance to satisfy the request.
- 2. If the request is still unmatched, and if a stateless transaction can produce a reply, then that is sent out. The current session continues to remember where it is in its conversational tree.

If that fails, the service image's response for unknown conversational requests becomes the reply.

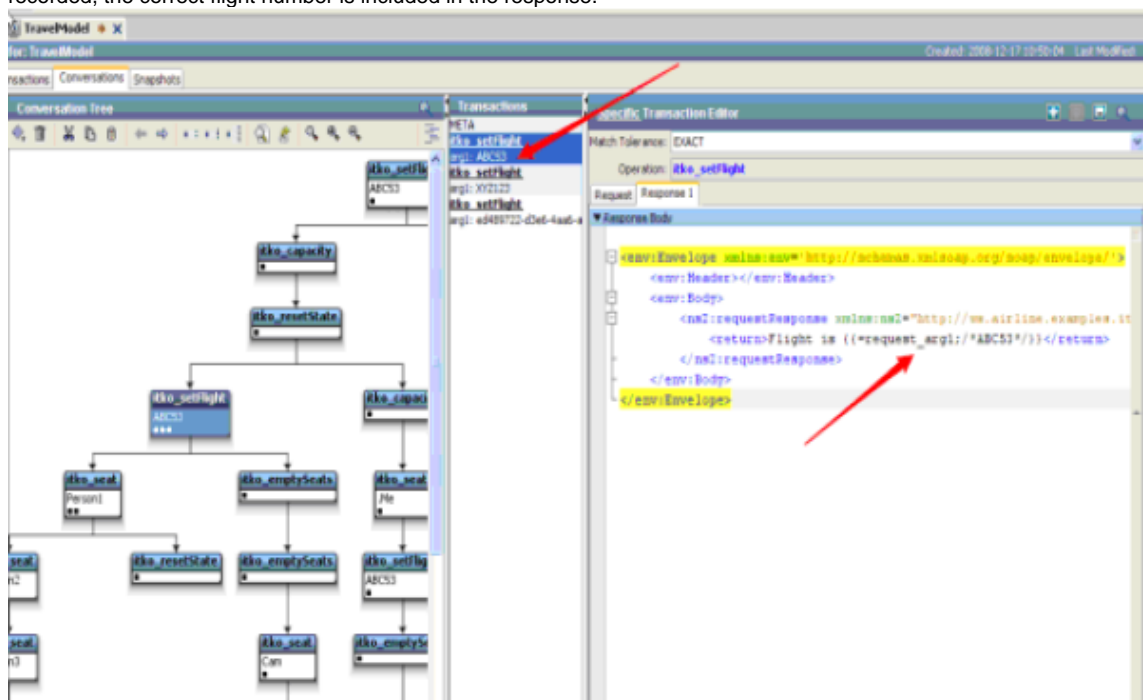
3.6 Magic Strings and Dates

3.6 Magic Strings and Dates

Magic strings and magic dates are central to the dynamic nature of a virtualized service. They enable the virtualized service to return meaningful results for request parameters that were never recorded.

Magic Strings

During the recording phase, each request is examined for arguments or parameters. For example, a weather forecasting web service call might include a city name or an airline booking system request might include a flight number. If the flight number is included in the recorded *response*, then it is classified as a 'magic' string. This means that if the VSE is in playback mode and a request comes in for a flight number that was never recorded, the correct flight number is included in the response.



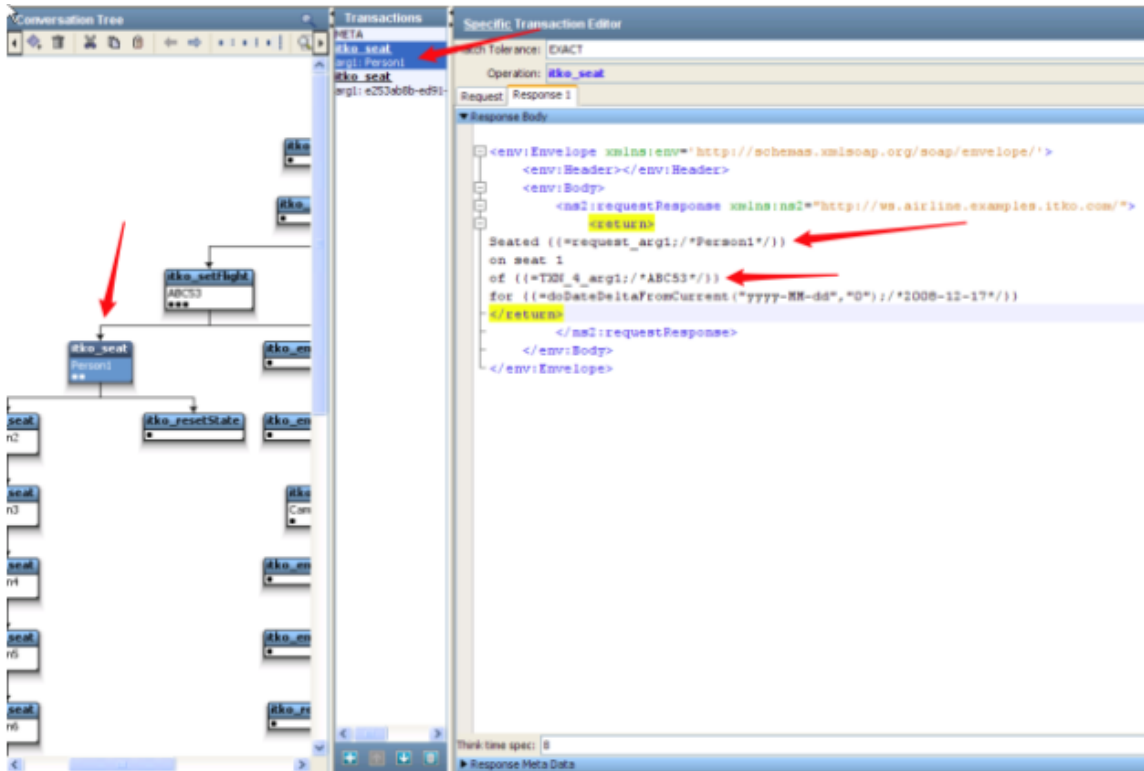
In the example above, we recorded a request to set some 'state' in the conversation, in this case the flight number (ABC53). The recorder noted that the same string ABC53 was contained in the response, so it converted it to a magic string, which is saved in the service image database as '=request_arg1/'ABC53/'.

The {{ }} notation is a standard LISA property substitution syntax. The meaning here is 'substitute this {{ }} text with the runtime value of the first argument of the request, and if there is no first argument then use ABC53 as the default'.

The practical significance of this is that if some future client requests flight ZZ99, the virtualized service will respond with the correct flight number, ZZ99.

Magic strings are detected not just in a simple request/response pair. If an argument is ever seen in any subsequent response in a conversation, it is deemed to be magic and the argument value is stored in the conversation state.

Here's an example, just a little further on in the same conversation. The request operation is 'itko_seat' and the single argument is a name, in this case 'Person1'. The response contains the name in the request, the previously set flight number and a date (more on dates later).



This is the canonical example of conversational state over a stateless protocol, in this case SOAP over HTTP. It means that if the VSE in playback mode sees a request to set the flight to ZZ99 and then a request to seat PersonSomeoneElse, then the correct string, 'Seated PersonSomeoneElse on seat 1 of flight ZZ99' will be included in the response, even though those details were never recorded.

Note that the seat number, 1 in this case, is not regarded as a magic string, since it was never seen in the original series of requests that led to this response. Nor is it 'big enough' to be regarded as a magic string - they need to be at least 3 characters long and optionally have whitespace on the left, right or both sides of the string. You can adjust the length and whitespace parameters in the **local.properties** configuration file.

The {{ }} property notation is extremely powerful and flexible and is not confined to using magic strings. For example, if we changed the '1' in the response above to DataSetValue and define a data set in the virtual service model, that data set would be used to generate the runtime response value. The dataset might be a random string generator, a counter, a call to a database, a reference to an Excel spreadsheet row, anything. {{ }} can execute arbitrary Java code and even generate realistic 'everyday' data such as a valid credit card number {{=:Credit Card:}}. Refer to the LISA User Guide, Chapter "Using BeanShell" for more information.

Magic Dates

Requests and responses are also scanned at recording time by a powerful date parser. Anything matching a very wide definition of date formats is recognized and translated into a 'Magic Date'. In the example above, the magic date is =doDateDeltaFromCurrent("yyyy-MM-dd","0");/ **2008-12-17**/ Note again the use of {{ }} notation.

At runtime, this is translated as 'generate a date of the format yyyy-MM-dd that is 0 days from the current date'. In other words, generate today's date.

If the original recording was taken on 1 February 2009 and the response contained a date 2009-02-10, then the magic date string would be =doDateDeltaFromCurrent("yyyy-MM-dd","10");/**2009-02-10**/. Note the '10' in the magic string; this means the VSE will generate a date in the response 10 days ahead of the current time. Thus if the VSE is in playback mode on 12 June 2010, the response will contain the string '2010-06-22'.

There is another variant of magic dates 'doDateDeltaFromRequest', as opposed to 'doDateDeltaFromCurrent'. The 'doDateDeltaFromRequest' variant is used when a date is used as a parameter in the request and a date is seen in the response. For example, an airline reservation system might accept a seating request for a particular flight on a particular day. If that date is seen in the response, the VSE will correctly substitute the date in any subsequent responses.

A more sophisticated example is if the flight request generated a response that detailed an international flight crossing the international date line - a flight from Los Angeles to Sydney would arrive two days later than the departure according to the calendar date even though the flight time is around 14 hours. In this example, the response would contain something like 'doDateDeltaFromRequest("yy-MM-dd", "2")' If VSE processed a similar request for a flight departing LAX on 19-june-2009, it would include the correct arrival date of 21-june-2009 in the response.

4. Understanding Transactions

4. Understanding Transactions

A transaction comprises a request and a response. In most cases, a transaction has only one response, such as in the case of an HTTP responder. However, the messaging protocol can return multiple responses (stored as a list) from a single request.

The following topics are available.

- [4.1 Stateless and Conversational Transactions](#)
- [4.2 Logical Transactions](#)
- [4.3 Match Tolerance](#)

4.1 Stateless and Conversational Transactions

4.1 Stateless and Conversational Transactions

Transactions are classified as either [Stateless](#) or [Conversational](#).

Stateless Transactions

Stateless transactions include no logical relationships between transactions. For example, HTTP and SOAP are stateless protocols. A stateless transaction always has a static response, regardless of what calls were made previously.

In a stateless conversation, each service call is independent of the others. For example:

1. What's the weather like in Dallas, TX? (Operation: weather; arg1-city)
2. What's the weather like in Atlanta, GA? (Operation: weather; arg1-city)
3. What will the weather be like in Dallas, TX tomorrow? (Operation: weather; arg1-city; arg2-date)

Conversational/Stateful Transactions

Conversations comprise stateful transactions and consist of the logical transaction that, if matched, starts a unique session and the information necessary to create and identify that session. A transaction in a conversation always depends on the context created by earlier conversation.

Here is an example of a stateful conversation that involves using an ATM.

1. Connect to the ATM. (Operation: log on)
2. What account do you want? (arg1-checking)
3. What's my balance? (Operation: balance)
4. Response.
5. Withdraw an amount of money. (Operation: withdrawal)
6. How much? (arg2-amount)
7. What's my balance? (Operation: balance)
8. Response (different based on the amount withdrawn in the previous request).
9. End session (Operation: log out)

A conversation (either during recording or playback) is identified by a unique string (for example, JSESSIONID used in HTTP or a GUID). LISA supports the following types of conversations:

Instance-based conversations: The protocol layer is responsible for identifying the unique string based on different instances of the client and that is used to identify server-side sessions for both recording and playback of the service image.

Token-based conversations: The token is generated by the LISA VSE using a string generation pattern stored with the conversation in the service image. After the token is generated, it operates the same as an instance-based conversation. Since token-based conversations cannot be automatically inferred during recording, you will need to use the VS Image Recorder to specify where tokens are found.

4.2 Logical Transactions

4.2 Logical Transactions

A logical transaction appears as a single node in a conversation. It comprises one *meta* transaction with one or more specific transactions.

When a logical transaction appears in the search for responding to a given request, the meta transaction is consulted to see if this transaction wants to respond to the given request. If the meta transaction decides to respond to the request, then all of the specific transactions are queried if they want to respond to the request. If none of the specific transactions want to respond to the request, then the response is taken from the meta transaction.

This logic can be expressed as the following pseudo code:

```

for each logical transaction \{

  if (meta transaction request matches the given request) \{
    // This node would handle the request
    for each specific transaction in this node \{
      if (the specific transaction matches the given request) \{
        return the response from the specific transaction
      \}
    \}
    // No specific transaction found for the given request
    return the response from the meta transaction
  \}
\}

```

Further, a physical meta transaction carries a list of specific transactions and a list of child meta transactions. The latter allows meta transactions to be structured into a decision tree.

4.3 Match Tolerance

4.3 Match Tolerance

The match tolerance of a transaction indicates how to match an incoming request against the one in the transaction. The types of tolerance include:

Type	Two requests match if...
Operation	Their operation names are the same.
Signature	An operation match occurs AND the list of argument names are the same. Argument values are not inspected.
Exact	A signature match occurs AND the argument values evaluate to equivalent. The match operator specifiable on every argument is only relevant for an exact match tolerance.

Match Operators

A match operator is specified for every argument in a request. For an **Exact** match operation, it controls how the value of the argument in the incoming request is matched against the value of the corresponding argument in the service image.

The following table describes the available match operators:

Type	Description
Anything	Always returns true. The virtual service recorder defaults the comparison to this when it determines that an argument is a date.
=	Equal. Returns true if the values are the same.
!=	Not equal. Returns true if the values are different.
<	Less than. Returns true if the inbound value is less, before, or earlier than the value from the service image.
<=	Less than or equal. Returns true if the inbound value is less, before, or earlier than or equal to the value from the service image.
>	Greater than. Returns true if the inbound value is greater, after, or later than the value from the service image.
>=	Greater than or equal. Returns true if the inbound value is greater, after, later than or equal to the value from the service image.
Regular Expression	Returns true if the inbound value (as a string) matches the value, as a regular expression, from the service image.
Property Expression	The value in the service image must be in the form of a double-braced script expression, <code>{ }</code> , which returns either true or false . The argument value in the inbound request is ignored unless referenced in the script.

Note: If an argument is marked as a date, the values from the requests being compared are first converted to a date before the comparison is done. This is not done for the Any, Property Expression or Regular Expression comparison types.

5. Recording a Service Image

5. Recording a Service Image

The following topics are available.

- 5.1 Virtual Service Model (.vsm)
- 5.2 Service Image
- 5.3 Identification of Conversations and Transactions
- 5.4 Virtual Service From WSDL


5.1 Virtual Service Model (.vsm)

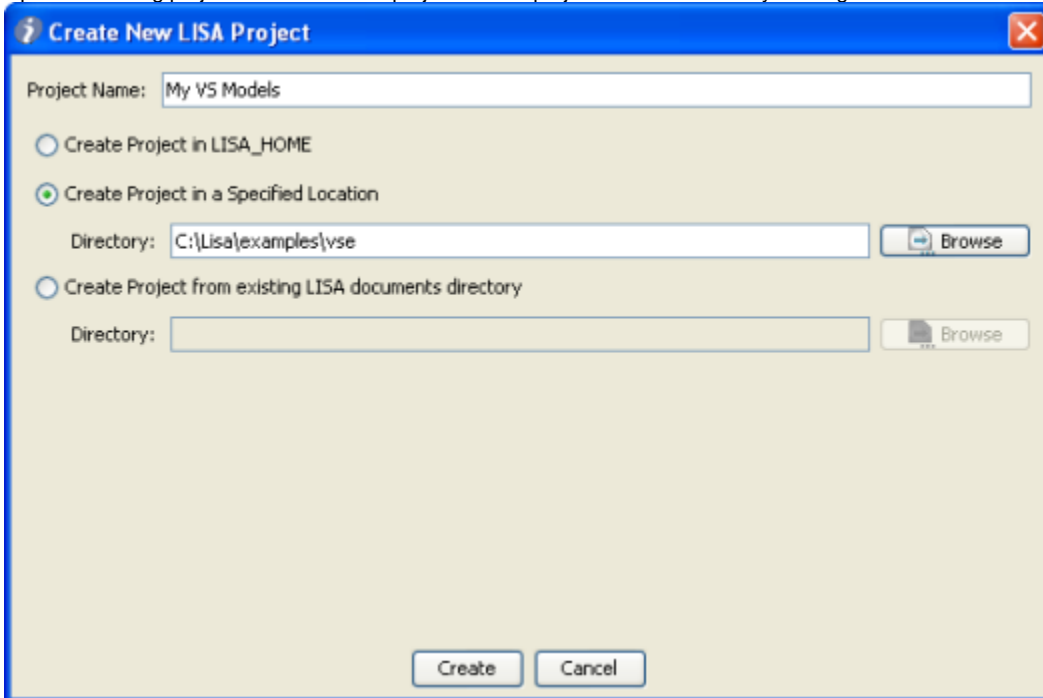
5.1 Virtual Service Model (.vsm)

A Virtual Service Model (VSM) is a specialized type of test case that becomes the end point of a service that has been virtualized. You must create a VSM to launch the Virtual Service Image Recorder.

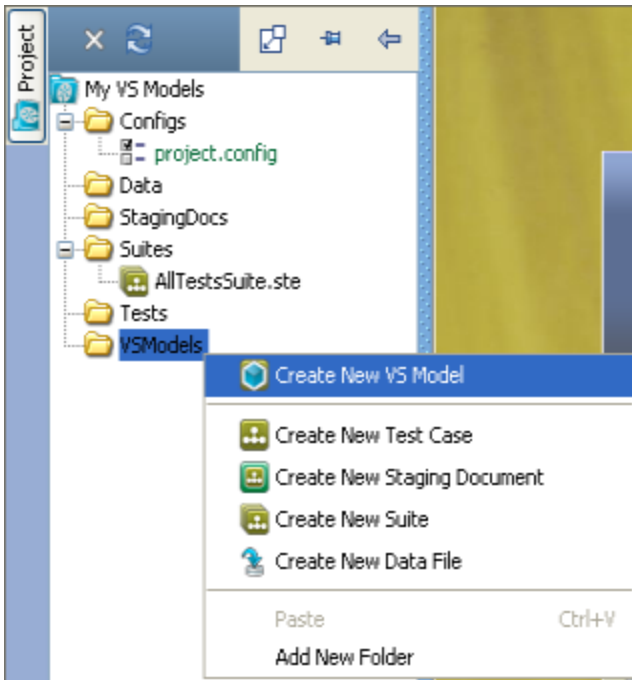
Creating a New VSM

You create a new VSM inside a LISA project.

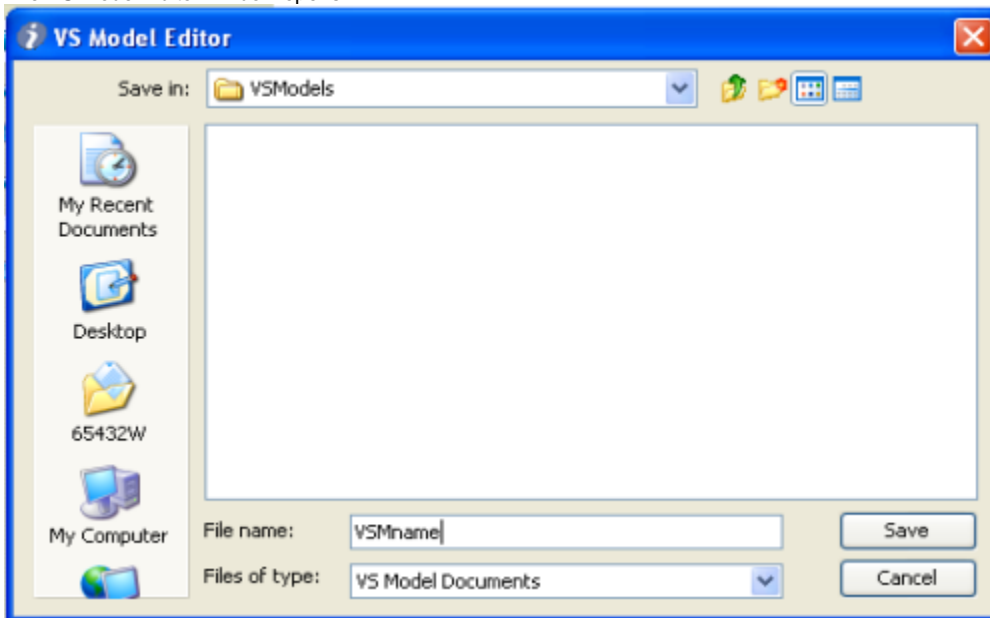
1. Open an existing project or create a new project. A new project can be created by clicking on the New  icon on the toolbar.



2. The project opens, and the left-hand pane shows the project folder and its subfolders. Right-click on the VSModels subfolder node and choose 'Create New VS Model'. Technically, you can create a VS Model in another folder as well. But it's a good practice to create it under the VSModels folder.



3. The VS Model Editor window opens.



4. In the VS Model Editor window, browse to the location for the new VSM.
5. In the File name field, enter a unique name for the VSM. The extension will be .vsm.
6. Click Save. The VS Model Wizard opens the new VSM.
7. Once a VSM is open, the Commands menu shows menu items relevant to a VSM.

Opening an Existing VSM

To open an existing VSM inside an existing project:

1. From the toolbar, click Open button to open the project, or choose it from the 'Open Recent' project list.
2. From the project pane that opens up on the left-hand side, select the VS model from its folder (usually VSMModels).

5.2 Service Image

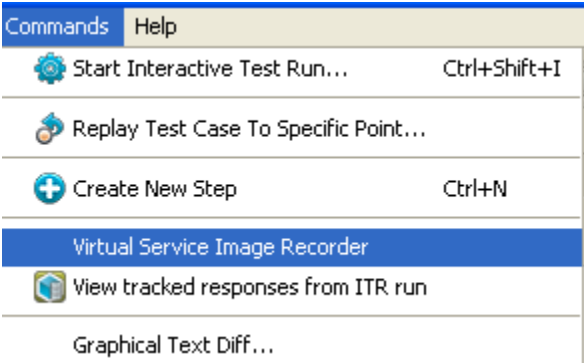
5.2 Service Image


Service images are generated using the Virtual Service Image Recorder and pretend to be what you recorded (a manipulated or altered version of your recorded raw traffic). You can view, edit, or create a service image by using the [Service Image Editor](#).

Due to their potentially large size, LISA stores service images in the internal Derby Reports database (**C:\Lisa\reports\lisa-reports.db**), which can be redirected to a different database instance by specifying the appropriate properties in the **local.properties** file as discussed in [VSE Installation and Configuration](#).

Virtual Service Image Recorder

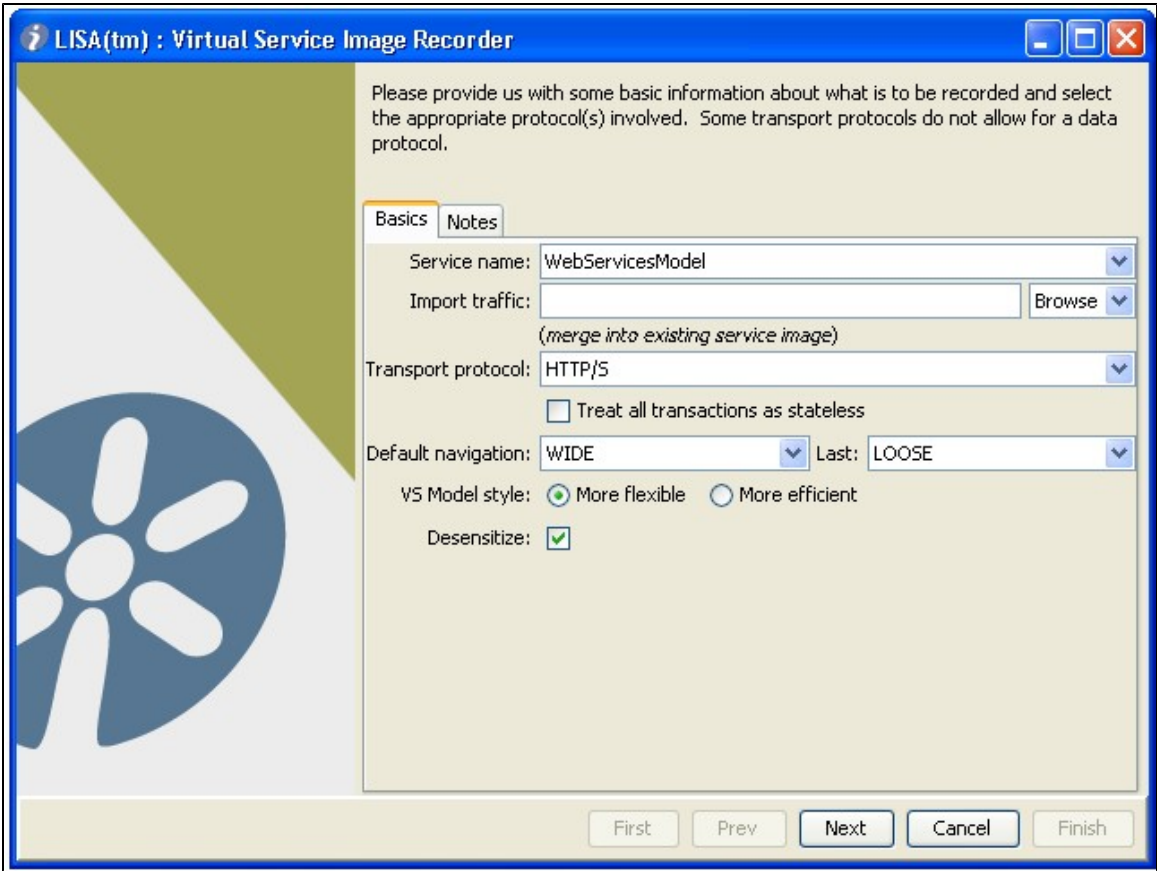
Access the LISA Virtual Service Image Recorder from the following menu: **Commands > Virtual Service Image Recorder**.



Alternately, you can click the test case toolbar button .

The Virtual Service Image Recorder opens a wizard in a new window. Initially, the Basics tab is shown.

Basics Tab



The first window in the VS Image Recorder wizard provides the name, protocol, and navigation options. Enter the information as needed in the fields depending on the protocol you are using. Subsequent windows are protocol specific. The Basics tab options are described below:

Field	Description
-------	-------------

Service name	A unique service image name.
Import traffic	Import a raw or conversational XML traffic file. This field could be blank if no such file exists. If a file is specified, then the transactions in the referenced XML document are merged into those resulting from the ensuing recording.
Transport protocol	Select one of the following options: <ul style="list-style-type: none"> • HTTP/S (for virtualizing a web server) • JDBC (for virtualizing a database) • IBM MQ Series (for virtualizing a messaging server connected to IBM WebSphere MQ series) • Standard JMS (for virtualizing a messaging server connected to any middleware that supports JMS API)
Data protocol	The Last Modified column lists the date and time when the virtual service image was last changed and saved.

The recorder understands several data protocols. Choosing the appropriate data protocol helps it to analyze the information it records to correctly distinguish the conversations from one another, and to identify transactions belonging to these conversations. If you selected HTTP/S as the transport protocol, select the appropriate option:

1. HTTP Traffic (web)---Default for HTTP/S transport protocol used in a web application typically using HTML.
2. Web Services (SOAP)---SOAP data protocol used in web services (written for consumption by a web service client).
3. Web Services Bridge---Data protocol for LISA Travel example. It is very specific to the example and would not be much useful in a general case. It can therefore be safely ignored.
4. Auto Hash Transaction Discovery – Identifies a message by the hash code of the data. Note that the hash code changes with even a slight change in the data, which effectively makes all requests unique. This is useful in case one will run exactly the same small set of requests against the service.
5. Generic XML Payload Parser – Identifies that the requests and responses are XML strings. If this protocol is used, then you can identify variables out of the XML messages that are used by the recorder.

The last three options are available for IBM MQ Series or Standard JMS transport protocols as well, in addition to the default processing.

JDBC does not allow a data protocol.

The usage of a [dynamic data protocol](#) is covered later.

Field	Description
Treat all transactions as stateless checkbox	This option is provided for very special cases where you may want to treat all recorded transactions as stateless. In most cases, you would leave this unchecked.
Default navigation	Choose the default navigation tolerance for all except the last (leaf) transactions. The default is WIDE.
Last	Choose the default navigation tolerance for the last (leaf) transactions. The default is LOOSE.
VS Model style	Set this option to indicate whether to generate a VSM including the prepare steps or not. The options are: <ul style="list-style-type: none"> • More flexible: Default. Include Prepare steps (leading to a five-step model in case of HTTP/S protocol). • More efficient: Prepare steps are absent (leading to a three-step model in case of HTTP/S protocol).

The available navigation buttons are:

First: Click to return to the first step.

Prev: Click to move to the previous step.

Next: Click to move to the next step.

Cancel: Click to close the wizard without saving.

Finish: Click to complete the recording and save the service image.

Clicking on the Notes tab will let you create any documentation for this service image.

[Next Steps - HTTP/S Protocol](#)

[Next Steps - JDBC Protocol](#)

[Next Steps - Messaging Protocols](#)

Next Steps – HTTP/S Protocol

1. If you are using HTTP as the Transport protocol in the Basics tab of the Virtual Service Image Recorder, complete the remaining wizard steps as given below.

LISA(tm) : Virtual Service Image Recorder

Please provide us with some basic information about what is to be recorded and select the appropriate protocol(s) involved. Some transport protocols do not allow for a data protocol.

Basics | Notes

Service name: WeBServicesModel

Import traffic: Browse

(create new service image)

Transport protocol: HTTP/S

☐ Treat all transactions as stateless

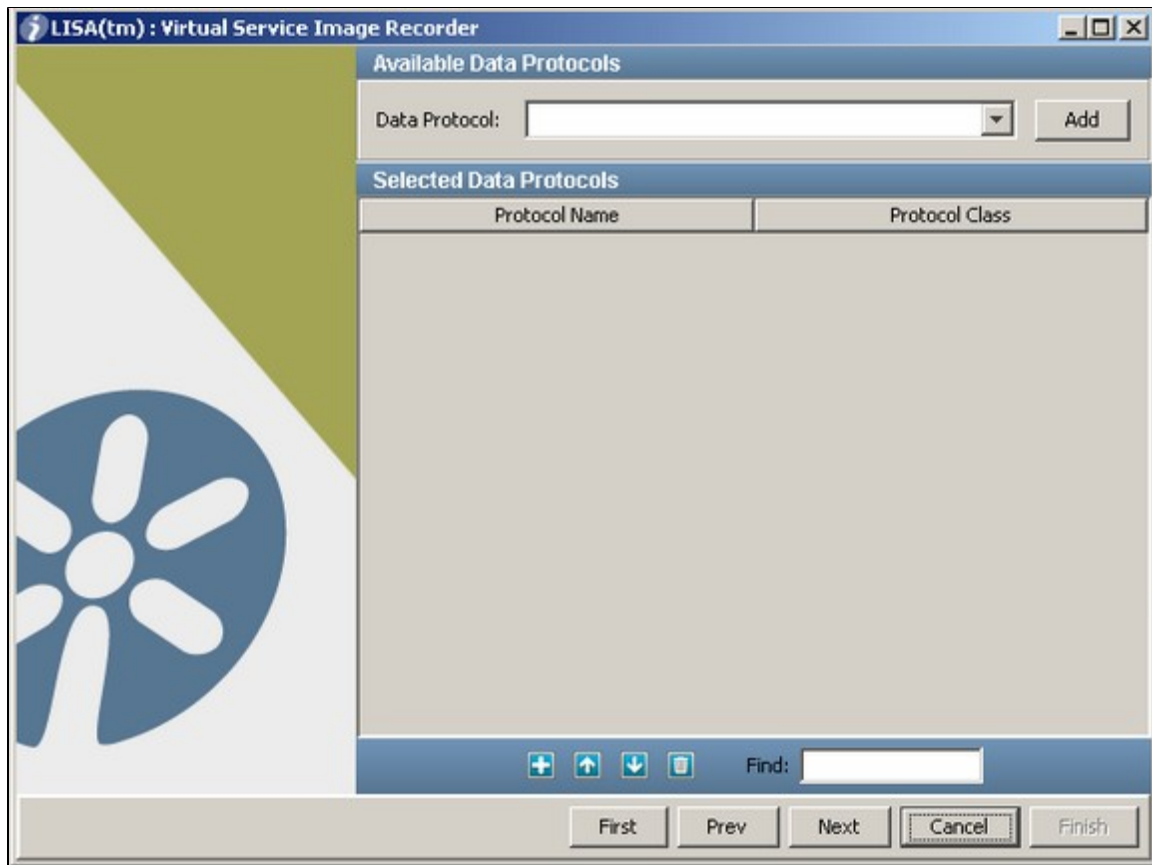
Default navigation: WIDE Last: LOOSE

VS Model style: ☒ More flexible ☐ More efficient

Desensitize: ☒

First Prev Next Cancel Finish

2. Click on **"Next"** on the recorder 1st screen.
3. Do NOT select any value for the data protocol on the recorder 2nd screen and click on **"Next"**.



4. Click on **"Next"** on the recorder 2nd screen.
5. Enter the port and host information on this step.

LISA(tm) : Virtual Service Image Recorder

Please provide us with the port your consumer will talk to us on, the target server's host name and port and how we should forward requests during recording. Use gateway unless your HTTP/S client requires the use of a proxy.

Listen/Record on port:

Target host:

Target port:

Recorder passthru style: ☒ Gateway ☐ Proxy

☒ Use SSL to server

☒ Use SSL to client

SSL keystore file:

Keystore password:

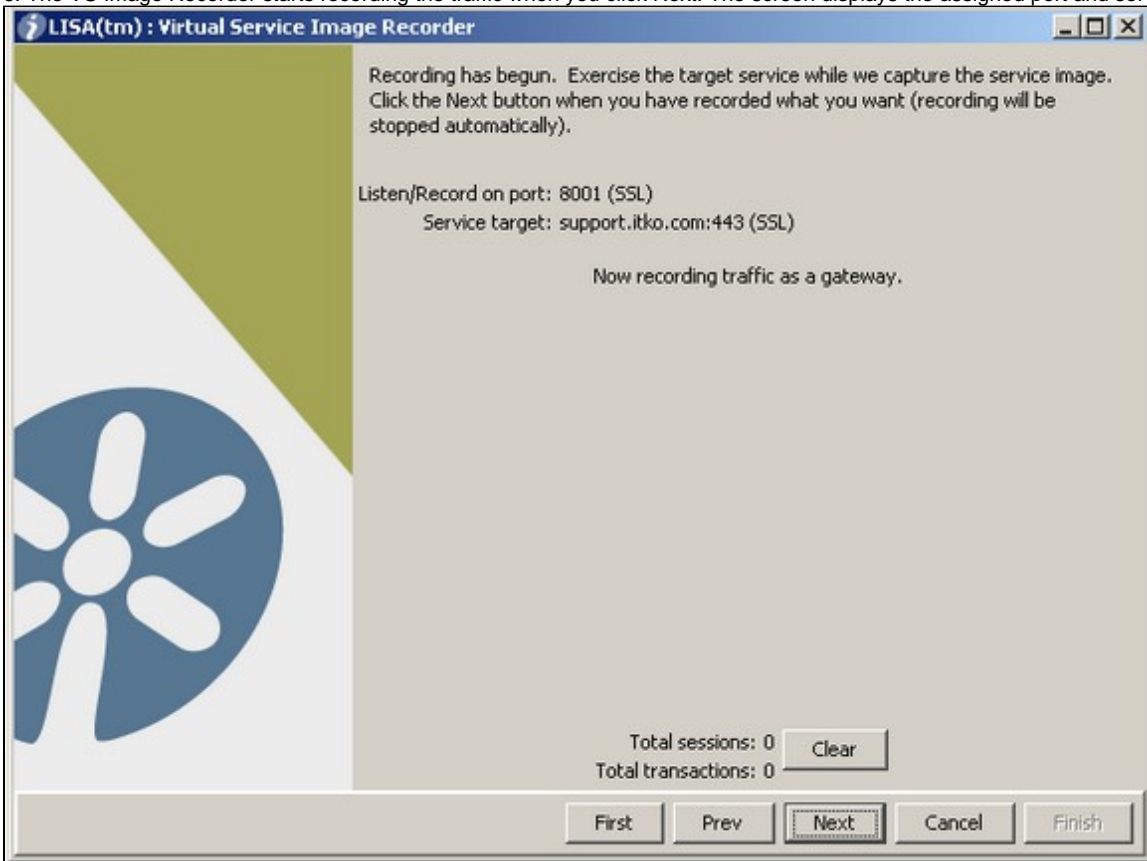
Recording will start automatically when Next is clicked.

The options are described below:

Field	Description
Listen/Record on port	Provide the port on which your consumer communicates to LISA. It is typical to choose 8001, but you can choose another port number.
Target host	Enter the name or IP address of the target host where the server is running. Leave blank if you are going to choose a Proxy pass-through style.
Target port	Enter the target port number listened to by the server. The defaults are 80 (HTTP) and 443 (HTTPS). Leave blank if you are going to choose a Proxy pass-through style.
Recorder pass-through style	<p>Indicate how VS Image Recorder will act during recording. The choices are Gateway and Proxy. If you select Proxy, the contents in Target host and Target port fields are cleared and the fields become disabled. This choice impacts how the client connects in the recording mode.</p> <ul style="list-style-type: none"> • If VS Image Recorder would listen in a gateway mode, the client will need to send http requests directly to the recorder and not to the server. (If the client is a browser, the URL will contain the host and port of the the recorder instead of that of the server.) • If VS Image Recorder would listen in a proxy mode, then the client will need to specify the recorder host and port as the proxy. (If the client is a browser, then the URL will contain the host and port of the server, but the proxy settings need to be set to route the request through the recorder.) <p>Note: Most HTTP clients have a setting for NOT using proxy for localhost. If your VS Image Recorder is running on localhost in proxy mode, then you will need to disable this setting for the traffic to get correctly passed via the recorder.</p>
Use SSL to server	If checked, sends HTTPS (secured layer) request to the server.
Use SSL to client	Check whether we can playback an HTTPS request from client using a custom client keystore. When the user specifies "Use SSL to client", it specifies to select a custom keystore, and a passphrase. If these are entered, LISA will use them rather than the hard-coded defaults.
SSL keystore file	Name of the keystore file.

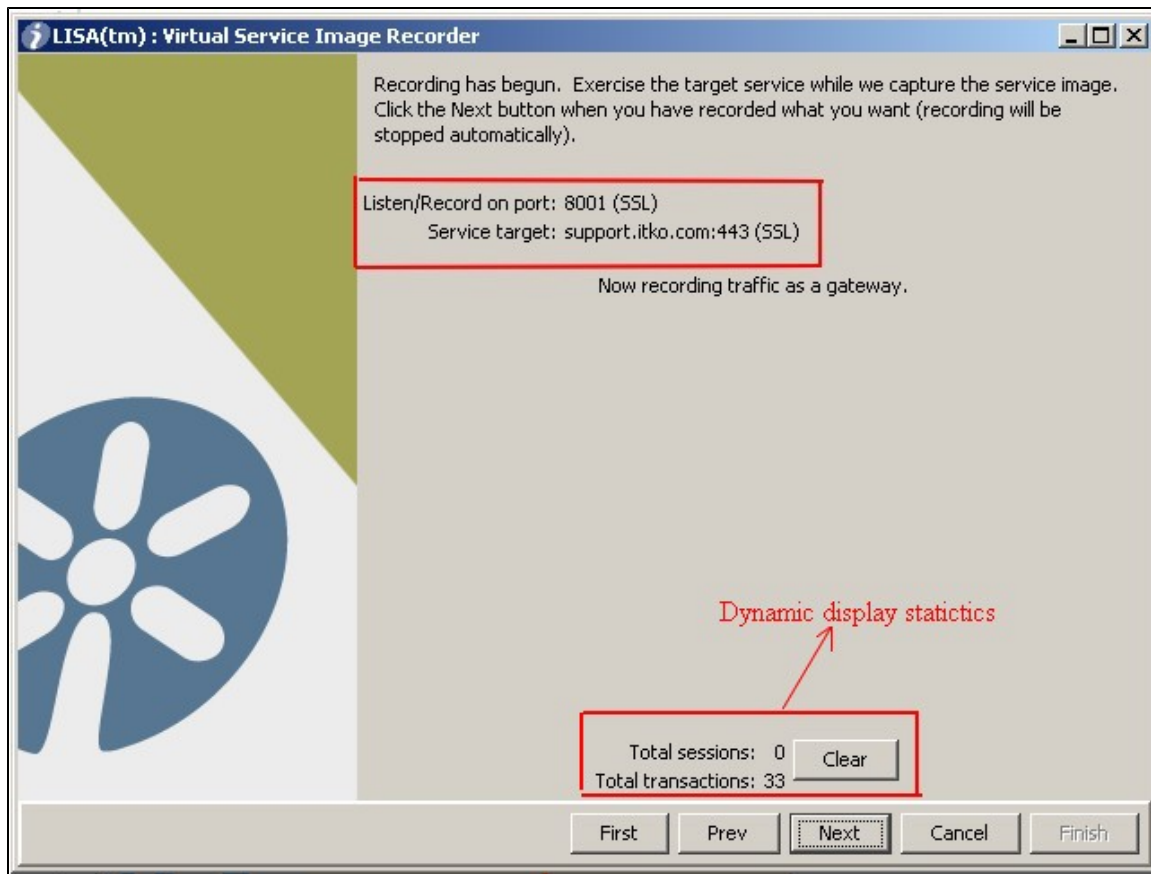
Keystore password	Password for the keystore file.
-------------------	---------------------------------

6. The VS Image Recorder starts recording the traffic when you click Next. The screen displays the assigned port and service target.



7. Start recording the traffic now. Use your HTTP client to send the requests to the server routed via the VS Image Recorder.

8. As the VS Image Recorder records transactions, they are reflected in the **dynamic display statistics** on the lower portion of the screen.



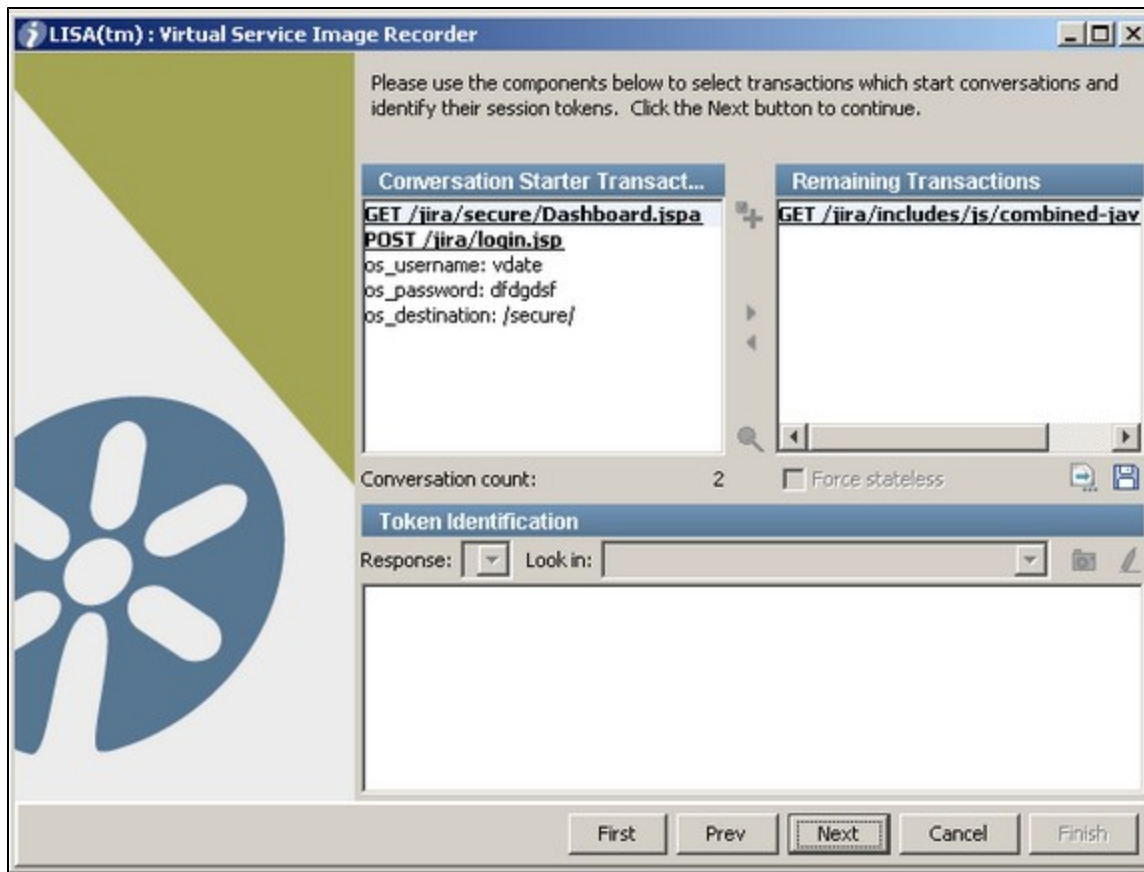
The options and dynamic display statistics are described below:

Field	Description
Total conversations	Indicates the number of conversations recorded.
Total transactions	Indicates the number of transactions recorded.
Clear	Click this button to clear out currently recorded transactions.
Next	When you have completed the recording, click to move to the next step. If you click Next and no transactions have been recorded, an error message appears. Click OK to continue recording.

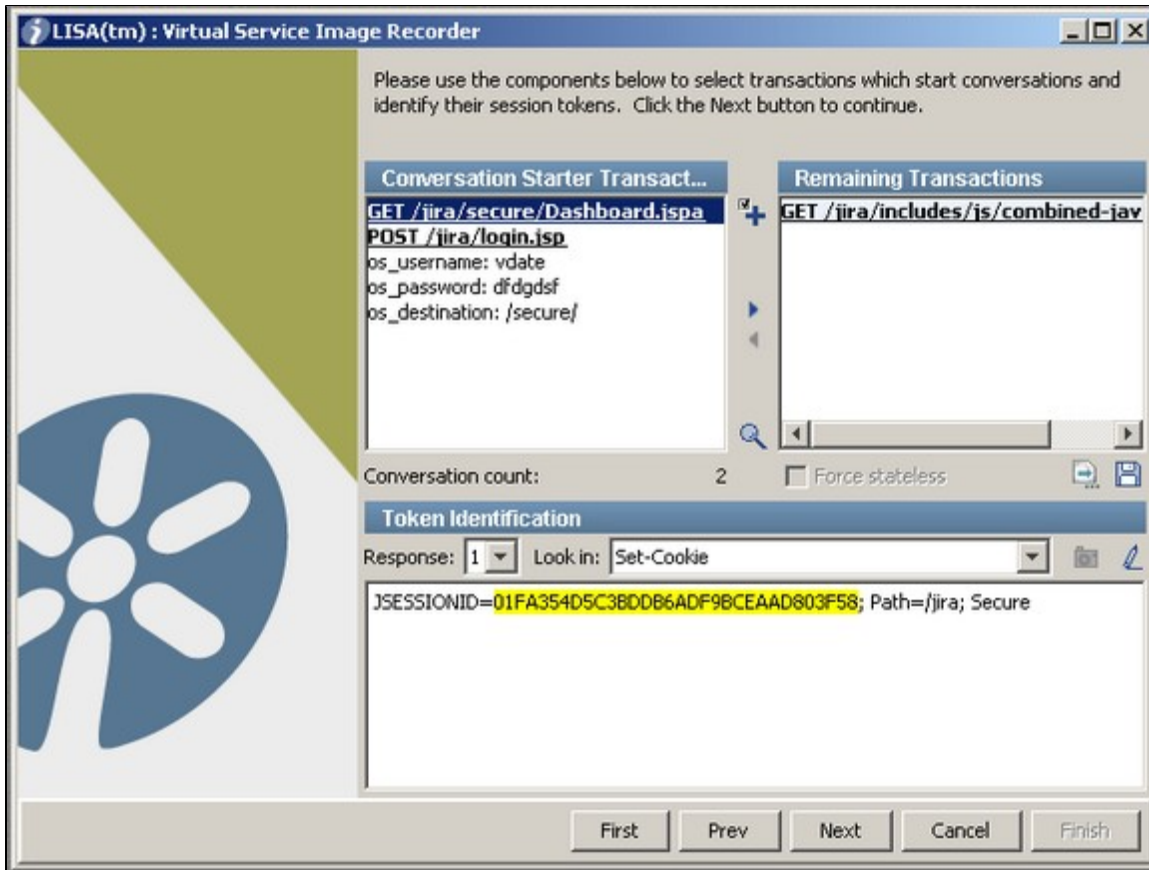
If transactions are not recorded...

You may have a port conflict, or the client sends transactions to the application instead of the Virtual Service Recorder. If another service is using that port, either stop that service or change the port setting so there is no longer a conflict.

9. After you record traffic, click **Next**.



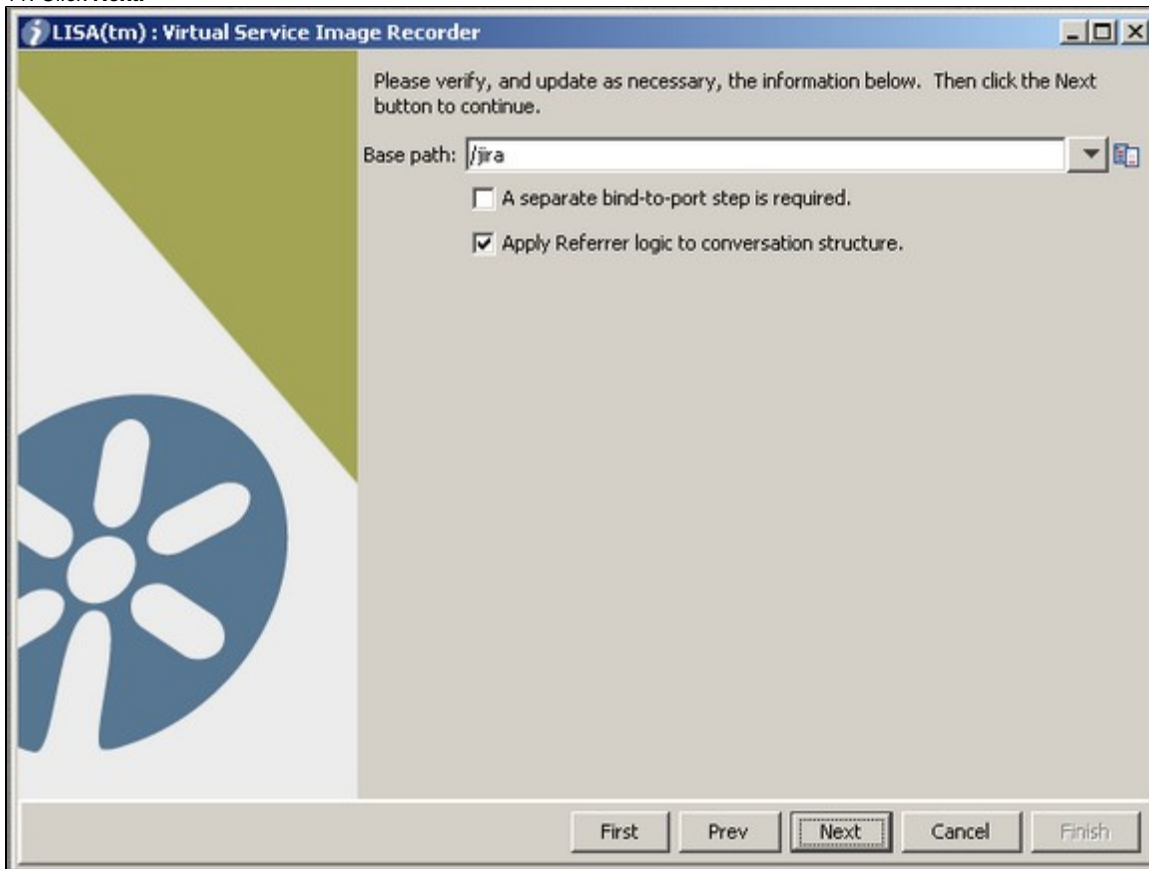
10. If no conversations were detected during the recording process, select transactions that start conversations. For token-based conversations, you must specify where tokens can be found. Use the **Token Identification** area in the VS Image Recorder wizards. Select transactions that start conversations and identify the session tokens. Select the **GET** Conversation Starter Transaction listed.



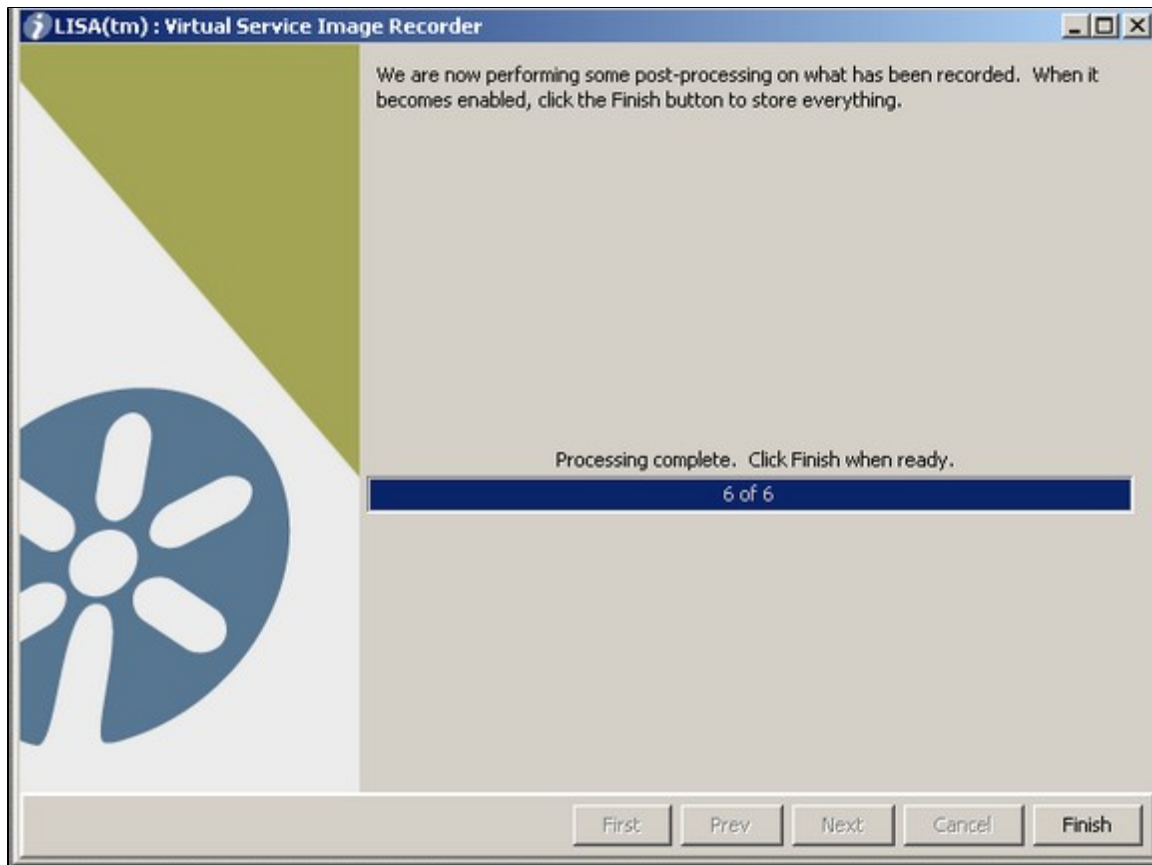
The step components are described below:

Field	Description
Conversation Starter Transactions	Lists the transactions you have selected as conversation starters. Select a transaction and click the arrow to move the transaction to the Remaining Transactions list if you do not want it to be a conversation starter.
Remaining Transactions	Lists the recorded transactions. Select a transaction and click the arrow to move the transaction to the Conversation Starter Transactions list.
Plus icon	Selects all transactions (either Conversation Starters or Remaining) in the list that are like a selected transaction. Use the appropriate arrow button to move all the selected transactions.
Conversation count	Displays the number of conversations in the recording. As you build conversations, the number is incremented.
Force stateless	From the Remaining Transactions list, select any transactions that should stay stateless and click the checkbox. For example, you may decide that a transaction that includes an image should stay stateless even though it contains a conversation starter token.
Stateless Transactions	Click to see a list of all transactions that will remain stateless, assuming the identified conversations on this panel. You can use the list to verify that you have identified all the conversation starter transactions.
Save	Click to save the raw recorded transactions. Click Browse to navigate to the location in which to save the file. You can import the raw traffic recording in the Basics tab before beginning a new recording.
Response	For the currently selected transaction, this identifies which of its responses to look in. In general, 1 is the only option.
Look in	Identifies the piece of the response you want to see when looking for conversation tokens. The drop-down list contains an entry for each of the response's meta data entries plus one for the response's body.
Token Identification area	Based on the selected transaction and response, the content of the selected Look in section of the response is displayed here. To mark a piece of the text as a conversation token, select the text and click the mark icon. The text is then highlighted in yellow. To mark the text as no longer a conversation token, either mark a different piece of the text or click the erase icon. After you mark a token, you can use the search icon to find similar transactions and mark their tokens. Click the search icon to open a dialog where you can highlight text (such as XML tags) that bound the conversation token, thus specifying the leading and trailing text to search for.

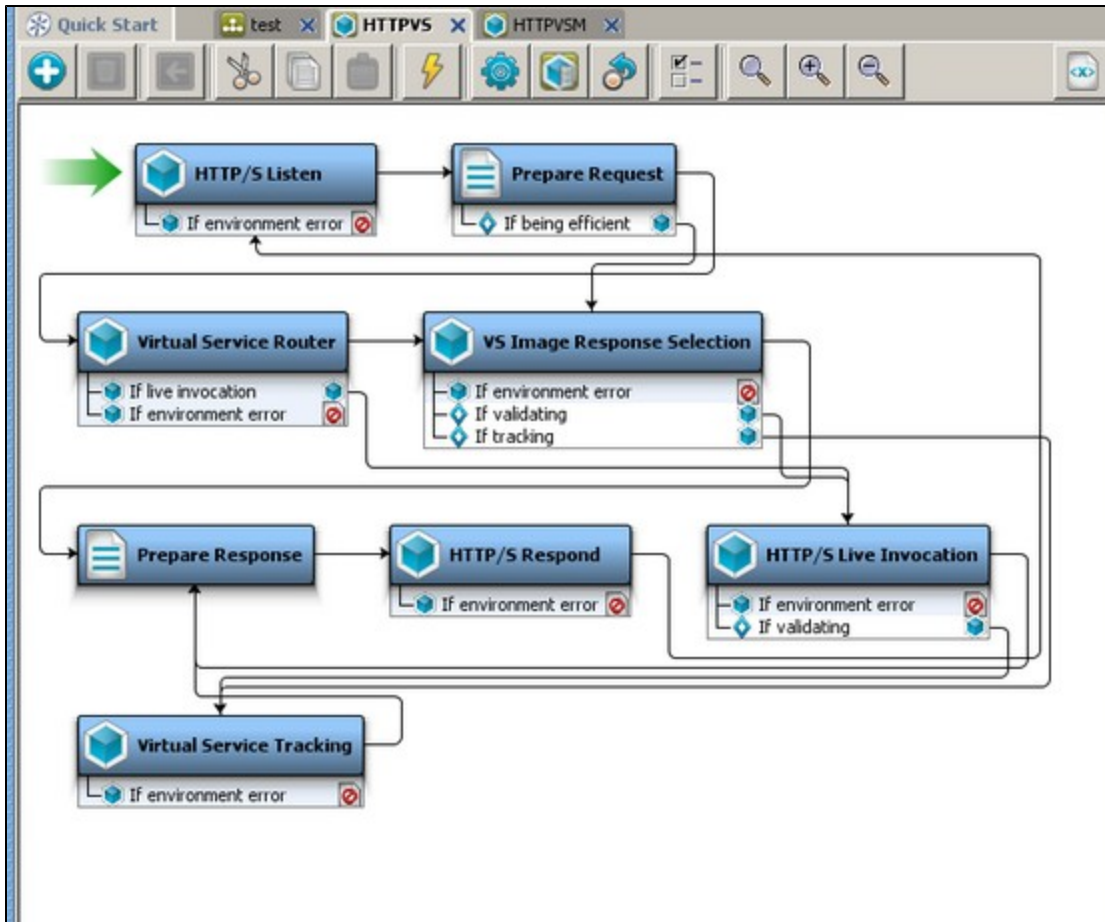
11. Click **Next**.



12. After you have identified the conversation starter tokens, verify the base path. Update the base path as needed. If needed, check the box to require a bind-to-port step before processing requests. Click **Next**.



13. During post-processing, the VS Image Recorder displays the processing status.
14. The recorder completes post-processing the recording. Click **Finish** to store the image.
15. In LISA Workstation, review and save the .vsm.



[Next Steps List](#)

[Return to Next Steps List](#)

Next Steps – JDBC Protocol

1. If you are using JDBC as the Transport protocol in the Basics tab in the Virtual Service Image Recorder, complete the remaining wizard steps as given below.

LISA(tm) : Virtual Service Image Recorder

Please provide us with some basic information about what is to be recorded and select the appropriate protocol(s) involved. Some transport protocols do not allow for a data protocol.

Basics | Notes

Service name:

Import traffic:

(create new service image)

Transport protocol:

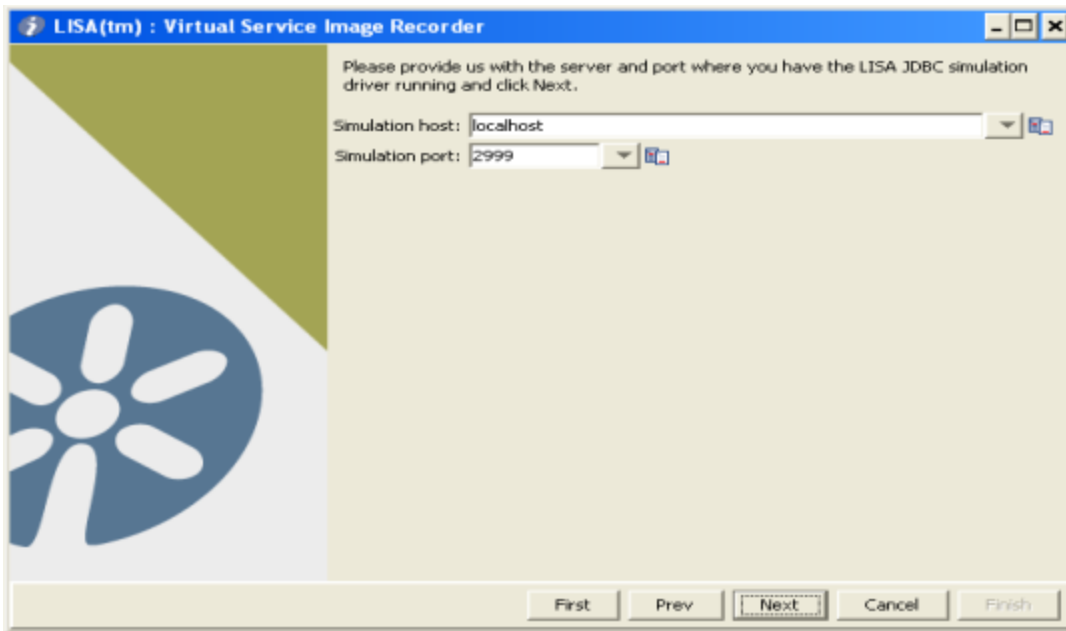
Data protocol:

☐ Treat all transactions as stateless

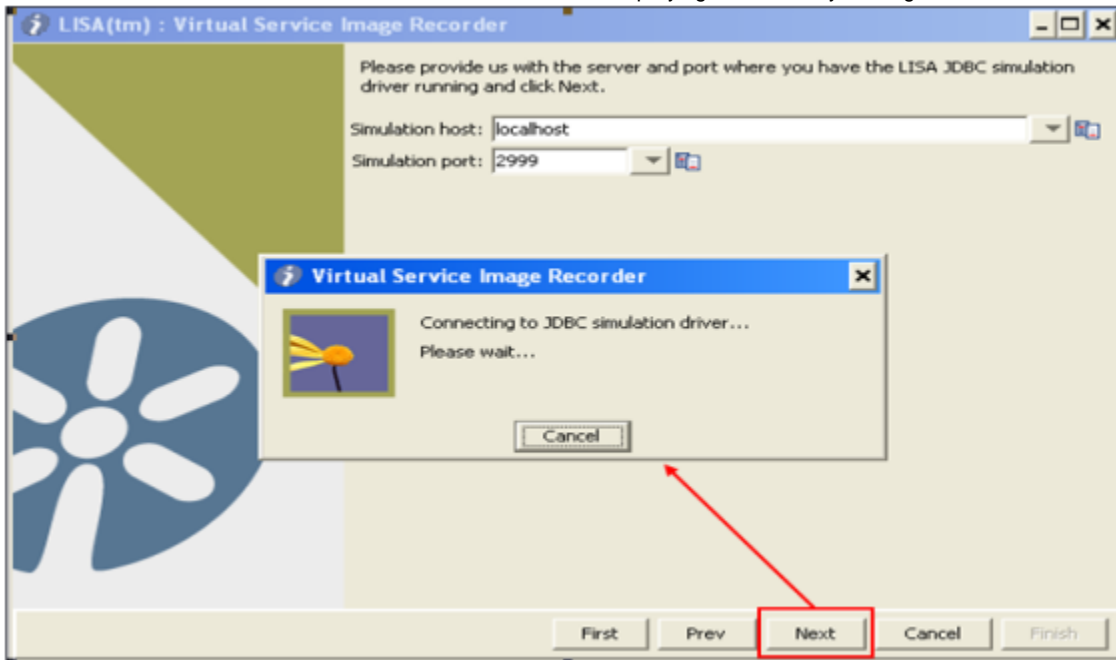
Default navigation: Last:

VS Model style: ☒ More flexible ☐ More efficient

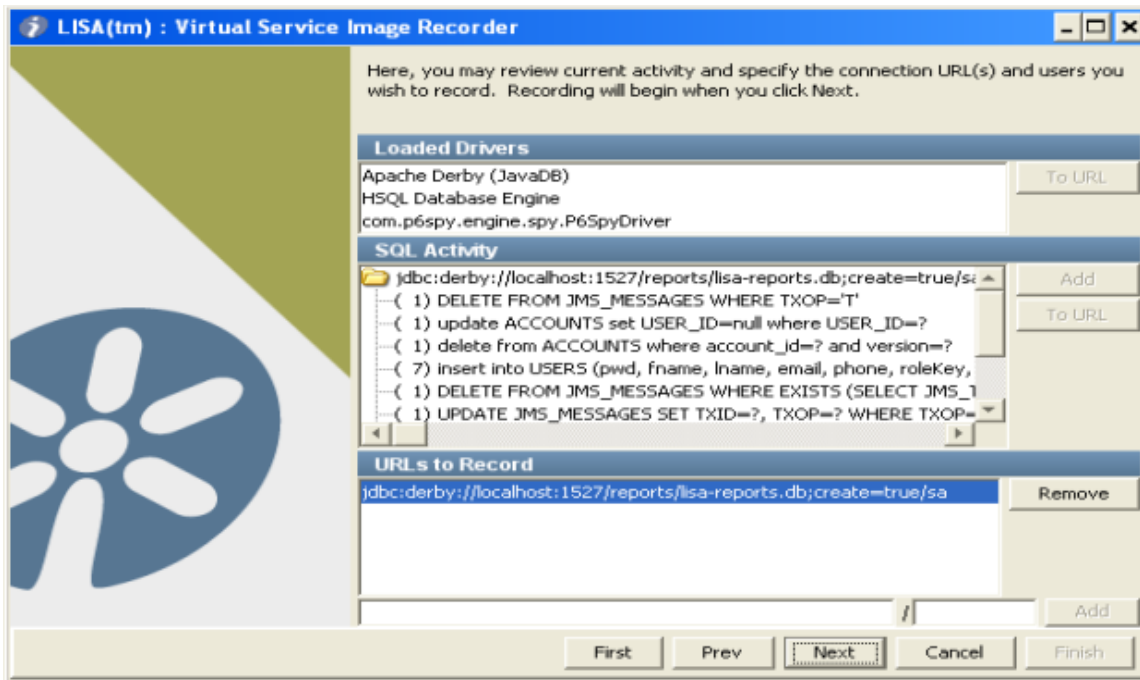
2. Set up the simulation host (default is localhost) and port (default is 2999) information, as appropriate.



3. Click **Next** to connect to the JDBC Simulation server. You can stop trying to connect by clicking **Cancel** in the status window, if needed.



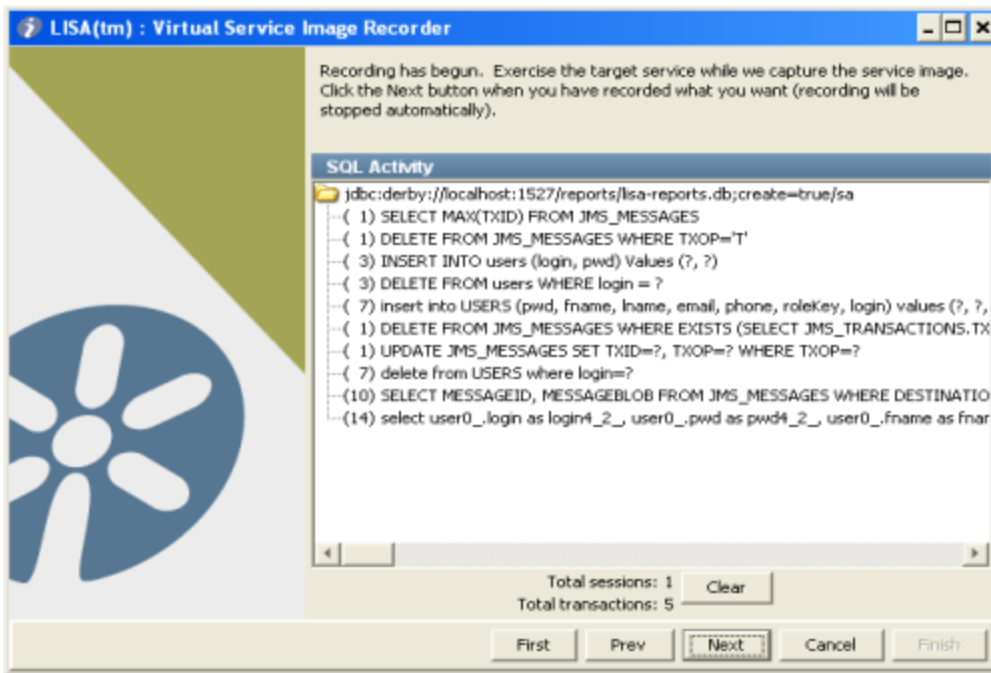
4. Specify connection URLs and users to record.



The options are described below:

Field	Description
Loaded Drivers	Lists the loaded drivers. By selecting a driver and clicking To URL, a regular expression that matches any connection through that driver is placed in the URL field at the bottom of the panel. Then click Add to add that pattern to the list of URLs to record.
SQL Activity	Select the URLs and users you want to record. Click To URL to place the selected connection string and user into the URL and user fields at the bottom of the panel. Click Add to put the URL in the URLs to Record list.
URLs to Record	The URLs added from the SQL Activity list or the URL/User entry fields are displayed. To delete an entry, select it and click Remove.
User	This is an unnamed area below the URLs to Record field where you can include a database user along with the URL. If you do not include a database user by leaving this blank, then the recording will be done for all users connecting to the URL. Note: You must supply a connection URL in the first box and a username in the next box before the Add button is enabled.

5. Click **Next**.



The options and dynamic display statistics are described below:

Field	Description
Total conversations	Indicates the number of conversations recorded.
Total transactions	Indicates the number of transactions recorded.
Clear	Click to clear out currently recorded transactions.
Next	When you have completed the recording, click to move to the next step. If you click Next and no transactions have been recorded, an error message will appear. Click OK to continue recording.

6. Record traffic. The number of Total conversations and Total transactions should increment with the number of transactions being recorded.

Note: If transactions are not recorded...

You may have a port conflict. The client sends transactions to the application instead of the Virtual Service Recorder. If another service is using that port, either stop that service or change the port setting so there is no longer a conflict.

If you face performance issues...

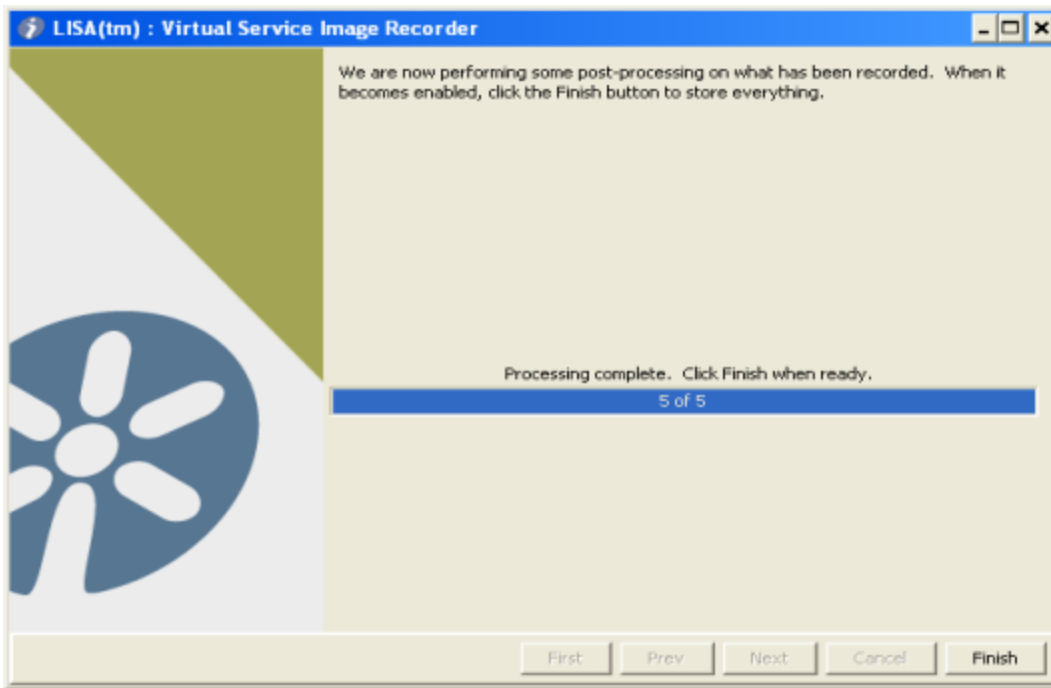
It is possible that the target database is responding slowly. Please check the **LISA_HOME/tmp/tmanager.log** file for debug messages which report the query execution time. A sample message is:

```
2009-07-01 15:35:39,248 [AWT-EventQueue-0] DEBUG com.itko.lisa.vse.stateful.model.SqlTimer - Exec time 72ms : SELECT TRAFFIC_ID, LAST_MODIFIED, SERVICE_INFO, CREATED_ON, NOTES, UNK_CONV_RESPONSE, UNK_STLS_RESPONSE FROM SVSE_TRAFFIC
```

Warning messages of the following kind are generated if the query time is above a threshold:

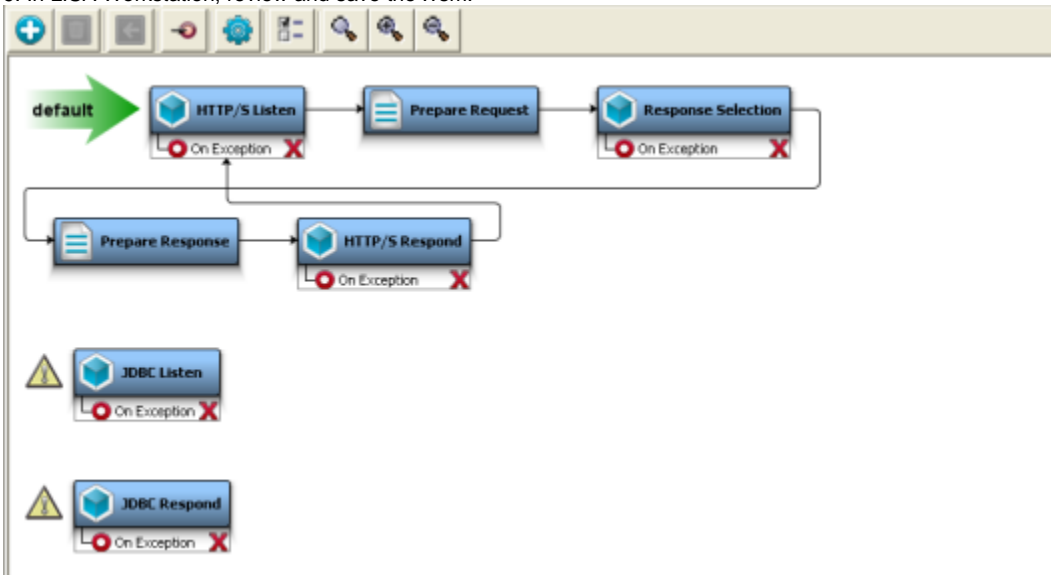
```
2009-07-01 15:17:11,161 [AWT-EventQueue-0] WARN com.itko.lisa.vse.stateful.model.SqlTimer - SQL query took a long time (112 ms) : SELECT TRAFFIC_ID, LAST_MODIFIED, SERVICE_INFO, CREATED_ON, NOTES, UNK_CONV_RESPONSE, UNK_STLS_RESPONSE FROM SVSE_TRAFFIC
```

7. If you are finished recording, click **Next**.



8. The recorder completes post-processing the recording. Click **Finish** to store the image.

9. In LISA Workstation, review and save the .vsm.



[Return to Next Steps List](#)

Next Steps – Messaging Protocols (IBM MQ Series and Standard JMS)

1. If you are using IBM MQ Series or Standard JMS as the Transport protocol in the Basics tab in the Virtual Service Image Recorder, complete the remaining wizard steps as given below.

LISA(tm) : Virtual Service Image Recorder

Please provide us with some basic information about what is to be recorded and select the appropriate protocol(s) involved. Some transport protocols do not allow for a data protocol.

Basics | Notes

Service name: JMS

Import traffic: Browse

(merge into existing service image)

Transport protocol: Standard JMS

☐ Treat all transactions as stateless

Default navigation: WIDE Last: LOOSE

VS Model style: ☒ More flexible ☐ More efficient

Desensitize: ☐

First Prev Next Cancel Finish

2. Click on "Next" on the recorder 1st screen.

3. Do NOT select any value for the data protocol on the recorder 2nd screen and click on "Next".

LISA(tm) : Virtual Service Image Recorder

Available Data Protocols

Data Protocol: Add

Selected Data Protocols

Protocol Name	Protocol Class

Find:

First Prev Next Cancel Finish

4. Set up the proxy queue and destination queue names. Remember to check the **Sessions** check box and use correlation ID to uniquely identify a session. (For this to work, the client needs to set a unique correlation ID to identify all requests in the session, and the server needs to return it in the response.) The alternative is to track by the message ID.

Note: In case of MQ, a 'Create Proxy Queue' check box appears. It is possible to create a temporary queue on the fly to be used as a proxy queue. Check it if you have not manually created the proxy queue on MQ. This option is not available in case of JMS, which means that you need to create the proxy queues manually on the MOM server.

LISA(tm) : Virtual Service Image Recorder

Provide the connection details for the request Destination to listen for messages and the Destination to forward the recorded messages to.

Destination Info | Connection setUp | Advanced setUp

Proxy Queue:

Forward To:

Forward Type:

Transactions:

☒ Sessions: ☒ Correlation ID ☐ Message ID

First Prev Next Cancel Finish

LISA(tm) : Virtual Service Image Recorder

Provide the connection details for the request Destination to listen for messages and the Destination to forward the recorded messages to.

Destination Info | Connection setUp | Advanced setUp

Proxy Queue:

☒ Create Proxy Queue

Forward To:

Forward Type:

☒ Track By: ☒ Correlation ID ☐ Message ID

First Prev **Next** Cancel Finish

5. Go to the **Connection Setup** tab and enter the connection parameters useful to connect to the MOM. These connection parameters are internally saved to save typing the next time. The Connection Setup tabs are different for MQ and JMS.

The Connection Setup tab for MQ looks like:

LISA(tm) : Virtual Service Image Recorder

Provide the connection details for the request Destination to listen for messages and the Destination to forward the recorded messages to.

Destination Info | **Connection setUp** | Advanced setUp

Host Name: localhost

TCP/IP Port: 1414

Channel: S_istanbul

Queue Manager: QM_istanbul

User:

Password:

Alt QManager:

Client Mode: Native Client

Main | Advanced

First Prev Next Cancel Finish

The Connection Setup tab for JMS looks as below:

LISA(tm) : Virtual Service Image Recorder

Provide the connection details for the request Destination to listen for messages and the Destination to forward the recorded messages to.

Destination Info | Connection setUp | **Advanced setUp**

JNDI Factory Class: org.jnp.interfaces.NamingContextFactory

JNDI Server URL: jnp://localhost:1099

JMS ConnectionFactory: QueueConnectionFactory

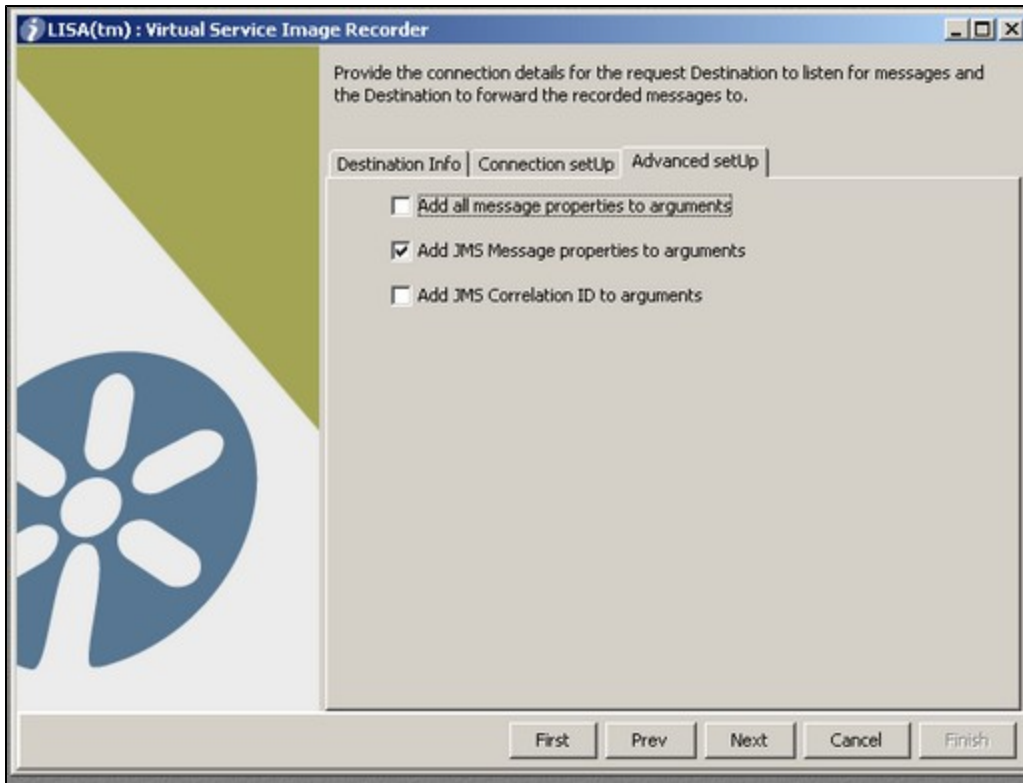
User:

Password:


Main | Advanced

First Prev Next Cancel Finish

6. In the **Advanced Setup** tab, in case of JMS messaging, you can help LISA Virtualize by identifying which arguments in the JMS message header it can use to identify the transactions differently from each other.



You will not see any options enabled in case of an MQ connection using native protocol. This is because it is not possible to set custom header parameters.

7. Click **Next**. On this tab, we set the response and response proxy queues. You may remember that in case of messaging, more than one response is possible for a given request. Use the  plus button to register a set of response and response proxy queues. The registered queues are visible in the list in the bottom portion of this screen. Before registering the response queue that you want to use for unknown requests, check the **Use for Unknown Responses** check box.

8. As you would see, in case of MQ you will find **Create Proxy Queue** check box which would be absent for the standard JMS provider.

The one for MQ looks like:

LISA(tm) : Virtual Service Image Recorder

Provide the connection details for the response Destinations to listen for messages and then the Proxy Queue the client application would like to receive these responses on.

Listen Destination:

Listen Type:

Proxy Queue:

☒ Create Proxy Queue

☐ use temporary queue/topic

Queue Model:

☒ Use For Unknown Responses:

Destination Name	Type	Connection Info	Message Count
ORDERS.RESPONSE	Queue - ASYNC	172.16.236.128:1...	Waiting...

Find:

Destination List | Current Connection Info

First Prev Next Cancel Finish

The one for JMS looks as below:

LISA(tm) : Virtual Service Image Recorder

Provide the connection details for the response Destinations to listen for messages and then the Proxy Queue the client application would like to receive these responses on.

Listen Destination:

Listen Type:

☐ use temporary queue/topic

Proxy Queue:

☐ Use For Unknown Responses:

Destination Name	Type	Connection Info	Message Count
queue/RESPONSE	Queue - ASYNC	jnp://localhost:109...	Waiting...

Find:

Destination List | Current Connection Info

First Prev Next Cancel Finish

9. It is possible to go to the **Current Connection Info** tab to ensure that the connection info is correct. It is copied from the connection info that was given earlier, and you may want to change it only in a very rare case when the response connection information is different.

Adding a temporary response destination

A 'temporary' destination in messaging is a destination created on demand for a messaging client. This is typically used in request-response scenarios:

- Before sending a request message the client creates a temporary destination to receive the response and sets the 'replyTo' value on the request message to point to their temporary destination. The server application receives the request, reads the replyTo value, and sends the response to that destination. Once the client receives the response it closes the temporary destination.
- The use of temporary destinations remove the need to have a separate administratively defined response queue. It also separates clients so each client can only receive responses for its own requests.

Key to the Message VSE Recorder's support for temporary queues is the 'replyTo' field. If the recorder encounters a request message with a replyTo field, then it has to take the following extra steps:

1. Look through its list of response recorders to see if the replyTo destination corresponds to a proxy destination that a recorder is already handling. If that's the case, then change the message's replyTo value so that the response messages goes through that recorder before being forwarded back to the client.
2. If no recorder was found handling the exact replyTo destination, then look through the list of response recorders for one marked for temporary destinations. If the client is using a temporary destination, then the first search will always fail and we will always use a response recorder marked for temporary destinations. The temp recorder will first be reset; any temporary destination it was already using will be closed and it will open a new temporary destination to receive responses from the server. Also, it will store the message's original replyTo destination so the response can be forwarded properly. After that it's same as with a non-temporary recorder; the message's replyTo value is set to be the recorder's response destination and it's forwarded on to the server's request destination.
3. If no matching temporary or non-temporary recorders are found then an error is logged.

This process for handling replyTo and temporary destinations has a couple of implications:

- You can have any number of non-temporary response recorders and one temporary response recorder and the recording process should always pick the most appropriate one to receive the response message.
- It doesn't make sense to have more than one temporary response recorder, unless you are recording from multiple separate messaging providers or servers.
- Proxy recording is **required** to support temporary destinations. *Bridge* recording is where messages are copied to another destination for us automatically by the messaging provider. If we ever add support for bridge recording then we **cannot** support bridge recording and temporary destinations at the same time.

Provide the connection details for the response Destinations to listen for messages and then the Proxy Queue the client application would like to receive these responses on.

Listen Destination:

Listen Type:

☒ use temporary queue/topic

Proxy Queue:

☐ Use For Unknown Responses:

Destination Name	Type	Connection Info	Message Count
queue/RESPONSE	Queue - ASYNC	jnp://localhost:109...	Waiting...
	Queue - ASYNC	jnp://localhost:109...	Waiting...

Find:

Destination List | Current Connection Info

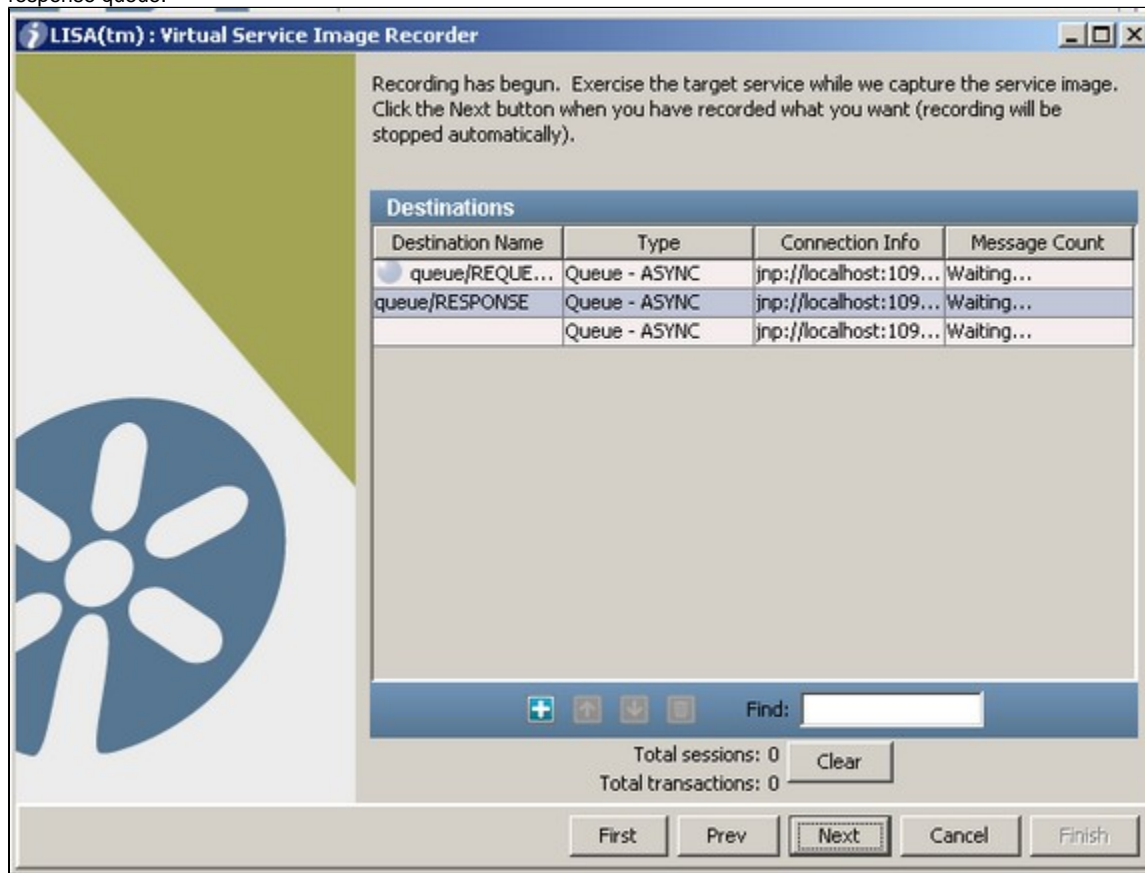
First Prev Next Cancel Finish

There is a new check box on the Response wizard page for temporary destinations. Checking this will disable the destination name and proxy destination name text fields and mark the response listener you're adding as a temporary response. The destination name will be blank.

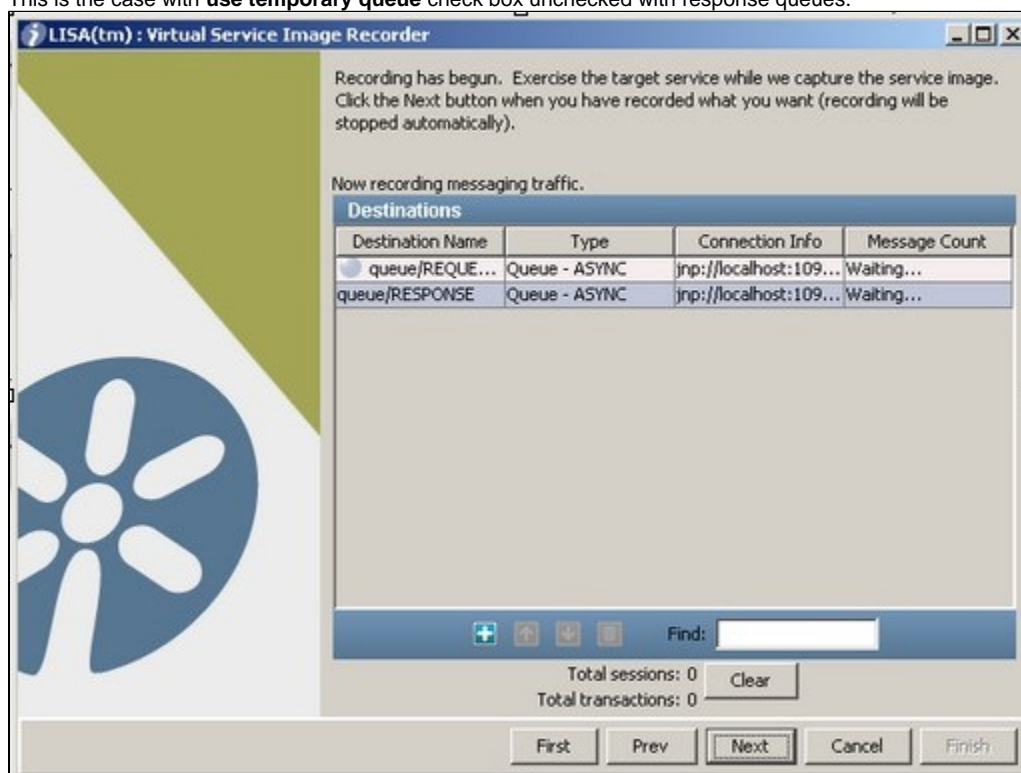
- Note: The only reason to add more than one temporary response to a recording session is if you're recording temporary destinations on multiple JMS providers or servers.

10. Click the **Next** button to begin recording. You will see the names of the queues to which LISA Virtualize is listening, and hence the status is Waiting. If you are connecting to MQ and have chosen to create the proxy queues on the fly, then at this point those proxy queues would be created.

This is the case with **use temporary queue** check box checked with response queues. The one without a destination name is the temporary response queue.



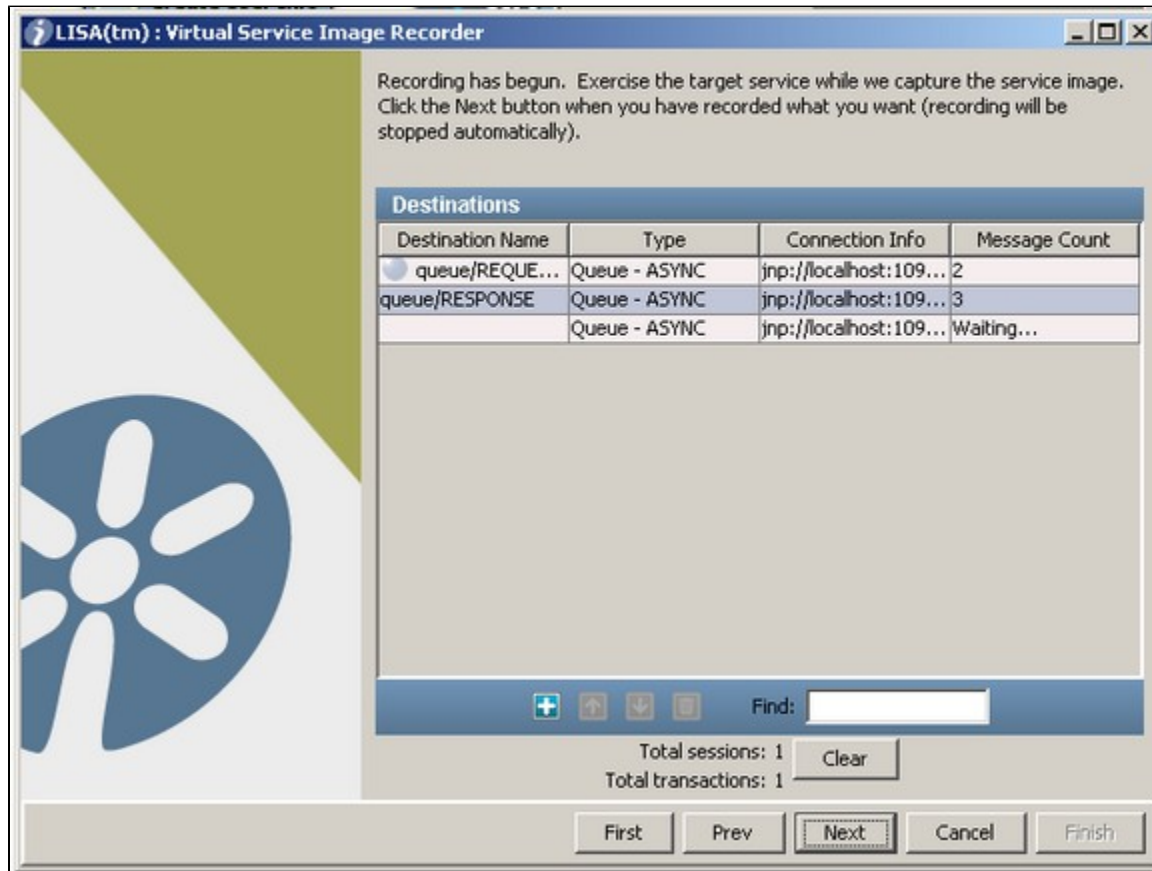
This is the case with **use temporary queue** check box unchecked with response queues.



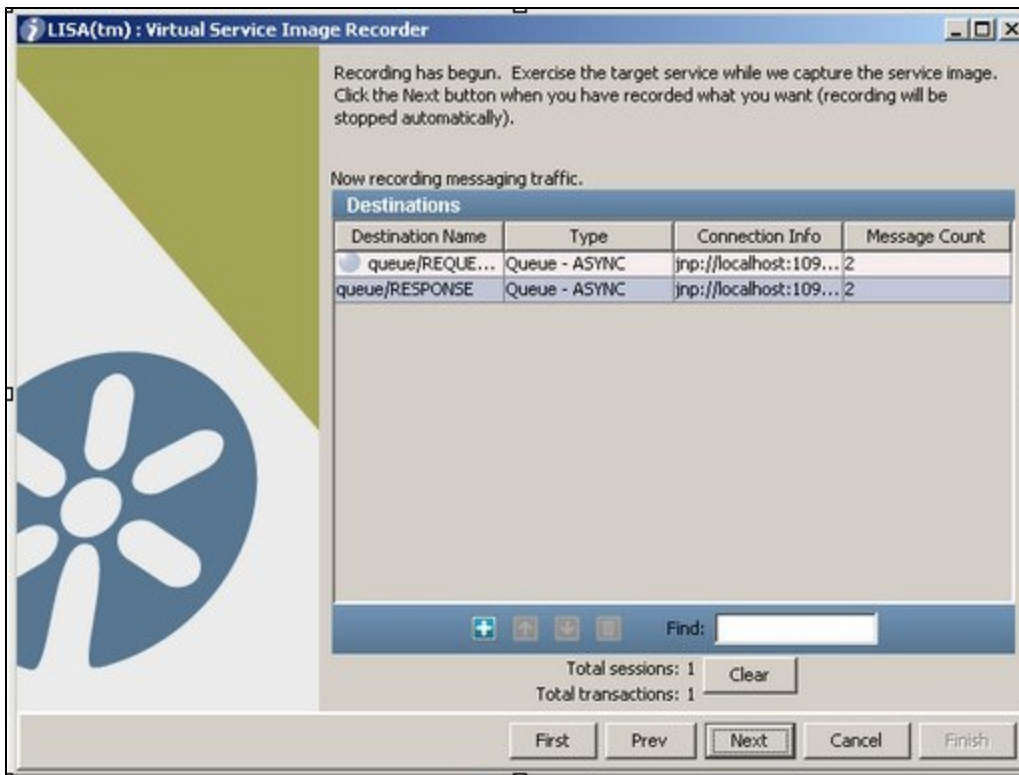
11. Run the client which would add messages to the request proxy queue. LISA Virtualize will copy those messages to the real request queue (ORDERS.REQUEST as in the screenshots). The server will pick those up from there and send responses to the response queue(s). Again, LISA Virtualize will pick those up and copy them to the response proxy queue(s), where the client listens to.

12. As the transactions are recorded, you will see the message count increasing and 'Total sessions' and 'Total transactions' count increasing on the Virtual Service Image Recorder dialog.

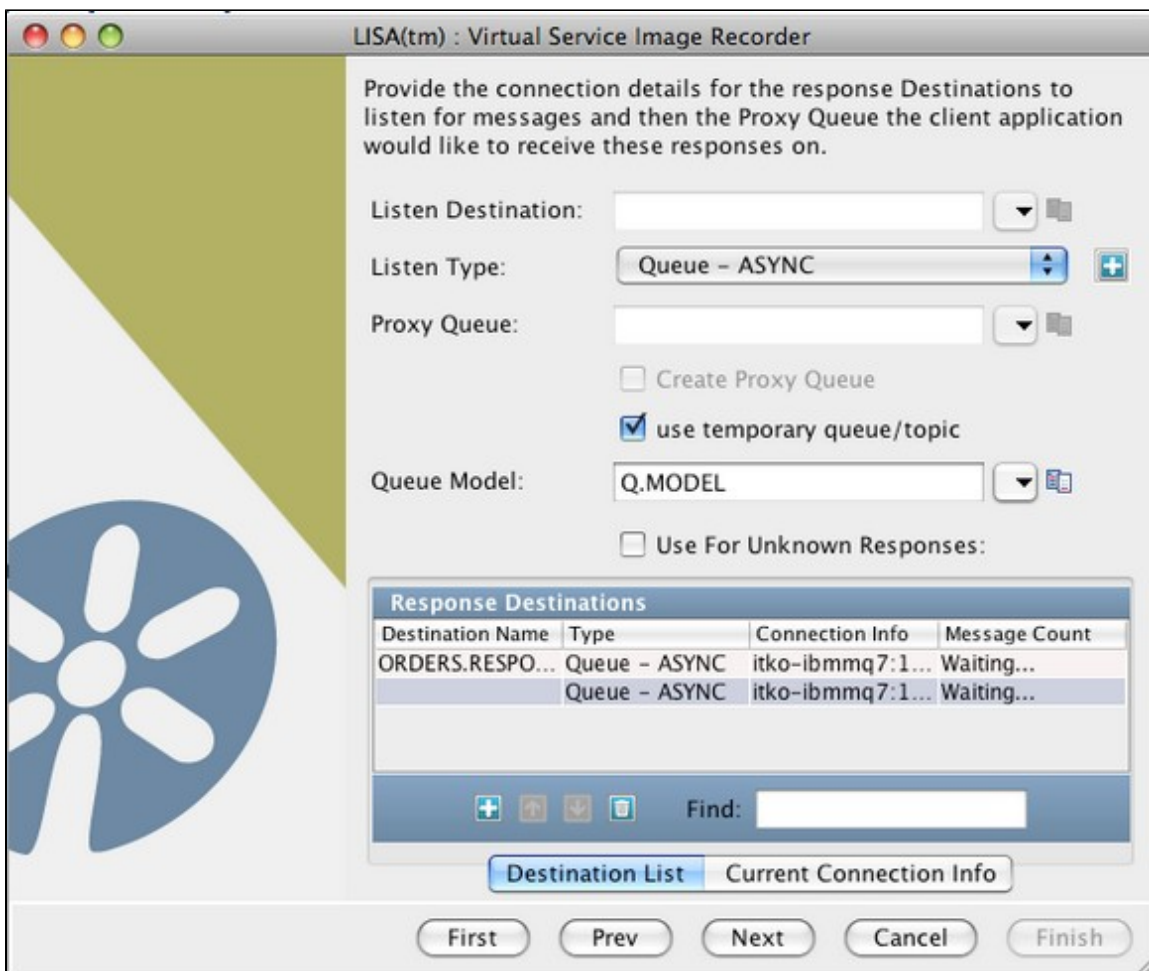
At the end of recording, all the request have gone through the same request queue, but about half of the responses have come back through temporary queues and the other half through the non-temporary response queue.



Following is the case of end of the recording without **use temporary queue** check box checked.



In case of MQ adding a temporary response destination



There is a new check box on the Response wizard page for temporary destinations, along with a new text field for the Queue Model. Checking the check box will disable the destination name and proxy destination name text fields, enable the Queue Model test field, and mark the response listener you're adding as a temporary response. A Queue Model must be entered in order for it to successfully create a temporary queue.

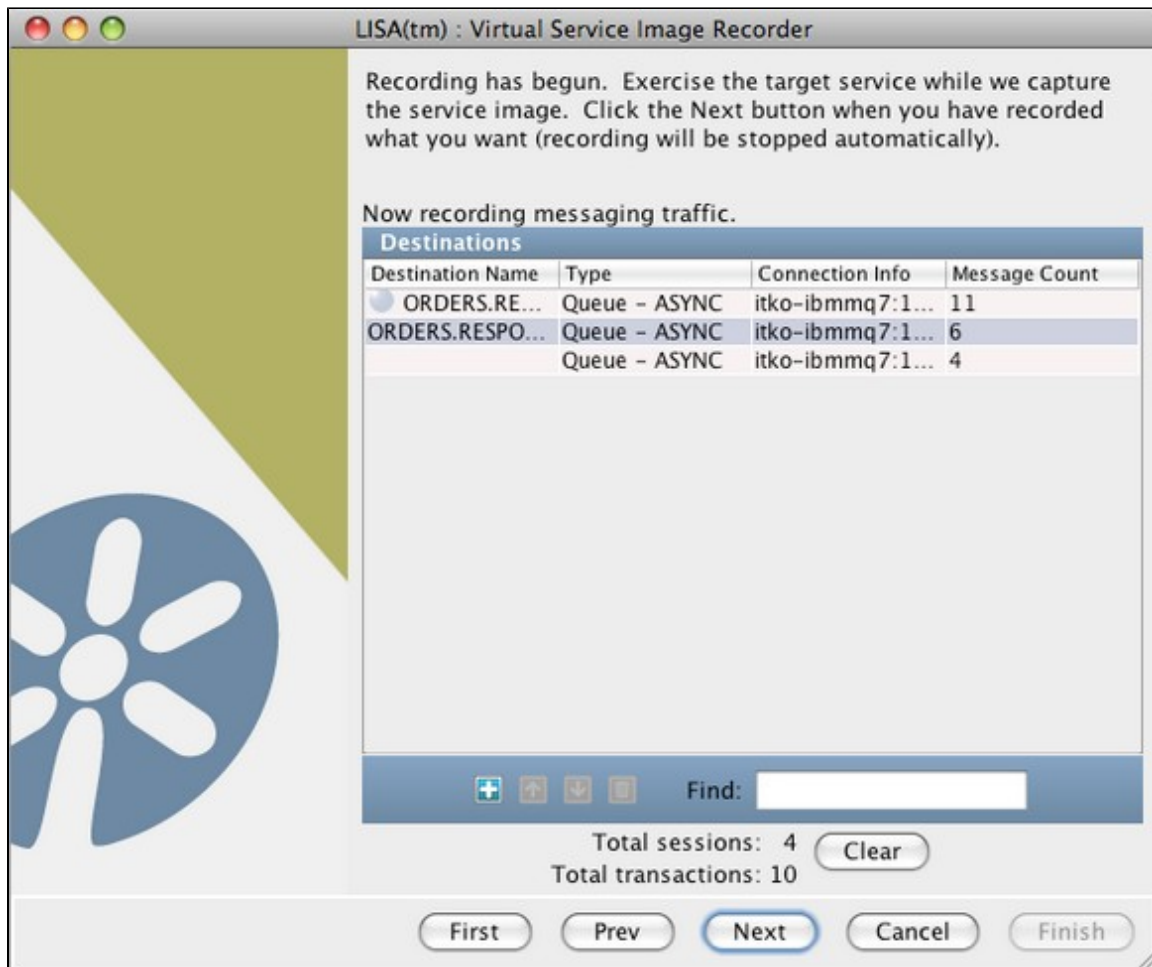
- Note: The only reason to add more than one temporary response to a recording session is if you're recording temporary destinations on multiple MQ servers.

Click the **Next** button to begin recording. You will see the names of the queues to which LISA Virtualize is listening, and hence the status is Waiting. At this point the proxy queues would be created.

Run the client which would add messages to the request proxy queue. LISA Virtualize will copy those messages to the real request queue. The server will pick those up from there and send responses to the response queue(s). Again, LISA Virtualize will pick those up and copy them to the response proxy queue(s), where the client listens to.

As the transactions are recorded, you will see the message count increasing and 'Total sessions' and 'Total transactions' count increasing on the Virtual Service Image Recorder dialog.

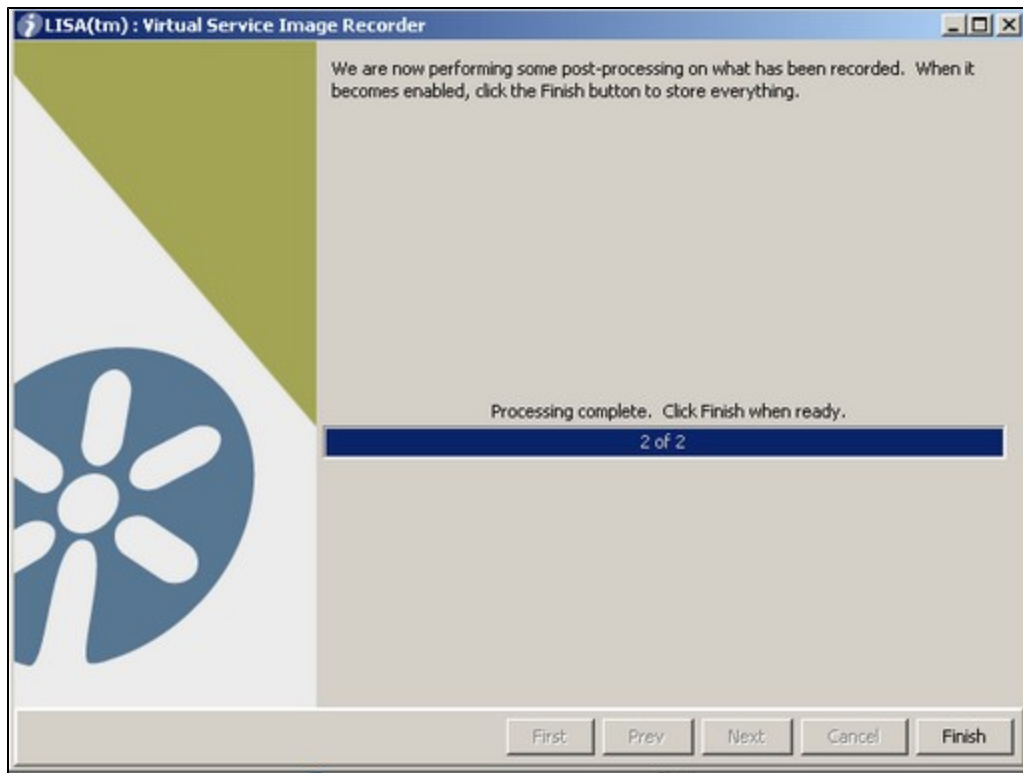
At the end of recording, All the request have gone through the same request queue, but about half of the responses have come back through temporary queues and the other half through the non-temporary response queue.



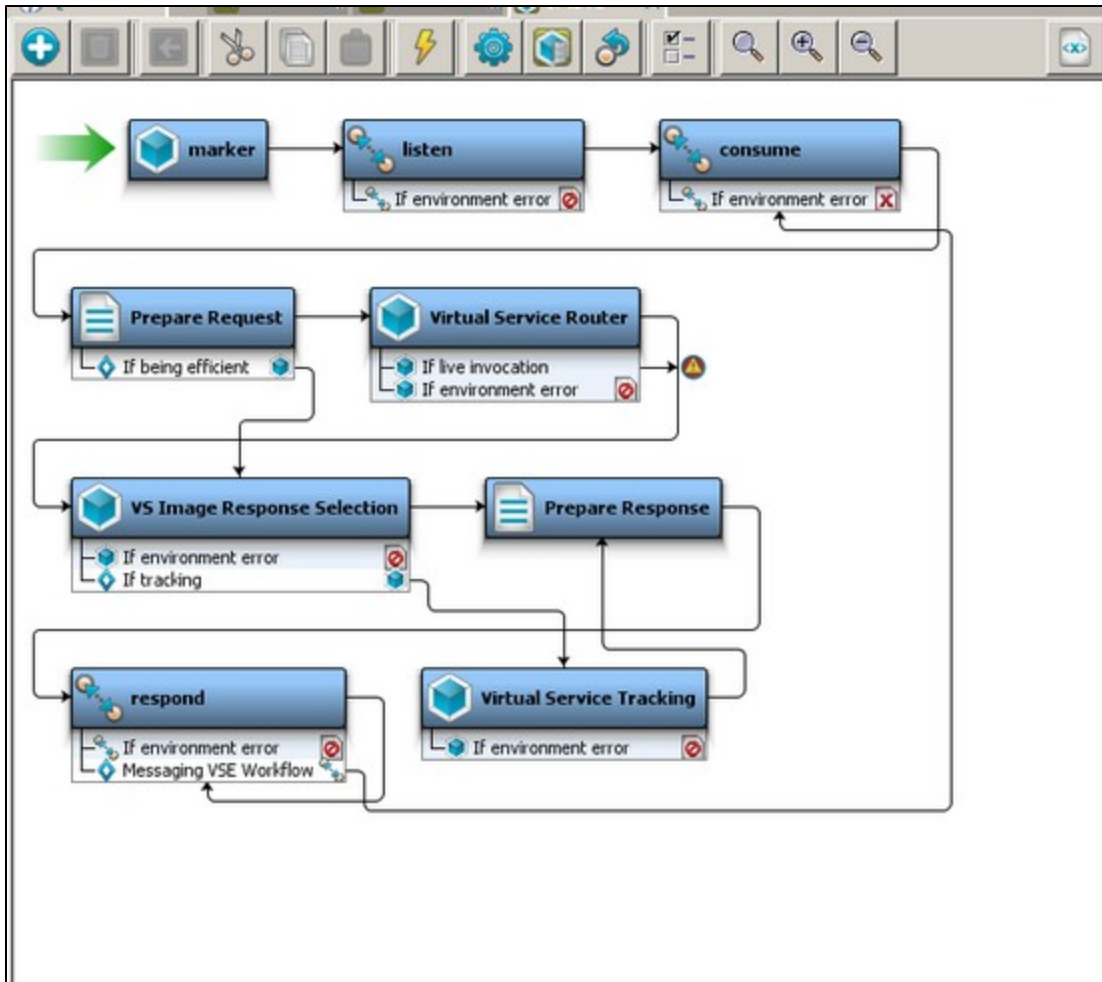
Note the two response queues. The one without a destination name is the temporary response queue.

13. When the run is complete, you may see the count of messages on the response queues less by 1. As a single request can have more than one response, LISA Virtualize would not have yet recognized the last of the transactions to have completed. The messages corresponding to the last transaction are therefore not counted.

14. Click Next. This serves as a trigger to close the last transaction and do the necessary cleaning. (You will see an intermediate screen if you were using a dynamic data protocol, which is elaborated in the next section.)



15. Click Finish to close this dialog and store the image.
16. Review and save the VSM generated in the main window.



Next Steps – Java Protocol

If you are using Java as the Transport protocol in the Basics tab in the Virtual Service Image Recorder, complete the remaining wizard steps as given below.

LISA(tm) : Virtual Service Image Recorder

Please provide us with some basic information about what is to be recorded and select the appropriate protocol(s) involved. Some transport protocols do not allow for a data protocol.

Basics | Notes

Service name: JAVA vs

Import traffic: Browse

(merge into existing service image)

Transport protocol: Java

☐ Treat all transactions as stateless

Default navigation: WIDE Last: LOOSE

VS Model style: ☒ More flexible ☐ More efficient

Desensitize: ☐

First Prev Next Cancel Finish

1. Choose "Java" as the transport protocol and click **Next**. Do not select any value for the data protocol on the recorder 2nd screen and click on **Next**.

LISA(tm) : Virtual Service Image Recorder

Available Data Protocols

Data Protocol: Add

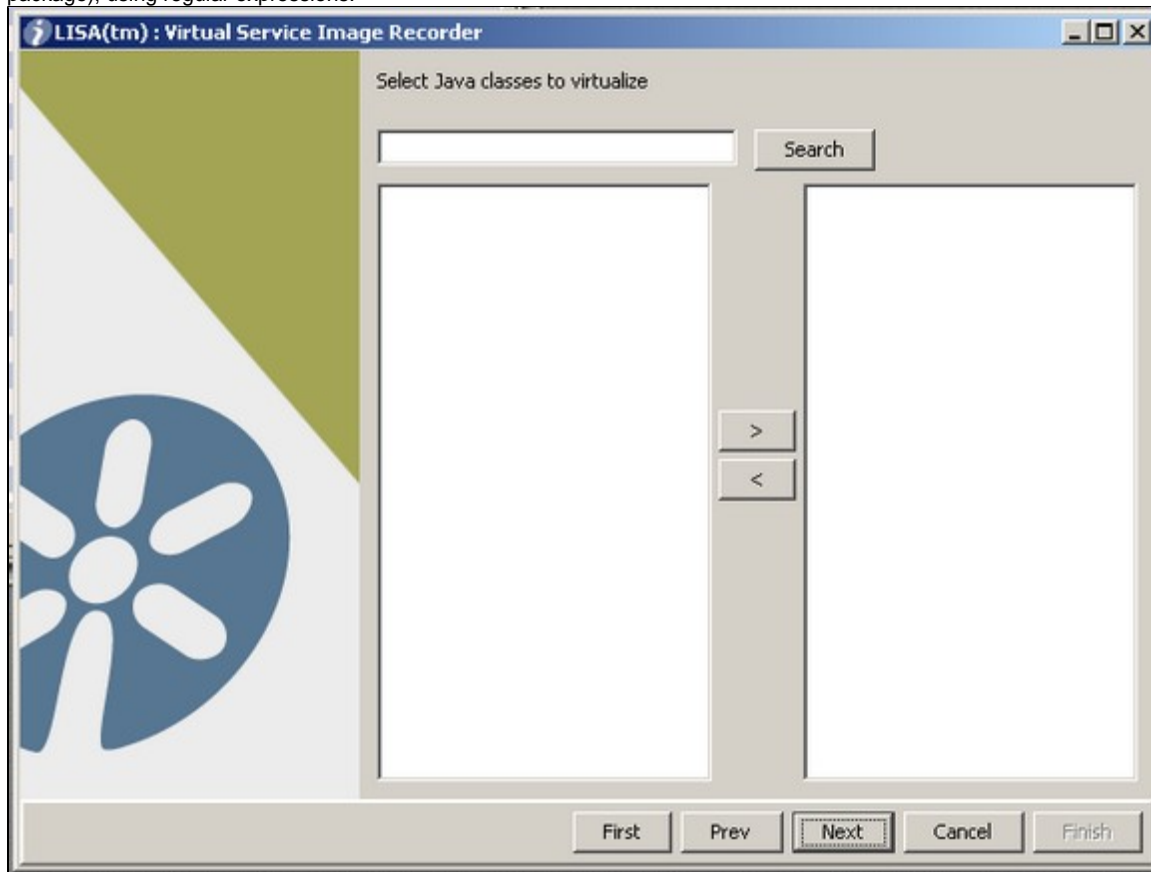
Selected Data Protocols

Protocol Name	Protocol Class

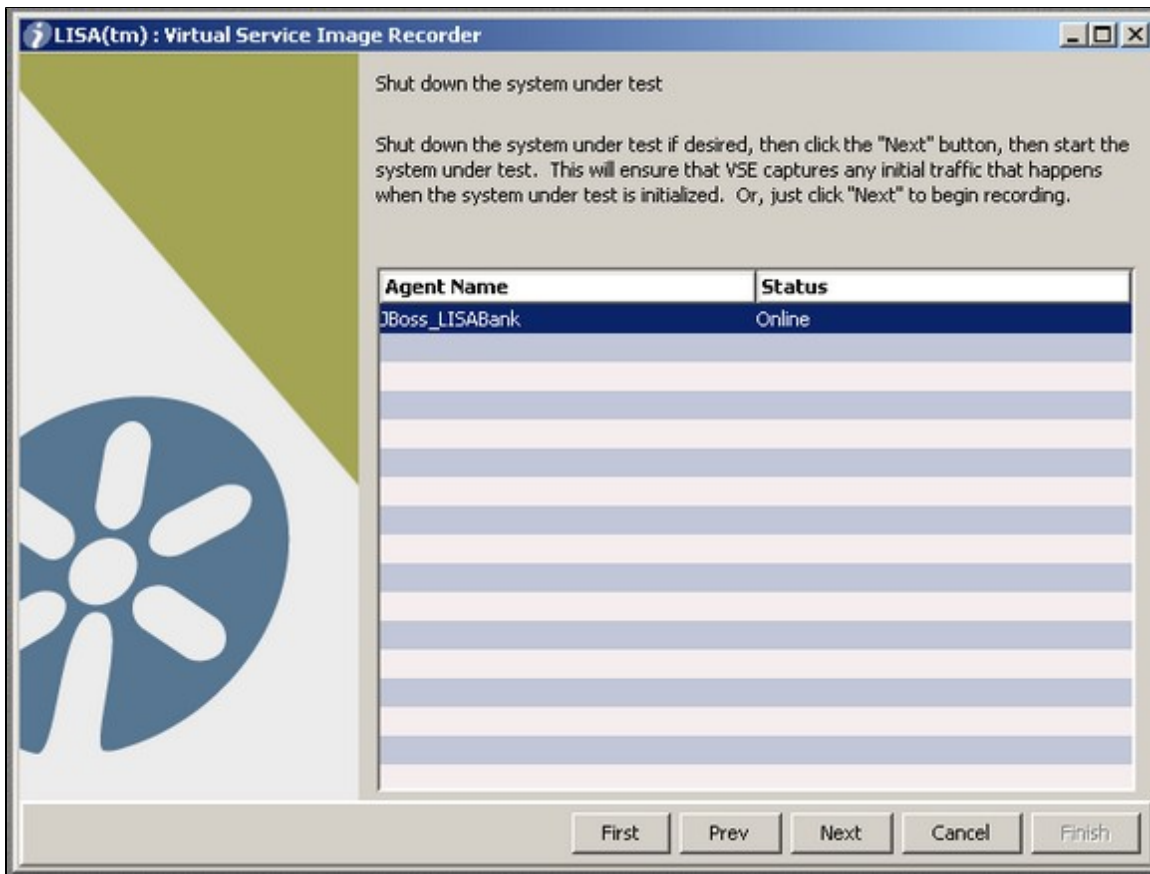
Find:

First Prev Next Cancel Finish

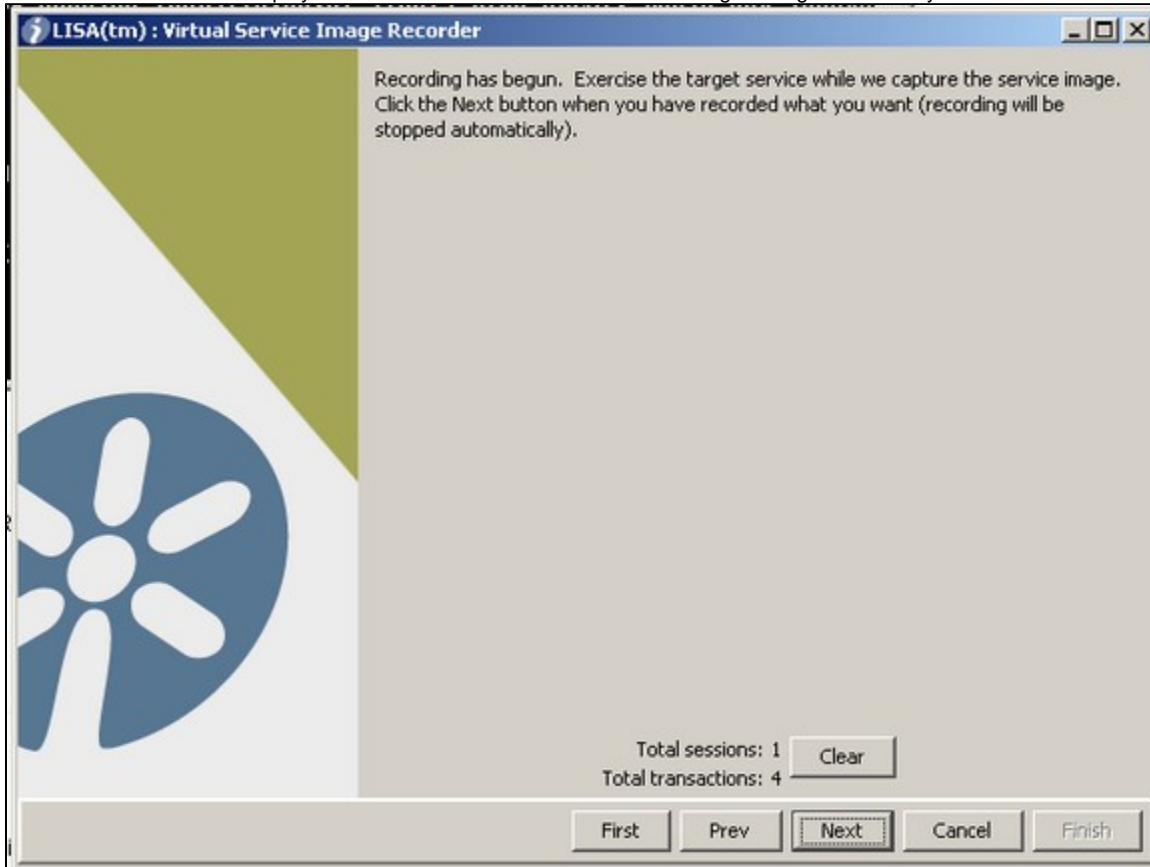
4. The next screen gives you the opportunity to enter other class names (if any) to virtualize. These are entered as fully qualified names (including package), using regular expressions.



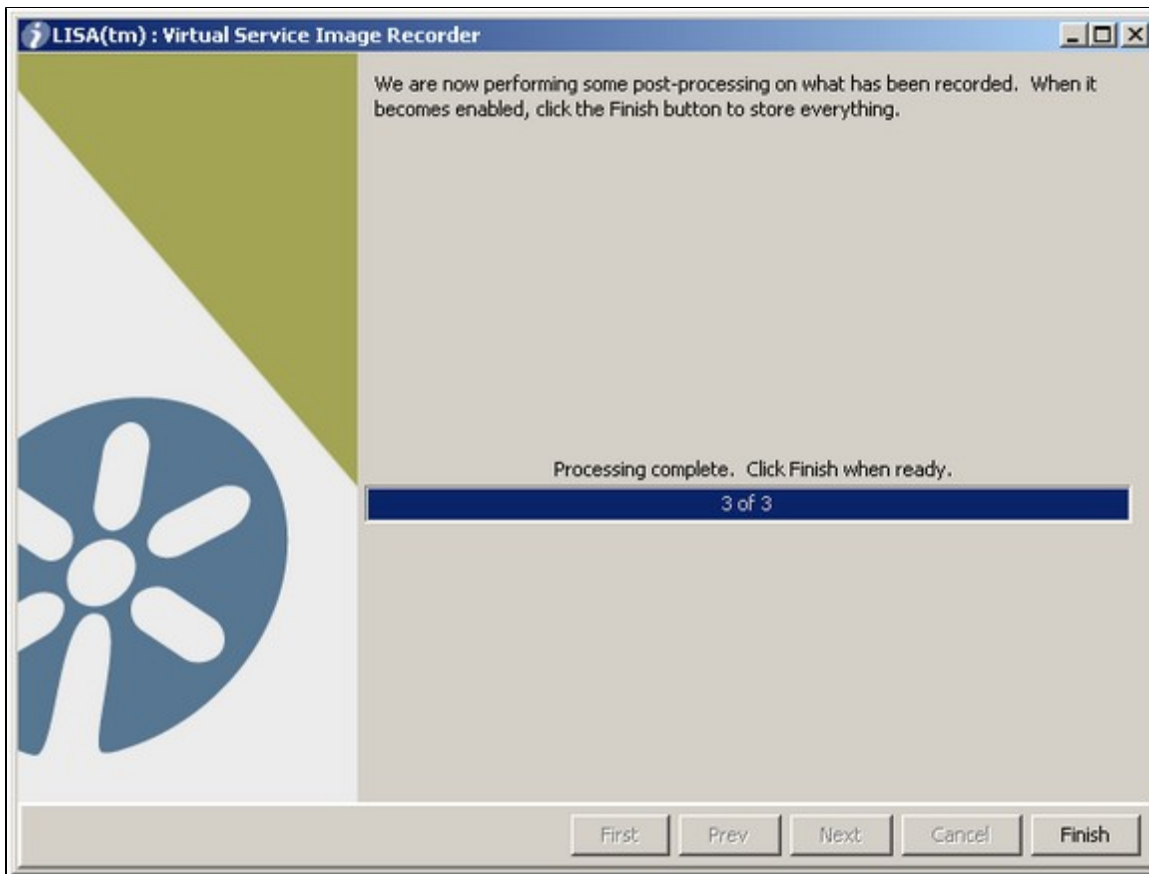
5. Click **Next** on this screen and you will again see the agent associated with the system under test.



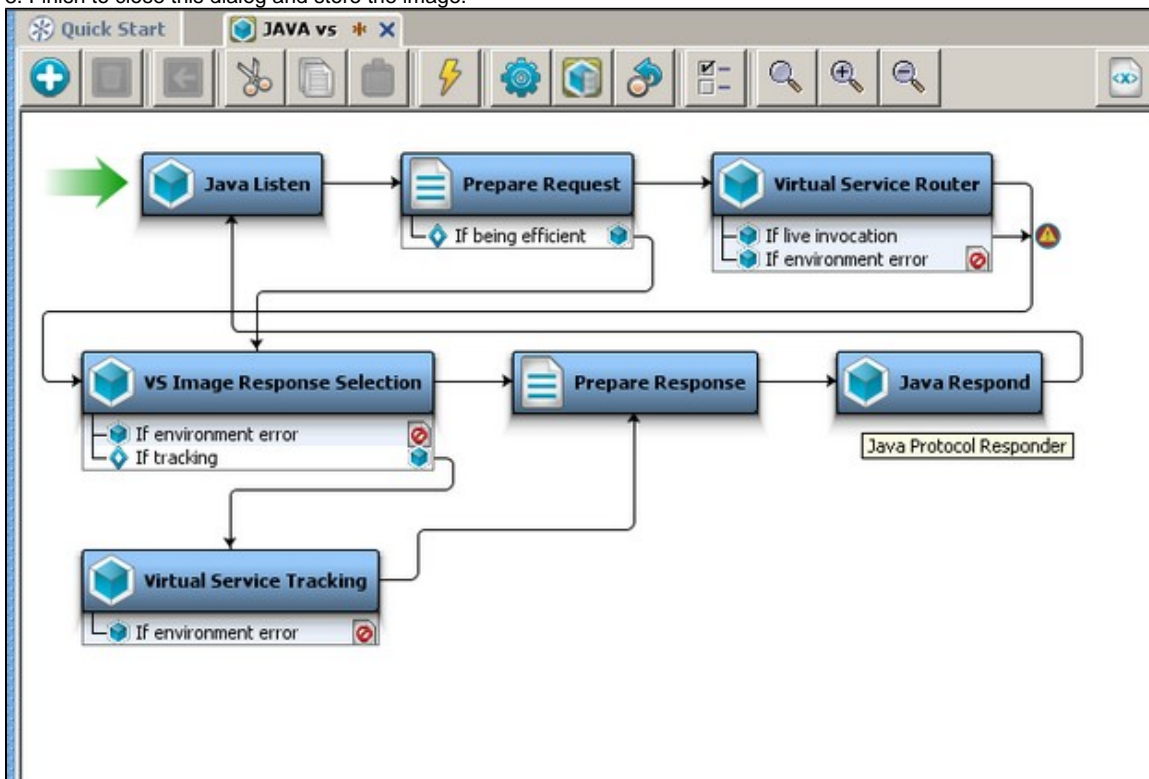
6. Follow the instructions displayed on the screen or click **Next** and recording will begin as with any VS Model.



7. Record your transactions, then click "Next".



8. Finish to close this dialog and store the image.



[Return to Next Steps List](#)

5.2.1 Virtualizing JDBC

5.2.1 Virtualizing JDBC

This page describes how to use LISA Virtual Service framework to virtualize JDBC-based database traffic. For this to make sense, you should be familiar with LISA's notion of service images.

- Virtualizing JDBC
 - o The Driver
 - + Driver Installation
- 1. DriverManager Style
- 2. DataSource Style
- 3. Other Startup Properties
 - o The Recorder
 - + Step 1
 - + Step 2
 - + Step 3
 - + Step 4 (and following)

The Driver

As with all things JDBC, the database simulator is presented to the system under test (SUT) as a standard JDBC driver. When installed and running in the SUT's JVM, this driver supports three different modes of processing on a per-connection basis: WATCH, RECORD, and PLAYBACK. By default, all connections are in WATCH mode.

In WATCH mode, the LISA driver wraps itself around each database connection and statement object and tracks SQL execution statistics (statements and counts of execution).

In RECORD mode, the LISA driver also wraps result sets and sends all information requested of the wrapped database connection to a LISA JDBC service image recorder which runs as part of the TestManager?.

In PLAYBACK mode, the LISA driver requests the data to provide to the SUT from a LISA virtual service model that contains the JDBC listen and respond steps along with a service image response selector step which will work together to satisfy the SUT's database requests. Such a model can be running either in the ITR or in a VSE server.

The RECORD and PLAYBACK modes require a connection to a LISA instance. To do this, the driver requires an IP port that it will listen on for connections from either the LISA service image recorder or from a JDBC listen step. If it is not specified, it will default to 2999. While executing, there may be only one active connection to a LISA instance. For example, if a VSE is connected to the driver for playback, the TestManager? (or any other VSE) will not be allowed to connect to the same driver.

Driver Installation

The driver is packaged in a jar file called **lisajdbcsim.jar**, which is located in the `LISA_HOME/lib` directory. This file must be placed somewhere in the SUT's classpath. For our JBoss demo server, you will find it is already placed in the `DEMO_HOME/jboss/server/default/lib` directory.

How you get the driver up and running in the SUT will depend on the style of JDBC that the SUT uses.

DriverManager Style

If the SUT uses Java's DriverManager to acquire connections (not likely in an app server), add the following to the startup command for the SUT:

```
-Djdbc.drivers=com.itko.lisa.vse.jdbc.driver.Driver
```

If this property is already being used, just add our driver to the front, separating it from the rest of the driver class names with a colon.

DataSource Style

If the SUT uses the newer DataSource approach for connection acquisition (as the demo server does), then you'll have to update the SUT's configuration. The method of changing this configuration will vary based on the SUT.

The following properties can be specified for the VSE Data Source:

1. driver - Specify the name of a driver class that VSE should delegate to for the real connection to the database.
2. url - The actual URL used to connect to the database. This should be in precisely the same form as the real database connection (e.g., for Oracle it will look like `jdbc:oracle:thin:@host:port:sid`).
3. user - The user name to connect to the database.
4. password - The password to connect to the database.
5. state - This must be one of watch, record, or playback (case insensitive) and defines the initial state for connections.
6. datasource - Instead of specifying a driver class to wrap, a datasource class can be specified. This will keep the VSE Driver from "hijacking" the driver manager and may be useful in situations where an app server is not allowing VSE to connect. Using this form will also permit multiple endpoints - VSE can service several connection in the same SUT.
7. jdbcSimPort - This is normally specified for the entire SUT (see the section on the **lisa.jdbc.sim.port** property below), but if multiple endpoints are desired, this can be specified on a per-datasource basis. This only works if "datasource" is specified, and not "driver".

Although it is supported, specifying a state of RECORD or PLAYBACK as the initial state in this manner can have undesirable side effects, as

each requires a live LISA instance which may not always be available due to the somewhat random nature of how data source style SUTs acquire connections. This will likely be more problematic for recording than for playback. See below for the preferred way of having the SUT sync up with LISA.

If you want to use both the simulation and Pathfinder drivers, make the simulation driver the "outside" driver and specify the Pathfinder class and URL as the real information described above. Refer to the **itko-example-ds.xml** file in the `DEMO_HOME/jboss/server/default/deploy` directory for an example that defines our data source to JBoss for our demo server.

Other Startup Properties

Regardless of the SUT's connection style, the following properties can be added to the startup command for the SUT to affect the simulation driver:

`lisa.jdbc.sim.require.remote`

This property must be set to true or false. The default value is false. If true, the driver will block until there is an active connection with a LISA TestManager? or VSE instance. This is the preferred way to have an SUT sync up with LISA (especially a VSE) when you want to record or play back any startup database activity performed by the SUT.

`lisa.jdbc.sim.port`

This property must specify a valid IP port. If it is not specified, it will default to 2999. This is the port the driver will listen on for connections from a TestManager?-based recorder or a running virtual service model.

The Recorder

The standard Virtual Service Image recorder is what you will use to capture JDBC database traffic. To use it, create a new or open an existing vs model. On the "Commands" menu, select the "Virtual Service Image Recorder" item. This will display a wizard to guide you through the recording process. If there are already steps present in your virtual service model, a warning message will be displayed.

Step 1

On the first page of the wizard, provide a name for the service image to record in the field labeled, "Service info:". Select an existing one from the drop-down if you would like newly recorded data to be added to an existing service image. Select "JDBC" as your transport protocol and click Next.

Step 2

On the next page, enter the host name (or IP address) and port where the LISA JDBC simulation driver is running (see above for details on installing and running the driver in your SUT). The port must be the same as what the driver is configured to listen on (2999 by default or the value of the **`lisa.jdbc.sim.port`** property). When you click Next, a processing dialog is displayed until a connection with the driver is established. Once a connection exists, the third page of the wizard is displayed.

Step 3

The third page of the wizard is presented in three sections. The top section shows the list of all the JDBC drivers installed and registered in the SUT. The middle section shows the 10 most recent statements executed (and how many times) for every unique connection URL/database user combination. This refreshes every 5 seconds or so. The bottom section shows the list of connection URL/database user combinations that are to be recorded. The URL may be a regular expression. (This list will normally start off empty unless the state attribute is set to RECORD on a simulation connection URL for DataSource style SUTs.)

If a driver is selected in the drivers list, the "To URL" button may be used to copy a generic regular expression that will match connection URLs for that driver to the URL entry field. If the connection URL (top level) node in the activity tree is selected, the "To URL" button may be used to copy the URL and user to the URL and user entry fields. Also, the "Add" button to the right of the activity tree may be used to add the connection URL and user directly to the recording list.

Connection URLs may be either exact or a regular expression that will match connection requests and may be added to the recording list with or without a database user. The absence of a database user will match any user attempting to connect. If the "Add" button to the right of the URL/user entry fields is disabled, it is likely the recording list already covers the URL/user combination.

Once the recording list contains all the URLs and users to record, clicking Next will display the "recording" wizard page.

Step 4 (and following)

The fourth page of the wizard is divided into two sections. The top section is the same activity tree as the middle section of the previous page. The bottom section displays the number of conversations and transactions that have been recorded. Once all the desired traffic has been captured, click the Next button. The rest of the wizard's pages act as they do for any other type of recording.

5.3 Identification of Conversations and Transactions

5.3 Identification of Conversations and Transactions

The quality of the recorded service image is directly dependent on how much information LISA Virtualize has in order to identify the conversations

and the individual transactions in them. In particular, LISA Virtualize needs help in differentiating the transactions from each other:

1. **As part of a conversation** – which means identifying some unique id representing a session.
2. **As individual operation** – which means identifying some information specific to the semantics of the operation – for example, distinguishing an 'order creation' operation from an 'order listing' operation.

LISA Virtualize internally knows many patterns to look for. For example, in case of virtualizing over the HTTP protocol, if the server is a Java server, it typically sets a cookie containing the value of a special variable named as sessionid, which uniquely identifies the session. LISA Virtualize can catch that to identify different sessions from each other.

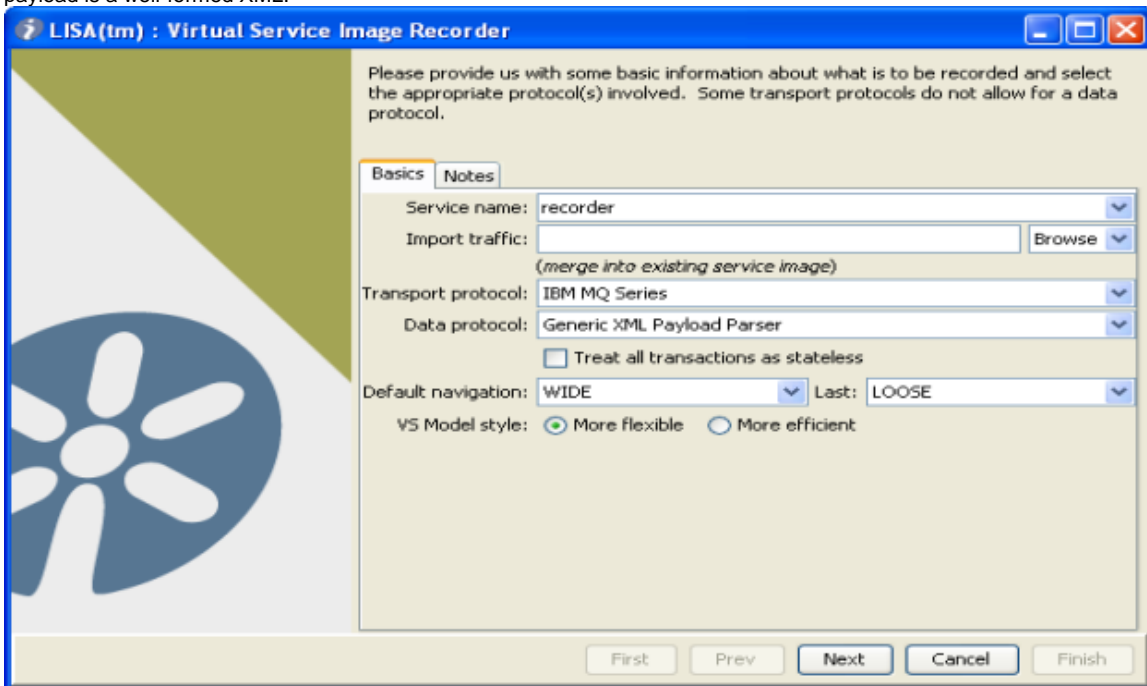
However, there are times when LISA Virtualize needs further help. It could be supplied in various forms:

1. In the HTTP recording, LISA Virtualize gives you the opportunity to identify tokens.
2. In case of JMS protocol, you could identify whether JMS correlation id, custom JMS headers or all JMS headers would be used by LISA Virtualize for this identification.
3. LISA Virtualize lets you specify a dynamic data protocol which lets you obtain meaningful information from the message itself. The following section describes this technique.

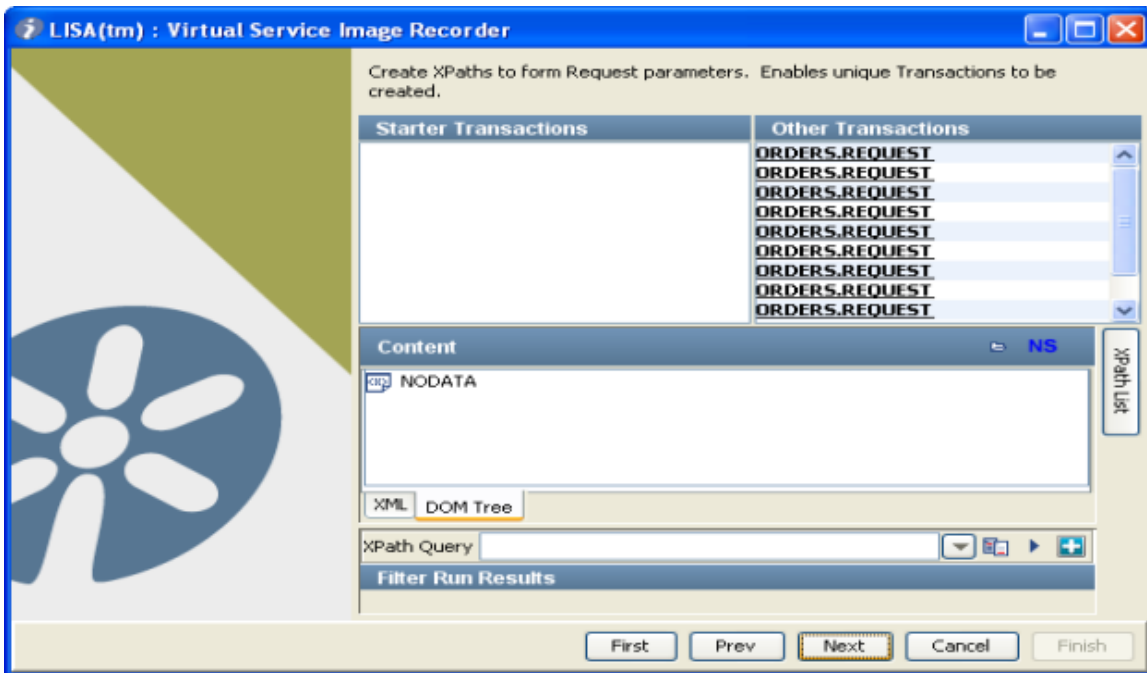
Dynamic Data Protocol

Using a dynamic data protocol is a technique to help LISA Virtualize look at the body of the recorded messages (payload) and extract meaningful information from them to help identify the transactions. Especially in case of very opaque protocols like the MQ native protocol, this is the only way to get meaningful conversation information.

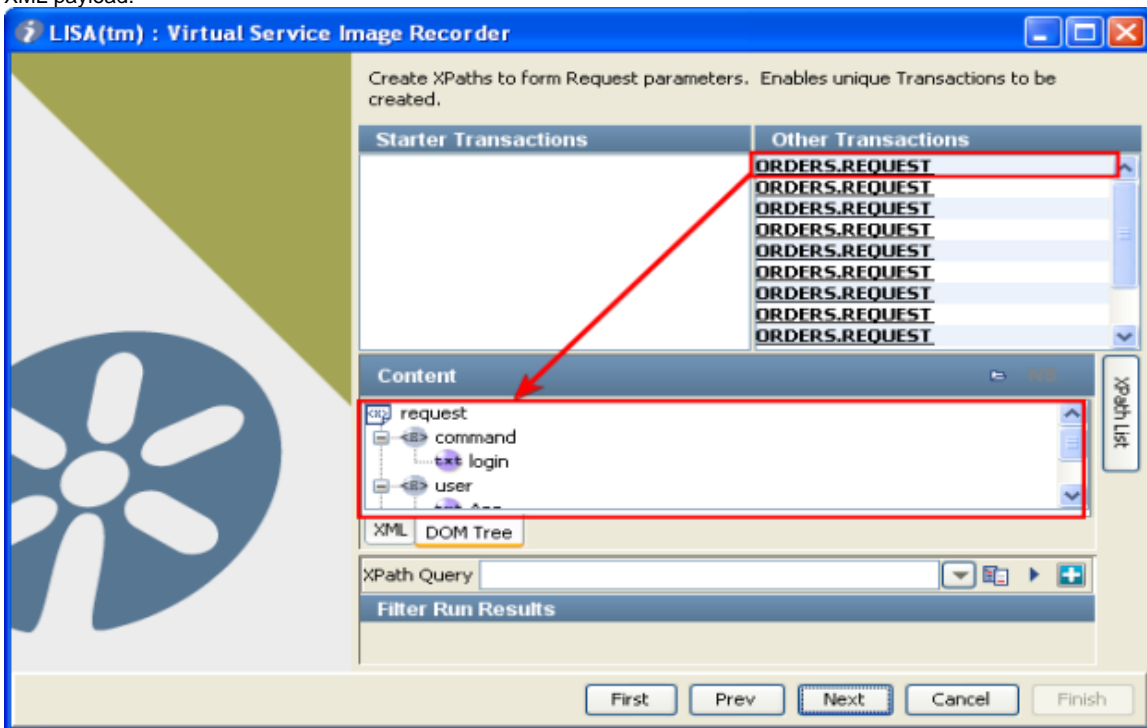
1. To choose a dynamic data protocol, in the Basics tab of the Virtual Service Image Recorder, choose 'Generic XML Payload Parser', if the payload is a well-formed XML.




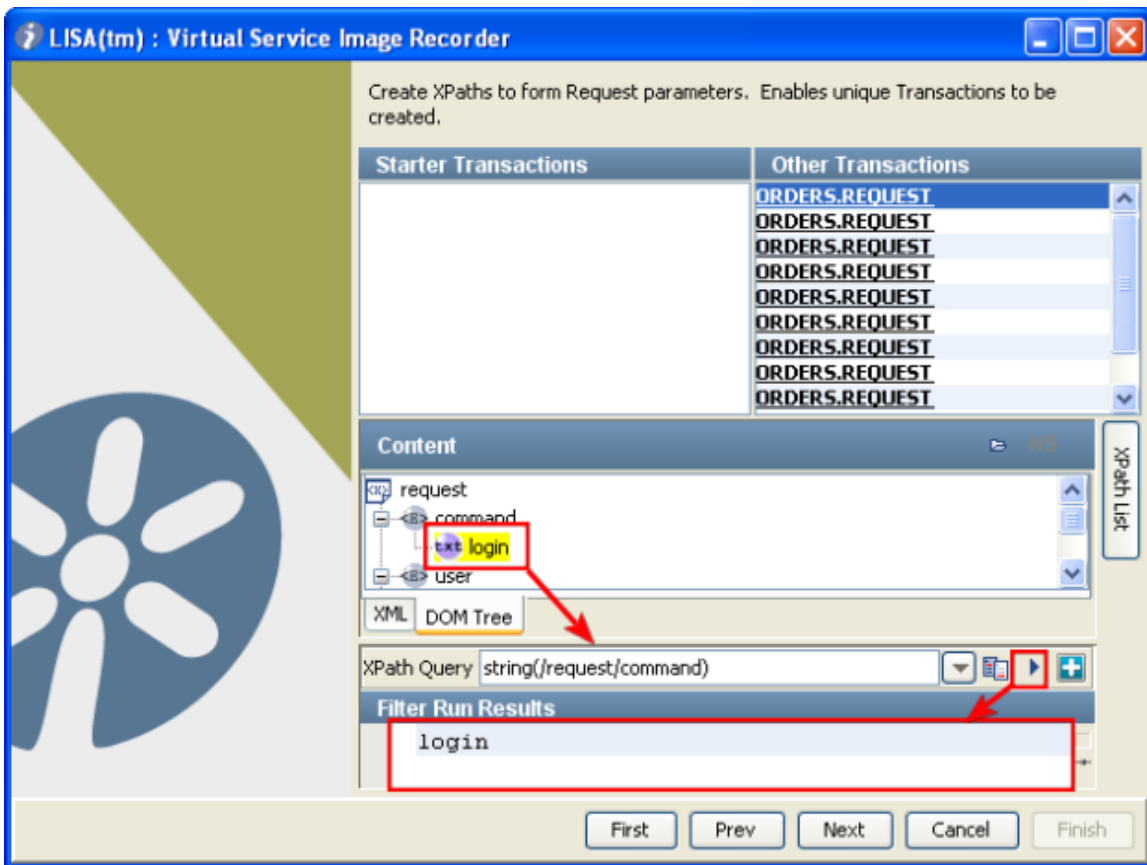
2. Proceed through the rest of the steps, up to the cleanup step.
3. After the cleanup step, the following screen appears:




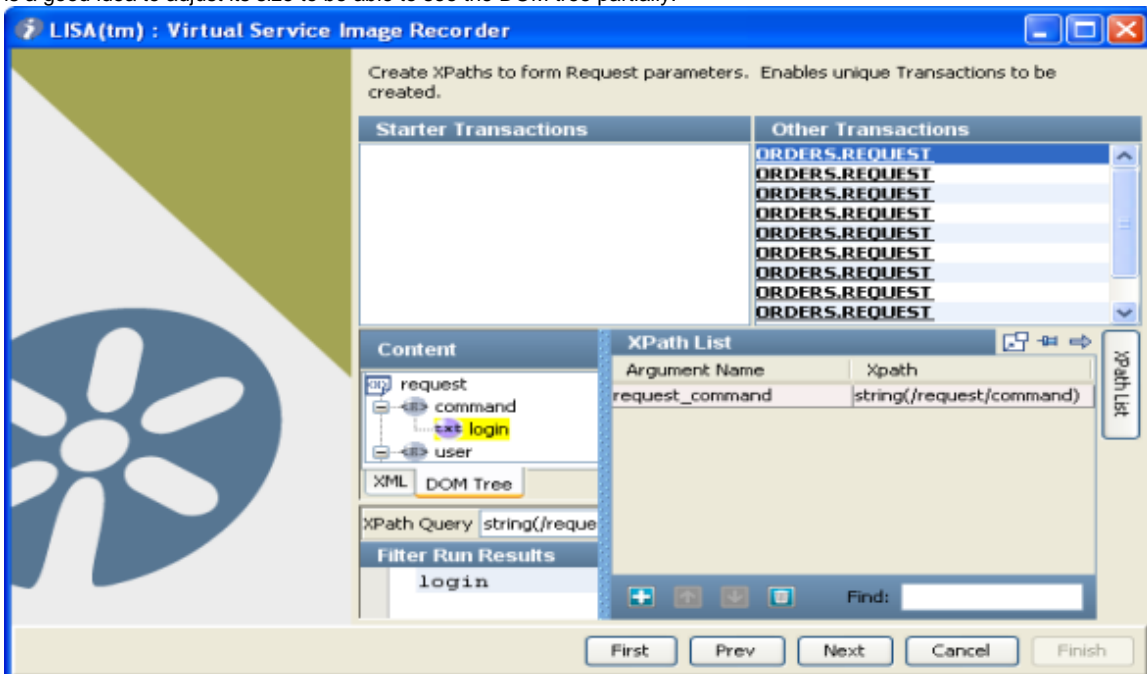
4. By default, there are no starter transactions identified, which means that it does not know which data to look at to identify the conversations. However, you will see the recorded transactions in the 'Other Transactions' list. Click on the first transaction to see the XML payload.



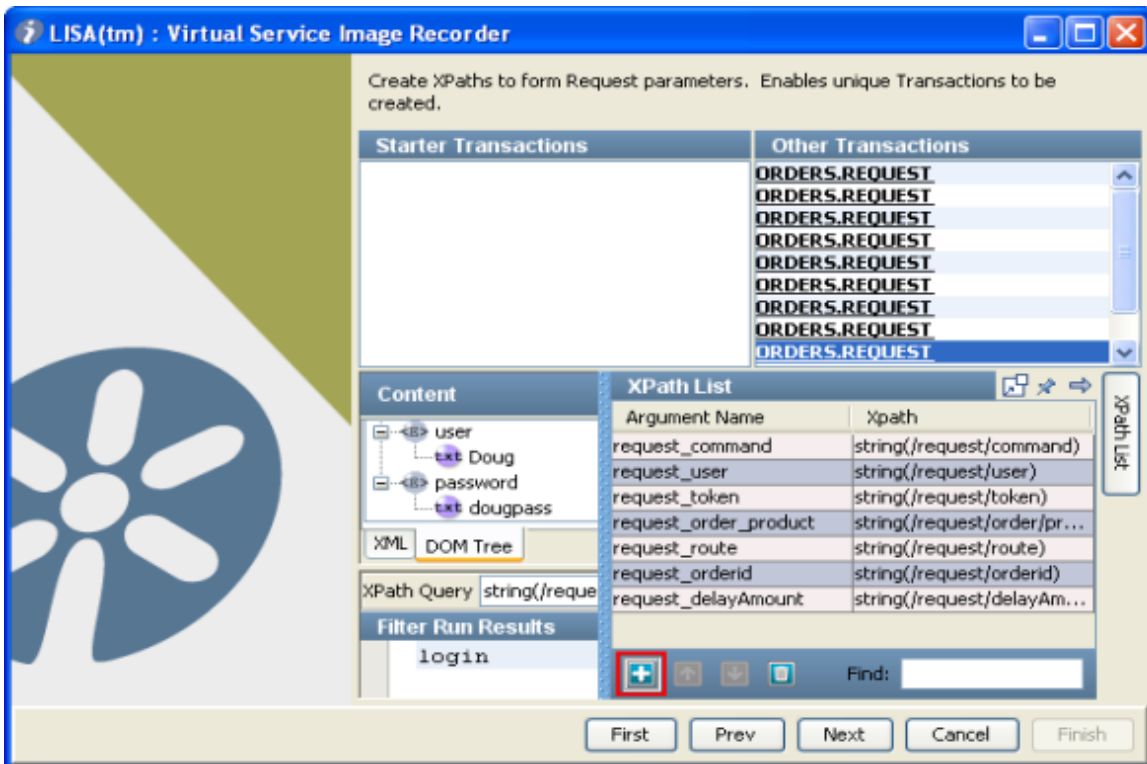
5. The XML payload can now be seen under Content. The XML tab shows the classic XML view. The DOM Tree tab shows it in the tree format.
6. In the DOM tree, to identify a particular node as a parameter for LISA Virtualize, click on the node. You will see the XPath query corresponding to the node in the text box against XPath Query. To evaluate the XPath query on the current payload, you can click the  arrow button. The result of the evaluation is shown under heading Filter Run Results.



7. To add this particular XPath query to the parameter list, click the  plus icon. This will automatically expand the XPath list subframe. It is a good idea to adjust its size to be able to see the DOM tree partially.



8. Note that LISA Virtualize has remembered the XPath string that you identified and given it an argument name. You are welcome to rename the argument.



9. Likewise, you are welcome to pull other variables from this transaction or others. Note that it is not required for all transactions to have the particular argument. At the time of processing, if a particular argument is not present in the payload then it is ignored.
10. Click **Next** once you are satisfied with your choice of variables. You are then directed to the post-processing screen as we have seen before.

5.4 Virtual Service From WSDL

5.4 Virtual Service From WSDL

Migrating to VSE and WSDL-generated Services


There are two aspects to migrate from the old Virtualize Web Service functionality:

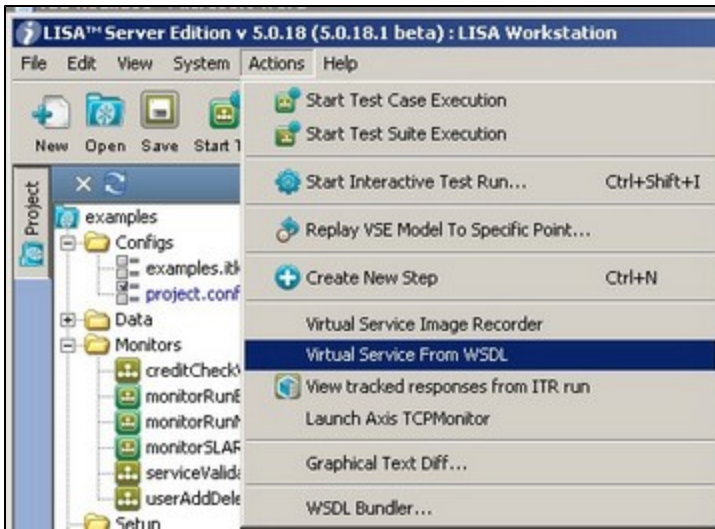
1. Generating the VSE Service
2. Deploying it.

Generating a VSE service from a WSDL

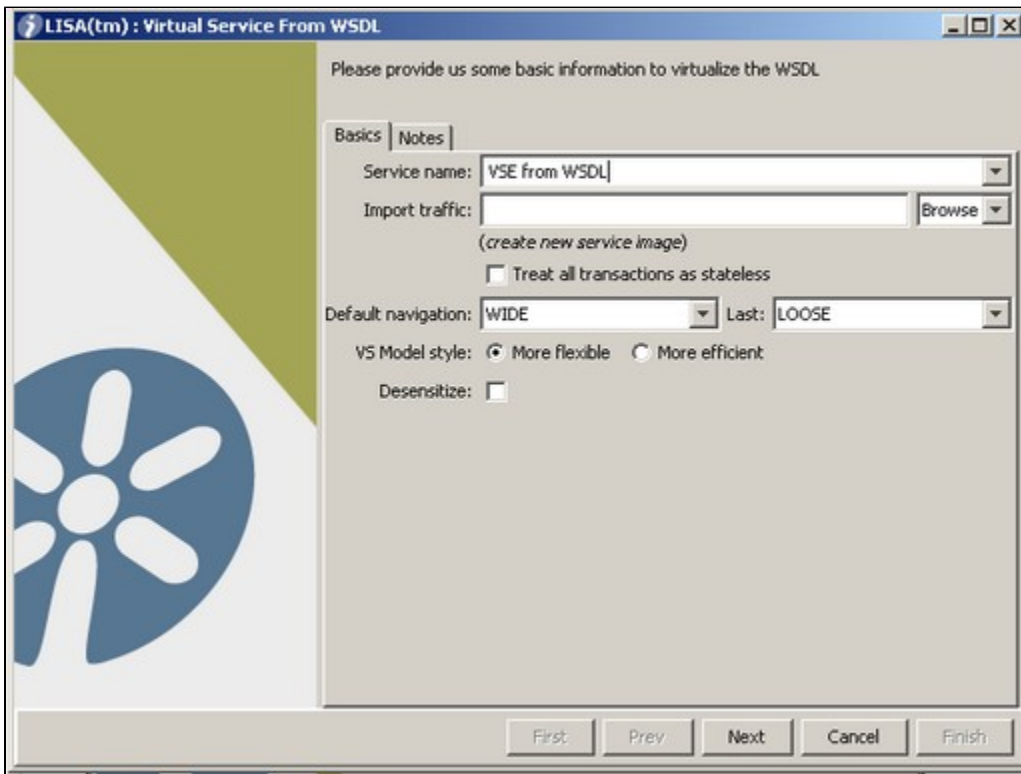
A web service can be generated from a WSDL using the following steps:

1. Creating a new, blank Virtual Service Model (VSM)

2. Under the Actions menu, choose "Virtual Service from WSDL", or you can click on the  icon and select "Virtual Service From WSDL" option.



3. Enter a service name. Defaults are fine for the rest of the fields on this screen. Click Next.



4. Enter a port number in **Listen on port** text box on which the VSE service should listen.

LISA(tm) : Virtual Service From WSDL

Please provide us with the wsdl of the web service and service name which you want to virtualize. Also select the operations which should be included in the service image.

Listen on port: 8001

WSDL: host:8080/itko-examples/services/SignedCalculatorService?wsdl Actions

Name: CalculatorService

Which operations do you want to virtualize? All None

☒ add

☒ subtract

First Prev Next Cancel Finish

5. Add WSDL to be virtualized in the **WSDL** text box. This can be a local file or a URL.
6. Choose particular service in that WSDL that needs to be selected, in the **Name** text box. There is usually only one choice.
7. Select the **operations** in that service that should be virtualized. By default, all of the operations will be virtualized.
8. Click Next. On the next screen, the service image is generated and the wizard is finished. Click Finish.

LISA(tm) : Virtual Service From WSDL

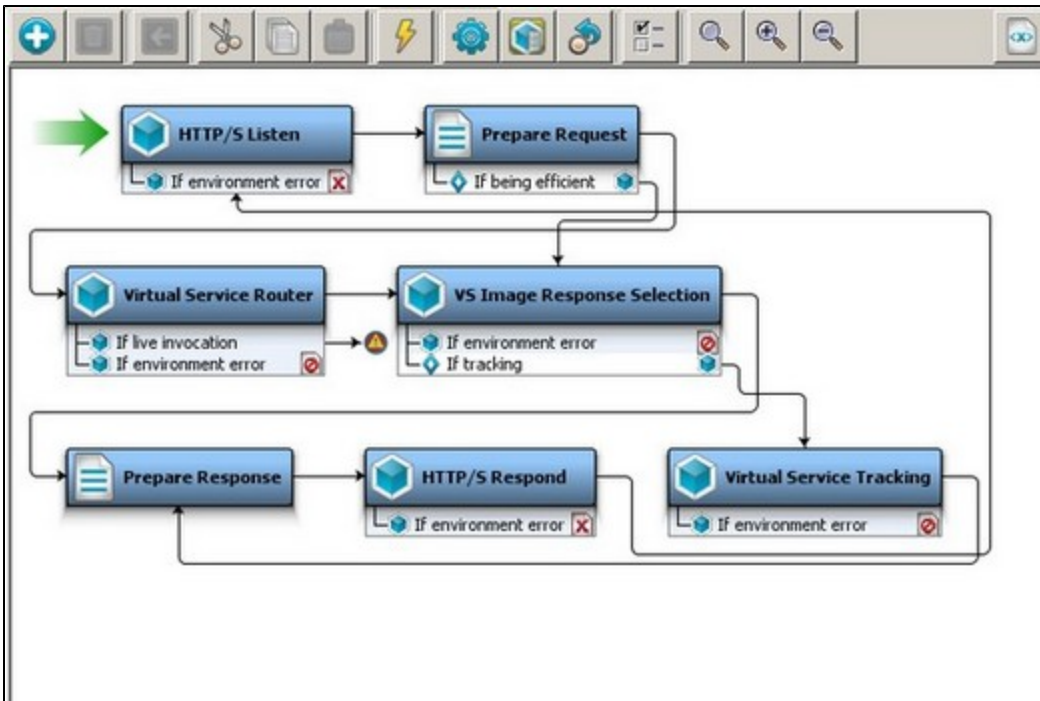
We are now performing some post-processing on what has been recorded. When it becomes enabled, click the Finish button to store everything.

Processing complete. Click Finish when ready.

2 of 2

First Prev Next Cancel Finish

9. The blank VSM will now be populated with steps. Save it.



The service image that was generated will be a *stub* service. It will return correctly formatted responses, but the values will be default values.

The Service Model (VSM) that was saved will be what gets deployed to VSE.

Server-side test cases

There is no direct equivalent in VSE for the old Server-side Test Case. This functionality has been replaced by VSE Service Images, which reside in the VSE database and are designed for recording existing services rather than writing the service logic from scratch.

If there is application logic that must exist in the VSE Service then the user is better off using the full Virtual Service Image Recorder to create their service rather than simply generating one from the WSDL.

It is possible to include some application logic directly in the service image created by the Virtual Service from WSDL function, however, that is an advanced topic.

Deploying a VSE Service from within a Test Case

Unlike the old Virtualize Web Service there is no equivalent step for deploying and undeploying VSE services from inside a Test Case. However, the same outcome can be achieved by using the VSEManager utility. This can be done with a Command Line step or a Java Script step.

Command Line

VSEManager is included as a tool under the bin/ directory in LISA Server.

For example, the command to deploy a VSE service looks like this:

```
LISA_HOME/bin/VSEManager.exe --deploy "service name" --model "LISA_PROJ_ROOT/VServices/service.vsm" --config "LISA_PROJ_ROOT/Configs/project.config"
```

And the command to undeploy that VSE service:

```
LISA_HOME/bin/VSEManager.exe --remove "service name"
```

Run VSEManager by itself on the command to get a complete listing of the available options.

Java Script

In some installations VSEManager may not be available as a tool under bin/. The same outcome can be achieved by using a Java Script step instead:

Deploying a VSE service:

```
arguments = new String[] {
```

```

"--deploy",
"service name",
"--model",
"LISA_PROJ_ROOT/VServices/service.vsm",
"--config",
"LISA_PROJ_ROOT/Configs/project.config"
};

com.itko.lisa.coordinator.VSEManager.main(arguments);

Undeploy that VSE service:

arguments = new String[]

Unknown macro: {  "--remove",  "service name" }

;

com.itko.lisa.coordinator.VSEManager.main(arguments);

```

6. Editing a Service Image

6. Editing a Service Image

In this chapter we shall see how to edit a virtual service image using a Service Image Editor. If you are not interested in editing a service image, then feel free to skip this chapter.

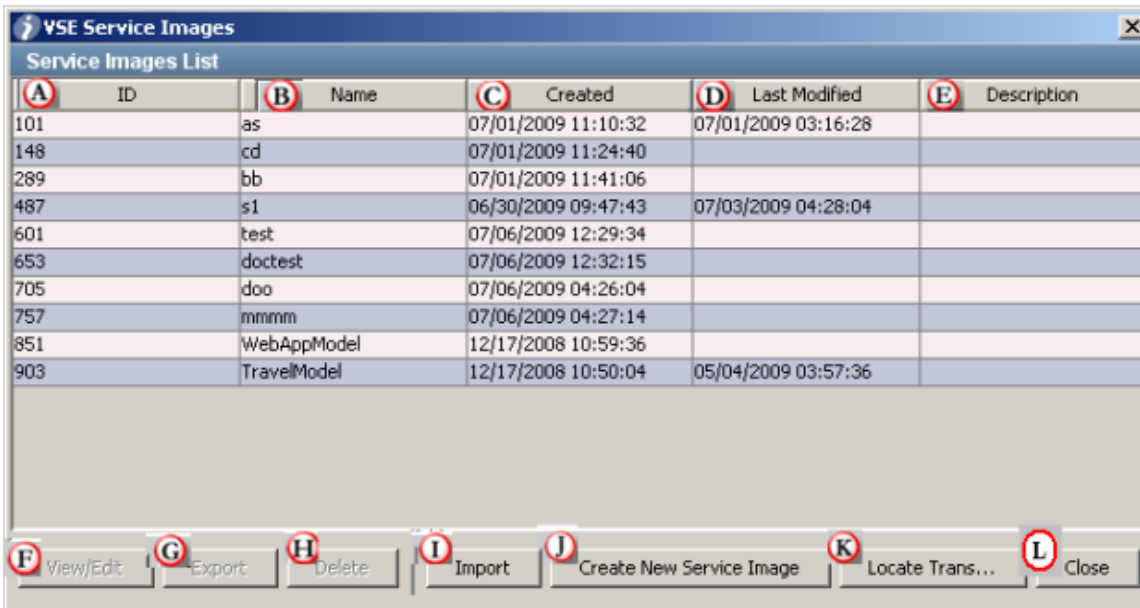
The following topics are available.

- [6.1 VSE Service Images Window](#)
- [6.2 Service Image Editor](#)
- [6.3 Service Image Editor Basic Info Tab](#)
- [6.4 Service Image Editor Stateless Transactions Tab](#)
- [6.5 Service Image Editor Conversations tab](#)
- [6.6 Conversation Tree Editor](#)

6.1 VSE Service Images Window

6.1 VSE Service Images Window

You can access the VSE Service Images window from the menu bar by selecting **System > View Virtual Service Images**.



The VSE Service Images window lists all the service images found in the LISA Workstation database.

VSE Services Images components

Callout	Description
A	The ID column lists the ID for the service image.
B	The Name column lists the virtual service image name on the system.
C	The Created column lists the date and time when the virtual service image was created.
D	The Last Modified column lists the date and time when the virtual service image was last changed and saved.
E	The Description column lists any documentation pertaining to the service image.
F	Click View/Edit to open the selected virtual service image in the Service Image Editor.
G	Click Export to export the selected virtual service image to an XML file which may be imported by another Workstation user.
H	Click Delete to remove the selected virtual service image from the database.
I	Click Import to import an XML file as a virtual service image.
J	Click Create New Virtual Service Image to create a service image manually in the Service Image Editor.
K	Click Locate Trans... , enter the transaction id in the resulting dialog box and click OK. It will give us details about which service image contains the transaction and where to find it.
L	Click Close to close the window.

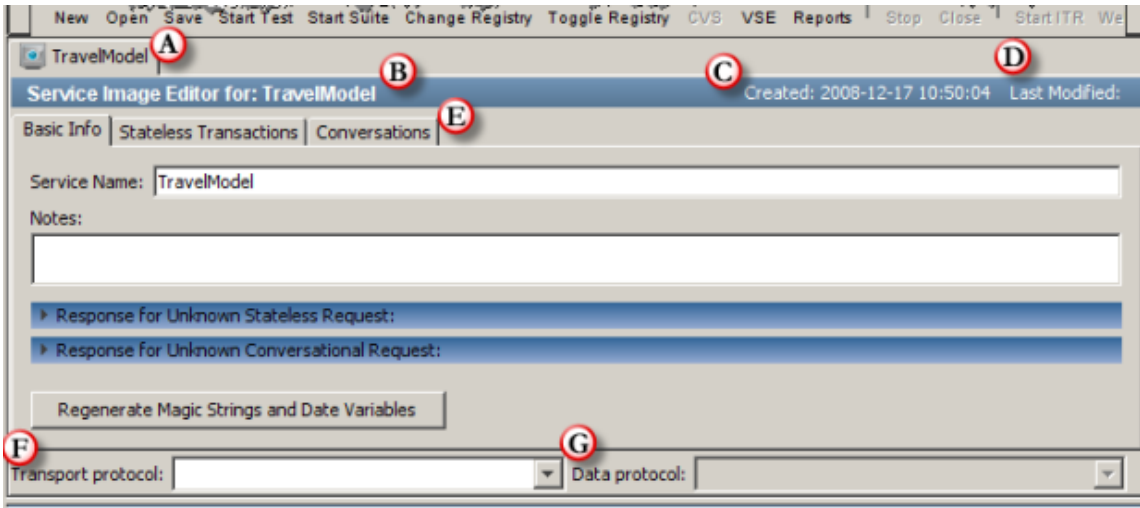
6.2 Service Image Editor

6.2 Service Image Editor


From the VSE Service Images window, clicking "View/Edit" or "Create New Virtual Service Image" will launch the service image editor. In the former case, the service image selected in the VSE Service Images window will be opened in the editor. In the latter case, a blank service image is created and opened.

You can alternatively access the editor from the Response Selection step in the .vsm by clicking **View/Edit Selected Service Image**.

The following screenshot shows the components which are visible through all the tabs of the Service image editor.



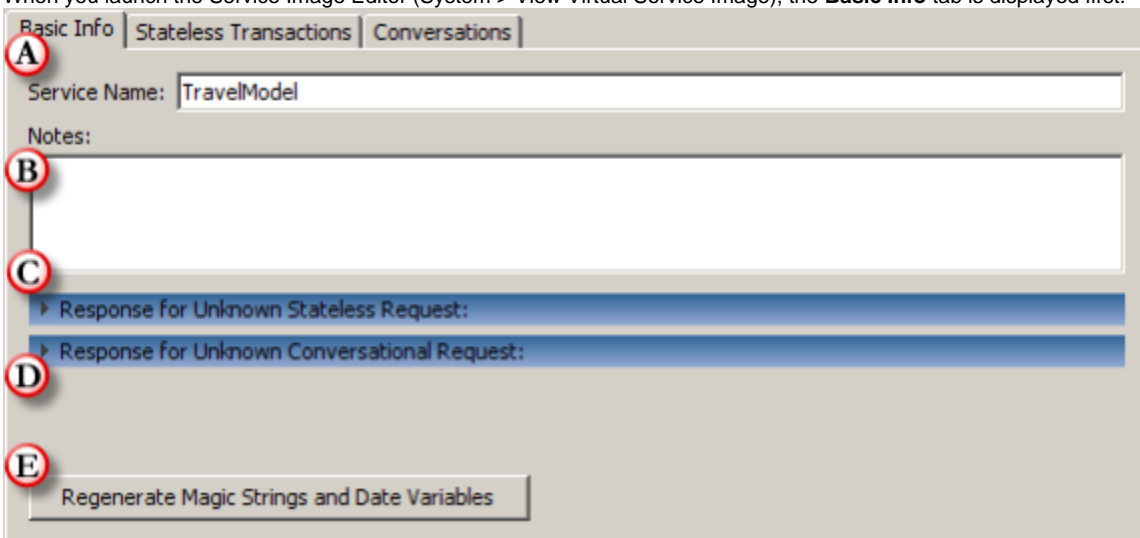
Service Image Editor Components

Callout	Description
A	LISA Workstation tab that identifies the virtual service image when you work in other LISA components. The  icon identifies the tab as the Service Image Editor.
B	Name field that identifies the virtual service image.
C	Date and timestamp when the service image was created.
D	Date and timestamp when the service image was last modified.
E	Information tabs for the service image.
F	Transport protocol setting. If needed, select the protocol from the list: HTTP/S or JDBC. The fields in the tabs change according to the protocol.
G	Data protocol setting. If you selected HTTP/S in the Transport Protocol field, select the appropriate data protocol from the list: HTTP Traffic (web), Web Services (SOAP), Web Services Bridge.

6.3 Service Image Editor Basic Info Tab

6.3 Service Image Editor Basic Info Tab

When you launch the Service Image Editor (System > View Virtual Service Image), the **Basic Info** tab is displayed first.



Basic Info Tab Components

Callout	Description
A	Use the Service Name field to edit the service image name, if needed.
B	Use the Notes field to document information about the service image.
C	Use the Response for Unknown Stateless Request area to enter the response to return for unknown stateless requests during playback. See below for more information.
D	Use the Response for Unknown Conversational Request area to enter the response to return for unknown conversation requests during playback. See below for more information.
E	Use the Regenerate Magic Strings and Date Variables if you have added or changed transactions.

Editing Responses for Unknown Requests

To enter a response for unknown stateless and conversational requests, expand the areas by clicking the arrows. The **Response Body** and **Response Meta Data** areas are displayed. Use the arrows to expand or collapse these areas as needed.

The screenshot shows the Service Image Editor interface. The 'Response for Unknown Stateless Request' and 'Response for Unknown Conversational Request' areas are expanded. The 'Response Body' area is also expanded. The 'Response Meta Data' area is expanded and contains a table with 'Key' and 'Value' columns. The 'Regenerate Magic Strings and Date Variables' button is at the bottom.

Response Body Area

In the **Response Body** area, complete the fields as needed:

Unnamed Text area: Contains the response to be returned for unknown stateless requests during playback.

Think time spec: Enter the amount of think time required in milliseconds. The think time is the time that is taken before sending out the response to a request. The default setting is **0**. If you enter a range, the think time is randomly selected within that range (for example, **100-1000** specifies a random think time between 100 and 1000 milliseconds). You can specify time measurements by adding a suffix to the numbers (case does not matter). For example, **10t-5s** indicates a random think time between 10 milliseconds and 5 seconds. The valid suffixes are:

t—milliseconds
s—seconds
m—minutes
h—hours

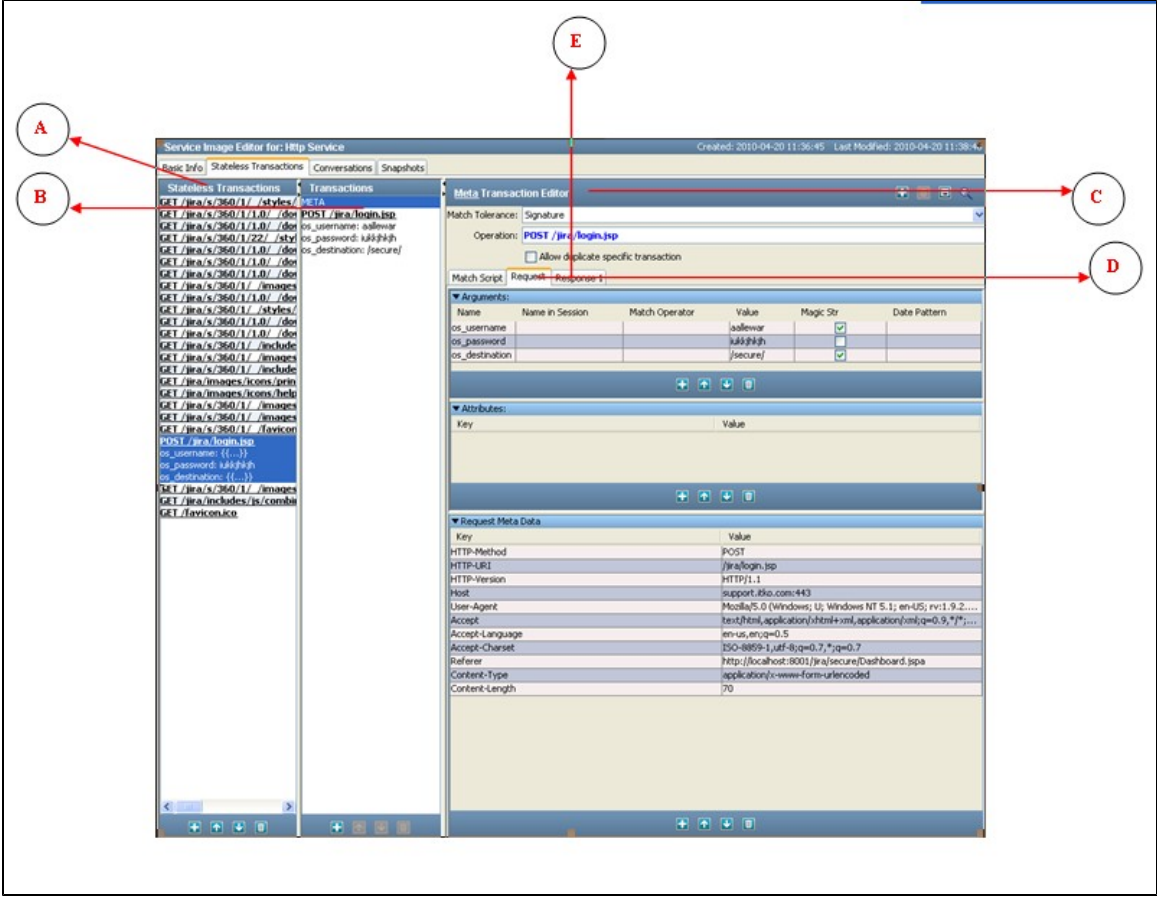
Response Meta Data Area

In the **Response Meta Data** area, use the toolbar at the bottom to add, move, or delete key-value pairs as needed.

6.4 Service Image Editor Stateless Transactions Tab

6.4 Service Image Editor Stateless Transactions Tab

Access the **Stateless Transactions** tab from the Service Image Editor (**System > View Virtual Service Image**). The editor title bar indicates the selection as **Specific** or **Meta**. Click the **Save** icon to save changes. Click the **Zoom** icon to expand the column to the full width of the window. Click the icon again to return the editor to its normal state and view all areas.



Stateless Transactions Tab Components

Callout	Description
A	Use the Stateless Transactions list to view and edit stateless transactions. Use the toolbar at the bottom of the pane to add, move, or delete stateless transactions.
B	One logical transaction (in the stateless transaction list) contains exactly one META transaction and any number of specific transactions. The Transactions list shows these transactions under the logical transaction. Use the toolbar at the bottom of the pane to add, move, or delete stateless transactions.
C	Use the Transaction Editor to view and edit transaction requests and response data for either Specific or Meta transactions, selected from the Transactions area. Use the small toolbar to add, delete, and save transaction information. You can also select a match tolerance for the given transaction here. More information on this is given below.
D	The Transaction Request Data tab shows the stateless requests. See below for more information.
E	The Transaction Response Data tab shows the response to the stateless requests. See below for more information.

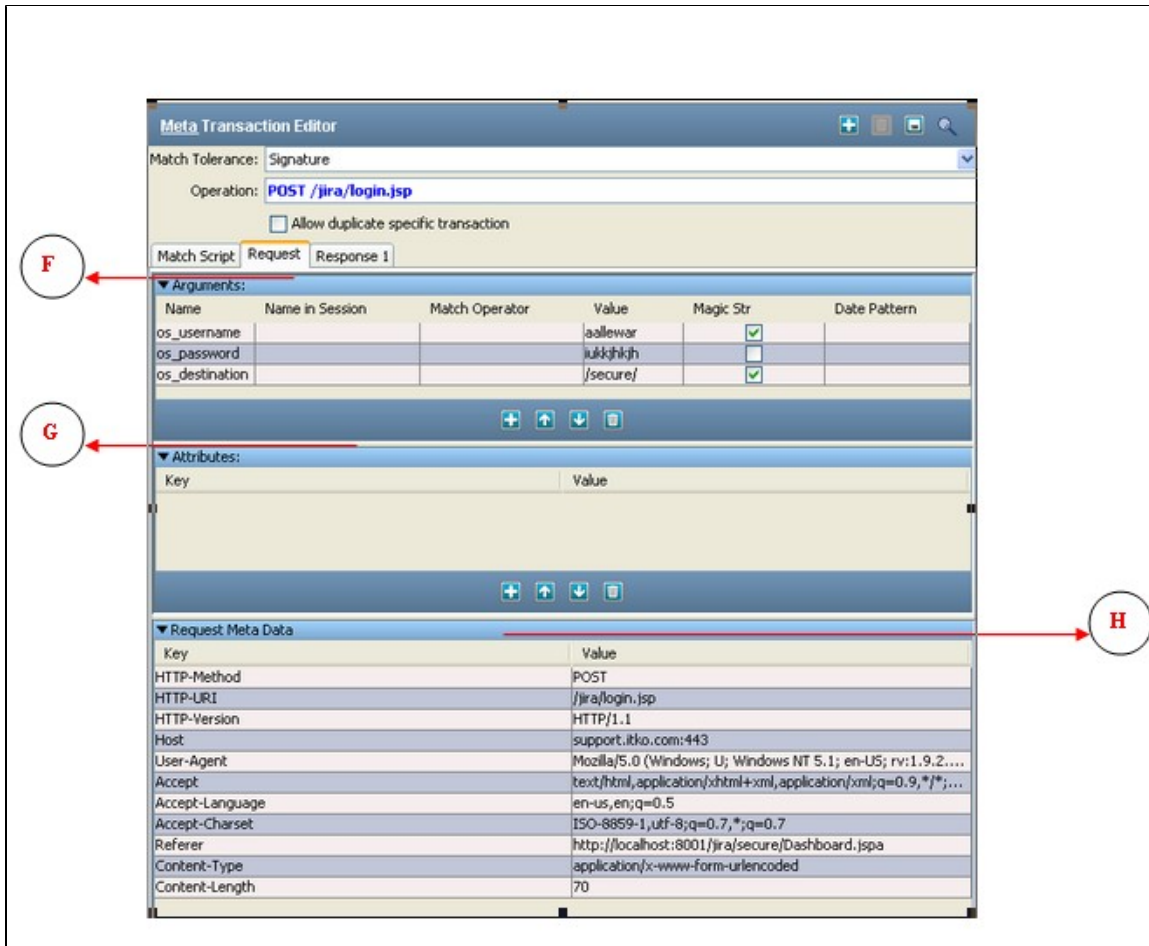
Note: If you add or change several transactions, return to the **Basic Info** tab to click **Regenerate Magic Strings and Data Variables**. LISA creates the magic string and date variables for you. Existing magic strings and variables are not modified.

Transaction Editor

Use the Transaction Editor to view and edit transaction data for specific or meta transactions. Select a specific transaction or **META** from the **Transactions** list.

The Transaction Editor includes **Transaction Request Data** and **Transaction Response Data**.

Transaction Request Data Editor

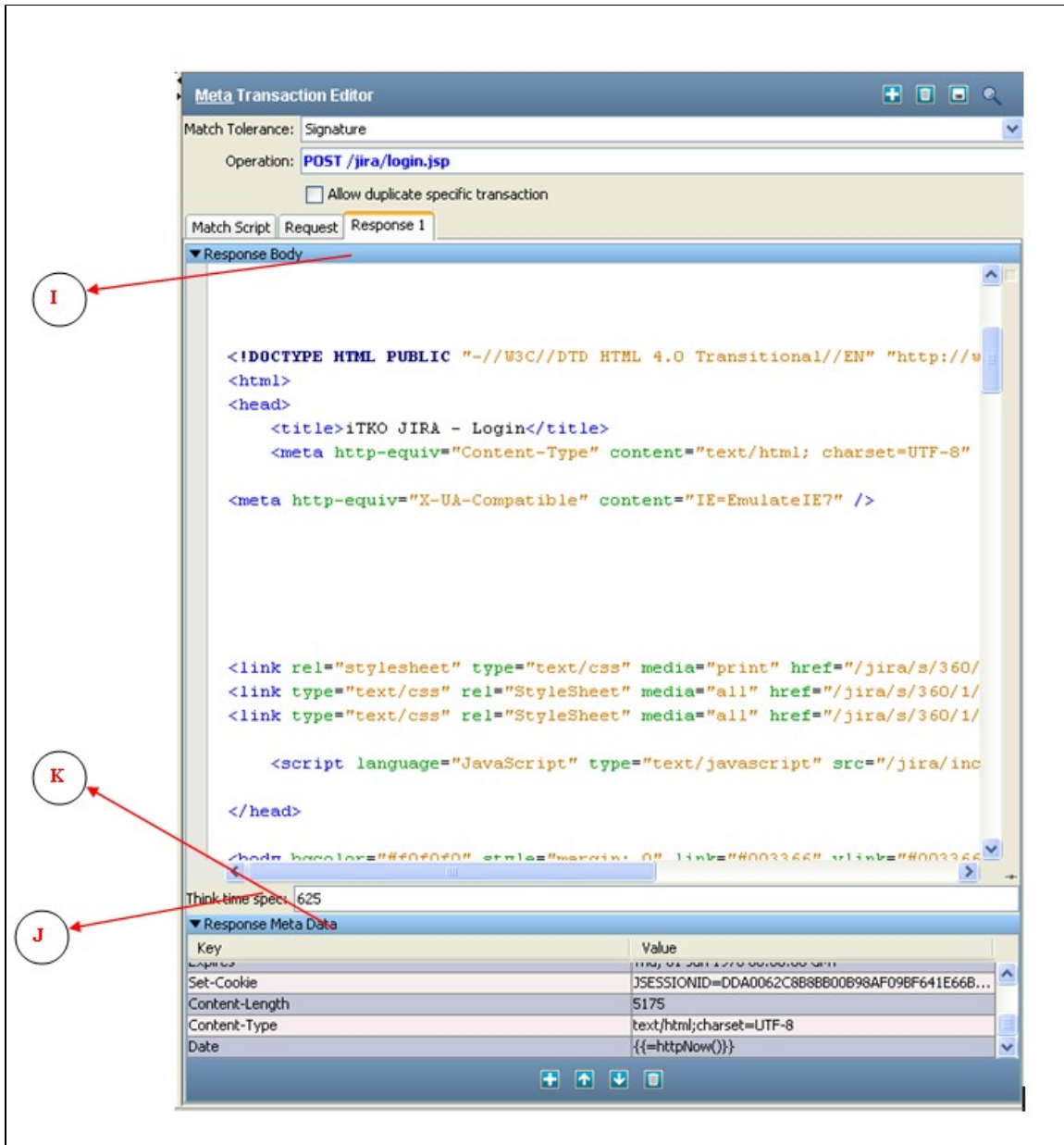


Transaction Request Data Editor Components

Callout	Description
F	Use the Arguments area to edit, add, move, or delete meta transaction arguments. For specific transactions, you can only edit aspects of the arguments. Note: To sort the arguments by data in a column, double-click the column heading to enable the sort arrows. Click the arrows to arrange the data in ascending or descending order. Click the arrows again to disable them.
G	Use the Attributes area to add, edit, move, and delete key-value pairs.
H	Use the Request Meta Data area to add, edit, move, and delete meta data key-value pairs.

Transaction Response Data Editor

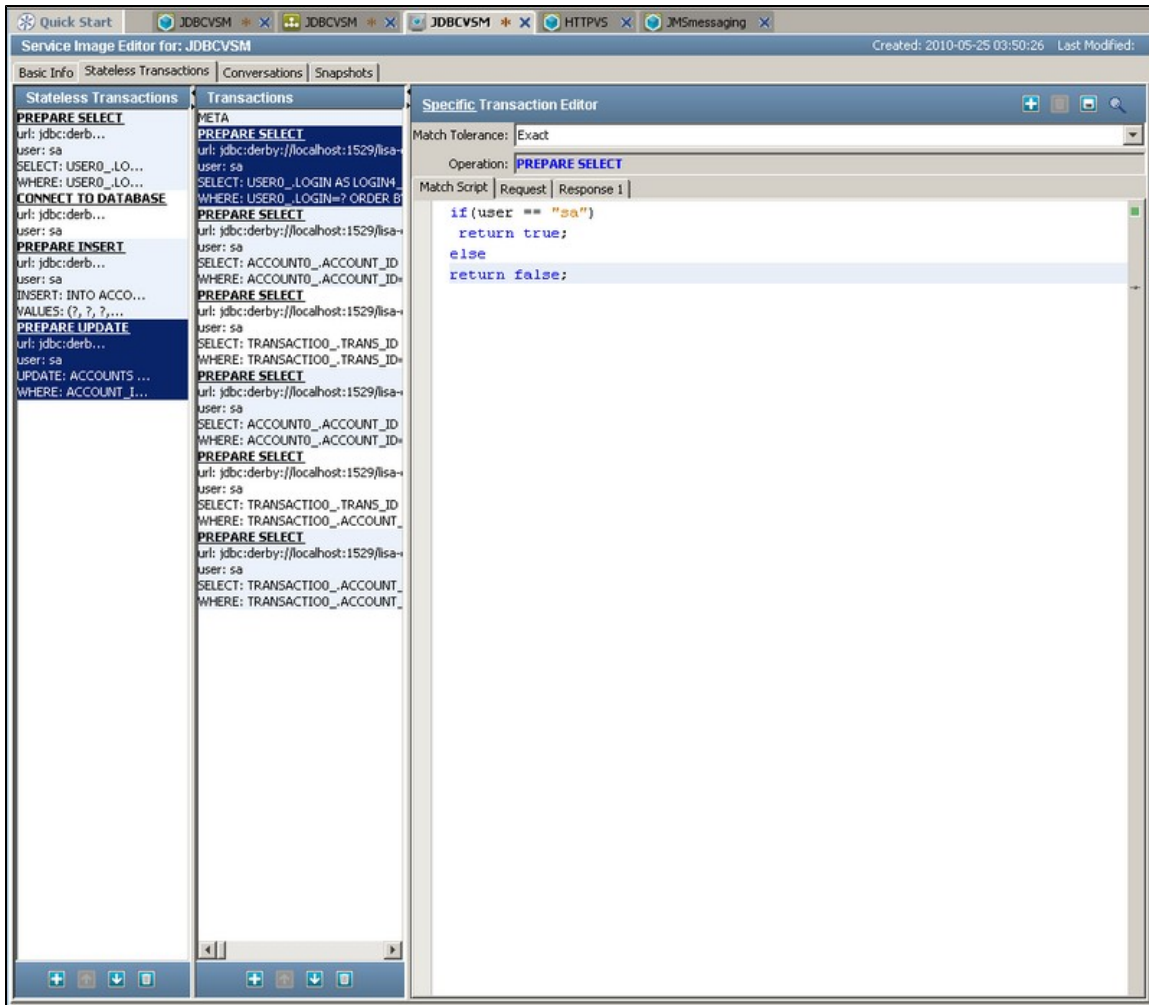
Use the **Transaction Response Data** to view and edit the response information. Click **Add** to add a response tab. Click **Delete** to remove a response tab. You cannot delete the first response because at least one is required.



Transaction Response Data Editor Components

Callout	Description
I	Use the Response Body area to edit the expected response for a transaction. Use the toolbar to edit, search, and format the response as needed. Note: Expand or collapse the area by clicking the arrow.
J	Edit the Think time spec as needed.
K	Use the Response Meta Data area to add, edit, move, and delete key-value pairs. Note: Expand or collapse the area by clicking the arrow.

Transaction Match Script Data Editor



Match Script Tab:

Match Script defines how VSE decides if a given transaction matches the incoming one. Unlike Match tolerance, here there is no need to specify levels of a match tolerance. We just write bean shell script performing certain action to return the specific match based on the given condition.

For example:

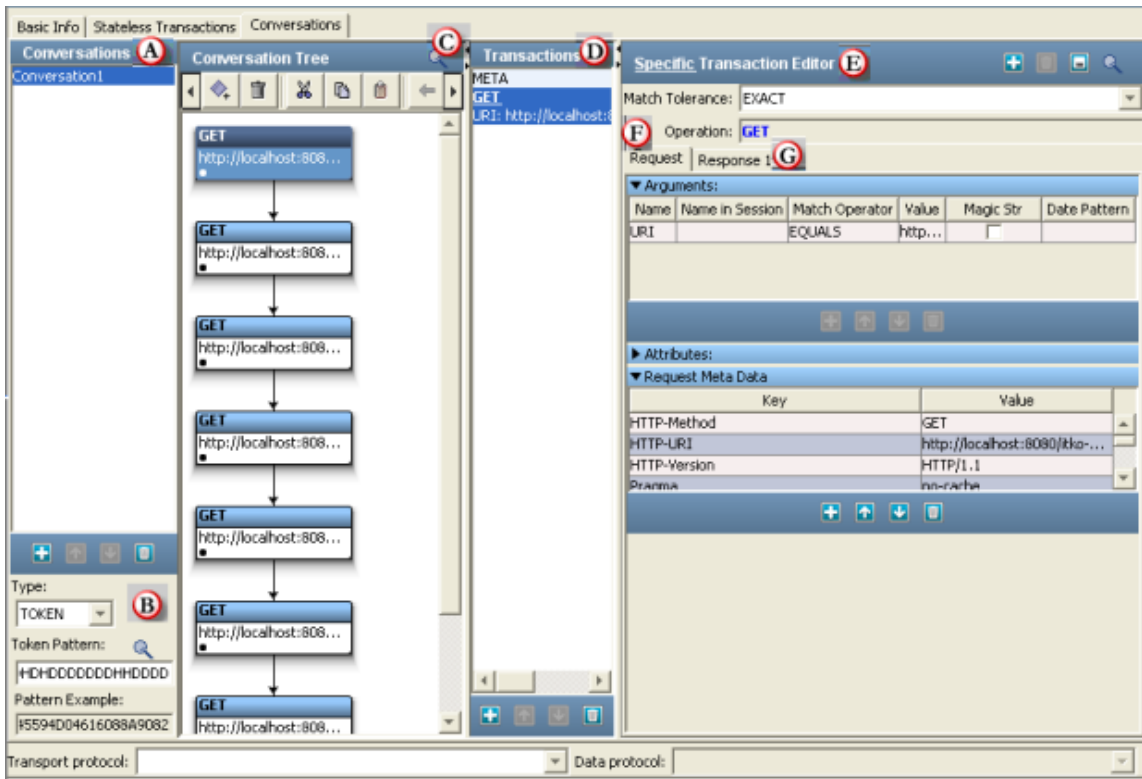
```
/* always match name=joe */
ParameterList args = incomingRequest.getArguments();
if ("joe".equals(args.get("name"))) return true else return defaultMatcher.matches();
```

For the match script to work there is no need to specify any match tolerance level, or there is no need to specify any match operator. It just finds the match based on the condition in the match script.

6.5 Service Image Editor Conversations tab

6.5 Service Image Editor Conversations tab

Access the **Conversations** tab from the Service Image Editor (**System > View Virtual Service Image**).



Conversations tab components

Callout	Description
A	The Conversations list shows all the conversations in the service image. Select a conversation to view and edit. A conversation consists of number of logical transactions.
B	The Type is either INSTANCE or TOKEN . If the conversation is to be token based, the Token Pattern field becomes available.
C	The Conversation Tree editor displays the logical conversation selected in the Conversation list in either a graph node tree view or a standard tree view.
D	Use the Transactions list to view and edit specific transactions or the meta data for transactions inside a selected logical transaction. Use the toolbar at the bottom of the pane to add, move, or delete stateless transactions.
E	Use the Transaction Editor to view and edit transaction requests and response data for either Specific or Meta transactions, selected from the Transactions area. The fields are dependent on the selected transport and data protocols. See below for more information. For more information about the Transaction Editor and its tabs, see the Transaction Editor section for the stateless requests .
F	Use the Transaction Request Data tab to enter the data for conversational requests during playback.
G	Use the Transaction Response Data tab to view and edit the response content, think time, and key-value pairs for the specific or meta transaction.

Note: If you add or change several transactions, return to the **Basic Info** tab to click **Regenerate Magic Strings and Data Variables**. LISA creates the magic string and date variables for you. Existing magic strings and variables are not modified.

6.6 Conversation Tree Editor

6.6 Conversation Tree Editor

Use the Conversation Tree editor to view and edit recorded transactions or create transactions manually. You can view the navigation trees in two display modes:

[Graph View](#)
[Tree View](#)

When you switch between views, the selected node remains selected. In both views, you can perform these actions from the editor toolbar,

right-click menus, or the view itself:

The Conversation Tree Editor Toolbar

The Conversation Tree editor toolbar varies slightly, depending on the display.

Graph View Tools










Tree View Tools



Note: If a tool is grayed out, it cannot be used with the selected transaction.

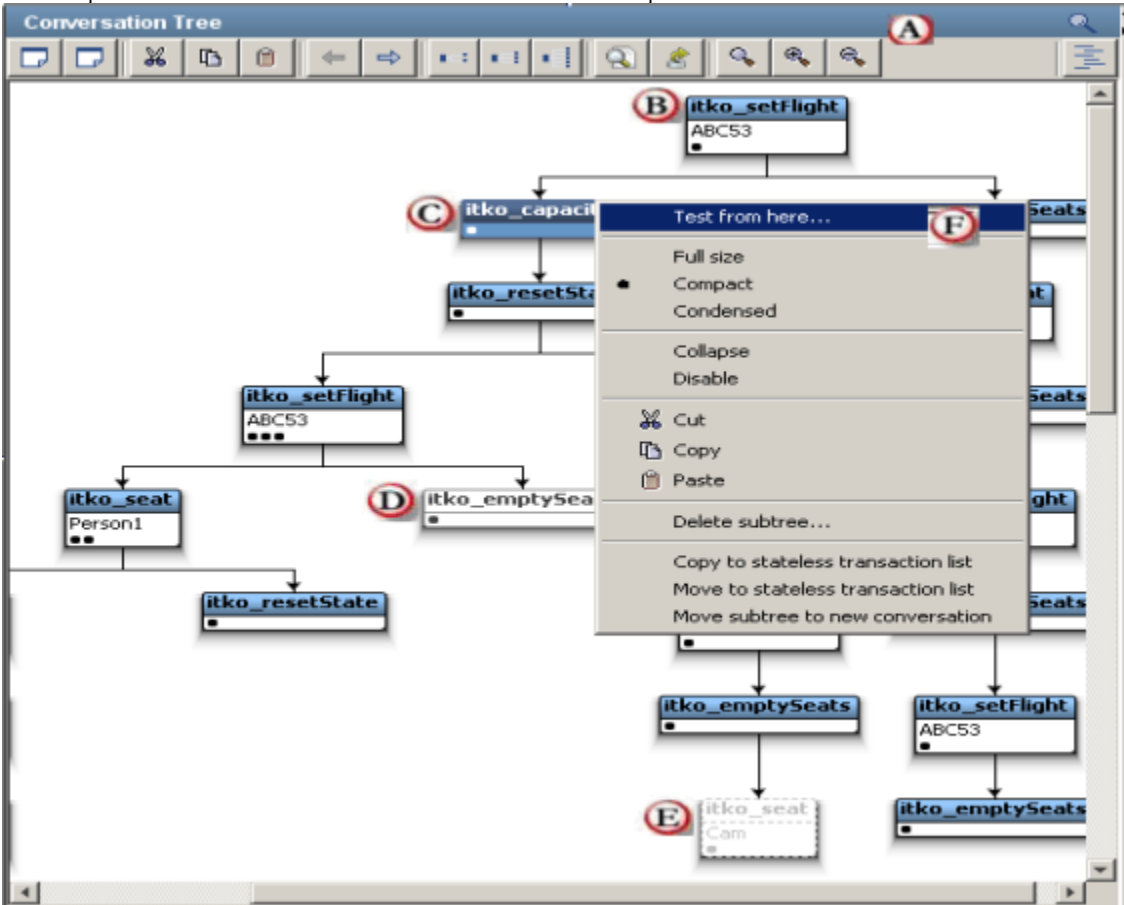
Conversation Tree Editor Tools

Tool	Icon	Description
Zoom		Expand the pane to fill the LISA Workstation window. Click the icon again to resize the pane.
New transaction		Create a new transaction below the selected node.
Delete transaction		Delete the selected node.
Cut		Cut the selected transaction.
Copy		Copy the selected transaction.
Paste		Paste the cut or copied transaction below the selected transaction.
Move earlier (Graph view)		Move the selected node earlier in the sibling list.
Move later (Graph view)		Move the selected node later in the sibling list.
Move earlier (Tree view)		Move the selected node earlier in the sibling list.
Move later (Tree view)		Move the selected node later in the sibling list.
Highlight CLOSE		From the selected node, highlight the nodes that can result in CLOSE navigation.
Highlight WIDE		From the selected node, highlight the nodes that can result in WIDE navigation.
Highlight LOOSE		From the selected node, highlight the nodes that can result in LOOSE navigation.

Display the Find bar		Toggle the visibility of the Find bar at the bottom of the pane.
Show/Hide Transaction id		Shows or hides the transaction id (useful in debugging).
Reset size		Reset the size to 1:1 ratio after using Zoom in or Zoom out tools.
Zoom in		Enlarge the node size in the view.
Zoom out		Decrease the node size in the view.
Switch display to tree view		Switch the display view to the tree view.
Switch display to graph view		Switch the display view to the graph view.

Conversation Tree Editor Graph View

This sample screen shows the conversation in the LISA Travel example:



Nodes are displayed according to status. The Conversation Tree editor components and node themes are described below by the image callout letter:

Callout	Description
---------	-------------

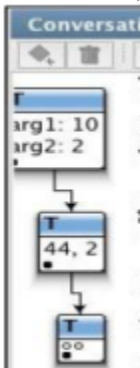
A	Toolbar. For more information, see Conversation Tree Editor Toolbar
B	Standard node appearance.
C	Selected node.
D	Collapsed node. Any children nodes are not displayed. Expand the node to view any children nodes.
E	Disabled node. This node and any children nodes (not displayed) are ignored during runtime.
F	<p>Right-click menu. When you right-click a transaction, the menu provides these actions:</p> <ul style="list-style-type: none"> • Test from here • Full size/Compact/Condensed • Collapse/Expand • Disable/Enable • Cut • Copy • Paste • Delete subtree • Copy to stateless transaction list • Move to stateless transaction list • Move subtree to new conversation

Node Display Styles

You can display nodes in three styles:

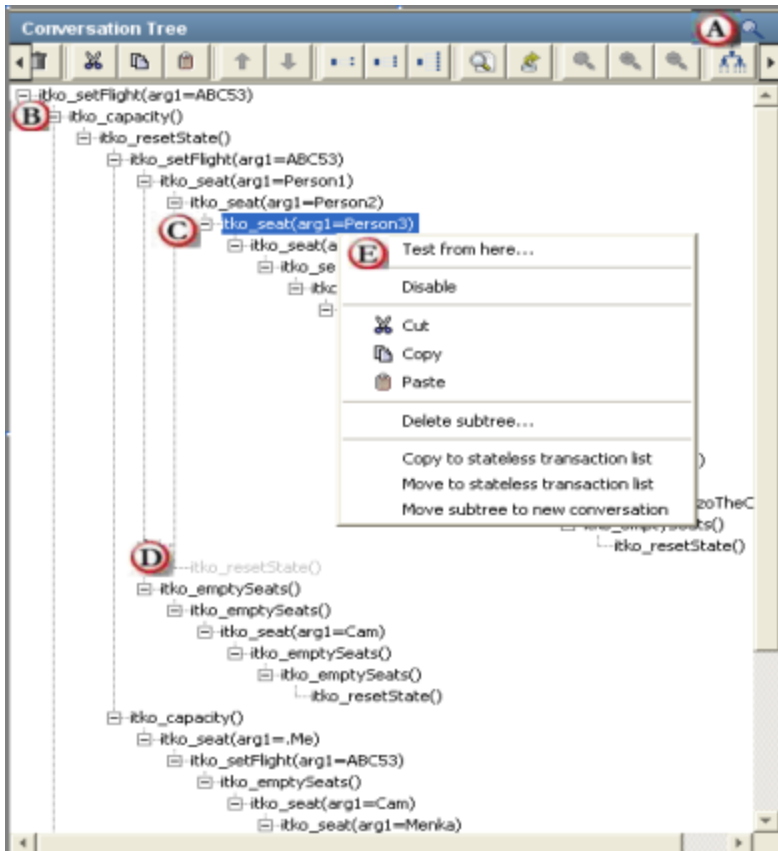
- Full size
- Compact (default)
- Condensed

In all three styles, the row of black dots at the bottom of each node shows how many specific transactions belong to the node. In the condensed style, the hollow dots show the number of arguments to the transaction's request. In the following figure, the first node is displayed full size, the second is compact and the third one is condensed.



Conversation Tree Editor Tree View

The tree view displays the same information as the graph view in a compact fashion.



Nodes are displayed according to status. The Conversation Tree editor components and node themes are described below by the image callout letter:

Callout	Description
A	Toolbar. For more information, see using the Conversation Tree Editor toolbar .
B	Standard node appearance.
C	Selected node.
D	Disabled node. This node and any children nodes (not displayed) are ignored during runtime.
E	Right-click menu. When you right-click a transaction, the menu provides these actions: <ul style="list-style-type: none"> • Test from here • Disable/Enable • Cut • Copy • Paste • Delete subtree • Copy to stateless transaction list • Move to stateless transaction list • Move subtree to new conversation

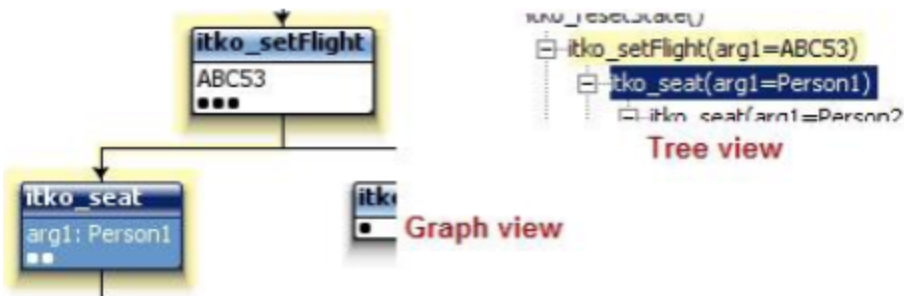
The Test from Here Action

You can use the Conversation Tree editor to test for an expected response from a selected transaction in both graph and tree views.

To test from a transaction:

1. In the Conversation Tree editor, select and right-click a transaction.
2. From the menu, select **Test from here**.

Graph View	Tree View
------------	-----------



Note: If the test is not successful, the following error will appear: "No transaction matching the request follows the selected one in this conversation." Click **OK** to continue.

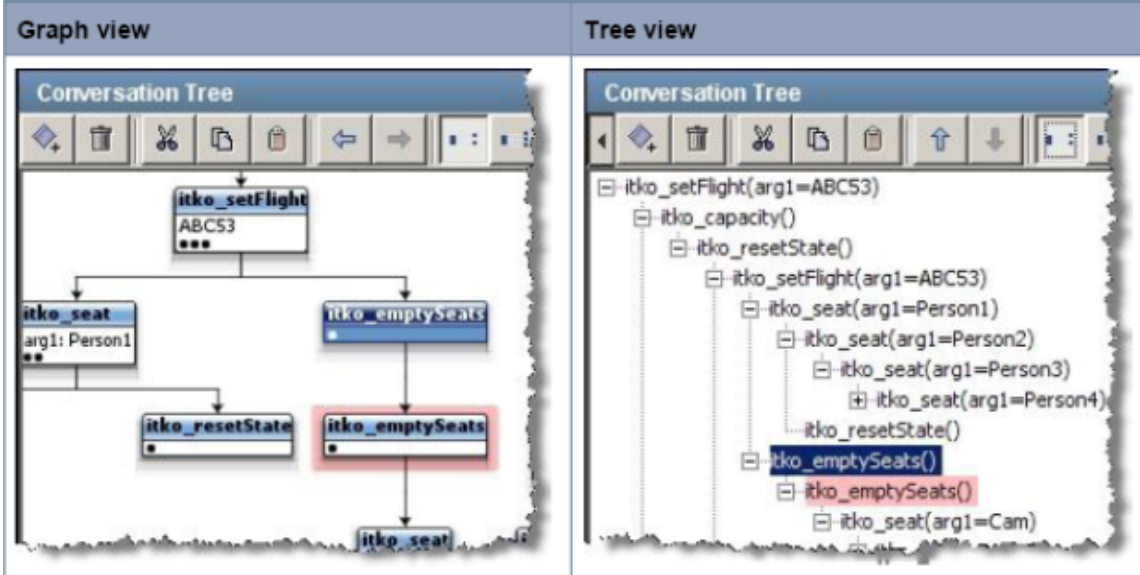
5. Click Close to close the window.

Highlighting Navigation Possibilities

Use the Conversation Tree to view navigation possibilities in a conversation based on the selected navigation tolerance.

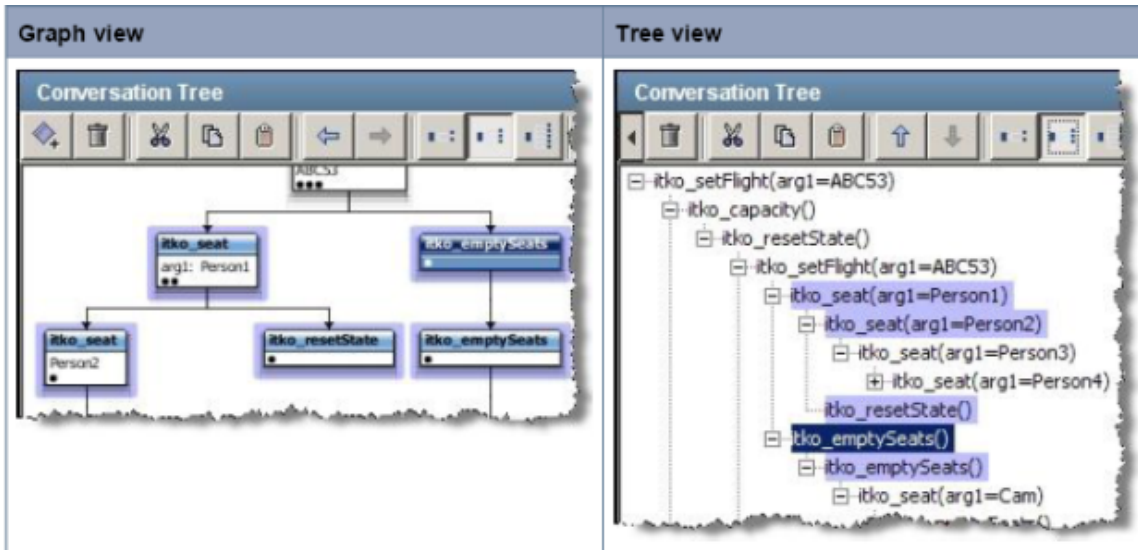
Viewing Close Navigation

Select a transaction and click the **CLOSE** button. The transactions that are searched in the selected transaction's subtree are highlighted in red.



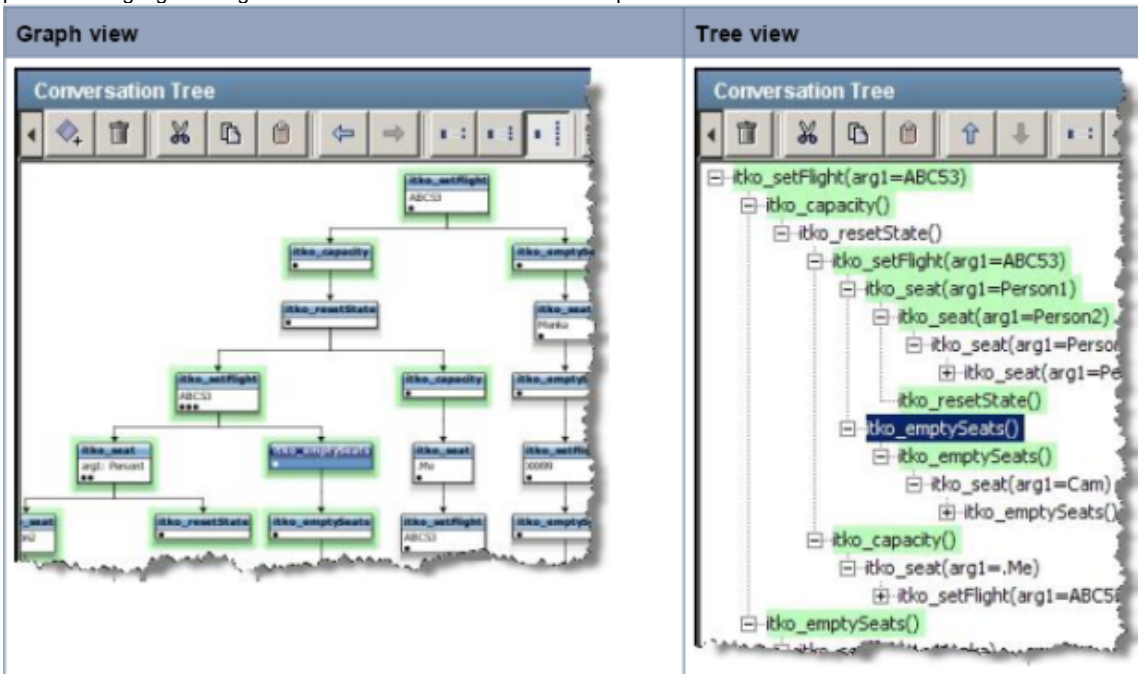
Viewing Wide Navigation

Select a transaction and click the **WIDE** button. The transactions that are searched in the selected transaction's subtree and sibling subtrees are highlighted in blue.



Viewing Loose Navigation

Select a transaction and click the **LOOSE** button. The transactions that are searched in the selected transaction's subtree, sibling subtrees, and parent are highlighted in green. A full conversation restart is also possible.

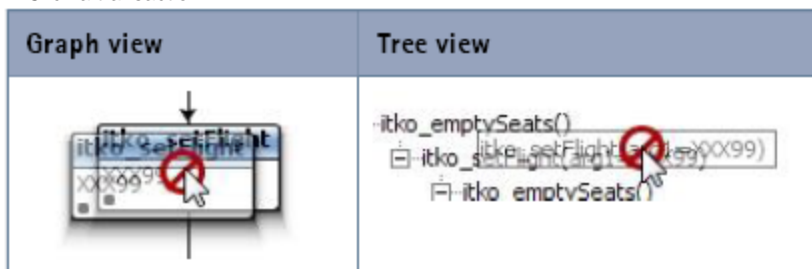


Restructuring the Conversation Tree

In some cases, you may want to restructure a conversation by dragging and dropping a transaction and its subtree, if it has one. You can perform this action in both the graph and tree views.

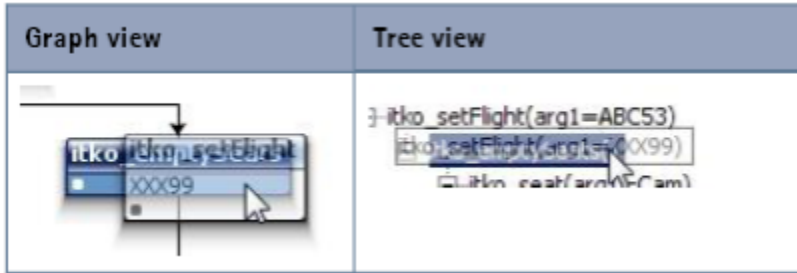
To restructure a conversation tree:

1. Click a transaction.



As you drag the transaction, it displays the universal symbol for "not possible."

2. Drag the transaction to the transaction that should be the new parent transaction. If it is possible to drop the transaction, then the "not possible" symbol disappears.



3. Drop the transaction. The transaction and any transactions in its subtree will be moved below the new parent.

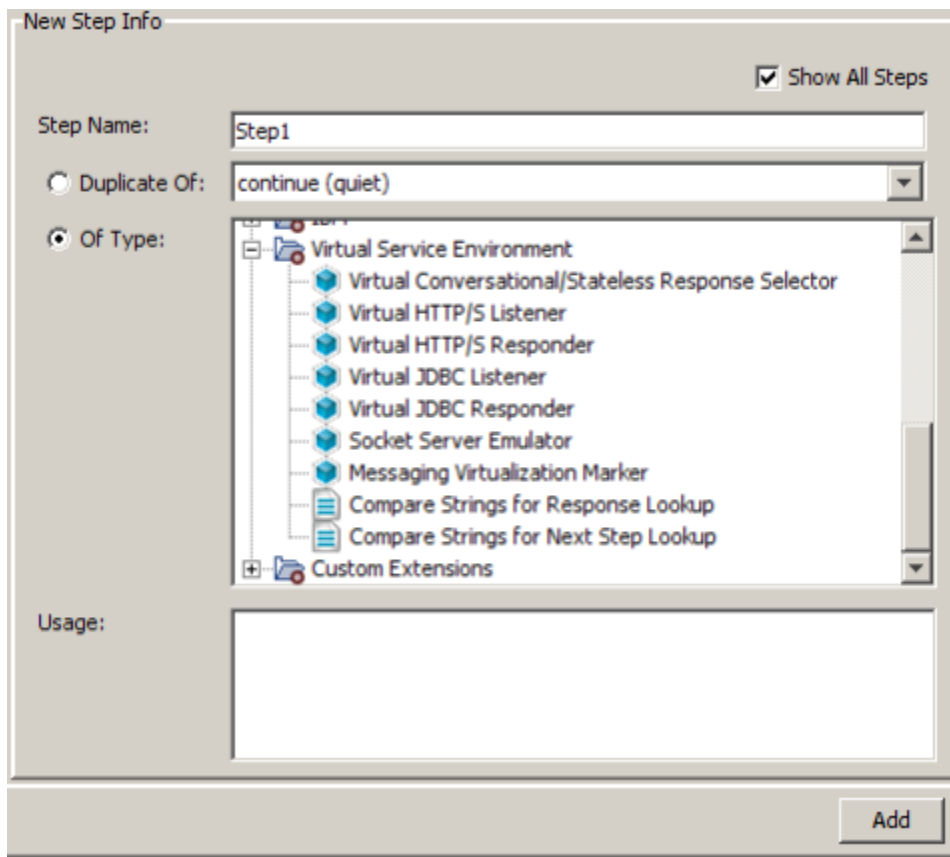
7. Editing a VSM

7. Editing a VSM

We have seen that a VS image recording creates a Virtual Service Model (VSM) with 3 steps or 5 steps, depending on the option chosen. There may also be situations when one would like to edit the VSM by editing generated steps or adding further steps to it.

Furthermore, the recorder cannot be used for message-based services. It is then necessary to create the VSM and service image manually. Feel free to skip this chapter if you do not need to edit a VSM or create it afresh without a recorder.

A VSM is a specialized LISA test case and hence editing/creation of a VSM is similar to a LISA test case. There are quite a few steps listed under "Virtual Service Environment" tab that may be added to a VSM, as shown below. It is also possible to add a step that does not appear under this tab. However, some of these other steps may make a VSM undeployable.



This chapter explains the steps under the "Virtual Service Environment" tab.

The following topics are available.

- [7.1 Virtual Conversational/Stateless Response Selector Step](#)
- [7.2 Virtual HTTP/S Listener step](#)
- [7.3 Virtual HTTP/S Responder Step](#)
- [7.4 Virtual JDBC Listener step](#)
- [7.5 Virtual JDBC Responder Step](#)
- [7.6 Socket Server Emulator Step](#)
- [7.7 Messaging Virtualization Marker Step](#)
- [7.8 Compare Strings for Response Lookup Step](#)
- [7.9 Compare Strings for Next Step Lookup Step](#)

7.1 Virtual Conversational_Stateless Response Selector Step

7.1 Virtual Conversational_Stateless Response Selector Step

This step is responsible for deciding on an appropriate virtual response for a given request, and can thus be seen as the main step in any VSM. It does that by looking at a service image that is set into it. It is possible that there can be more than one response for a request; therefore, the responses are always emitted as a list. It is typically created by recording and virtualizing some form of service traffic.

Virtual Response Selector Setup Information				
Service Image to use:	<input type="text"/> Set to Selected			
Request property name:	<input type="text" value="conversational.request"/> ▼			
<input checked="" type="checkbox"/> Format step response as XML				
If exception:	<input type="text" value="fail"/> ▼			
Service Images List				
ID	Name	Created	Last Modified	Description
1	si-database	02/10/2009 11:18:37		
View/Edit Selected Service Image		Refresh Service Image List		

Enter the data for the fields as described:

Field	Description
Service Image to use	From the Service Images List area, select the service image you need. Click Set to Selected . If the service image list is empty and you are not going to record one, go to System > View Virtual Service Images and import one.
Request property name	Set the property name to define the property to look in for the inbound request. Required. Note: This is usually the response of the previous step.
Format step response as XML	By default the step response is formatted as XML. The VSE framework expects Respond steps to accept either a response object, a list of response objects, or an XML document that represents either. If this field is unchecked, the step outputs a list of response objects, even if the list contains only one response.
If exception	Select the step to go to if an exception occurs. The default is fail .
Service Images List	Lists all the available service images. Select one from the list to view, edit, or set as the service image the step is to use by then clicking Set to Selected .
View/Edit Selected Service Image	After you select a service image from the list, click the View/Edit button to open the Service Image Editor. For more information, see Service Image Editor .
Refresh Service Image List	Click to refresh the list of service images to display changes.

7.2 Virtual HTTP_S Listener step

7.2 Virtual HTTP_S Listener step

This step is used to simulate an HTTP server, including SSL support. It listens for incoming HTTP requests and converts them to a standard

virtual request format.

Virtual HTTP/S Listen Setup Information

Listen port: [] Bind address: [] ☐ Bind only

☐ Use SSL SSL keystore file: [] Select...

Keystore password: [] Verify...

Base path: []

Data protocol: HTTP Traffic (web)

☒ Format step response as XML

If exception: fail

Enter the data for the fields as described:

Field	Description
Listen port	Enter the port on which LISA listens for the HTTP/S traffic.
Bind address	Enter the local IP address on which connections can come in. By default with no bind address specified, the listen step accepts connections on the specified port regardless of the NIC (or IP address) it comes in on.
Bind only	Check this box to acquire the network resource and move to the next step. A second Listen step is required that does not use the Bind only option. This option allows the model to listen on a port (requests are queued until a listen step consumes them) and perform setup tasks before dropping into the wait/process/respond loop. For example, Step 1 of the model acquires the listening port (using Bind only) and step 2 triggers external software that sends requests.
Use SSL	Check this box if you want to simulate a secure HTTPS Web site. Then supply the SSL keystore information.
SSL keystore file	Click Select to browse to your SSL keystore file. The same keystore file must be available to the VSE server to which the VS model is deployed.
Keystore password	Enter the keystore password, and click Verify .
Base path	Identify the HTTP requested resource URIs that the listen step is to process. When the request comes in, the list of queue names is scanned for a name (base path) that starts the URI on the request. The queue name that matches is the one into which the request is placed and the listen step that is associated with the queue (by base path) processes the request.
Data protocol	Select one of the available protocols: HTTP traffic (web) Web Services (SOAP) Web Services Bridge
Format step response as XML	By default the step response is formatted as XML. The VSE framework expects Respond steps to accept either a response object, a list of response objects, or an XML document that represents either. If this field is unchecked, the step outputs a list of response objects, even if the list contains only one response.
If exception	Select the step to go to if an exception occurs. The default is fail .

7.3 Virtual HTTP_S Responder Step

7.3 Virtual HTTP_S Responder Step

This step is used in conjunction with the **Virtual HTTP/S Listener** step to transmit responses to HTTP requests produced by the listener. It takes a virtual response and uses it as the reply to the corresponding request using the HTTP/S protocol.

You can create this step by recording and virtualizing HTTP traffic.

Virtual HTTP/S Responder Setup Information

Responses list property name:

If exception:

Conversational Model Properties

Enter the data for the fields as described:

Field	Description
Responses list property name	The name of the property to look in for the response to send.
If exception	Select the step to go to if an exception occurs. The default is blank.
Conversational Model Properties	Enter a property and click Add . To delete a property, select it from the list and click Remove . The properties listed here are associated with the current conversation session, which allows their values to be available to downstream conversational requests.

7.4 Virtual JDBC Listener step

7.4 Virtual JDBC Listener step

This step is used to control the simulation of JDBC database traffic. It manages the communication with the simulation driver that is embedded in the database client.

Virtual JDBC Request Listener Setup Information

JDBC simulation driver host:

JDBC simulation driver port:

☒ Format step response as XML

If exception:

Installed and Initialized JDBC Drivers

Current SQL Activity

Enter the data for the fields as described:

Field	Description
JDBC simulation driver host	Enter the host name or IP address of the database client where the simulation driver is running.
JDBC simulation driver port	Enter the port number for the JDBC simulation driver. The default is 2999 .
Format step response as XML	By default the step response is formatted as XML. The VSE framework expects Respond steps to accept either a response object, a list of response objects, or an XML document that represents either. If this field is unchecked, the step outputs a list of response objects, even if the list contains only one response.
If exception	Select the step to go to if an exception occurs. The default is fail .

Connect/Disconnect	Click Connect to connect to the JDBC simulator. If connected, click Disconnect to end the connection. You can use this feature to validate the connection information.
Installed and Initialized JDBC Drivers	Displays the JDBC drivers installed and initialized in the database client.
Current SQL Activity	Displays the current SQL activity within the database client.

7.5 Virtual JDBC Responder Step

7.5 Virtual JDBC Responder Step

This step is used to send the result of a JDBC data call as selected by a conversational response selection step to the simulation driver that is embedded in the database client.

Enter the data for the fields as described:

Field	Description
Responses list property name	The name of the property to look in for the response to send.
If exception	Select the step to go to if an exception occurs. The default is blank.
Conversational Model Properties	Enter a property and click Add . To delete a property, select it from the list and click Remove. The properties listed here are associated with the current conversation session, which allows their values to be available to downstream conversational requests.

7.6 Socket Server Emulator Step

7.6 Socket Server Emulator Step

This step can be used to simulate any text-based (typically HTTP) server socket. It supports listening, responding, and binding.

Virtual Socket Service - Step1

Virtual HTTP/Socket Setup Info

Process mode: Full Process

Listen port: 8080 Bind address: localhost

☐ Use SSL SSL keystore file: Select... Keystore password: Verify...

Base path: /

If environment error: Abort the Test Record terminator: ☐ Ensure proper HTTP response format ☒

Listener status: Not running Test Clear Listener

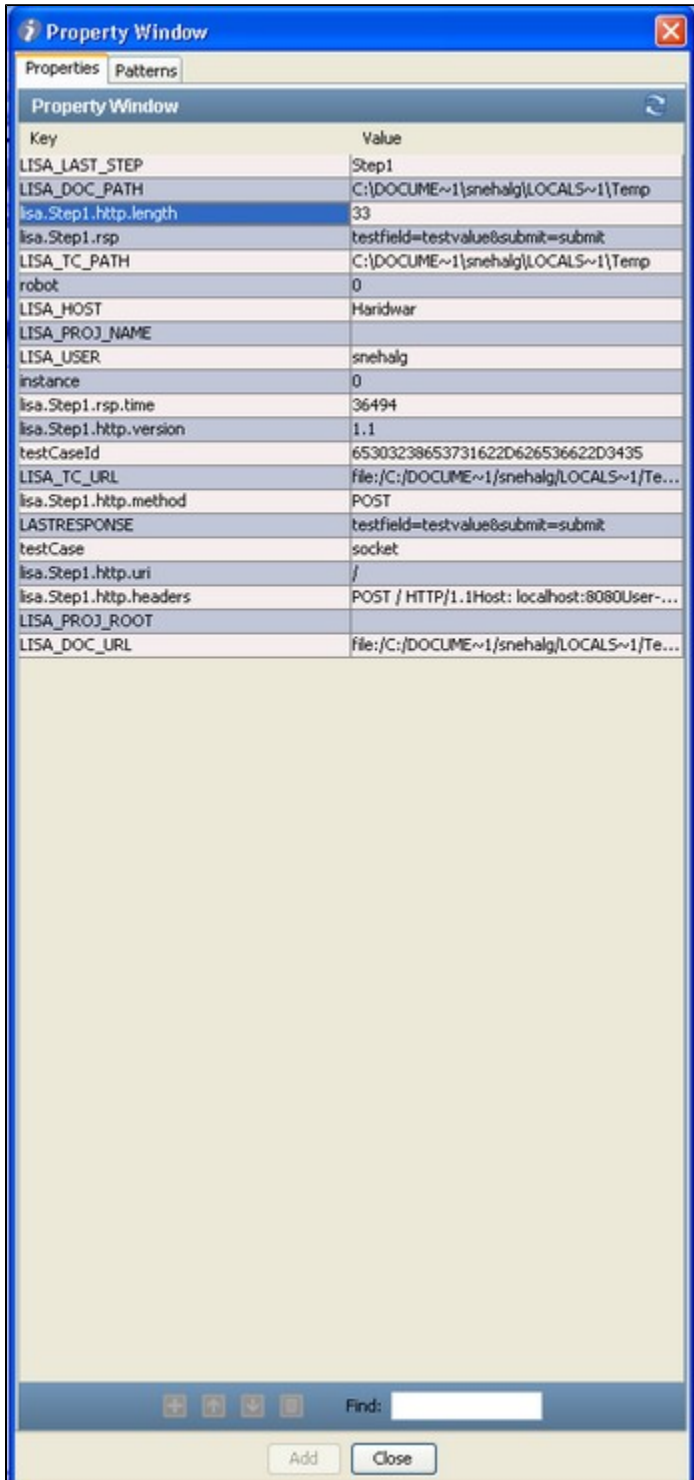
Response to Send

```
<html>
<head>Test</head>
<body>
  <h1>Test</h1>
</body>
</html>
```

Enter the data for the fields as described:

Field	Description
Process mode	From the list, select the process mode. The valid options are: Full Process--Default Asynchronous setup--Acquires the network resource and moves to the next step. Listen Only Respond Only
Listen port	Enter the port on which LISA listens for the HTTP or Socket traffic.
Bind address	Enter the local IP address on which connections can come in. By default with no bind address specified, the listen step accepts connections on the specified port regardless of the NIC (or IP address) it comes in on.
Close immediately	Check to use design-time testing of this step. This option instructs the step to do the configured work and then immediately cleanup its network resources.
Use SSL	Check this box if you want to simulate a secure HTTPS Web site. Then supply the SSL keystore information.
SSL keystore file	Click Select to browse to your SSL keystore file. The same keystore file must be available to the VSE server to which the VS model is deployed.
Keystore password	Enter the keystore password, and click Verify .
Base Path	Identify the HTTP requested resource URIs that the listen step is to process. When the request comes in, the list of queue names is scanned for a name (base path) that starts the URI on the request. The queue name that matches is the one into which the request is placed and the listen step that is associated with the queue (by base path) processes the request.
If exception	Select the step to go to if an exception occurs. The default is blank.
Record terminator	If the socket emulator is to simulate a record-based service, enter the character that marks the end of a record. If you leave this field blank, either line-oriented records or the HTTP protocol are simulated.
Ensure proper HTTP response format	By default, this option is checked. When in a process mode that sends a response and the response is to be a valid HTTP response, this option guarantees that the HTTP headers in the response text are correctly formatted and, if needed, that the Content-Length : HTTP response header is present and correct.
Listener status	Indicates whether the listener is running or not.
Test	Click to test the listener setup.

Clear Listener	Click to stop the test of the step.
Response tab	The Response to Send includes the text for the response.
Read Response From File	Click to browse the file system for a response.
Request tab	Last/Original Request is used only during design time. It displays the last request the step received.



lisa.step1.http.length property is seen getting added.

Request Tab

Listener status: Not running

Test

Clear Listener

Last/Original Request

Request

Response

7.7 Messaging Virtualization Marker Step

7.7 Messaging Virtualization Marker Step

This step is used to designate that a message-based test case is designed for use in the Virtual Service Environment. If the test case is listening or responding through JMS, this step should be added to the VS model to ensure it can be deployed to the VSE.

7.8 Compare Strings for Response Lookup Step

7.8 Compare Strings for Response Lookup Step

This step is used to look at an incoming request to a virtual service and determine the appropriate response in a completely stateless fashion, without referring to any service image. The stateful portions of VSE are not supported. You can match incoming requests using partial text match, regular expression, among others.

▼ Compare Strings for Response Lookup - Step1

Text to match:

Range to match: Start: End: "No Match" next step: "On Exception" step: HTTP/S Listen

☒ Store responses in a compressed form in the test case file.

Case Response Entries

Enabled	Name	Delay Spec	Criteria	Compare Type	Response
---------	------	------------	----------	--------------	----------

Find:

Criteria

Response

The step components are:

Field	Description
Text to match	Enter the text against which criteria should be matched. This is typically a property reference, such as LASTRESPONSE .
Range to match	Enter the start and end of the range.
"No Match" next step	From the list, select the step to go to if no match is found.

"On Exception" step	From the list, select the step to go to if the test fails.
Store responses in a compressed form...	Selected by default. This option compresses the responses in the test case file.
Case Response Entries	Add, move, and delete entries.
Enabled	Selected by default when you add an entry. Deselect to ignore an entry.
Name	Enter a unique name for the case response entry.
Delay Spec	Enter the delay specification range. The default is 1000-10000 , which indicates to use a randomly selected delay time between 1000 and 10000 milliseconds. The syntax is the same format as Think Time specifications.
Criteria	This area provides the string to compare against the Text to match field. To edit the criteria, in the Case Response Entries area select the appropriate row, and then select a different setting from the Criteria list.
Compare Type	Select an option from the list: <ul style="list-style-type: none"> Find in string (default) Regular expression Starts with Ends with Exactly equals
Response	This area provides the response of this step if the entry matches the Text to match field. To edit the response, in the Case Response Entries area select the appropriate row, and then select a different setting from the Response list.
Criteria	Allows for the update of the criteria string for an entry.
Response	Allows for the update of the step response for an entry.

7.9 Compare Strings for Next Step Lookup Step

7.9 Compare Strings for Next Step Lookup Step

This step is used to look at an incoming request and determine the appropriate next step. You can match incoming requests using partial text match, regular expression, among others.

Each matching criterion specifies the name of the step to which to transfer if the match succeeds.

The step components are:

Field	Description
Text to match	Enter the text against which criteria should be matched. This is typically a property reference, such as LASTRESPONSE .

Range to match	Enter the start and end of the range.
"No Match" next step	From the list, select the step to go to if no match is found.
"On Exception" step	From the list, select the step to go to if the test fails.
Next Step Entries	Add, move, and delete entries.
Enabled	Selected by default when you add an entry. Deselect to ignore an entry.
Name	Enter a unique name for the next step entry.
Delay Spec	Enter the delay specification range. The default is 1000-10000 , which indicates to use a randomly selected delay time between 1000 and 10000 milliseconds. The syntax is the same format as Think Time specifications.
Criteria	This area provides the string to compare against the Text to match field. To edit the criteria, in the Next Step Entries area select the appropriate row, and then select a different setting from the Criteria list.
Compare Type	Select an option from the list: <ul style="list-style-type: none"> Find in string (default) Regular expression Starts with Ends with Exactly equals
Next Step	From the list, select the step to go to if the match is found.
Criteria	Allows for the update of the criteria string for an entry.

8. Doing the Virtualization

8. Doing the Virtualization

Virtualization is the process where LISA Virtualize responds to the client in the absence of the server. It uses the **VSM** and the **Service image**. This chapter covers the virtualization process.

The following topics are available.

- [8.1 Preparing for Virtualization](#)
- [8.2 Deploying and Running a VSM](#)
- [8.3 Running Live Request Against LISA VSE](#)
- [8.4 Execution Mode](#)
- [8.5 Session Viewing and Model Healing](#)

8.1 Preparing for Virtualization

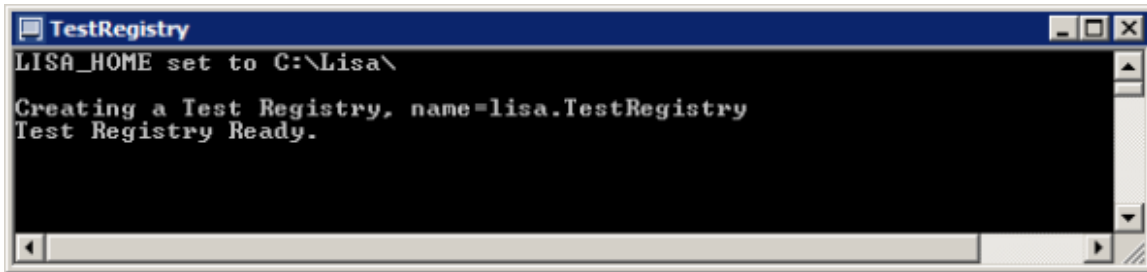
8.1 Preparing for Virtualization

Starting Test Registry

LISA Virtual Services Environment must be attached to a **Test Registry**. The Test Registry maintains a register of the Coordinator, Simulator Servers, and Virtual Service Environment.

To launch the Test Registry:

1. From the Start > Programs > LISA menu, select TestRegistry, which launches the TestRegistry window. The name of the default test registry is **lisa.TestRegistry**.



2. You can minimize the TestRegistry window. Do not close the window.

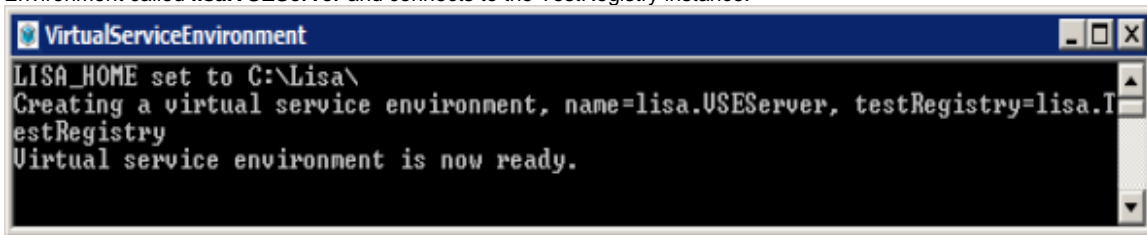
Note: You can use the Windows system service if this option was chosen during install process.

You can also start a named test registry from a command prompt by entering `[LISA_HOME]\bin\TestRegistry.exe -m TestRegistryName`, where **TestRegistryName** is the name of the test registry.

Starting Virtual Service Environment

The Virtual Service Environment (VSE) has to be started for virtualization to run. The VSE needs to register with the Test Registry.

1. From the Start menu, select Programs > LISA > VirtualServiceEnvironment, which launches a window that creates a Virtual Service Environment called **lisa.VSEServer** and connects to the TestRegistry instance.



2. You can minimize the Virtual Service Environment window. Do not close the window.

You can use the Windows system service if this option was chosen during install process.

You can also start a named virtual service environment from a command prompt by entering `[LISA_HOME]\bin\VirtualServiceEnvironment.exe --n VSEName -m TestRegistryName`, where **VSEName** is the name of the VSE and **TestRegistryName** is the name of an existing test registry. If the Test Registry is installed on a different computer, use the following RMI syntax to reach the server:
rmi://servername:port/test-registry-name|

Running Multiple Virtual Service Environments

If you want to run multiple VSE Servers on one computer, you must do one of the following:

- Configure LISA RMI port strategy (**lisa.rmi.port.strategy**) in the local.properties file to **anon**.
- Add **-p port** command line option while running **VirtualServiceEnvironment.exe**, where **port** is the port number which will be different for different VSE instances.

The **maxvirtualservices** in your license limits the number of virtual services you can run.

Note: RMI port strategy information can also be found at [LISA Port Usage](#).

Using VSE Manager to Setup the VSE

VSE Manager lets you make changes to your virtual service environments. Use the following at a command prompt:

```
[LISA_HOME]\bin\VSEManager.exe --help
./lisa.sh com.itko.lisa.coordinator.VSEManager command [ ... command ]
```

Where "command" may be one or more of the following:

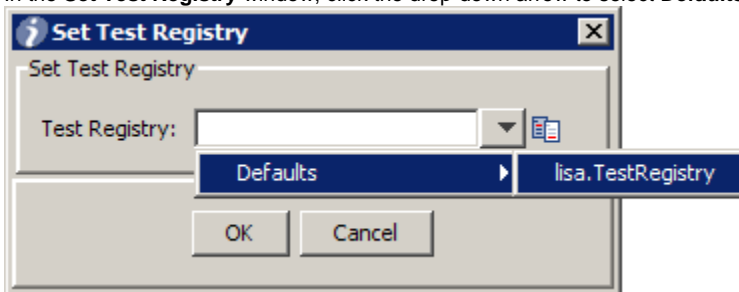
Command	Description
--registry <name>	Used to set the name of the test registry to work through.
--vse <name>	Used to set the name of the virtual service environment to work with.
--status [<vs-name>all]	Used to print out the status of one or all virtual services in the current environment.
--list-images	Used to list out the currently known service images in the current environment's database. The virtual services that refer to each service image is also displayed.

--deploy <vs-name> --model <to-file> [config <config-spec>] [+auto-restart]	Used to deploy a new virtual service to the current environment. The "config-spec" may either be the name of a config file or of a configuration internal to the test case model.
--deploy-image --file <xml-file>	Used to deploy a virtual service image to the current environment. The "xml-file" must contain an XML document of the form created by exporting a service image with LISA. If the image already exists in the current environment, an error is displayed.
--redploy <vs-name> (deploy arguments) [+newname <new-name>]	Used to redeploy an existing virtual service to the current environment. The --model and --config arguments are the same as for --deploy. The --newname argument may be used to change the name of the service
--redploy-image --file <xml-file>	Used to deploy a virtual service image to the current environment. The "xml-file" must contain an XML document of the form created by exporting a service image with LISA. If the image already exists in the current environment, it is replaced.
--update <vs-name> [capacity <capacity>] [+thinkscale <scale>]	Used to update either the capacity or think scale (or both) of the name virtual service.
--remove <vs-name>	Used to remove the named virtual service from the current environment.
--remove-image <service-image-name>	Used to remove the named service image from the current environment's database.
--start <vs-name>	Used to start the named virtual service in the current environment.
--stop <vs-name>	Used to stop the named virtual service in the current environment.
--stopvse	Used to stop the current virtual service environment.
--help	Displays this text.

Using VSE Dashboard to Connect to the Test Registry and VSE

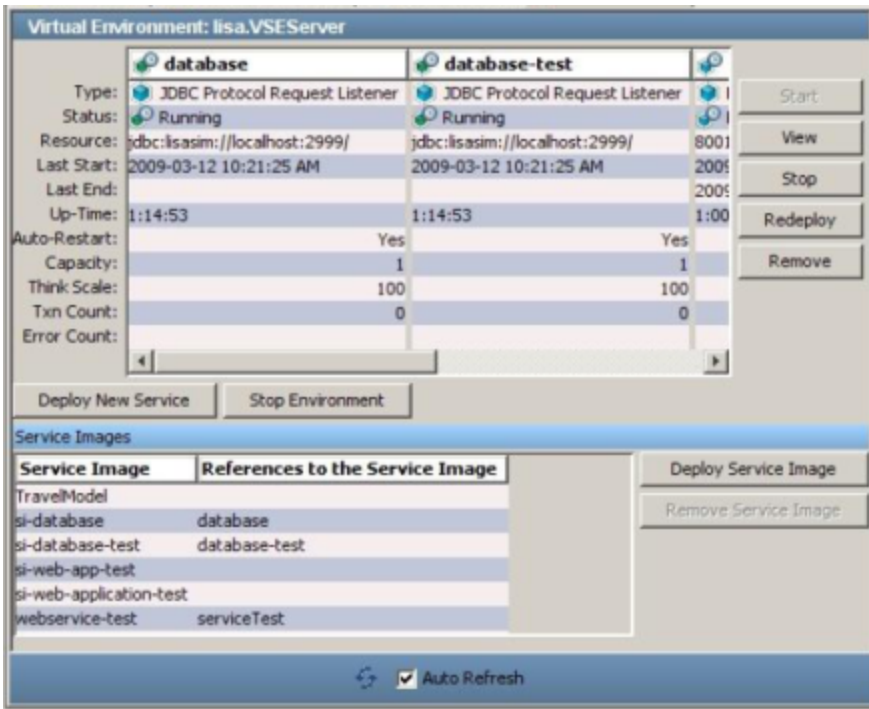
Open LISA Workstation to verify VSE has started:

1. From the desktop, double-click the **LISA Workstation** shortcut to start LISA workstation.
2. From the toolbar, click **VSE**, or from the menu bar, select **System > VSE Dashboard**.
3. In the **Set Test Registry** window, click the drop-down arrow to select **Defaults > lisa.TestRegistry**.



4. Click OK.

The VSE dashboard requires a **Test Registry** and at least one **Virtual Service Environment (VSE)**. If the VSE is not running, a message displays that there are no virtual environments currently registered. Otherwise, the VSE dashboard shows all virtual environments one after the other.



VSE Dashboard allows you to deploy, start, view, stop, redeploy, and remove virtual services. It also displays all service images stored in the VSE database including:

1. Type.
2. Status.
3. Resource.
4. Last Start.
5. Last End.
6. Up-Time.
7. Auto-Restart.
8. Capacity.
9. Think Scale.
10. Transaction (Txn) Count.
11. Error Count.

8.2 Deploying and Running a VSM

8.2 Deploying and Running a VSM

1. In the VSE dashboard, click **Deploy New Service**. The Deploy New Virtual Service window opens.

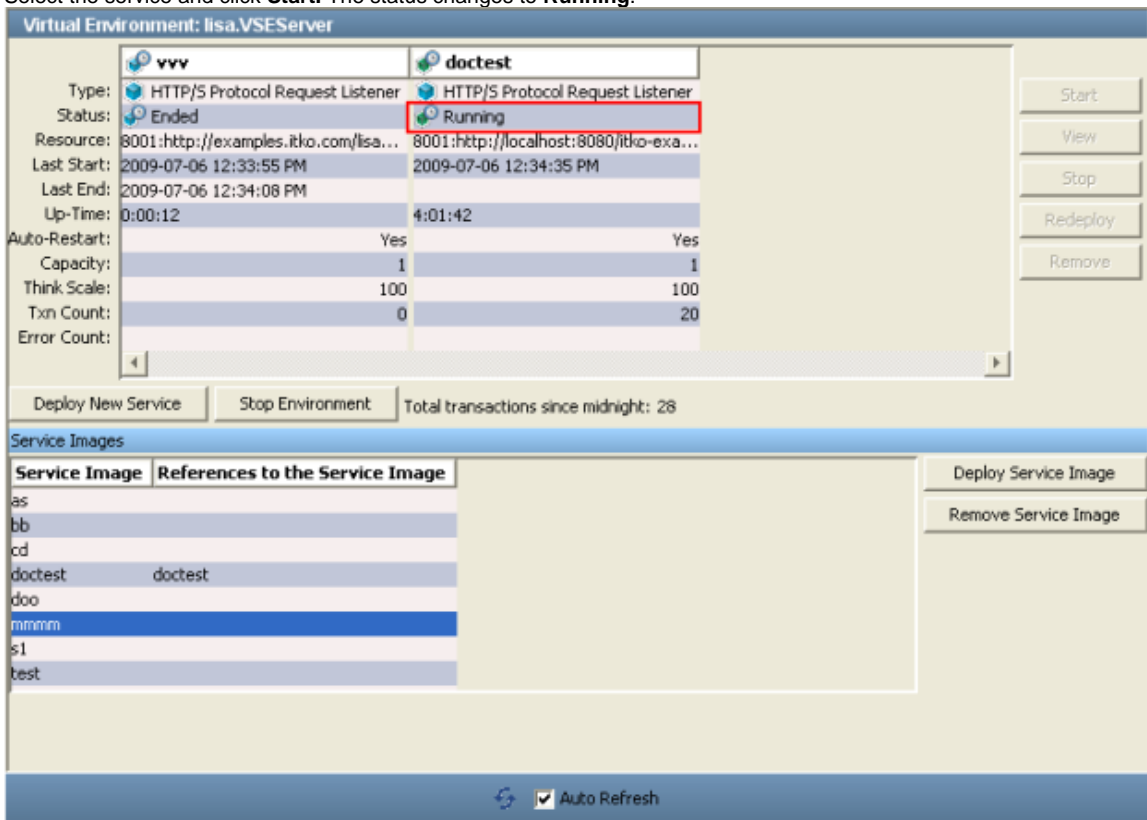
The screenshot shows the 'Deploy New Virtual Service' dialog box. It has a title bar with a question mark icon and a close button. The fields include: 'Virtual service name:' (text input), 'VS model:' (dropdown menu with a 'Browse' button), 'Configuration [opt]:' (dropdown menu with a 'Browse' button), 'Concurrent request capacity:' (spin box set to 1), and 'Think time scale factor:' (spin box set to 100 with a '%' sign). There is a checkbox labeled 'If service ends, automatically restart it' which is checked. At the bottom are 'OK' and 'Cancel' buttons.

2. Modify the fields as needed:

Field	Description
Virtual service name	Enter a unique name for the virtual service. You may leave this blank at first; since as you choose the VS model, the Virtual service name referenced by it is chosen automatically.

VS model	From the drop-down list, select the .vsm . You can also browse the file system to the path of the .vsm file you want to deploy. Note: If you do not see the VS model in the list, save the .vsm file and try again.
Configuration [opt]	This field is optional.
Concurrent request capacity	Enter a number to indicate the load capacity. The default is 1 .
Think time scale factor	Enter the think time percentage with respect to the recorded think time. To double the think time, use 200. To halve the think time, use 50. The default is 100 .
If service ends, automatically restart it	Keeps the service running even after an emulation session has reached its end point. Selected by default.

3. Click **OK**.
4. The virtual service status should be displayed in lisa.VSEServer environment as loaded.
5. Select the service and click **Start**. The status changes to **Running**.

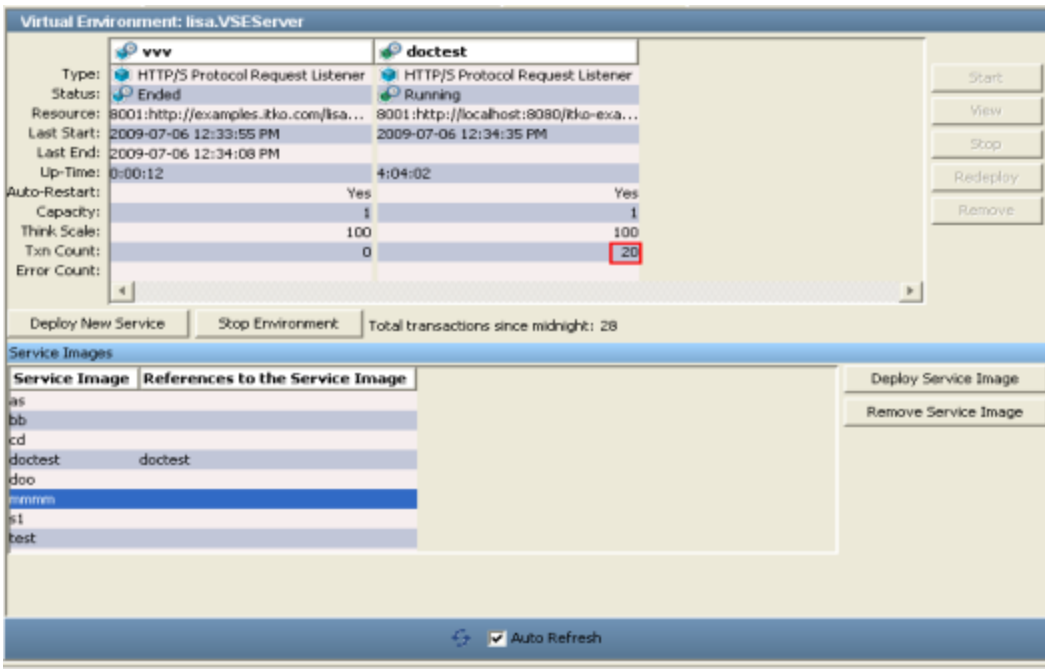


8.3 Running Live Request Against LISA VSE

8.3 Running Live Request Against LISA VSE

After you deploy the virtual service, run live requests against LISA VSE. If possible, take down the live service and configure the client to talk to LISA VSE by configuring the gateway / proxy settings.

1. In VSE Dashboard, verify that LISA VSE received the requests (status is **Running**) and view the transaction count (Txn count).



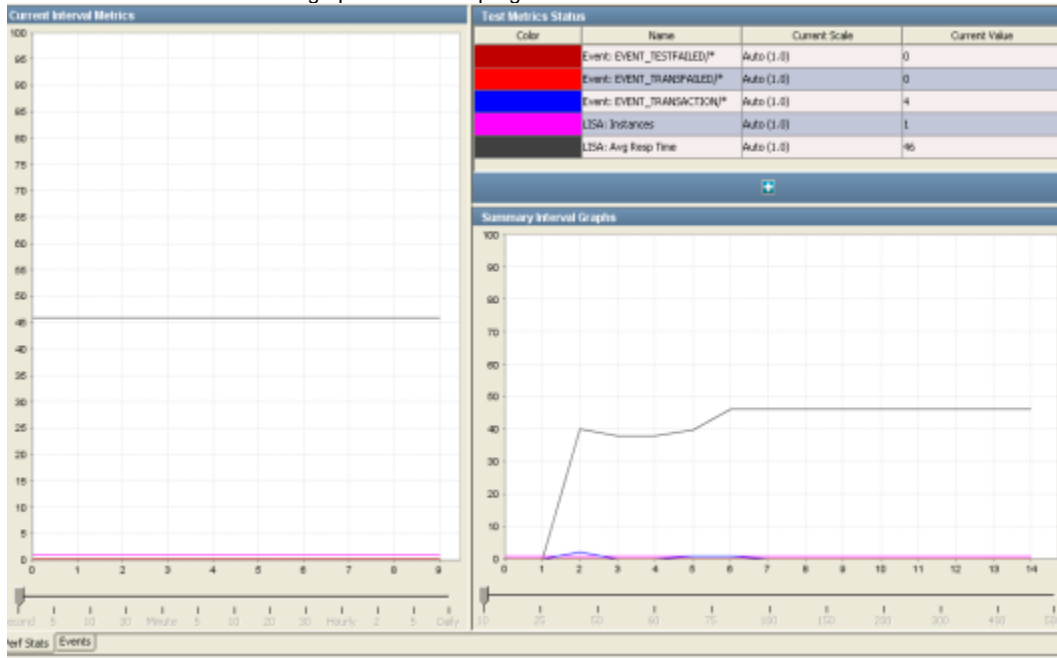
If transactions are not recorded...

If a deployed VS model is not seeing any transactions, then the client is not configured properly. Reconfigure the client to point to the virtual model rather than the real system.

If another service is using that port, either stop that service or change the port setting so there is no longer a conflict.

2. While the test case executes, click **View** to monitor performance statistic and events.

3. The PerfStats tab shows line graphs of the test progress.



4. Click **OK** when the test ends.

5. If you edit the service image or the VSM, save the changes and redeploy the modified service image in the VSE Dashboard by clicking Redeploy.

8.4 Execution Mode

8.4 Execution Mode

The concept of an execution mode is how the VSE runtime framework makes all this happen. Lisa Virtualizer supports the following execution modes.

1. Most Efficient –

The Most Efficient execution mode uses VSE service image to resolve request to response. This mode is the fastest and will not execute the routing or tracking steps.

2. Transaction Tracking –

This mode works same as 'Most Efficient' execution mode but same time Lisa virtualize records response to request in database for transaction tracking. This mode fires more events than 'Most Efficient' execution mode and remembers transaction flow through session.

3. Live System -

In this mode Lisa virtualizer fires request against the live system to retrieve response instead of looking into VSE database. This mode will use the live invocation step of the model to determine a response to the current request. The target system of the live invocation will control the performance.


4. Image Validation -

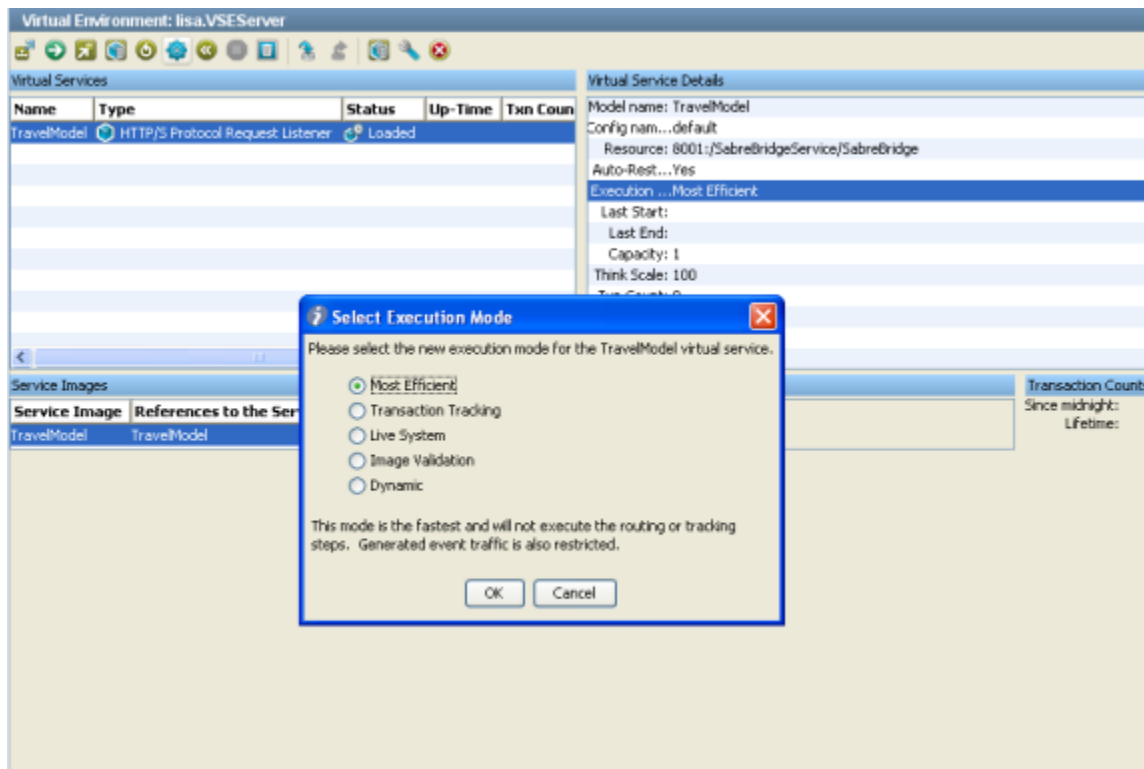
This mode will use both the VSE database and the live system to derive a response to the current request. The responses are compared and appropriate history remembered. It is least performant (TODO) of all the mode.

5. Dynamic –

This mode allows the model to determine on a per request basis which if the other modes to use. Performance is therefore non-deterministic.

Selecting Execution Mode

In order to set Execution mode for VSM, first you need to deploy VSM using VSE dashboard. After successful deployment, click on the  'Set Execution mode for selected virtual service' icon.



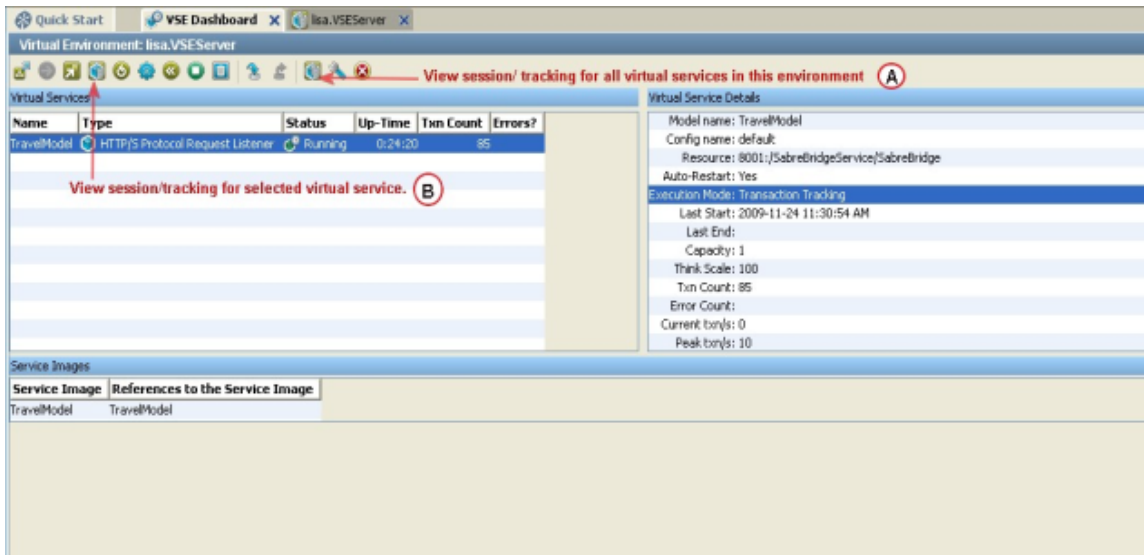
8.5 Session Viewing and Model Healing

8.5 Session Viewing and Model Healing

This feature allows a user of VSE to actually see what is going on with current (or recent) sessions on a VSE server and to track down why the particular response for a given request was served up. It also allows a live comparison between the responses provided by VSE and a corresponding live system and, where differences exist, patch or heal the VSE service image to keep in sync with the live system. The latter activity is referred as model healing.

Viewing/Healing Panel

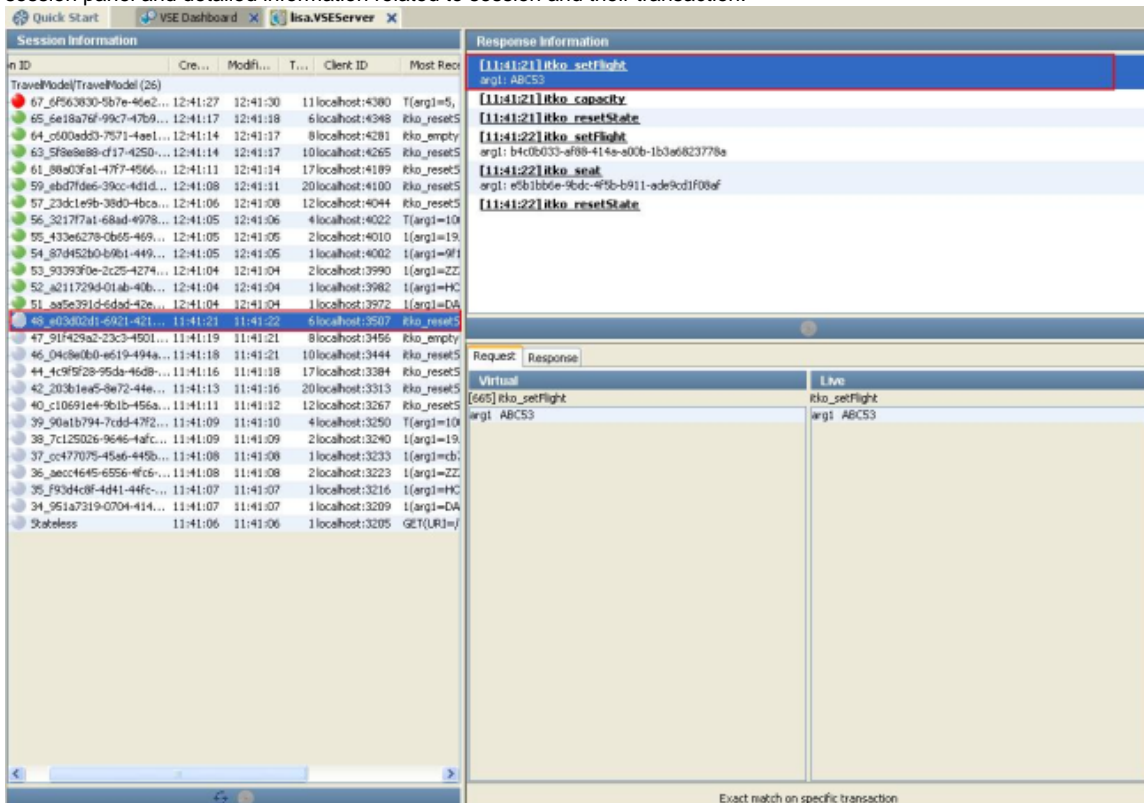
Session viewing and model healing are accomplished via a new panel that is accessible from either the VSE dashboard or while editing a virtual service model.



There are two ways from VSE dashboard to go the session panel.

1. Click on the icon to view the session tracking for the all virtual services deployed in VSE.
2. Click on the icon to view the session tracking for selected virtual service.

Once you click the icon 'B' shown in above figure, the session panel opens and shows the recorded session and transaction. If deployed virtual serve doesn't have any recorded transaction then session panel open without any session information. The following screenshot shows the session panel and detailed information related to session and their transaction.



This session panel divided into two panes. These are

Session Information –

This pane lists all the session for the selected virtual service in tabular format. Each column listed below.

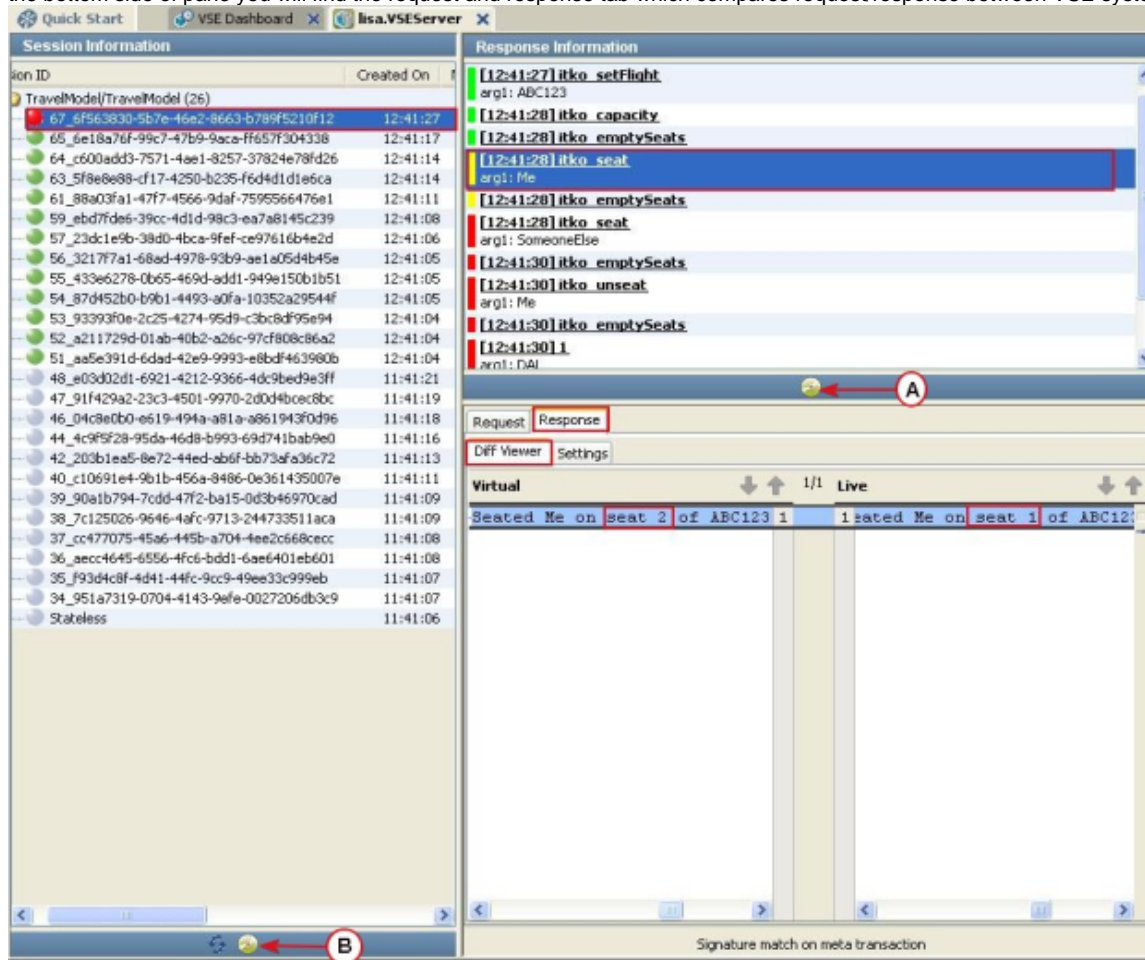
1. **Session ID** – The Unique Id for each session.
2. **Created On** – Timestamp of first transaction in that particular session.
3. **Modified On** – Timestamp of most recent transaction.
4. **Txn Count** – Number of transaction in that particular session.

5. **Client ID** – (TODO)
6. **Most Recent Request** – (TODO)

Please observe carefully the listed sessions in the above screenshot. Some of the sessions are represented using gray colored ball, which means they are tracked and recorded using Transition Tracking execution mode. Some of the Sessions represented using green and red (Session having problem) colored ball, which means they are recorded using Image validation execution mode.

Response Information –

The response information pane shows the list of transaction for the selected session and tooltip on each transaction tells us how that request matched. It may be exact match on specific transaction or signature match on Meta transaction. If you click on any specific transaction, then on the bottom side of the pane you will find the request and response tab which compares request/response between VSE system and Live System.



Look into the Response Information pane of below attached screenshot. Here transactions are represented using Green, Yellow and Red colors.

1. Green colored transaction - Signature match on meta transaction.
2. Yellow colored transaction - Signature match on meta transaction also image navigation is ok but response body differs between VSE and Live system.
3. Red colored transaction – Conversational transaction diverged between the live system and VSE image.

The buttons A and B in the above screenshot are used to update the service image with live session/stateless transaction. This process is called model healing. In the above figure session marked with red colored ball indicates disparity between the VSE image and live system. Therefore there is a need of model healing to remove the disparity for the VSM to work correctly. Once the user clicks on the button the session marked with red colored ball represented with gray colored ball as this is now tracked.

1. Using Button A, you can update the service image with transaction of selected session.
2. Using Button B, you can select multiple sessions displayed in the Service Information pane and update the service image.

VSE Command Line Options

VSE Command Line Options

VSE Manager

This command line tool is used for managing virtual service environments.

Syntax:

```
./lisa.sh com.itko.lisa.coordinator.VSEManager command [ ... command ]
```

where "command" may be one of the following:

--registry <name>

Used to set the name of the test registry to work through.

--vse <name>

Used to set the name of the virtual service environment to work with.

--status [<vs-name> | all]

Used to print out the status of one or all virtual services in the current environment.

--list-images

Used to list out the currently known service images in the current environment's database. The virtual services that refer to each service image is also displayed.

~~**--deploy <vs-name> --model <to-file> [config <config-spec>] [--auto-restart]**~~

Used to deploy a new virtual service to the current environment. The "config-spec" may either be the name of a config file or of a configuration internal to the test case model.

--deploy-image --file <xml-file>

Used to deploy a virtual service image to the current environment. The "xml-file" must contain an XML document of the form created by exporting a service image with LISA. If the image already exists in the current environment, an error is displayed.

--redploy <vs-name>

Unknown macro: {deploy-arguments}

[--newname <new-name>]

Used to redeploy an existing virtual service to the current environment.

The --model and --config arguments are the same as for --deploy. The --newname argument may be used to change the name of the service

--redploy-image --file <xml-file>

Used to deploy a virtual service image to the current environment. The "xml-file" must contain an XML document of the form created by exporting a service image with LISA. If the image already exists in the current environment, it is replaced.

~~**--update <vs-name> [capacity <capacity>] [--thinkscale <scale>]**~~

Used to update either the capacity or think scale (or both) of the named

virtual service.

--remove <vs-name>

Used to remove the named virtual service from the current environment.

--remove-image <service-image-name>

Used to remove the named service image from the current environment's,
database.

--start <vs-name>

Used to start the named virtual service in the current environment.

--stop <vs-name>

Used to stop the named virtual service in the current environment.

--stopvse

Used to stop the current virtual service environment.

--help

Displays this text.

ServiceImageManager

Usage:

```
java com.itko.lisa.vse.stateful.ServiceImageManager [options]
```

where "options" may be:

-h, --help

Displays this help text.

-i <file-name>, --import <file-name>

Imports the named file as an XML document that describes a complete
service image.

-m <image-name> <file-name> [<tol1> [<tol2>]],

--merge <image-name> <file-name> [<tol1> [<tol2>]]

Imports the named file as an XML document that describes a collection
of linear conversations, merging them into the named service image.

"tol1" represents the default tolerance to use for non-leaf

transactions and "tol2" represents the default tolerance to use for

leaf transactions. Both must be one of CLOSE, WIDE or LOOSE. If not
specified, "tol1" will default to WIDE and "tol2" to LOOSE.

-e <image-name> <file-name> [raw], --export <image-name> <file-name> [raw]

Export the named service image as an XML document written to the
specified file. If 'raw' is specified, the xml will be in raw traffic format

-d <image-name>, --delete <image-name>

Delete the named service image from the database.

-l, --list

Displays a list of the known service images.

LISA Web 2.0 Guide

LISA Web 2.0 Guide

This LISA Web 2.0 guide user documentation is divided into six parts.

PART 1 - LISA Web 2.0 - User Guide
PART 2 - LISA Web 2.0 - How Tos
PART 3 - LISA Web 2.0 - Reference
PART 4 - LISA Web 2.0 - Videos
PART 5 - LISA Web 2.0 - Repository
PART 6 - LISA Web 2.0 - FAQ

PART 1 - LISA Web 2.0 - User Guide

PART 1 - LISA Web 2.0 - User Guide

Recording a Web Site via DOM Events

LISA Web 2.0 testing, works by letting LISA **emulate a web browser**.

LISA Web 2.0, allows you to record events at the **DOM** (Document Object Model) **level** (such as mouse clicks, mouse movements, keys being typed, etc.) and play those events back as a browser during test execution.

DOM-level testing gives you fine-grained control over what a test can and cannot do, what type of object it can access and what kind of result it can return. In particular, all client-side logic is accessible to it. A Web 2.0 test can easily interact with frames, JavaScript, CSS, Ajax, Plugins, Applets and so forth. Many modern web sites make heavy use of these technologies, Ajax in particular, and those are usually called Web 2.0 sites, hence the term **Web 2.0 tests**. However, any web site can be tested in this fashion.

For more information on Web 2.0, you can also refer to its [How To doc](#) and [Reference Guide](#) in the [LISA Web 2.0 Guide](#).

The following topics are available in this section.

1. [Introduction to Web 2.0](#)
2. [Recording Mode](#)
3. [Playback Mode](#)
4. [Edit Mode](#)
5. [Debugging](#)
6. [Setting up ADF Extensions](#)
7. [Running Browser Standalone](#)
8. [Troubleshooting](#)
9. [Known Limitations](#)

1. Introduction to Web 2.0

1. Introduction to Web 2.0

One of the major strengths of LISA is its outstanding ability to create and run tests that make use of a mix of different technologies (web, j2ee, web services, swing, etc...) as is so often necessary in the enterprise software world.

Web 2.0 tests are no exceptions in this regard and can be mixed with any other type of step. Nothing special is required to achieve this.

Until version 3.5, LISA has fully supported **HTTP-level web testing** and it will continue to do so in future releases.

HTTP-level testing works by having LISA install a **proxy** between itself and the web server. It then captures the HTTP and HTTPS traffic flowing between the client and the web server during a recording session. It submits GET or POST requests during a playback session. For more details you can consult the LISA web testing chapter in the User Guide.

Later, LISA supported the testing and recording of the Web 2.0 browser. By contrast, the Web 2.0 testing works by letting LISA emulate a web browser. It allows you to **record events at the DOM level** (such as mouse clicks, mouse movements, keys being typed, etc...) and play those events back as a browser during test execution.

There are advantages to each approach:

The HTTP-level testing might be more resistant to client-side changes during test execution since it is only aware of URLs. In addition, it is very lightweight so it is well suited to massive load-testing. DOM-level testing however is more resistant to server-side changes, it gives you much finer-grained control over what a test can and cannot do, what type of object it can access and what kind of result it can return. In particular, all client-side logic is accessible to it. A Web 2.0 test can easily interact with Frames, Javascript, CSS, Ajax, Plugins, Applets and so forth. Many modern web sites make heavy use of these technologies, Ajax in particular.

In addition to web testing, the LISA Web 2.0 can record, replay, and validate other so-called RIA (Rich Internet Applications) such as Java Applets (Swing and AWT), ActiveX controls and in particular Flash and Flex applications.

The following topics are available in this section.

- [1.1 System Requirements](#)
- [1.2 Technologies and Platforms](#)
- [1.3 Getting Started with LISA Browser](#)

1.1 System Requirements

1.1 System Requirements

In addition to the LISA installation on a Windows NT or better machine, the following is also required.

- An install of the .NET 2.0 SP1 runtime (or newer)
- A public jre (1.4 or greater) if you intend to test java applications or use HTTP recording

1.2 Technologies and Platforms

1.2 Technologies and Platforms

Web 2.0 tests might be better described as GUI tests because they support a lot more than basic web tests.

In the pure web realm, any server side-technology or platform is supported (because it's irrelevant from the client's perspective), whereas on the client side Web 2.0 can run Internet Explorer, Mozilla Firefox, or Safari (initial support) making it a truly cross-browser solution (those browsers constitute about 99% of the browser market at the time of this writing).

In addition to web testing, Web 2.0 can record, replay and validate other so-called RIA (Rich Internet Applications) such as Java Applets (Swing and AWT), ActiveX controls and in particular Flash and Flex applications.


Finally Web 2.0 supports non-web hosted technologies. It can record, replay and validate Java desktop applications (Swing, AWT), .NET WinForms or native Win32 applications.

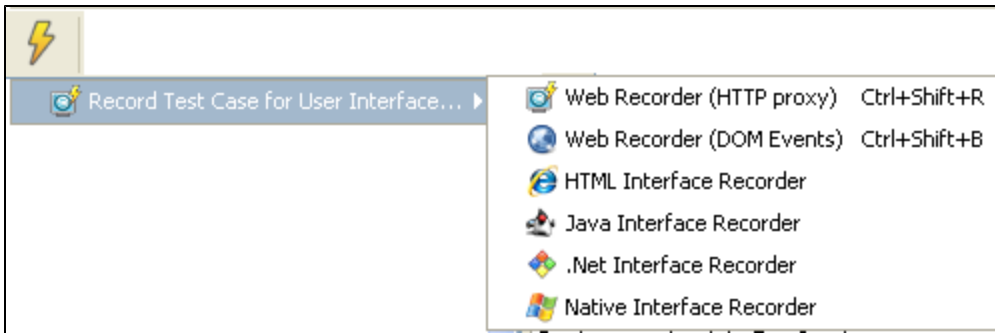
Important Note: The support for all these technologies is deep and does not rely on so-called analog record and replay technology that many tools only have. Analog testing is technology-agnostic because it relies on screen coordinates, which makes it very brittle and not very powerful.

1.3 Getting Started with LISA Browser

1.3 Getting Started

To open the Web Browser,

- Click the  icon on the test case menu. This will open a menu further -



Click **Record Test Case for User Interface > Web Recorder (DOM Events)**...

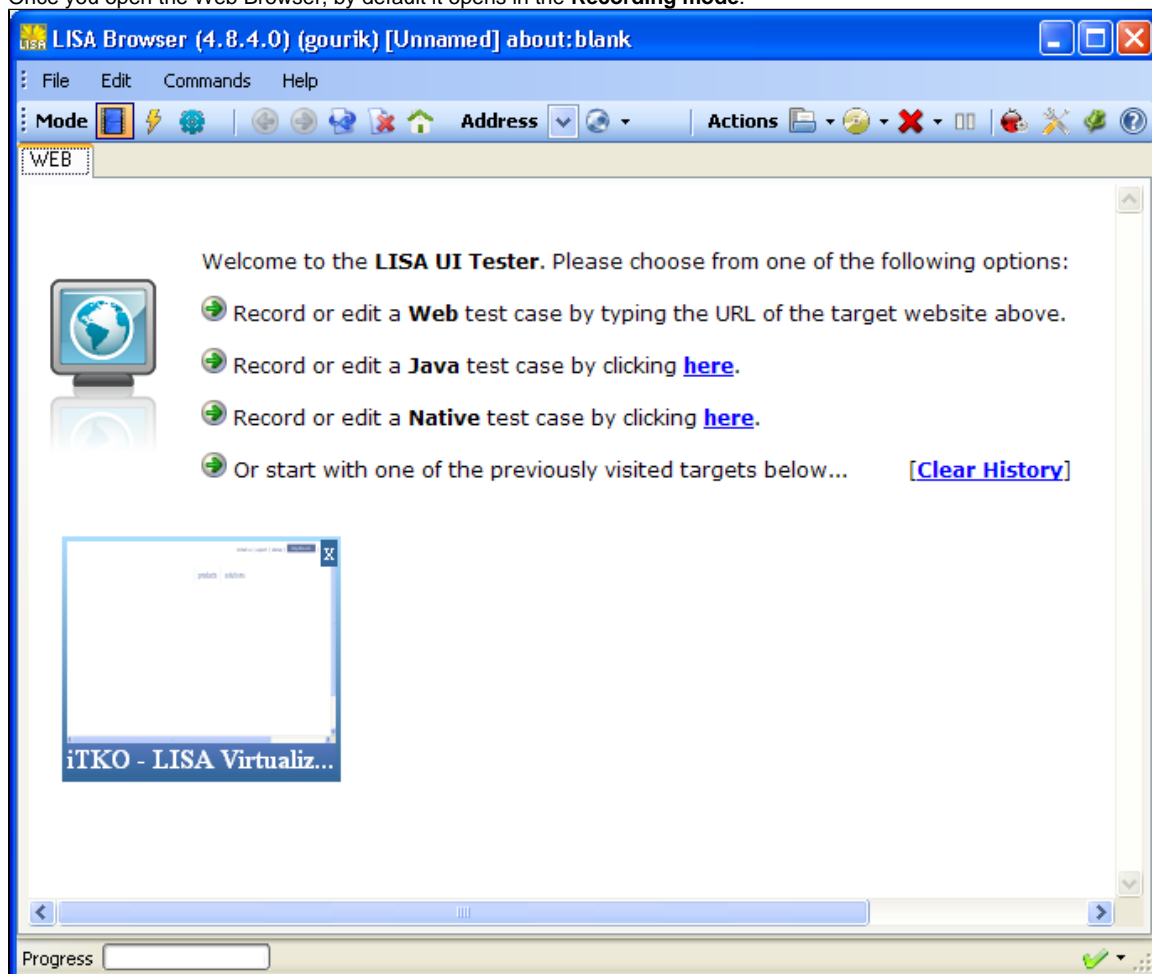
Or click **Actions > Record Test Case for User Interface > Web Recorder (DOM Events)** from the main menu.

This allows you to launch the browser that LISA uses to record and playback Web 2.0 tests.

The first time the browser is launched; there is a wait dialog that indicates it is synchronizing its initial state with LISA. Subsequent invocations will not do this since synchronization will happen on the fly.

Typically, the first interaction you will have with a web test is of recording a session.

Once you open the Web Browser, by default it opens in the **Recording mode**.



There are three main sections in the Web 2.0 Recorder:

- Recording Mode
- Editing Mode
- Playback Mode

Within the browser, there are menus which allow you to Record or edit a **Web or Java or a Native** test case.

You can also see the recently opened web pages if there are any recordings done previously.

More details regarding the same can be found in the subsequent sections.

- 1.3.1 Browser Menu
- 1.3.2 Browser Toolbar
- 1.3.3 Browser Settings
- 1.3.4 Browser & Extension Updates
- 1.3.5 Browser Architecture

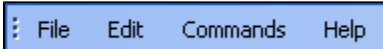
1.3.1 Browser Menu

1.3.1 Browser Menu

The LISA Browser opens in the Recording mode by default. The LISA Browser has a typical main menu and a toolbar, which has functions and icons depicting various activities or actions.

Browser Menu

The Browser menu is as shown below and explained.



File Menu

- File > Load** – Loads the current web address
- File > Save** – Saves the current recording
- File > Save As** – Saves the current recording under a different name
- File > Close** – Closes the current recording
- File > Exit** – Exits the LISA Browser

Edit Menu

- Edit > Pause Recording** – Pauses the recording
- Edit > Clear Steps** – Clears all steps
- Edit > Browser Settings** – Opens the Settings dialog box, where you can set the browser settings.
- Edit > Internet Options** – Opens the Internet Properties dialog box, where you can set the Internet options.

Commands Menu

- Commands > Back** – Loads the previous page
- Commands > Forward** – Loads the next page
- Commands > Reload** – Reloads the page
- Commands > Stop** – Stops the recording
- Commands > Toggle Debug Window** – Toggles the debug window
- Commands > Capture Session** – Captures the current recording session

Help Menu

- Help > Documentation** – Opens the documentation page for LISA web 2.0
- Help > Browser Updates** – Opens the LISA Component update dialog box.
- Help > Extension Updates** – Opens the LISA Extension update dialog box.

1.3.2 Browser Toolbar

1.3.2 Browser Toolbar

The LISA Browser opens in the Recording mode by default.

The LISA Browser has a **toolbar** as shown below:



On the **left** of the toolbar, there is the Mode button, where in you can select the Mode of operation within the browser:

- Recording Mode - Where you can record the operation.
- Edit Mode - Where you can edit the transactions.
- Playback Mode - Where you can playback the recorded operation.

By default the Recording mode is selected.

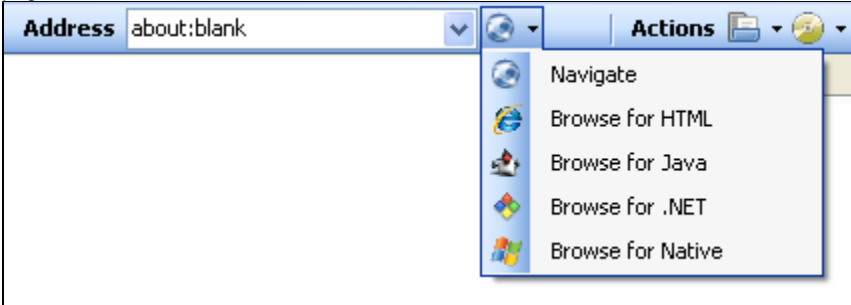
You can Record the web page and playback the recording by clicking on the Playback mode.

You can click on the Edit mode to view add/delete the Logical, Physical events and view the Object details which are described later.

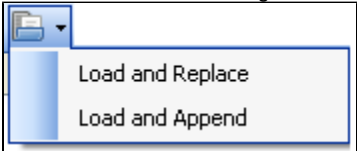
There are the usual **Web** page buttons to take you – Back, Forward, Reload, Abort and Home page.



You can enter the Web page address in the Address bar provided or select the "Go" button to navigate and open HTML/ Java/.Net or Native page.

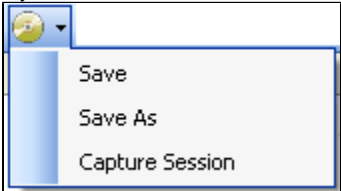


You can Load and Replace or Load and Append from the "Actions" button. The Load button is not normally used when using the browser within LISA. It is useful when using the browser is standalone. You can either Load and Replace or Load and Append a previous recording here.

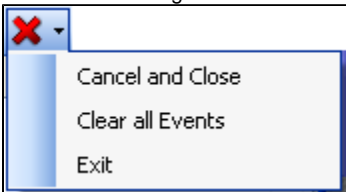


You can Save the recordings from the "Save" button. The Save button is what commits the recording to LISA's Test Case, and closes the browser.

If you are in standalone mode it saves the recording to a file. You can do a Save, Save as or Capture a Session here.








You can Close and Cancel the recording from the "Cancel" button. The Cancel button simply discards the current recording and closes the browser without modifying the Test Case. The window close button has the same effect. You can also Clear all previous Events here and Exit from the Recording window here.





The following icons are described below:



Icon	Description
	The Pause button allows you to navigate without recording. Recording resumes when it is pressed again. It is useful to skip some undesired events.
	The Debug button allows you to open or close the debug window.

	The Settings button opens the Settings dialog that allows you to configure global behaviors of the both the recorder and the debugger.
	The Pin window button makes the browser (in recorder or debugger mode) stay as a topmost window, which means that it will stay on top of any other window on the screen, even if it does not have the focus. This is useful when you want to observe the tests running while other windows try to grab the focus, especially when you test external applications (like Swing or .NET Winforms).
	The Help button displays the download area from where you can download this document.

The Debug  and the Settings  buttons are described in detail in the next section.


1.3.3 Browser Settings

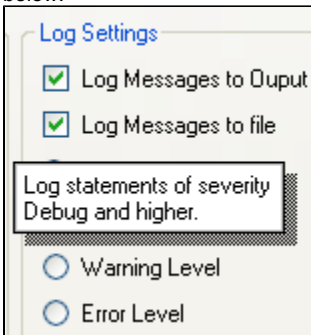
1.3.3 Browser Settings

The LISA Browser Settings allows you to control how the **Recording and Playback** will work.

From the main menu, click **Recording > Settings** or Click the **Settings icon** from the top right toolbar.

The Settings are divided in 4 sections: General, Recording, Playback and Environment.

While the settings window is open, you can quickly lookup the meaning of a setting by clicking the  icon at the top right corner of the window and then the desired setting, or position the mouse over the desired setting and type the F1 key. You will get a tooltip for that field as shown below:



Each playback setting can be overridden on a per test case basis by defining a property (usually but not necessarily in the configuration of the test case or suite), whose name is indicated in the reference and in the help pop-up described above.

For example, there is a setting called "Synchronize Ajax Calls" (which is used to force the browser to treat all ajax calls as synchronous calls). If you bring up the help pop-up you will see this setting can be overridden with the `SYNC_AJAX` property, which means if a test case defines the `SYNC_AJAX` property with an appropriate value (true or false in this case), the behavior for this test case as defined by this property will override the one defined in the settings window.

General Tab

The General Tab is used to set basic options.

Settings

General | Recording | Playback | Environment

Information

BUILD: 4.8.4.0 (LATEST AVAILABLE: 4.8.4.0)

MACHINE: GAURILAPTOP

OS: Microsoft Windows NT 5.1.2600 Service Pack 3

IE: 8.0.6001.18702 Firefox: 3.6

.NET: 2.0.50727.3615 JRE: 0.0

EXE: "C:\Lisa5.0GA\bin\browser\lisa_browser.exe" -m recorder -p 2641 -pid 2556

MEMORY: 86941 KB

General Behavior

☐ Disable browser popups

☐ Hide Error Dialogs

☐ Enable ActiveX / Flash / Flex (beta)

☐ Enable Applets (Needs public JRE)

☐ Capture HTTP traffic (Needs public JRE)

☐ Enable Pathfinder Integration (Needs public JRE)

☐ Enable SWT (preview)

Remote Applications Port: 0

Maximum Bandwidth: 0 KB/s

Output Directory: {{BROWSER_HOME}}\out

Scripts Directory: {{BROWSER_HOME}}\scripts

Maps Directory: {{BROWSER_HOME}}\maps

Log Settings

☒ Log Messages to Output Window

☒ Log Messages to file

☒ Debug Level

☐ Info Level

☐ Warning Level

☐ Error Level

Custom HTTP Headers (requires restart)

Name	Value

For help click on the ? icon in the top-right corner, then on a setting's label.

Save Cancel

In the **Information** box – you can see the machine information which runs LISA browser

In the **General Behavior** box – you can check/uncheck general behavior of the browser properties.

In the **Log Settings** box – you can check/uncheck log related information.

- **Disable browser popups**: acts like a popup blocker. Overrideable in a test with `DISABLE_POPUPS`.
- **Suppress Javascript Dialogs**: alert and confirm dialogs will auto-respond during playback `SUPPRESS_DIALOGS`.
- **Enable ActiveX / Flash / Flex**: Turns on support for ActiveX events. Only reason to turn it off might be faster startup time.
- **Enable Applets**: Turns on support for ActiveX events. Only reason to turn it off might be faster startup time or improved stability (the Java Plugin Interface has some known bugs in certain versions of it, notably 1.5.0_01 through 1.5.0_14, that could cause crashes).
- **Enable SWT**: Turns on support for testing non-Eclipse based SWT applications.
- **Capture HTTP traffic**: Turns on a proxy to capture all HTTP(S) traffic and stores all headers in the event requests. Only reason to turn it off might be faster startup time or already having a proxy.
- **Enable Pathfinder integration**: this causes the browser to receive and decrypt Pathfinder payloads for Pathfinder-enabled applications.
- **Remote Applications Port**: specifies the port to use the control remote applications (Swing, SWT or WinForms). The default, 0, picks the port dynamically.
- **Maximum Bandwidth**: throttles request and response speed to simulate a network with the specified throughput. 0 means no limit.

- **Log messages to Output window:** self-explanatory.
- **Log messages to Output file:** self-explanatory. The log files go in the same directory as the DOM browser.
- **Log Level:** the level of log statements required to be logged in the Output window or file if turned on above.

Recording Tab

The Recording Tab is used to set Recording options.

Settings

General **Recording** Playback Environment

Recording Strategy

All

Ignore ids and names whose value matches:

☐ Externalize recorded text

DOM

Ignore frame names whose value matches:

Ignore text whose value matches:

Use the following attributes (in this order):

1) <input type="text" value="id"/>	5) <input type="text" value="<none>"/>
2) <input type="text" value="name"/>	6) <input type="text" value="<none>"/>
3) <input type="text" value="<none>"/>	7) <input type="text" value="<none>"/>
4) <input type="text" value="<none>"/>	8) <input type="text" value="index"/>

Or use the following locator javascript function
☐

☒ Ignore invisible elements

Java

Record using:

☐ Component names ☒ Deep paths

☒ Component text ☐ Geometry

Recording Options

☒ Use context menus for filters and assertions

☐ Write Traffic to Disk

☒ Compress recording files

Capture Level

☐ Capture DOM Level 2

☐ Verbose HTML recording

☐ Ignore HTML responses

☐ Capture HTML changes

☐ Capture Applet snapshots

☐ Capture ActiveX snapshots

Capture Diff Size Bytes

Capture Max Time ms

Capture DOM Events

<input checked="" type="checkbox"/> navigate	<input checked="" type="checkbox"/> docload
<input checked="" type="checkbox"/> focus	<input checked="" type="checkbox"/> dblclick
<input checked="" type="checkbox"/> mousedown	<input checked="" type="checkbox"/> change
<input checked="" type="checkbox"/> mouseup	<input checked="" type="checkbox"/> contextmenu
<input checked="" type="checkbox"/> mouseover	<input checked="" type="checkbox"/> drag/drop
<input type="checkbox"/> mouseout	<input checked="" type="checkbox"/> mousemove
<input checked="" type="checkbox"/> click	<input checked="" type="checkbox"/> keypress
<input checked="" type="checkbox"/> open/close	<input type="checkbox"/> ajax callback

For help click on the ? icon in the top-right corner, then on a setting's label.

Save Cancel

- **Use context menus for filters and assertions:** Override the right-click menu in web pages to popup a custom menu that offers choices about filters, assertions or debugging. Turn it off if your site already uses custom context menus.
- **Capture applet snapshots:** stores in the test the applet screenshots used in applet test editing (browse mode). Turn it off if the tests get too large.
- **Compress recording files:** keep that turned on (used for debugging).
- **Verbose recording:** Records events for which we could not detect an event handler (possibly DOM Level 2). Turn it off for most sites, try to turn it on if you notice some necessary event does not get recorded (happens using ExtJS for instance).
- **Capture HTML changes:** Store the HTML at any click or change event to make it easier for later editing.
- **Capture Diff size:** changes over this size will store the whole response, changes under this size will store the diff.

- **Capture Max time:** the diff above won't be captured if it takes more than this amount of time.
- **Save Dialog Triggers:** for pages with a non text/plain mime type, this decides whether to pop up a "Save As" dialog instead of performing a navigation if the target url matches the regular expression (by default, it does this for Excel, Word, PDF, Text, PowerPoint, Executable and Zip files).
- **Recording strategy:** which HTML attributes to use and in which order when generating XPath expressions.
- **Exclude ids matching:** any element id matching this regular expression won't be used in the auto-generated XPaths.
- **Record component names/text:** use java component names or text (or not) in the XPath generated when recording Java based applications.
- **Capture DOM events:** Turn off any type of DOM event you're not interested in capturing.

Playback Tab

The Playback Tab is used to set Playback options.

The screenshot shows the 'Settings' dialog box with the 'Playback' tab selected. The dialog has four tabs: General, Recording, Playback, and Environment. The Playback tab contains several sections:

- Playback Options:**
 - ☐ Suppress Javascript Dialogs
 - ☐ Show Browser Dialogs
 - ☐ Synchronize Ajax calls.
 - ☐ Use Hardware Input
 - ☐ Enable mouse movement
 - ☐ Send Responses back to LISA
 - Max missing targets: 2
 - Min matching score: 1
 - Failed assertions screenshot directory: (empty text box with a browse button)
 - Playback Speed (left is slowest - right is fastest): (slider bar)
 - Wait multiplier: 0x
- Timeout Settings:**
 - DOM Load Timeout: 10000 ms
 - DOM Lookup Timeout: 1000 ms
 - Applet Load Timeout: 10000 ms
 - Applet Lookup Timeout: 5000 ms
 - ActiveX Load Timeout: 10000 ms
 - ActiveX Lookup Timeout: 5000 ms
- Browser Options:**
 - Use the following browsers by default:
 - ☒ Internet Explorer
 - ☐ Firefox
 - ☐ Safari
 - Installation Directories:
 - C:\Program Files\Internet Explor... (with a browse button)
 - C:\Program Files\Mozilla Firefox... (with a browse button)
 - (empty text box with a browse button)

At the bottom of the dialog, there is a help icon and text: "For help click on the ? icon in the top-right corner, then on a setting's label." and two buttons: "Save" and "Cancel".

Of particular interest on this tab is the **Synchronize Ajax calls** checkbox. This if checked, the step will not return until the Ajax call completes.

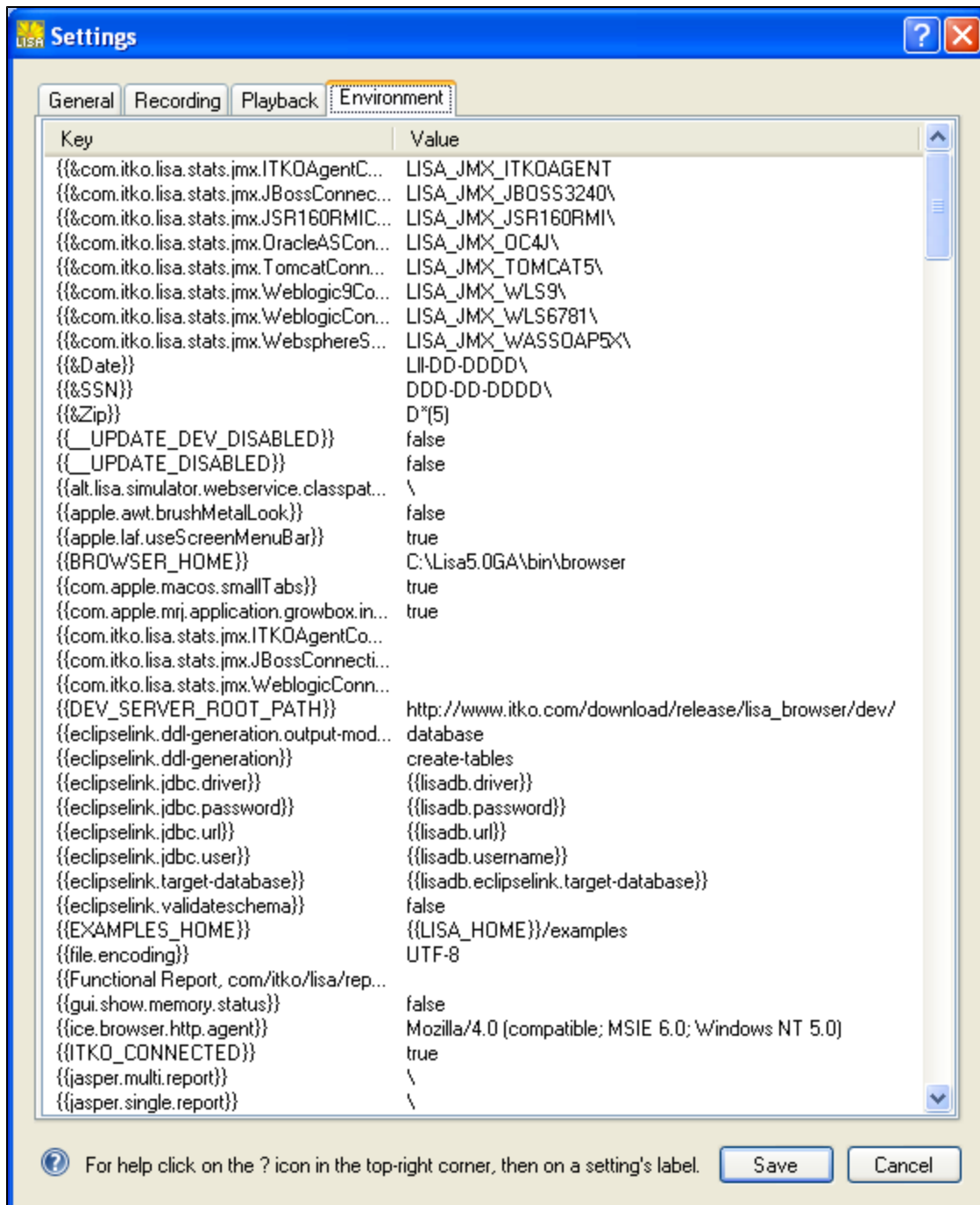
- **Fail step on missing target:** Generates a failure instead of the default warning when a target can not be found for a step (that includes

browser window, frame, element). Overrideable in a test with `FAIL_MISSING`.

- **Synchronize Ajax Calls:** forces all ajax calls to be executed synchronously. Overrideable in a test with `SYNC_AJAX`.
- **Use Hardware Input:** Replays tests by controlling keyboard and mouse. Overrideable in a test with `USE_HARDWARE`.
- **Send Responses back to LISA:** By default responses will not be sent back to LISA to improve performance and memory since it's usually not necessary.
- **Min Matching Score:** How many differences are allowed between a recorded XPath value and the best match found during playback. Overrideable in a test with `MIN_BACKTRACKING`.
- **Default Browser Mode:** What browsers to enable by default during playback (pipe-delimited combination of IE, FF and WK). Overrideable in a test with `DEFAULT_BROWSER`.
- **Playback Speed:** The default speed to use to replay test as a multiplier of the recorded speed. 0x is usually the best choice. Overrideable in a test with `PLAYBACK_SPEED` (integer between and 10).
- **XXX Load Timeout:** The maximum amount of time waited before a new page/applet/control load before proceeding. Overrideable in a test with `XXX_LOAD_TIMEOUT`.
- **XXX Lookup Timeout:** The maximum amount of time waited to find an element on a page/applet/control before proceeding. Overrideable in a test with `XXX_LOOKUP_TIMEOUT`.

Environment Tab

The Environment Tab is used to display the Environment settings.



This tab shows the list of global environment variables available to the browser, as read from **lisa.properties** and **local.properties**.

LISA Driver Settings:

In addition to the settings above that can be overridden from LISA in a test case or in the **local.properties**, the following are available:

- **lisa.browser.launch.timeout**: The amount of time allowed for a browser to launch (default is 10,000). Specify in **local.properties**.
- **lisa.browser.exec.timeout**: The amount of time allowed for a step to execute (default is 300,000). Specify in **local.properties**.
- **lisa.browser.max.instances**: The maximum number of browser instances per machine (default is 25). Specify in **local.properties**.
- **lisa.browser.client.user.single**: Whether to run staged browsers using the same user account as the currently logged-in users (default is true). Specify in **local.properties**.
- **lisa.browser.base.port**: The first port to use in the range of ports available to control web browsers (default is 0 for dynamic value). This normally does not need to be modified except in very secure environments that lock down some local ports. Specify in **local.properties**.
- **lisa.browser.client.user.<user name>=<encrypted password>**: when **lisa.browser.client.user.single** is set to false, browser instances will use the specified windows user accounts to run tests. If not enough user accounts are specified in this manner and the currently logged-in user has admin privileges, user accounts will be dynamically created to run the tests (and deleted at the end). To obtain an encrypted password, run the command line: `lisa_browser.exe -m encrypt -in <clear text>`.

- **lisa.browser.share.subprocess.state**: Whether to run a sub-process using the same browser instance as parent test (default is false). Specify in configuration of sub-process.
- **lisa.browser.swing.port**: The first port to use in the range of ports available to control Swing applications (default is 0 for dynamic value). Specify in local.properties.
- **lisa.browser.base.port**: The first port to use in the range of ports available to control web browsers (default is 0 for dynamic value). Specify in local.properties.

1.3.4 Browser & Extension Updates

1.3.4 Downloading Browser and Extension Updates

Because the web 2.0 environment evolves so fast, there is the ability to update it directly without waiting for a whole LISA release. Note that these intermediate releases are considered **unsupported**, but since most customer environments are not available to the outside world, we can not test a bug fix or feature enhancement against them so we work through this mechanism to allow a fast turnaround cycle. When the changes have been approved by a customer they are submitted to the official build environment for full testing.

One way to obtain the update is to run the update command from the command line:

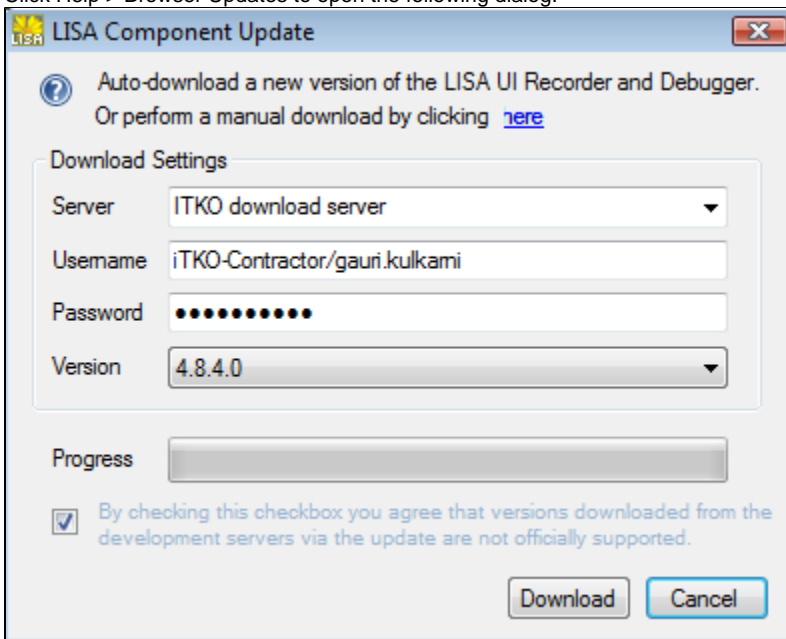
```
lisa_browser.exe -m update.
```

Or you can do it from the GUI by going to the Help menu and selecting the Update menu.

To download the updates of the web 2.0 browser,

Open the Web 2.0 browser within LISA.

- Click Help > Browser Updates to open the following dialog:

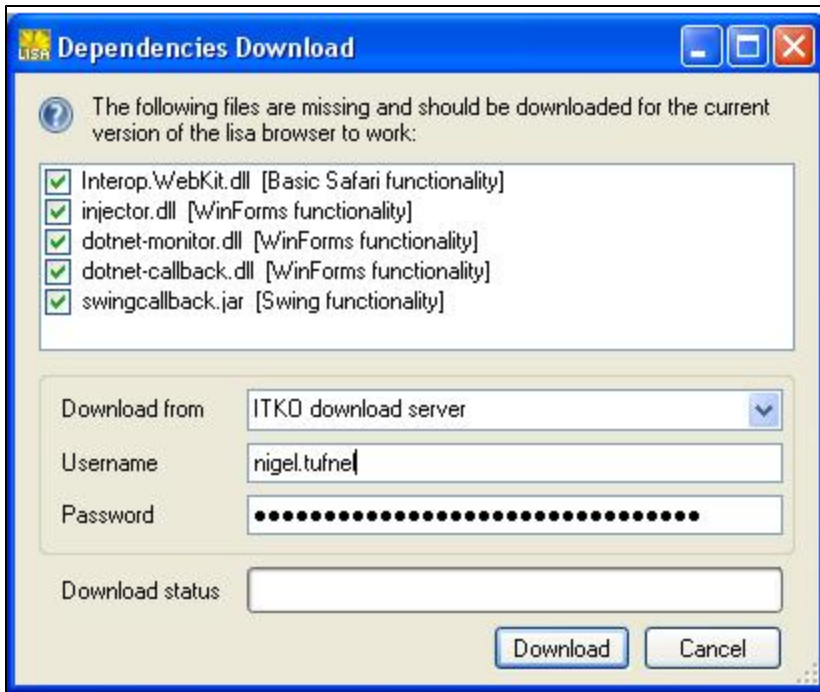


Click Download to start the download. You can monitor the download progress through the progress bar.

After the download is completed you will be notified that the update will be effective on restart.

Finally, when major new functionality is added, some dependencies will be added or modified that won't be available until you install from a full LISA installer.

This is why the browser checks for those on start up and prompts you for a download on start up if they are missing:



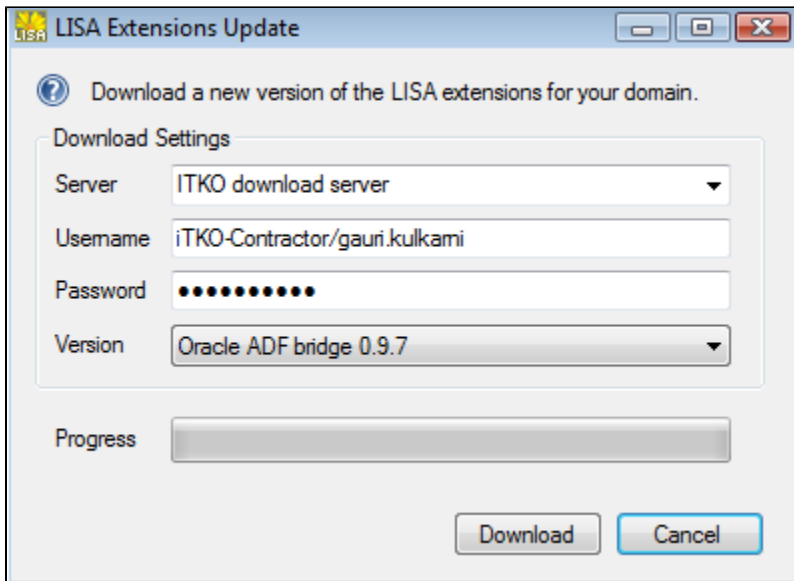
The browser won't start until those missing dependencies have been downloaded from one of the available locations: one of the release servers or one of the development servers. If none of these are accessible from the machine (no internet access for example), the listed files will have to be downloaded manually (see [LISA Web 2.0 Update Repository](#)).

Extension Updates

To download the updates of the Extensions,

Open the Web 2.0 browser within LISA.

- Click Help > Extensions Updates to open the following dialog:

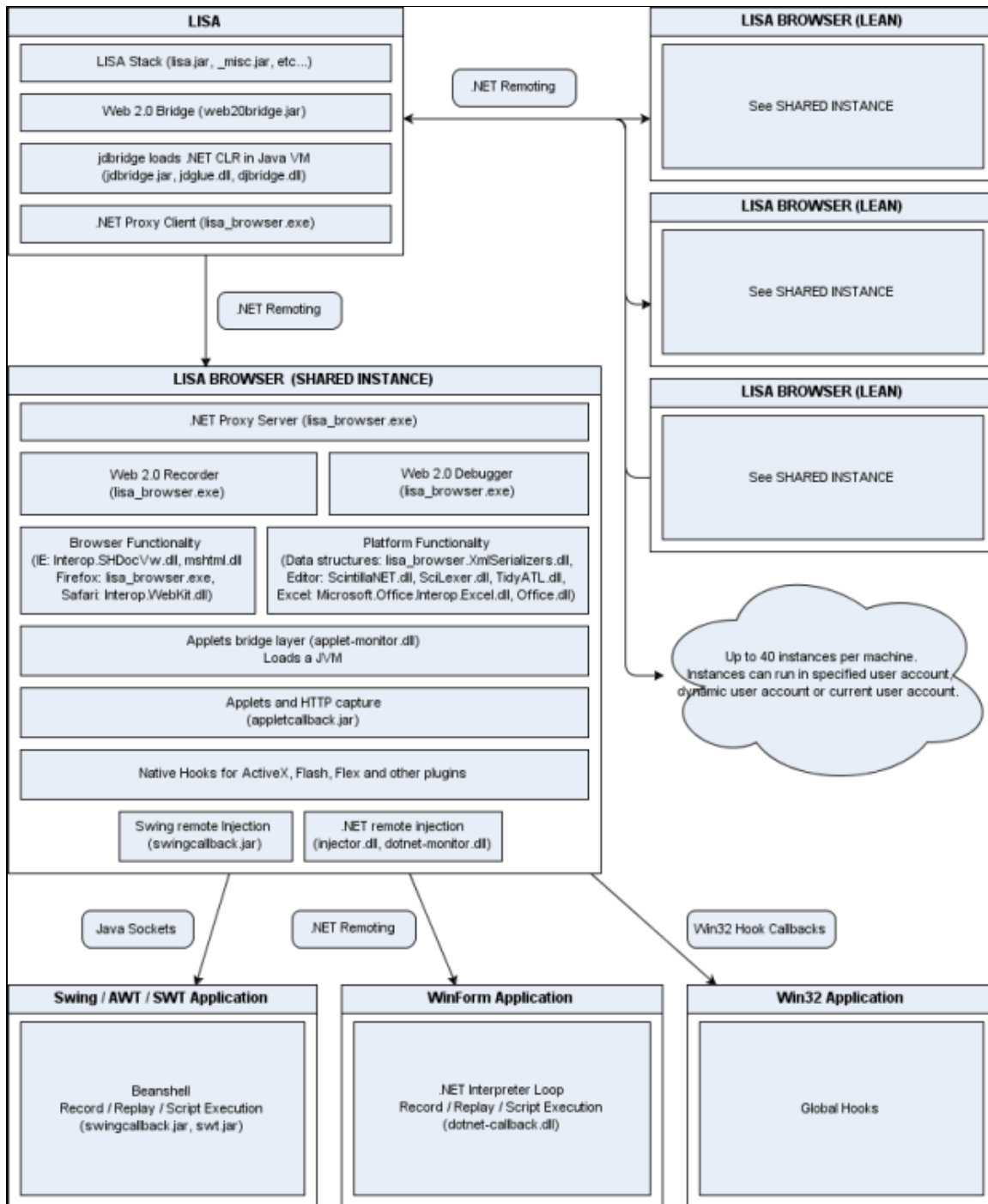


- Click **Download** to start the download.

1.3.5 Browser Architecture

1.3.5 Web 2.0 Browser Architecture

The Web 2.0 architecture in 4.6 and newer versions is as below:



Notes:

Q: Why is the browser hosted in .NET and native code rather than in TestManager (java)?

A: Hosting browsers in java correctly is difficult. There are a lot of products, open-source (jdic, jrex, etc...) and commercial that claim to do this but our experience is that they are quite prone to crashing or freezing. Using a native environment (mostly) eliminates these problems.

In addition, even if this approach was more stable, we could not run java applets. The reason is that a JVM would host a browser, which would then try to launch another JVM in the same process to run the applets. Since only one JVM per process is currently allowed, this would cause a crash.

Q: Why is the communication mechanism between LISA and the browser .NET Remoting?

A: Given that we have a java process and a .NET process that need to communicate, we had a few options:

- In-process is not doable for the applets reason mentioned above, as well as the required ability to run browsers under different user accounts.

- A proprietary protocol over raw sockets was too much work, especially since the communication must be bidirectional.
- Web Services over HTTP is the approach used in 4.5 and earlier but it is slow, verbose and exhibits bugs in the Web Service stacks of those platforms that undermine reliability.
- The most elegant approaches are to either load a JVM in the .NET processes and use RMI, or load a .NET CLR in the JVM and use .NET remoting. The first approach suffers from the applets limitation outlined above, which left us with the second one.

In 4.5 and earlier:

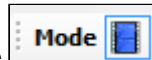
Will be completed upon request...

2. Recording Mode

2. Recording Mode

Within the Web 2.0 browser, you can record many type of applications like a simple web application, java application, swing application, .net application to name a few. In the recording mode, you can test any website and record all the events in the browser. These recorded events can later be replayed in the Playback mode.

When you open the LISA Browser, by default it starts in the Recording mode



The LISA Browser has a **toolbar** as shown below:



On the **left** of the toolbar you can use the **Mode** button to select the Mode of operation within the browser:

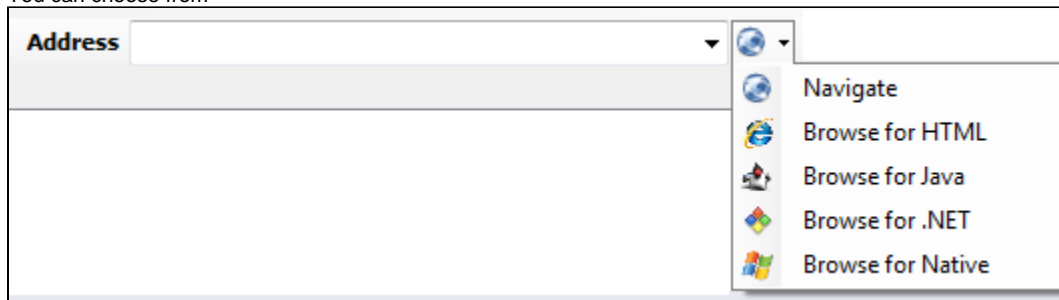
- **Recording Mode** - Where you can record the operation.
- **Edit Mode** - Where you can edit the transactions.
- **Playback Mode** - Where you can playback the recorded operation.

You can Record the web page in the Recording mode and playback the recording by clicking on the Playback mode.

You can view add/delete the Logical, Physical events and view the Object details in the Edit mode, which is described later.

To start Recording, select the appropriate option of recording from the Address bar drop down.

You can choose from -



Enter the website address in the Address bar.

You can navigate the website for testing and in the background it will record. Once done, click Save to save the recording.

This will save the recording and exit the web browser.

To playback or edit the events, open the web browser again. By default it will now open in the **Edit mode**, with the last saved recording.

For additional information see the topics below.

- [2.1 Recording Example](#)
- [2.2 Recording a Swing Test](#)
- [2.3 Recording an Applet Test](#)
- [2.4 Different Views of a Web Page](#)

2.1 Recording Example

2.1 Recording Example

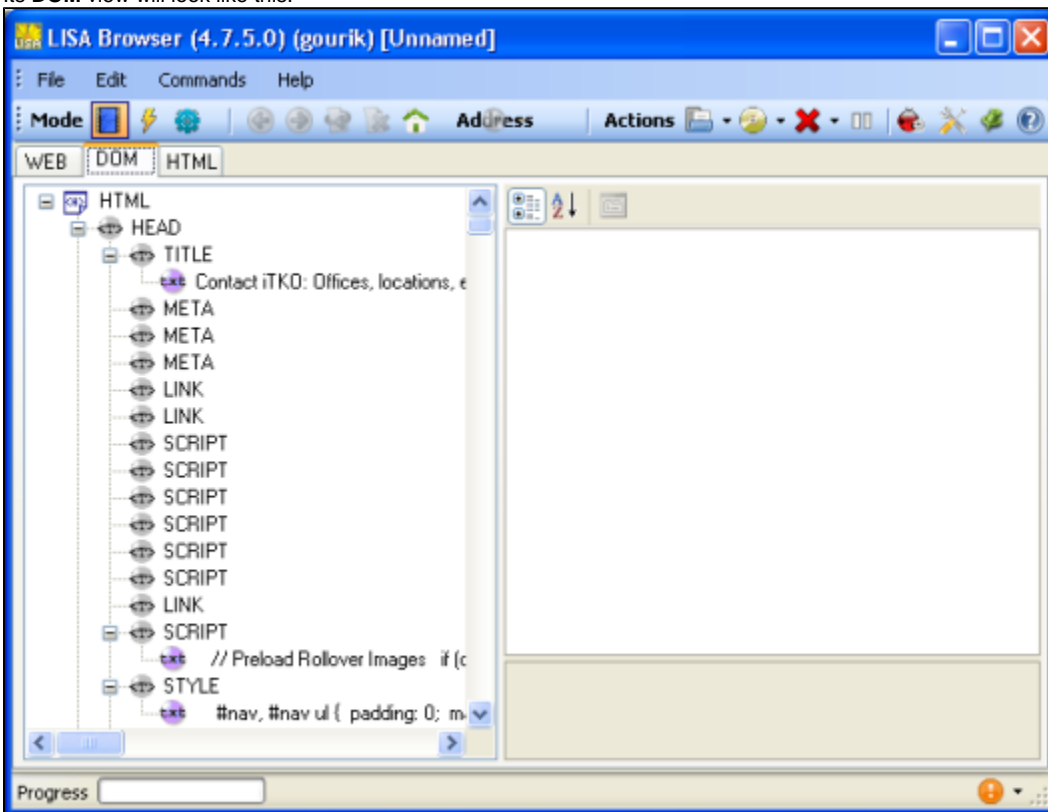
To start the recording,

- Type in the some web address, say: <http://www.itko.com/>
- Browse through the web pages, so that they get recorded.

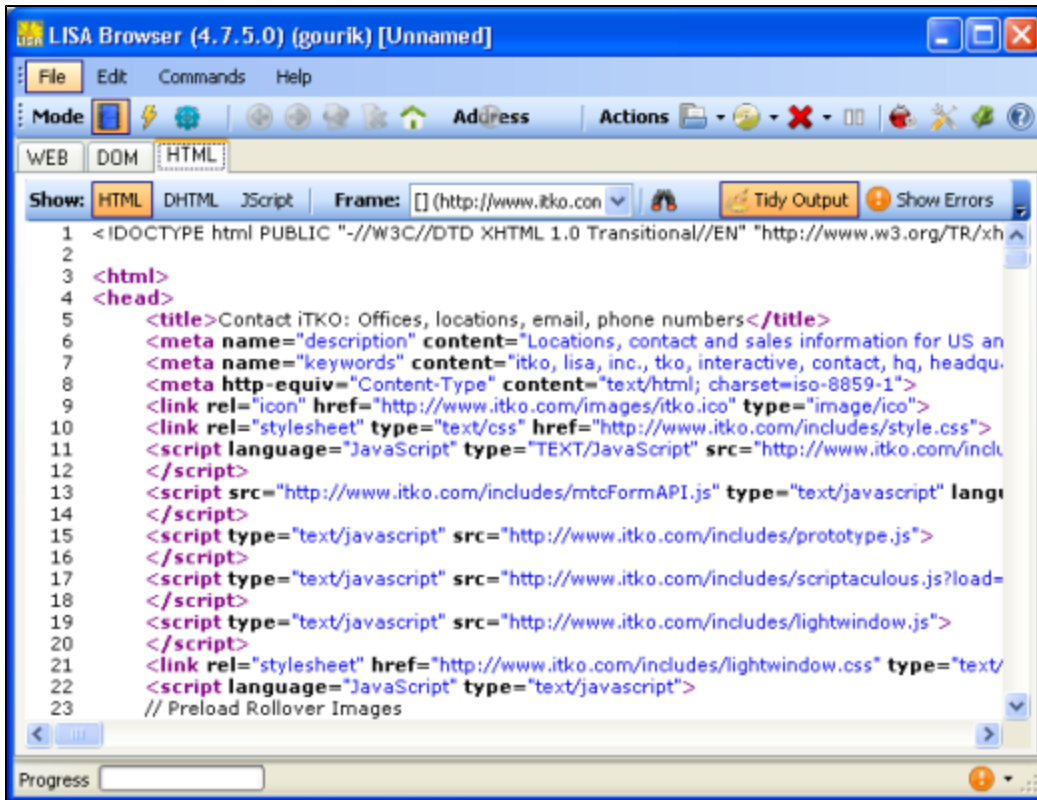
Its **Web** view will look like this:



Its **DOM** view will look like this:



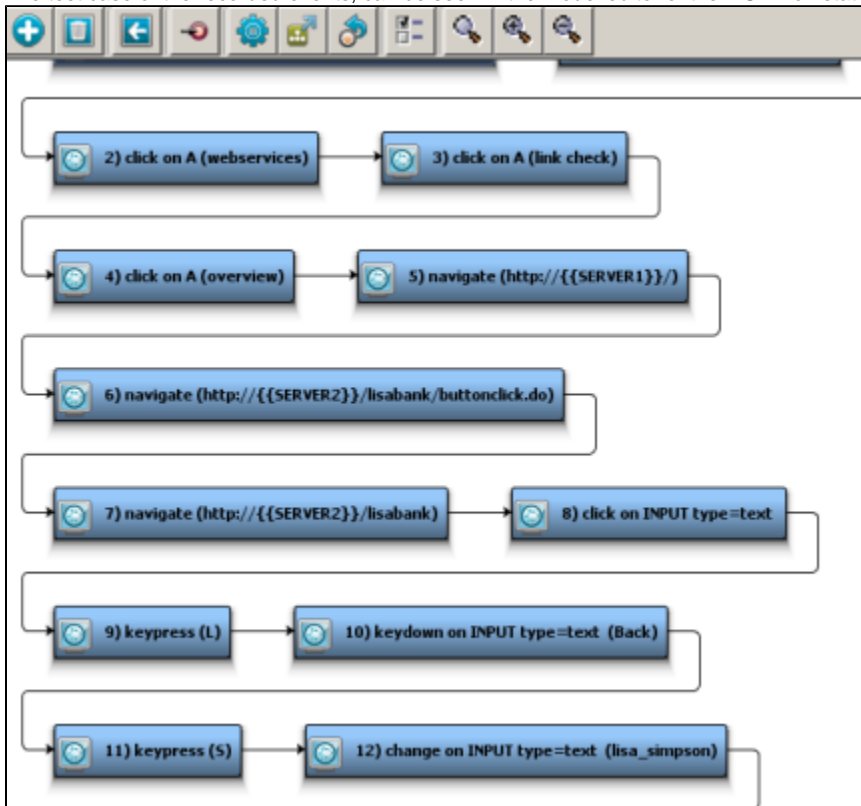
Its **HTML** view will look like this:



Once you are satisfied with the recording,

- Click the **Save** button to save and close the LISA browser.

The test case of the recorded events, can be seen in the Model editor of the LISA workstation as shown below:



The execution of this test can be triggered in the normal LISA ways, through:

- Staging a test or
- Using Interactive Test Run (ITR).

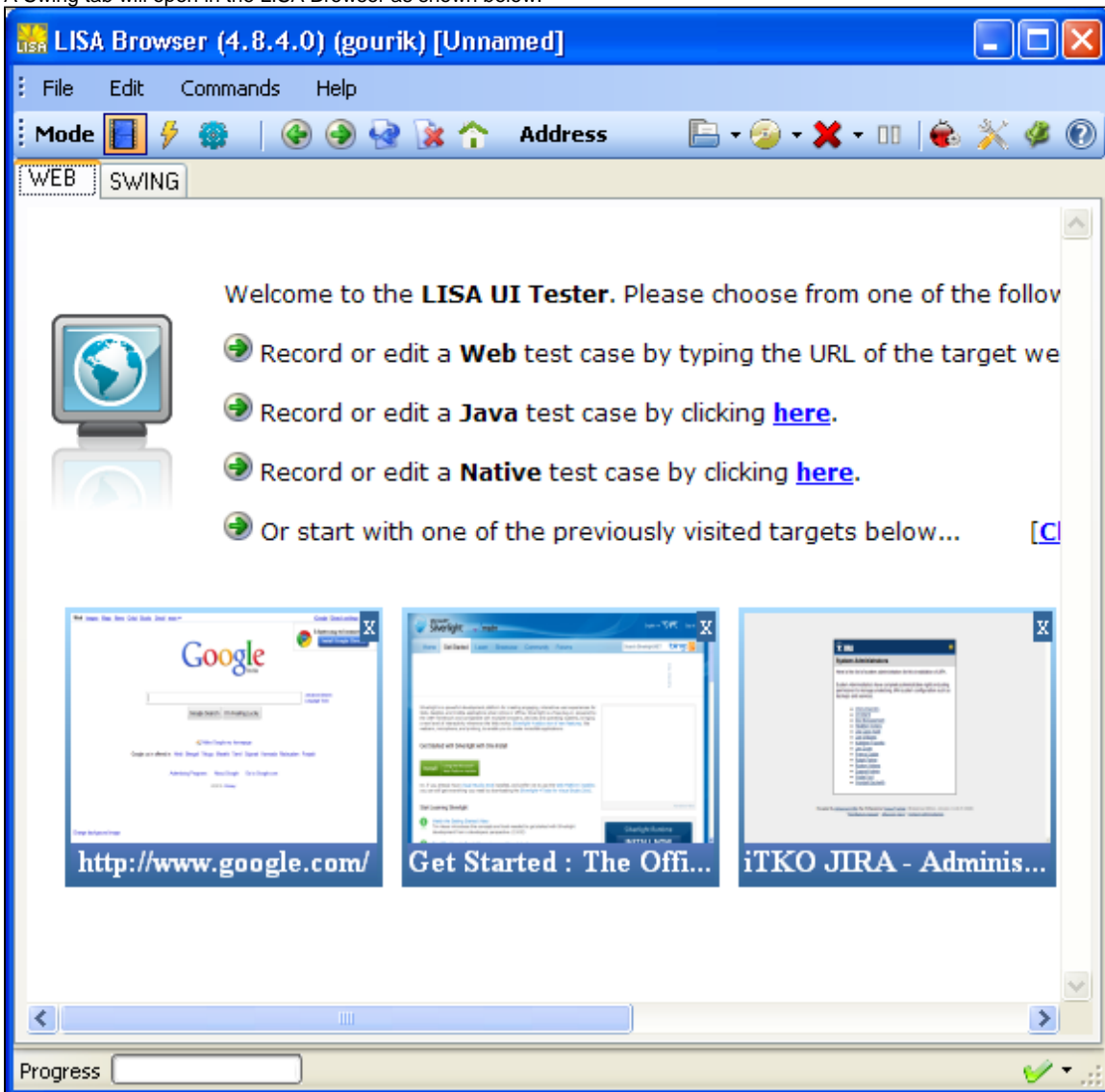
Reopen the LISA browser to view the recording in the Playback mode.

2.2 Recording a Swing Test

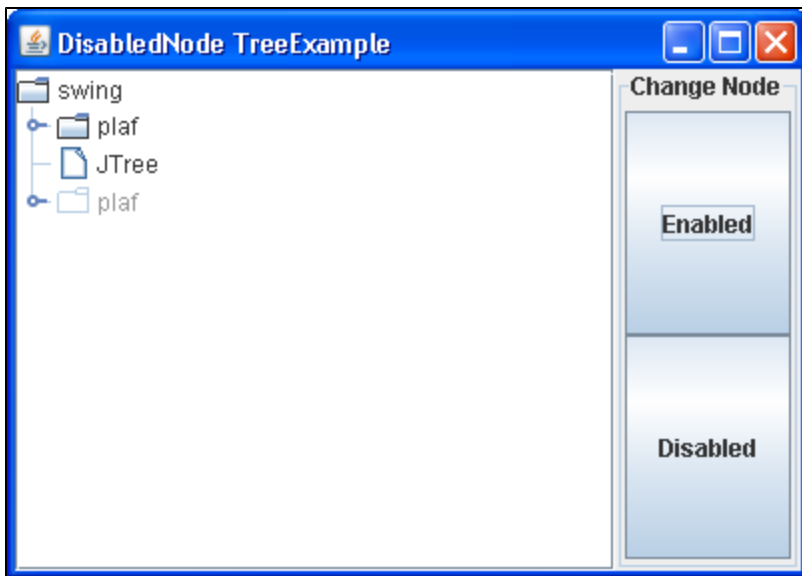
2.2.1 Recording a Swing Test case

To record a Swing test, you need to have a swing application.

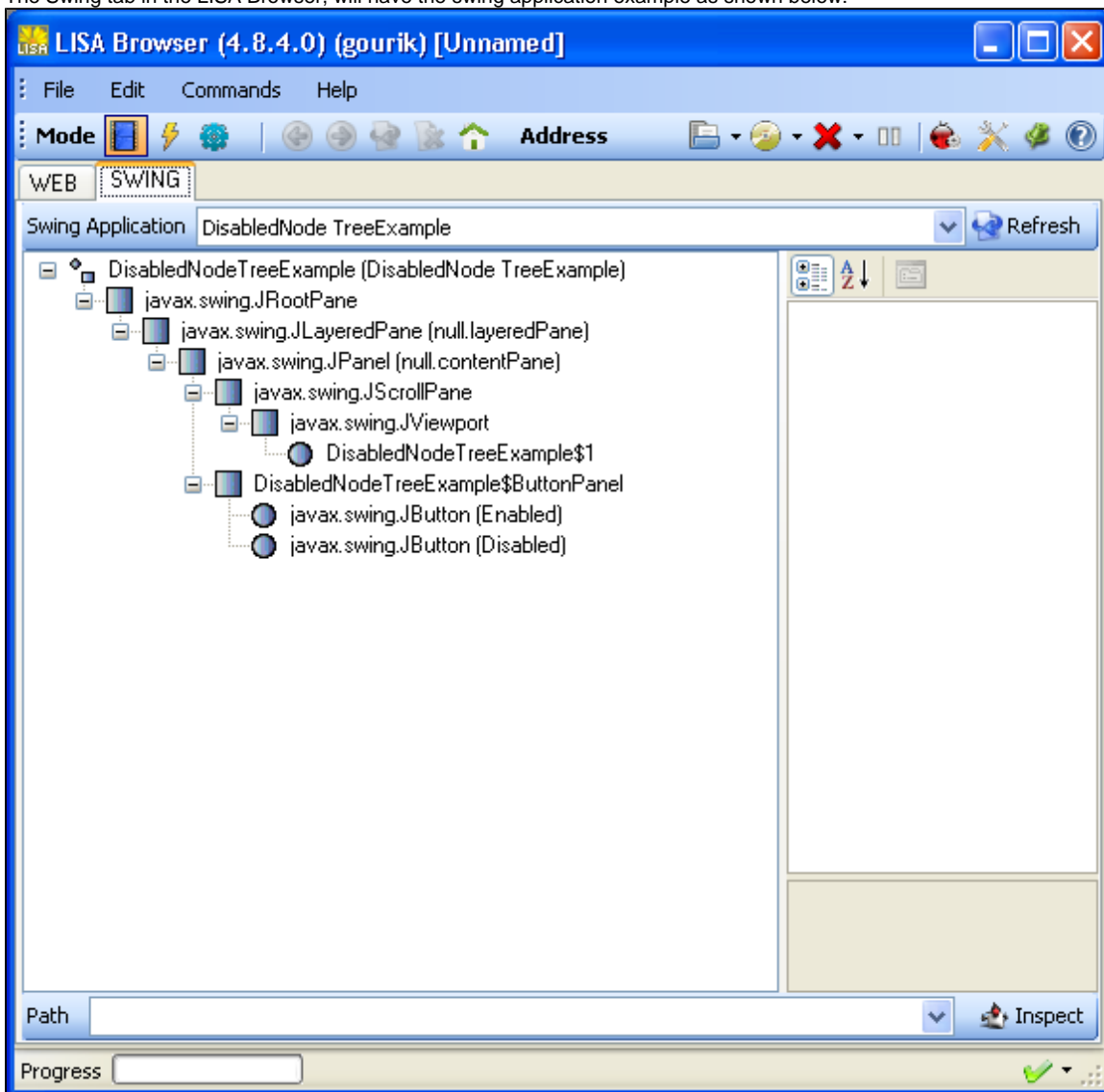
- Open the LISA Browser and click on "**Record or Edit a Java test case by clicking here**"
- Click on "here" and **browse** to the path of the swing application.
- Open the Swing application and browse for recording.
- A Swing tab will open in the LISA Browser as shown below:



A Swing example window will open as below:



The Swing tab in the LISA Browser, will have the swing application example as shown below:



2.3 Recording an Applet Test

2.3 Recording an Applet Test case

To record an Applet test, you need to have a web page which uses an Applet.

You also need to check the Applet related options in the Settings dialog box.

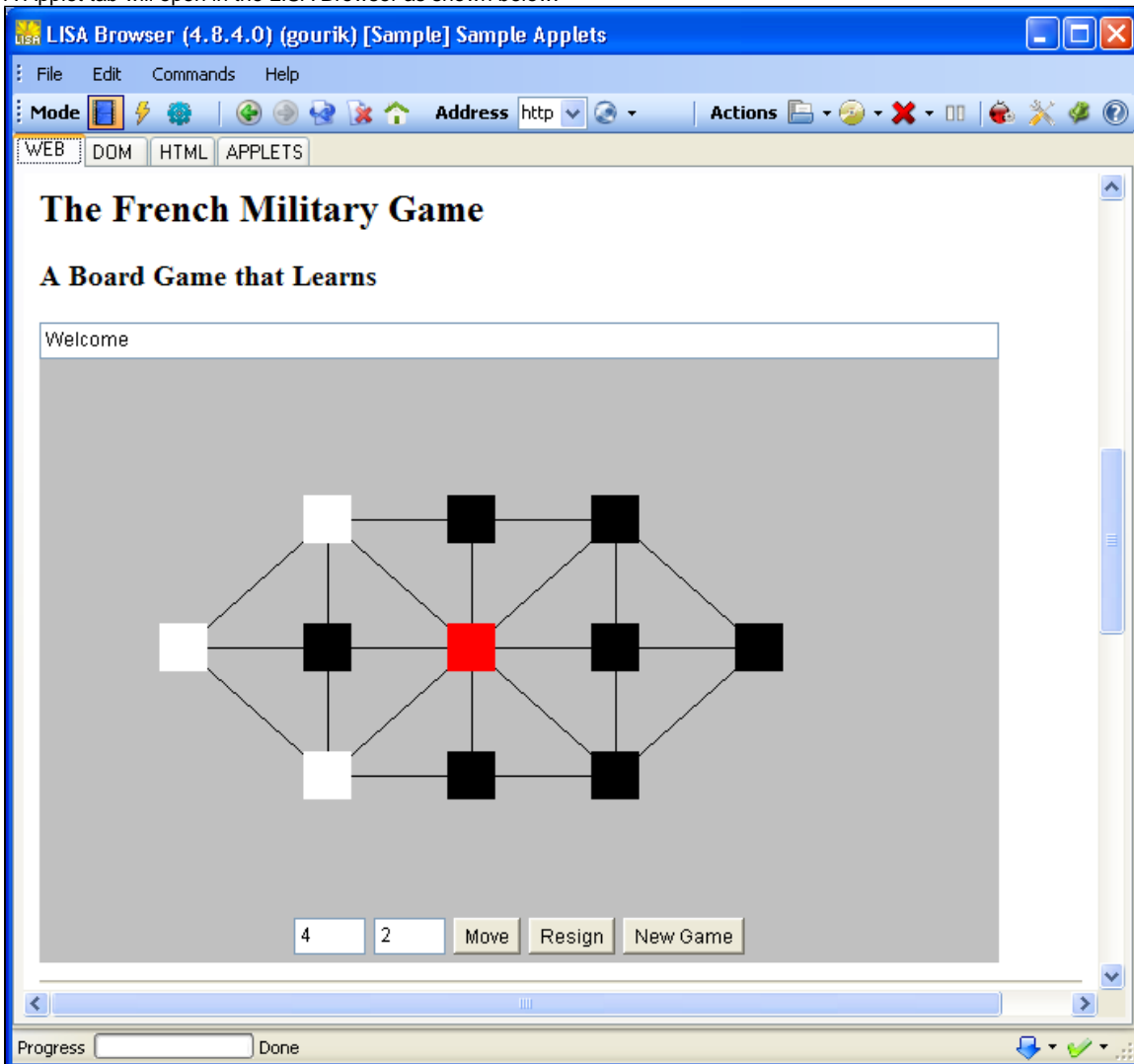
☒ Enable Applets

- In the General Settings tab

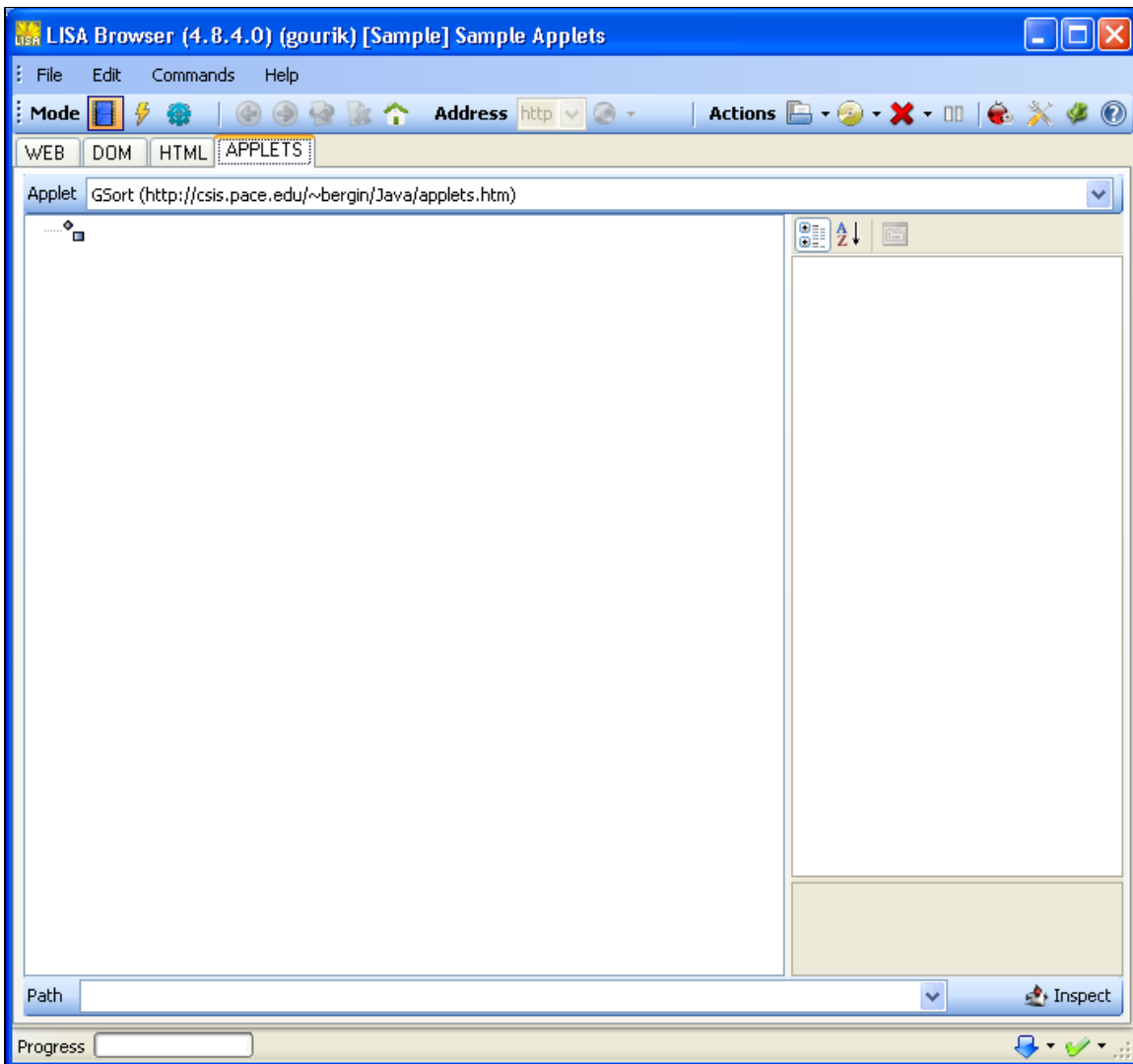
☒ Capture Applet snapshots

- In the Recordings Settings tab

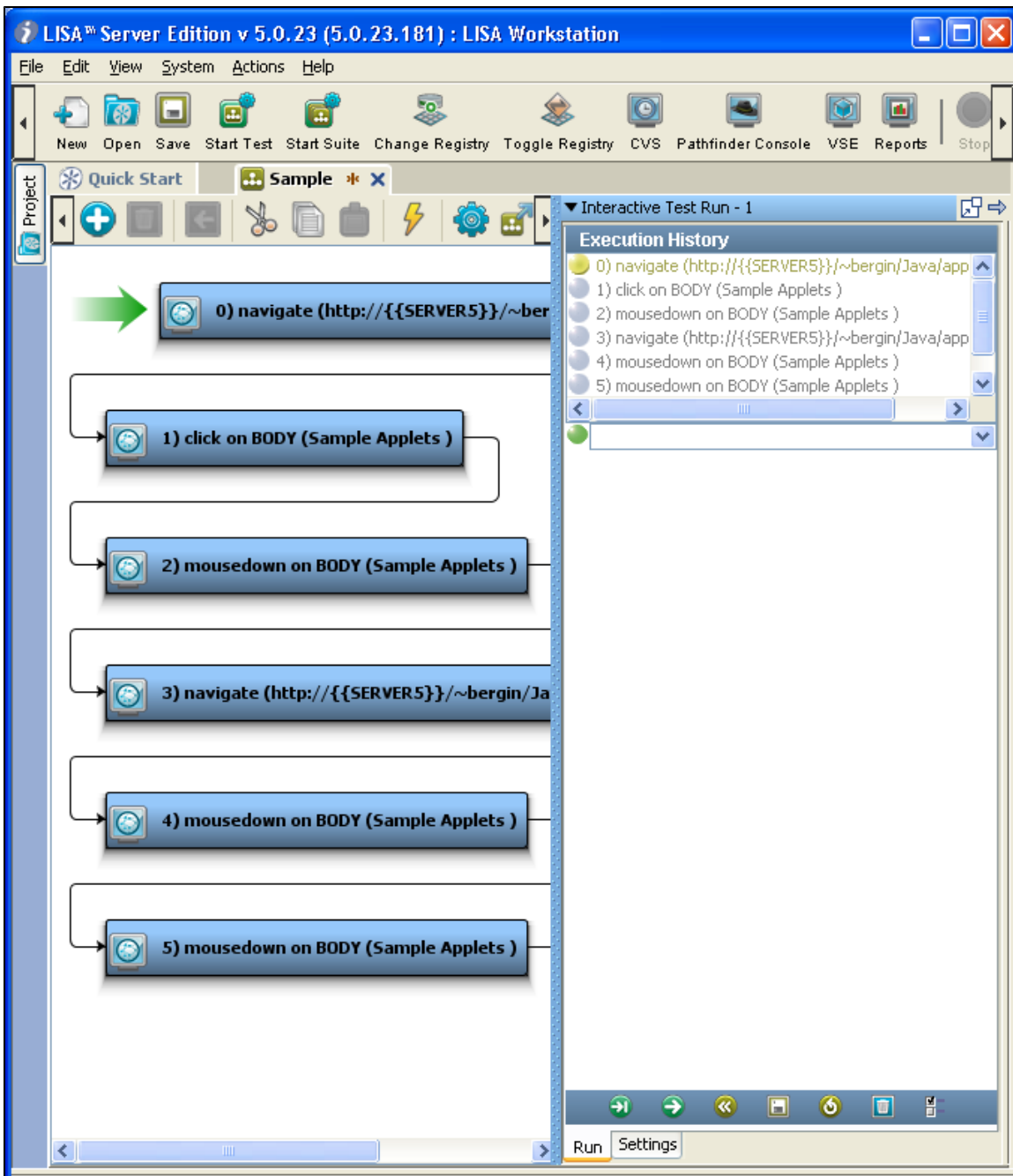
- Open the LISA Browser and click on "**Record or Edit a Java test case by clicking here**"
- Click on "here" and **browse** to the path of the web page which uses an Java Applet.
Example - <http://csis.pace.edu/~bergin/Java/applets.htm>
- Open the web page and start recording.
- A Applet tab will open in the LISA Browser as shown below:



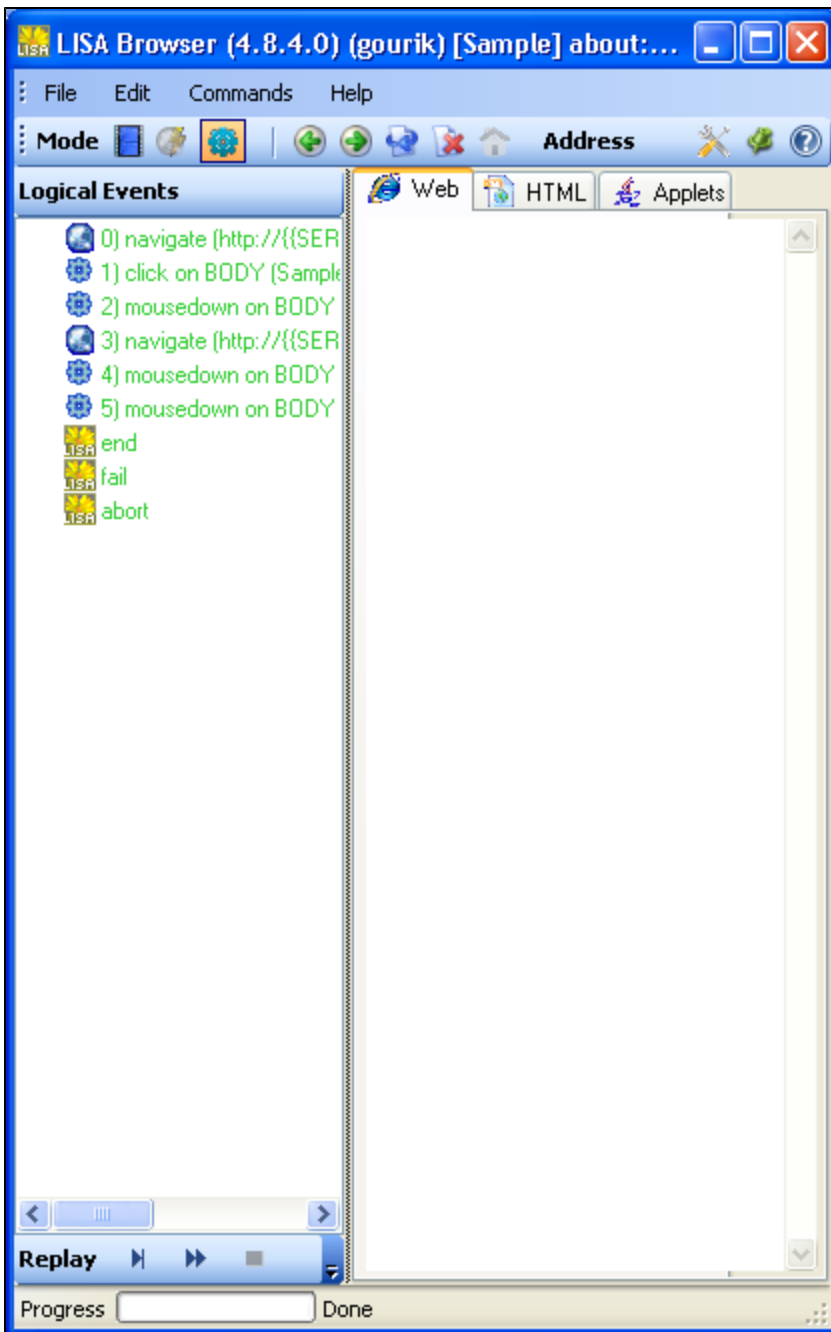
- Click on the Applet tab -



- Click Save to save the Recording. The browser closes upon saving.
- Open the LISA Workstation, where the recording appears as a test case and run it in ITR.



- When you play it in ITR, it will open the LISA Browser in the Playback Mode as shown below:



You can also see the Applets on the LISA Website.



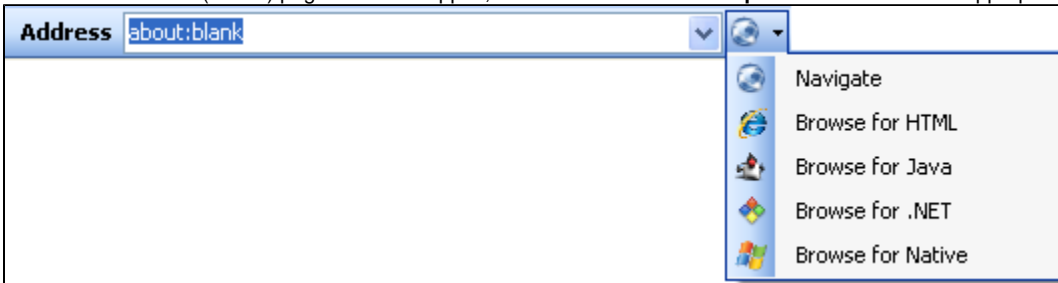
2.4 Different Views of a Web Page

2.4 Different Views of Recordings

When you record a test, it is sometimes desirable to understand how the HTML behind the page is organized. In particular, when you define Filters and Assertions, you sometimes need to know something about an html attribute on the page, or some text that might be hidden, etc.

To facilitate this, the LISA Browser is capable to show different views of pages within the browser window.

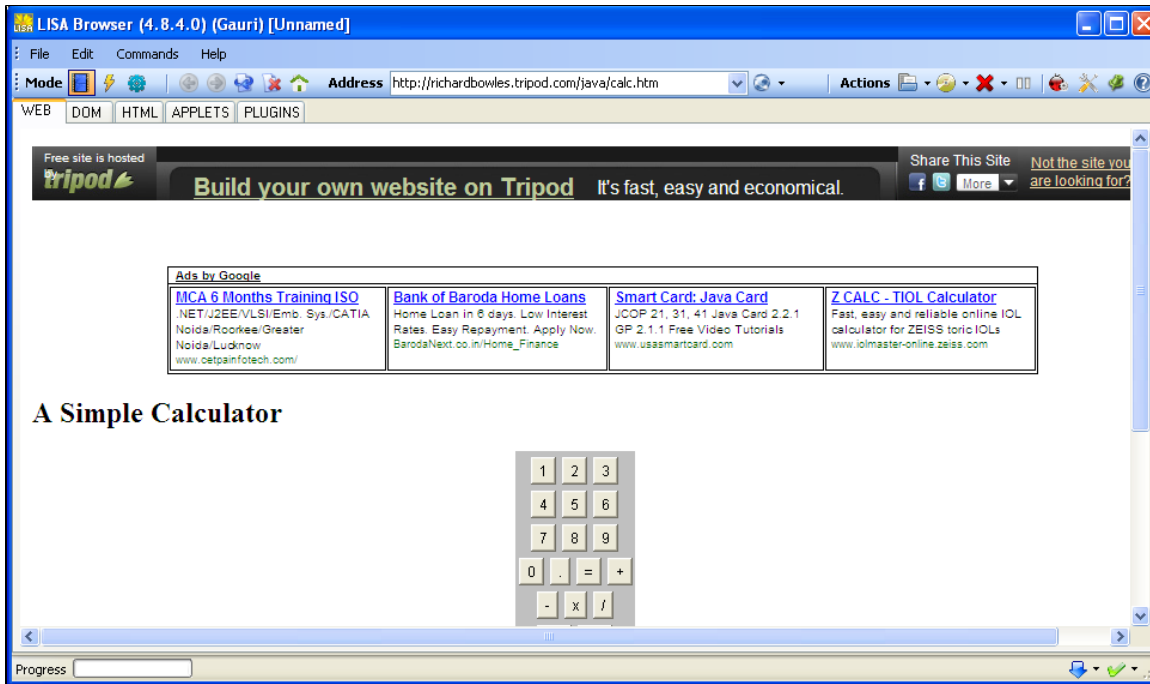
To browse for a web (HTML) page or a Java applet, click the **Address bar drop down** and select the appropriate link as shown below.



Web View

For the purpose of illustration, we have taken the example of a online calculator as shown below:

This web page is using an applet and a plugin, hence these two tabs are seen.

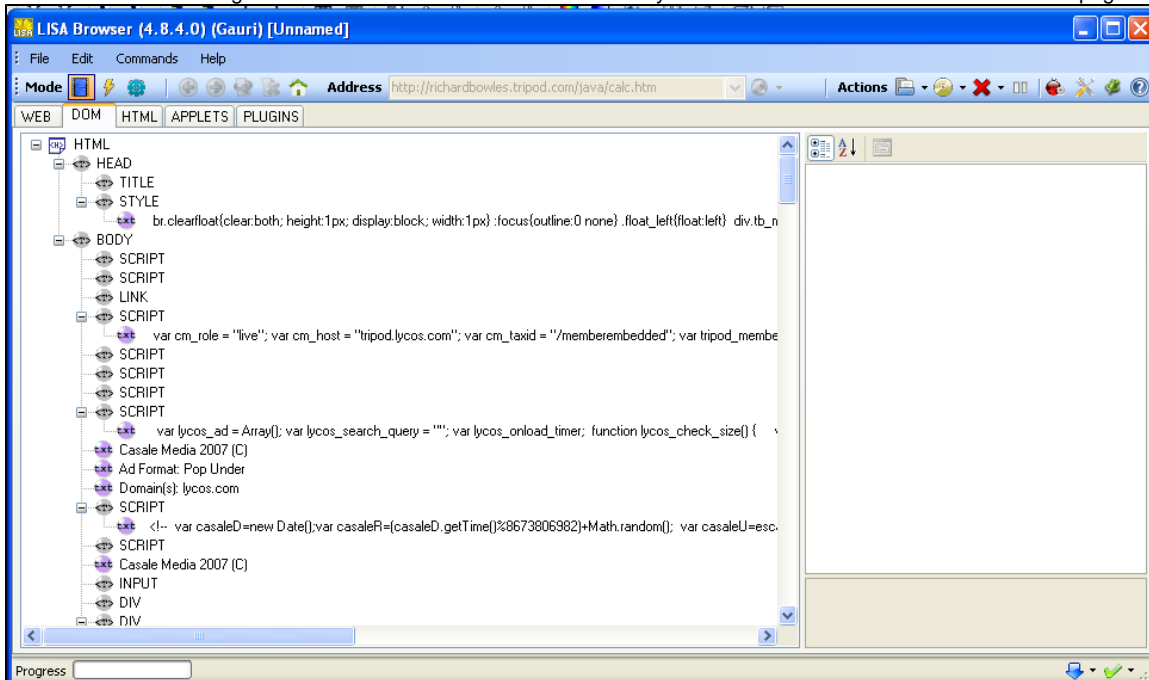


DOM View

The DOM view is a hierarchical view of the web page HTML document currently rendered in the browser. When you select an element in the tree, you can see all its attributes (name and value) displayed in the right hand-side grid.

Note that when you have a page made of frames, each frame node has its entire document available as child node so you can examine the whole hierarchy at once.

The **DOM view** is the regular view of a browser as seen below: Here you can see the DOM tree of the above web page.



HTML View

The **HTML view** is the textual source of the rendered page.

In addition, there are a few tabs at the top **Show** column, where you select the display.

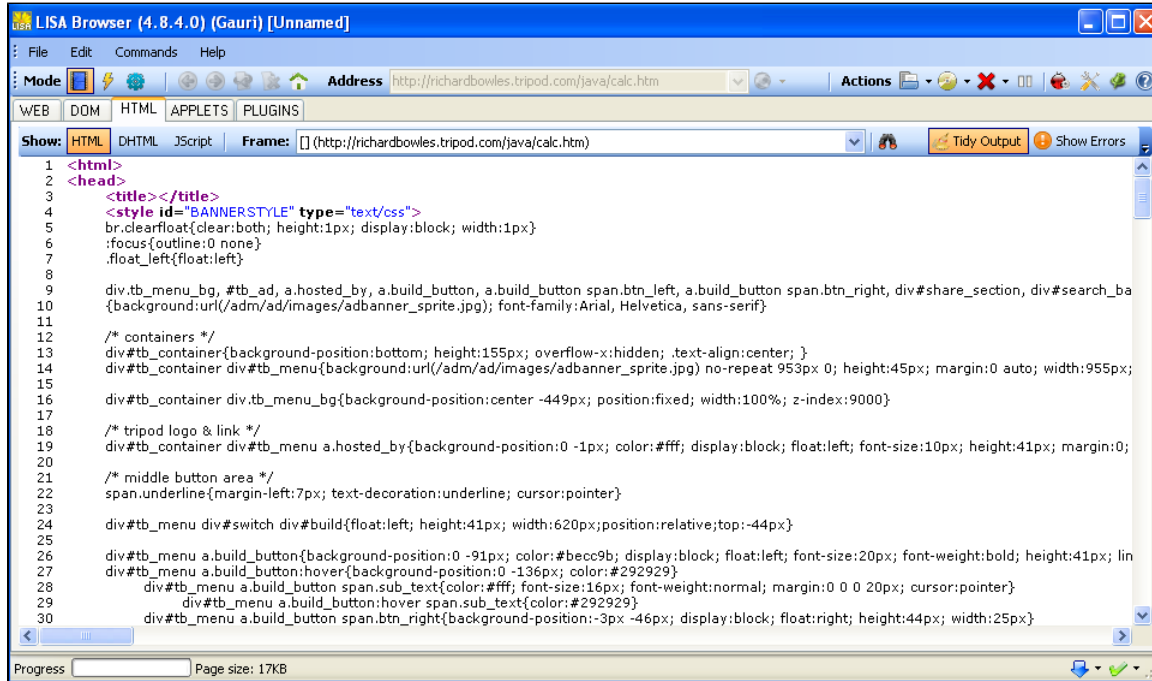
The Frame combo box lets you select which frame to display the source of in case of multiple frames, and the HTML / DHTML / JScript buttons

lets you select respectively whether you're seeing the HTML that comes from the server (static HTML), or the HTML at it is currently being seen by the client (Dynamic HTML), or the Javascript files and snippets used by the current page and frame.

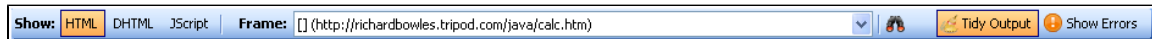
This is especially useful when pages make use of javascript or ajax that dynamically modify the html in the browser without reloading a whole page from the server.

Select **Browse for HTML** button from the Address bar drop down and enter the web page address. Here you can see the HTML view of the Java animation web page.

The page below shows the html view of the above web page.



In HTML view, there are a couple of controls at the top of the window as shown below:



- **Show HTML/DHTML/Jscript:** You can select the required script from HTML/DHTML/Jscript. The Static/Dynamic HTML group lets you select whether you're seeing the HTML that comes from the Server, or the HTML at it is currently being seen by the client. This is especially useful when pages make use of JavaScript or Ajax that dynamically modify the html in the browser without reloading a whole page from the Server.
- **Frame:** The Frame drop down box lets you select which frame to display the source of in case of multiple frames
- **Find:** The find button opens a Search dialog and **allows** you to search.
- **Tidy Output:** Shows or hides the **Tidy HTML output**
- **Show Errors:** Shows or hides the HTML errors.

Applet View

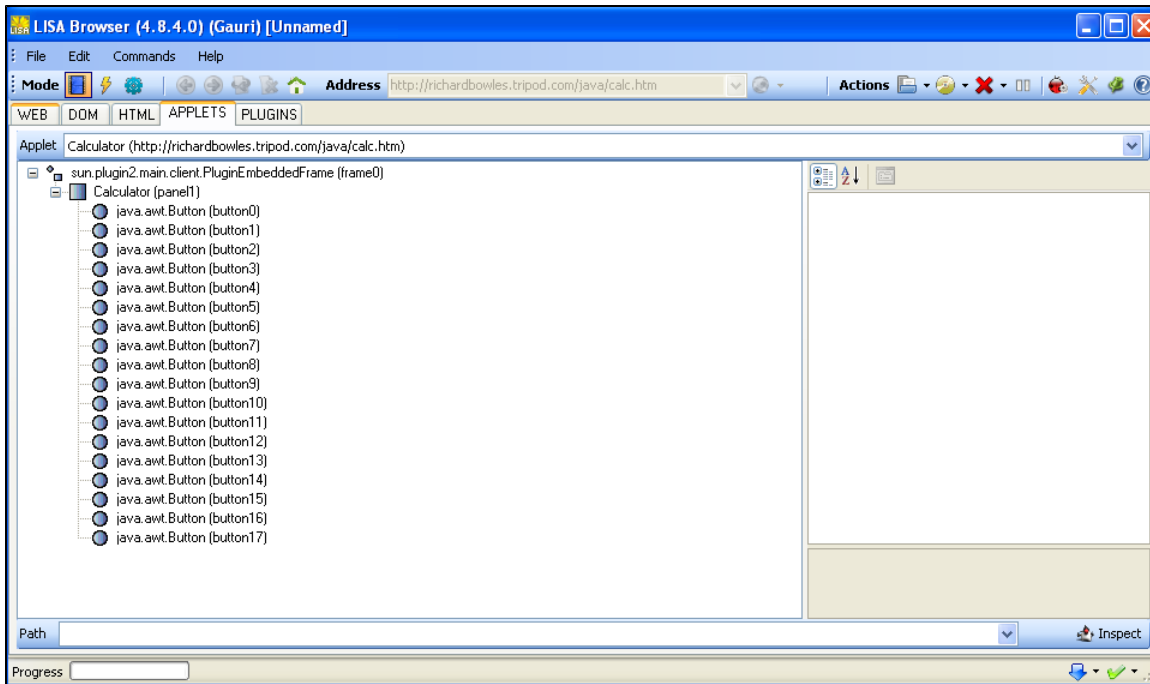
The **APPLET view** is a hierarchical view of (one of) the applet(s) currently displayed in the browser.

At the top of the panel, there is a combo box that allows you to select which applet to display the properties of in case there are several on the page.

In the tree on the left is the component hierarchy of all the UI elements that make up the applet. They are identified by class name and label or text. On the right side is the property grid that displays all the names and values of the fields of the java object that backs up the selected UI component in the tree. Those are organized by java class hierarchy (e.g. java.awt.Component properties, javax.swing.JComponent properties, etc...)

At the top of the panel, there is a drop down box that allows you to select **which applet** to display the properties of, in case there are several on the page.

Here you can see the Applet view of the Java animation web page.



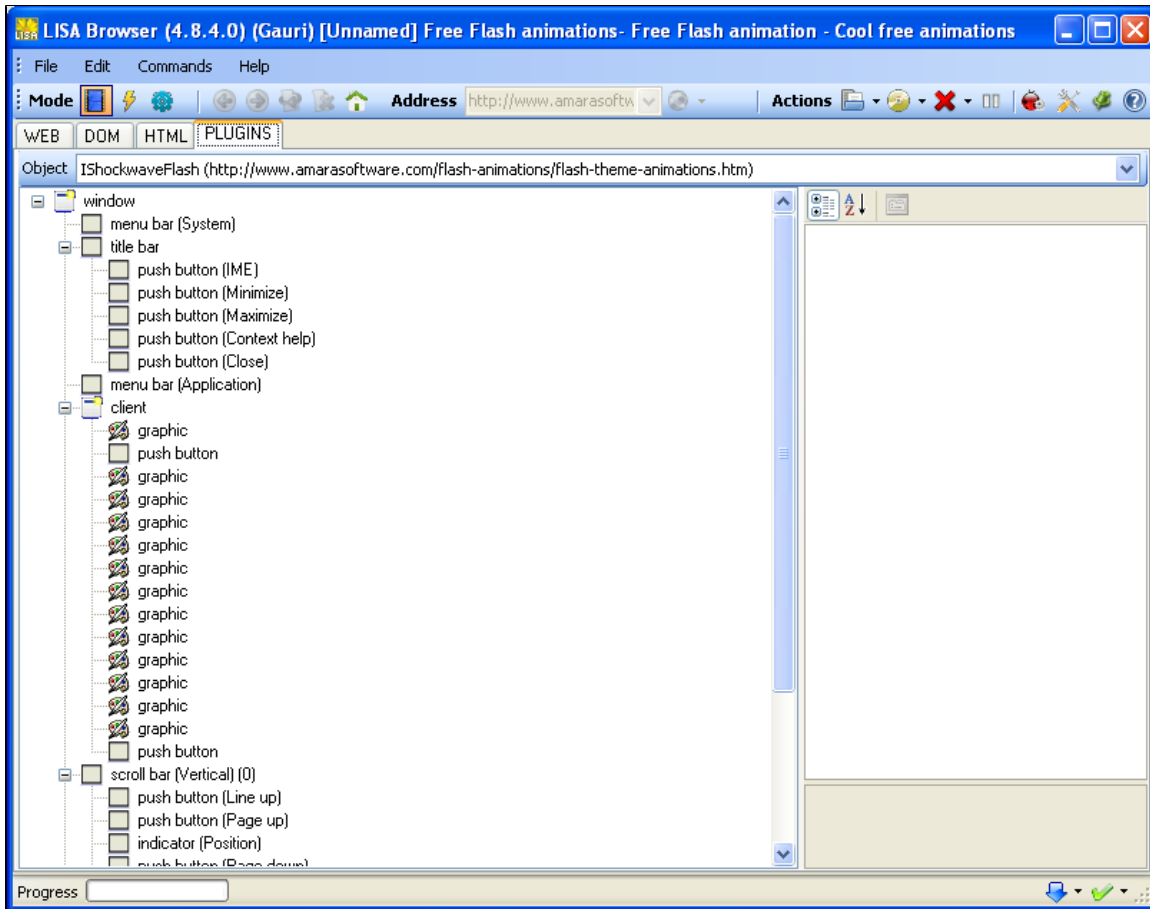
Plugins View

The **PLUGINS** view is a hierarchical view of (one of) the Active X control(s) currently displayed in the browser.

In particular those can be FLASH or FLEX controls. At the top of the panel, there is a combo box that allows you to select which object to display the properties of, in case there are several on the page. On the right side is a property grid that displays all the available information of a sub control identifiable in the object.

For the Plugins, we have taken an example of a Flash animation.

When the flash animation invokes, you can see the plugins tab as shown below:

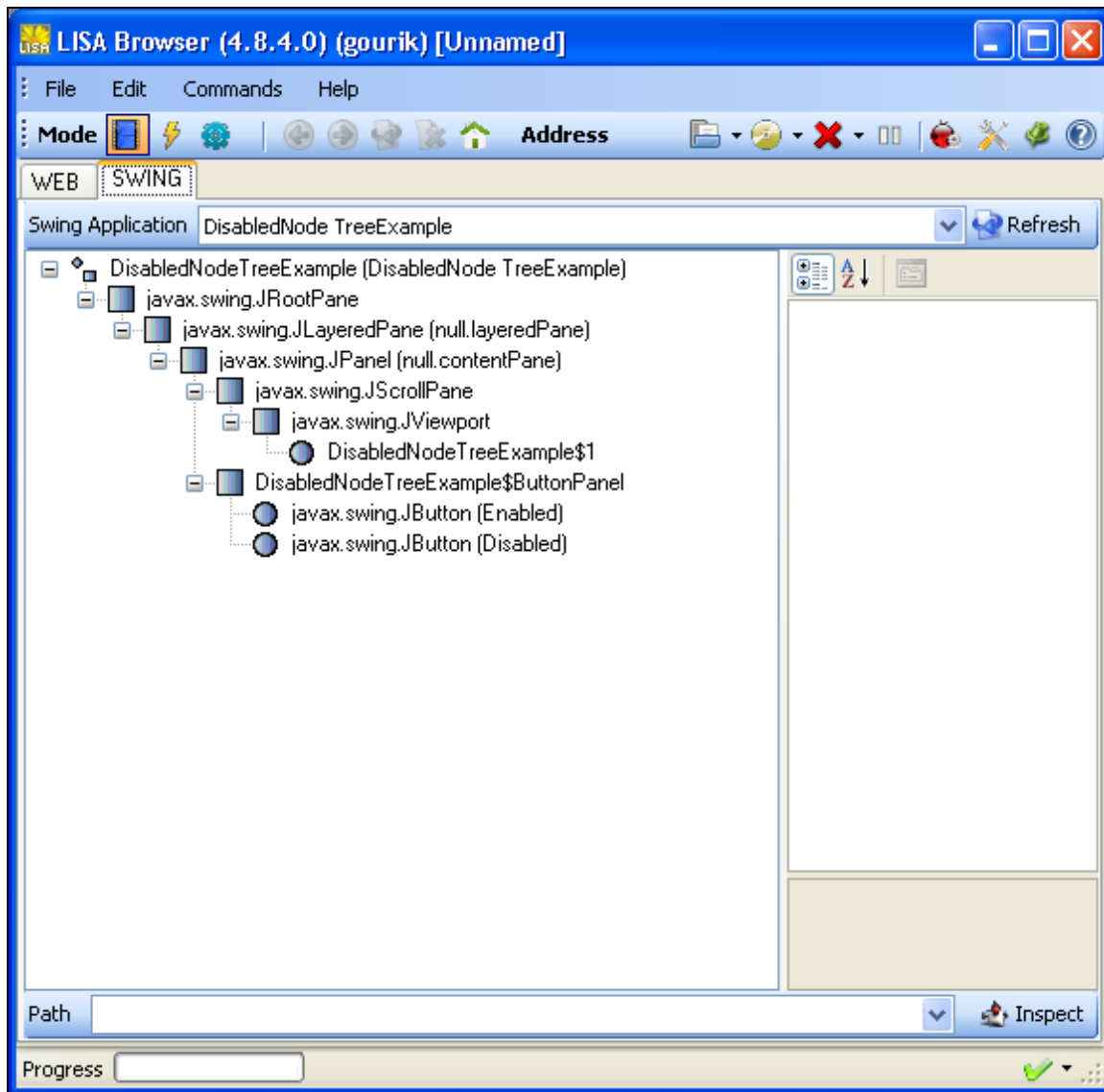


Swing View

The **SWING** view is identical to the **APPLET** view but represents the hierarchy of a recorded (or executed) Swing (or AWT) application.

In addition, selecting a node in the hierarchy will highlight the corresponding component in the Swing application.

The **Swing view** is a view of Swing operations. Here, we have running a swing application:



.NET View

The **.NET view** is a view of .Net operations.

The **.NET** view is identical to the **APPLET** and **SWING** views but represents the hierarchy of a recorded (or executed) .NET WinForms application. In addition, selecting a node in the hierarchy will highlight the corresponding control in the WinForms application.

In the tree on the left is the component hierarchy of all the UI elements that make up the applet. They are identified by class name and label or text. On the right side is the property grid that displays all the names and values of the fields of the java object that backs up the selected UI component in the tree.

2.5 Post Recording

2.5 Post Recording

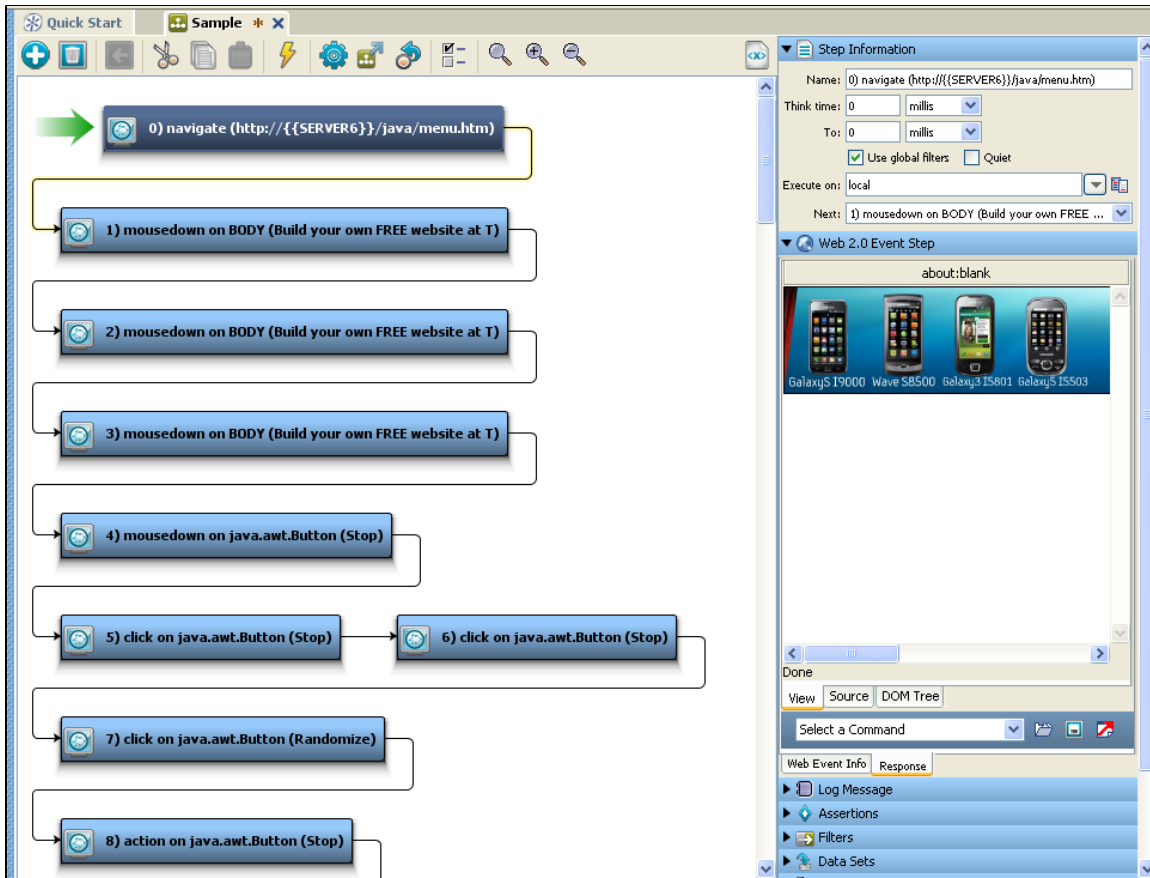
Once the recording is done and you save the recording, the browser closes the window.

You can now see the recording as a LISA Test case in LISA Workstation.

For example, we record a few pages of a website. Once we save this recording, it will close the browser window and you can see the test case formed in LISA workstation as shown below:

You can edit any of the test steps by double clicking on the test step in the LISA Workstation.

Select a particular step, to open its editor in the right panel:



There are two tabs at the bottom in the right panel.

Web Event Info tab - Click on this to see the Web Event Information. Click on the "Edit Events Via Recorder" button to open the LISA browser for editing as shown below:

▼ Step Information

Name:

0) navigate (http://{{SERVER6}}/java/menu.htm)

Think time:

0

millis

▼

To:

0

millis

▼

☒ Use global filters

☐ Quiet

Execute on:

local

▼

Next:

1) mousedown on BODY (Build your own FREE ...

▼

▼ Web 2.0 Event Step

vent

Launch Web Recorder

Edit Events Via Recorder

Event Type

navigate

▼

Window ID

0

▼

Frame ID

▼

URL

http://{{SERVER6}}/java/menu.htm

XPath

Value

Think Time

0

Web Event Info

Response

Response tab - Click on the Response tab to see the response as shown below:

▼ Step Information

Name:

0) navigate (http://{{SERVER6}}/java/menu.htm)

Think time:

0

millis

To:

0

millis

☒ Use global filters

☐ Quiet

Execute on:


local

Next:

1) mousedown on BODY (Build your own FREE ...)

▼ Web 2.0 Event Step

about:blank



Galaxy S I9000 Wave S8500 Galaxy 3 I5801 Galaxy S I5503

Done

View

Source

DOM Tree

Select a Command

Web Event Info

Response

Similarly, you can click on the Source and DOM tree tabs to see the respective outputs.

▼

Step Information

Name:

0) navigate (http://{{{SERVER6}}}/java/menu.htm)

Think time:

0

millis

▼

To:

0

millis

▼

☒ Use global filters

☐ Quiet

Execute on:

local

▼

Next:

1) mousedown on BODY (Build your own FREE ...

▼

▼

Web 2.0 Event Step

<?xml version="1.0" ?><!DOCTYPE

<STYLE>A:link {

COLOR: #000000

}

A:visited {

COLOR: #000000

}

A:hover {

COLOR: #000000

}

A:active {

COLOR: #000000

}

#abgt .curve DIV {

BACKGROUND-COLOR: #666

}

</STYLE>

<SCRIPT>

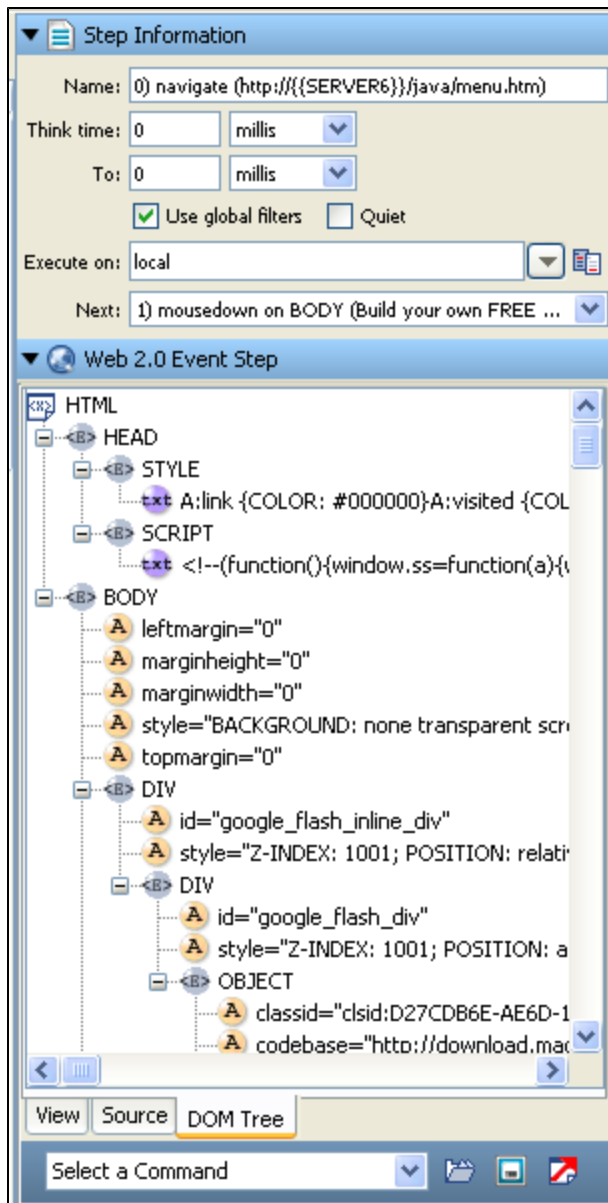
View

Source

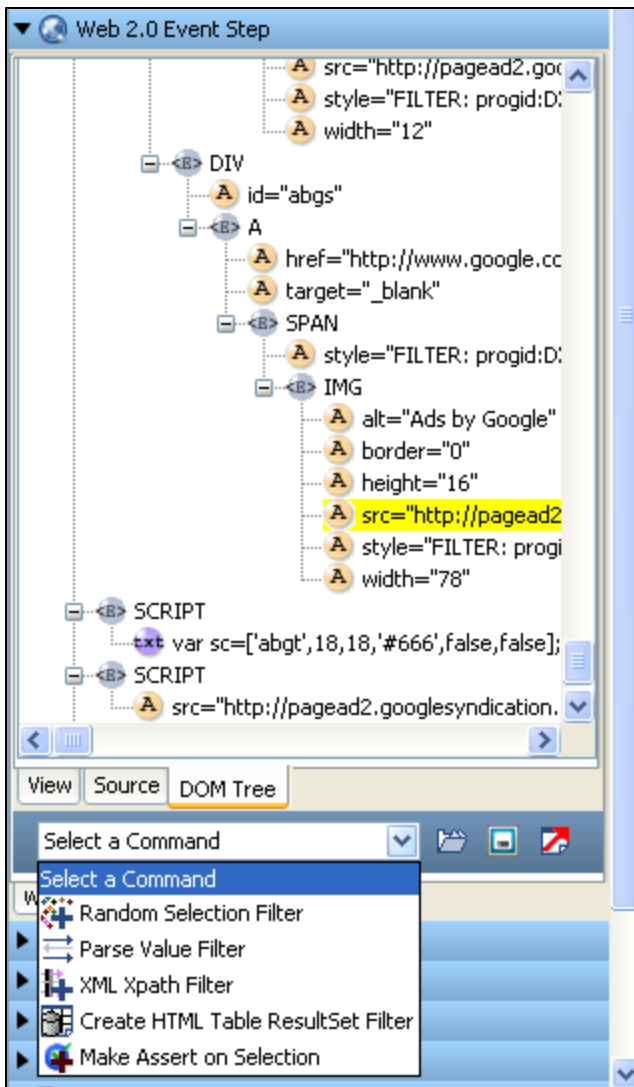
DOM Tree

Select a Command

▼



Within the DOM tree view, you can select a Attribute and select a command to be applied to it as shown below:



3. Playback Mode

3. Playback Mode

To execute or debug a Web 2.0 test, you will normally save it to LISA and execute it through **Staging** or in the **ITR** Interactive Test Run.

However the Web 2.0 browser also provides some facilities to do some quick runs and debugging of your tests. It is primarily designed to work with Web 2.0 specific tests but has some powerful debugging capabilities.

A typical usage of this is right after a recording session, **doing a playback** to make sure everything goes as you expect. Once you have saved the Recording, you can **Replay** the same using the Playback mode.




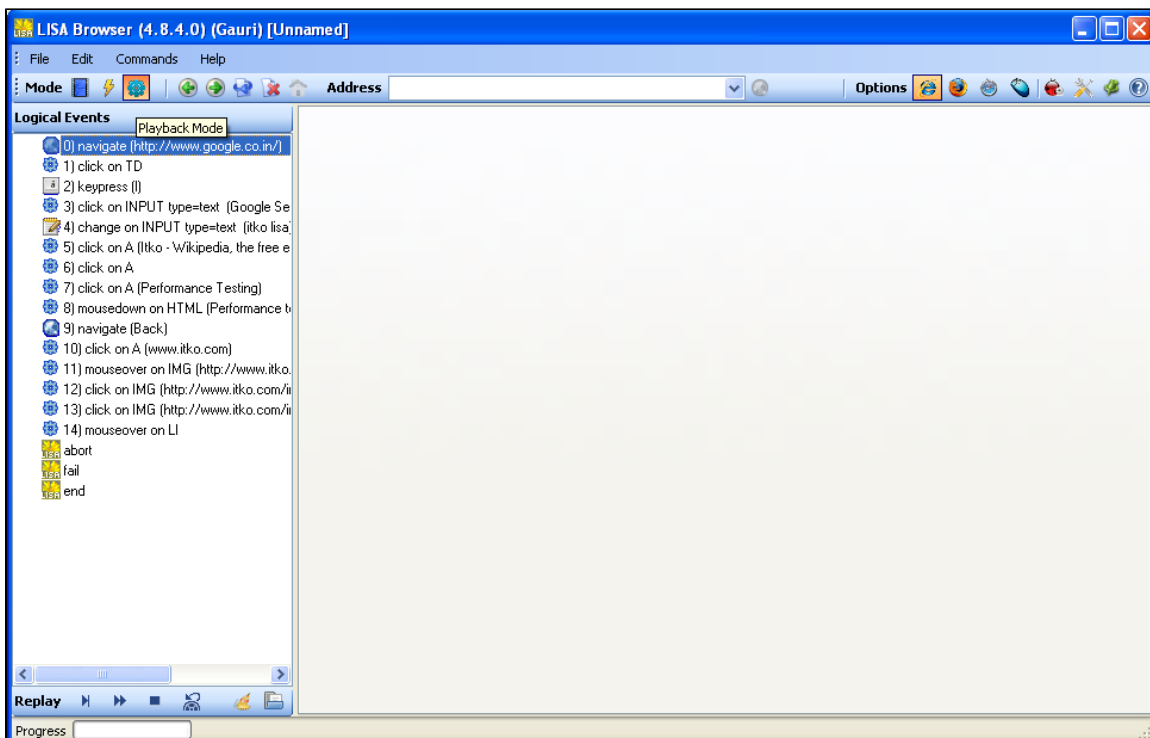
Click on the **Playback** button on the toolbar to view the details of the last recording.


During the playback, the browser toolbar is disabled.


This is the recording mode of a web page:




The playback of the above recording is enabled by clicking on the playback mode button  on the toolbar as shown below:



The playback toolbar has a few different buttons than the recording mode, like , wherein you can select the **browser** for playback from the options given options..

The Internet Explorer, Mozilla Firefox, Safari buttons,  control which browser is being used to execute the web test. You can select 1, 2 or 3 of them at a time. If more than one is selected, the selected browsers are used side-by-side. If none is selected, Internet Explorer is used as the default (since it is automatically installed on Windows).

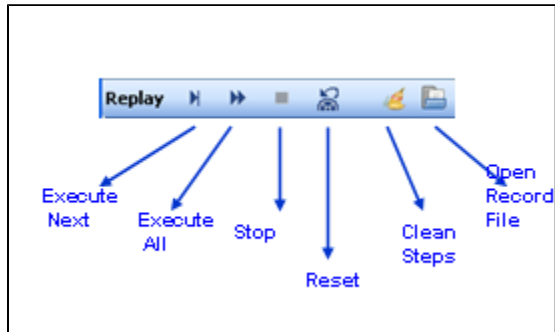
You can use the **Move mouse** button  to turn on or off mouse movement during playback.

Playback Toolbar



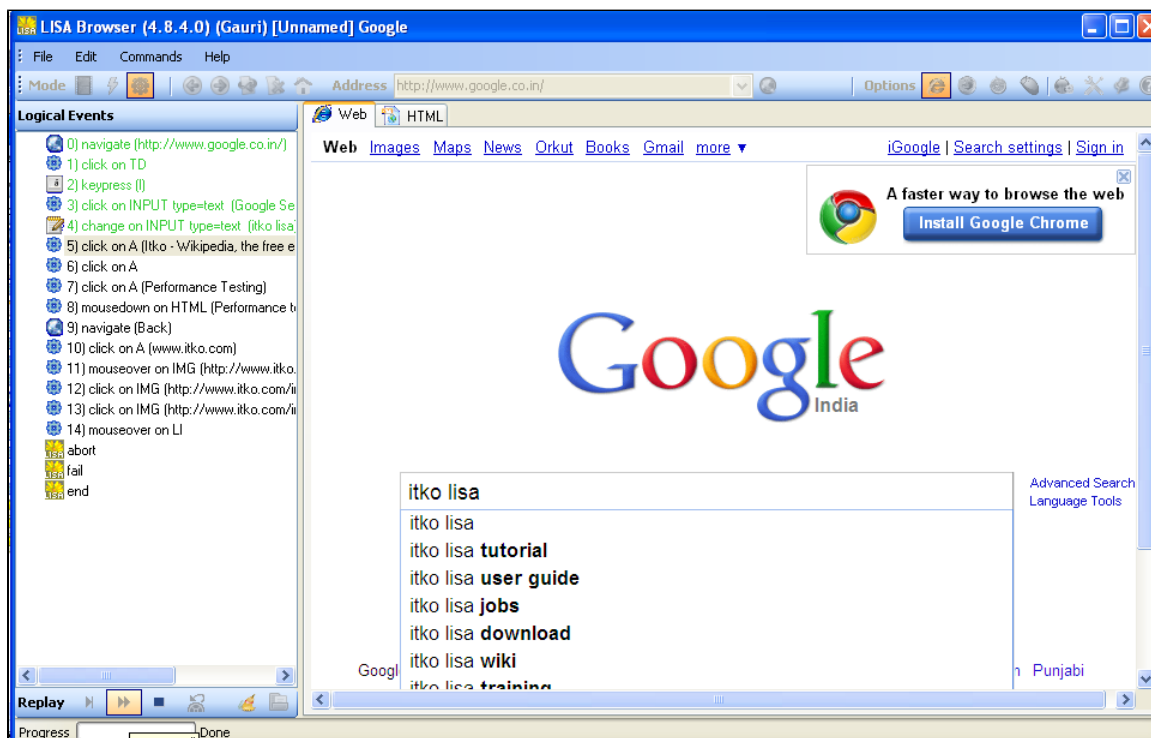
The playback mode has a toolbar at the bottom of the Logical events pane, to control the movement of the playback activities.

Click on Execute Next or Execute all steps to run the playback as shown below:



- **Execute Next** executes the selected event in the list.
- **Execute All** executes all the events starting at the selected one.
- **Stop** will stop the playback
- **Reset** will reset all the variables and executes all the events in the list starting with their first one.
- **Clean Steps** will disable all events that generated a warning in the last run.
- **Open Record File** will open a previously recorded file.

The Logical Events are listed on the left side and the entire recording will be displayed on the right side.

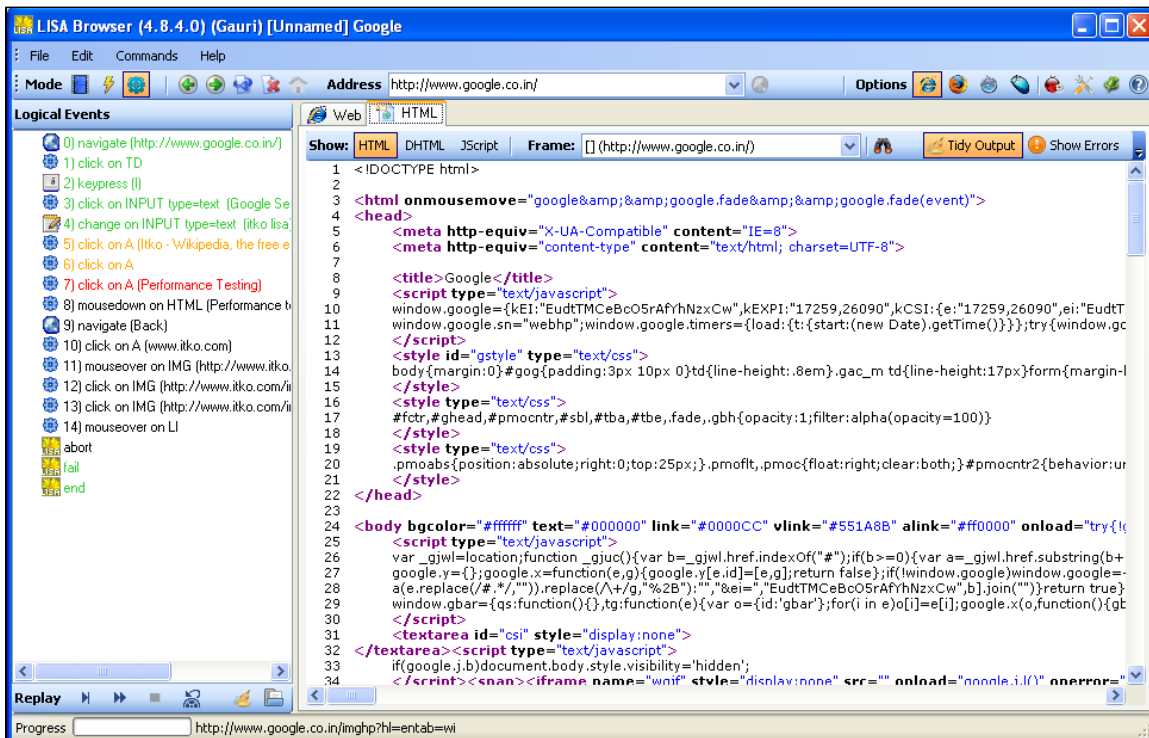


Once the playback starts, the steps that have run, are shown in green color in the Logical Events panel.

Tip: You can manually alter the flow by simply selecting an event in the list and clicking **Next** or **Play**. Alternatively, you can also press <CTRL><E> to execute the next node, which is useful if mouse movement is turned on.

If something is not going as you expect during a long test, you can also press CTRL-C to stop execution.

Click on the **HTML** tab within the Playback mode,



This will open a new menu bar as shown below:



- **HTML** - Click to view the html source
- **DHTML** - Click to view the DHTML source
- **JScript** - Click to view the Java Script source
- **Frame** - Will display the frame name
- **Find** - Will open a search window
- **Tidy Output** - Will show the tidy output
- **Show Errors** - Will open the error window

4. Edit Mode

4. Edit Mode

To open the Edit mode,

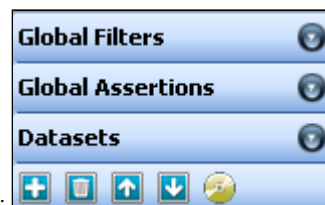


Click the Edit mode option in the toolbar to open the browser in the Edit mode as shown below:

In the Edit mode, you can view the Logical and Physical Events, Object Details and Response Panel.

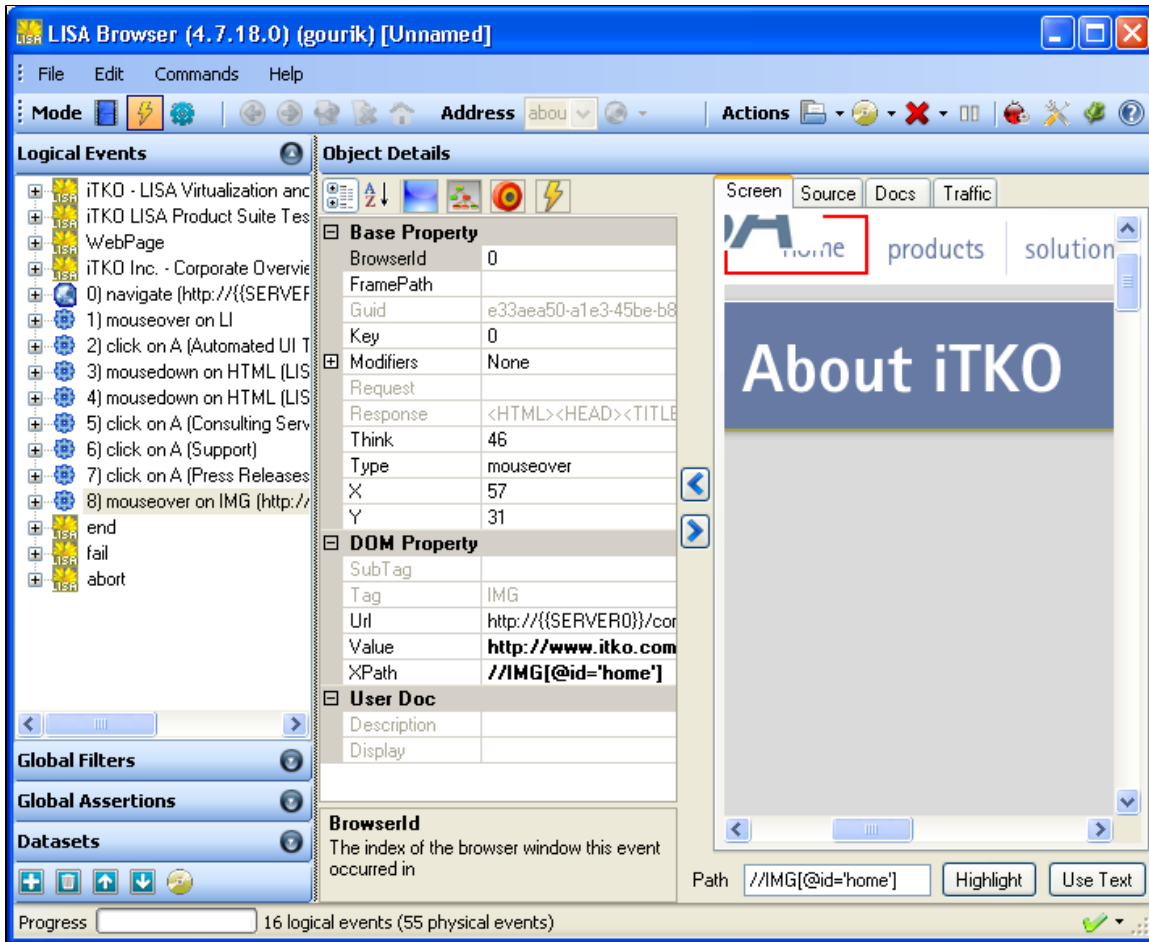


By default, the Logical events tab is open in the Edit mode.



In addition to this, there are the following tabs in the Edit mode which can be expanded:

The main edit mode window is as seen below:



In the left panel, a list of Logical Events is seen and on the right panel, you can see the Object details.

Please see the available subtopics for more information.

- [4.1 Event Types](#)
- [4.2 Logical Events](#)
- [4.3 Object Details](#)
- [4.4 Filters](#)
- [4.5 Assertions](#)
- [4.6 Datasets](#)
- [4.7 Editing Steps in Workstation](#)

4.1 Event Types

4.1 Event Types

HTML pages can generate many different types of events and we're going to review them here to make it clearer when we talk about them in the rest of the document.

There are several **sources for Events**:

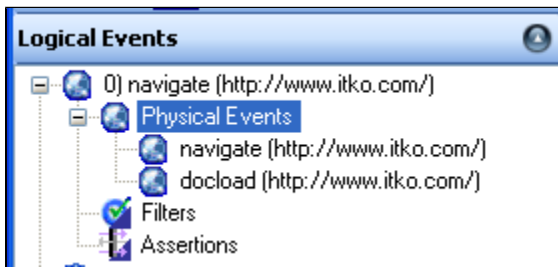
- Html page (DOM events),
- Browser environment (Native events),
- Plugins (Applet events and others),
- and finally actual events that are imported by LISA from steps that make use of other technologies or are used as markers (such as the fail or end events).

There are two types of events:

- Physical Events
- Logical Events

Physical Events - When we record or playback a Web 2.0 test, all the events occurring within or all actions taking place, are the physical events.

In the Web 2.0 browser, it is seen here -



Logical Events - However, from the user's point of view, only a small subset of these events is interesting and those are the logical events.

In the Web 2.0 browser, it is seen here -






For example, clicking on an html link could result in a mouse down, focus, mouse up and click events (Physical events), but the user sees this as only a click event (Logical Events).










LISA will actually group these physical events together into an event bucket and mark the click as the bucket's logical event.

DOM Events








| Icon | Event | Description |
|------|--------------|--|
| | Navigate | the user navigates to a page by using the address bar or the navigation buttons |
| | Doc Load | a html page or frame is loaded as a result of a user action |
| | KeyPress | the user presses a key |
| | Change | a DOM element's value is changed (inputs, selects or text areas are subject to this for example) |
| | Focus | a DOM element receives the focus in a page or frame |
| | Click | a DOM element is clicked |
| | Double Click | a DOM element is double-clicked |
| | Mouse Down | the user presses a mouse button |



| | | |
|---|------------|--|
|  | Mouse Up | the user releases a mouse button |
|  | Mouse Over | the mouse hovers over a DOM element area |
|  | Mouse Out | the mouse leaves a DOM element area |

Applet Events

| Icon | Event | Description |
|---|---------------------|---|
|  | Applet Load | a new applet is loaded by a Doc Load |
|  | Asynchronous Change | the applet hierarchy or visibility changed as the result of a user action |
|  | Focus | an AWT or Swing component receives the focus |
|  | Click | an AWT or Swing component is clicked |
|  | Double Click | an AWT or Swing component is double-clicked |
|  | Change | an AWT or Swing component's value is changed (text fields or com boxes are subject to this for ex.) |
|  | Mouse Down | the user presses a mouse button |
|  | Action | AWT/Swing's notion of an action event |
|  | KeyPress | the user presses a key |

Swing Events





| Icon | Event | Description |
|---|---------------------|---|
|  | Applet Load | a new applet is loaded by a Doc Load |
|  | Asynchronous Change | the applet hierarchy or visibility changed as the result of a user action |
|  | Focus | an AWT or Swing component receives the focus |
|  | Click | an AWT or Swing component is clicked |
|  | Double Click | an AWT or Swing component is double-clicked |
|  | Change | an AWT or Swing component's value is changed (text fields or com boxes are subject to this for ex.) |
|  | Mouse Down | the user presses a mouse button |

| | | |
|---|----------|---------------------------------------|
|  | Action | AWT/Swing's notion of an action event |
|  | KeyPress | the user presses a key |

Native Events

| Icon | Event | Description |
|------|-------------|---|
| | Open | a new browser window is opened |
| | Close | a browser window is closed |
| | Alert | a JavaScript alert dialog is clicked by the user |
| | Confirm | a JavaScript confirm dialog is clicked by the user |
| | File Dialog | a native File Open or File Save dialog is clicked by the user |

External Events

| Icon | Event | Description |
|---|----------|---|
|  | Continue | a no-op event |
|  | End | marks a test end |
|  | Fail | marks a test failure |
|  | | Any other non-Web 2.0 event imported from a LISA step |

.NET Events:

Each event has many properties attached to it that describe all the information necessary to replay it. We will go into those properties in detail in the next section, as we discuss how to modify these events after a recording.

4.2 Logical Events

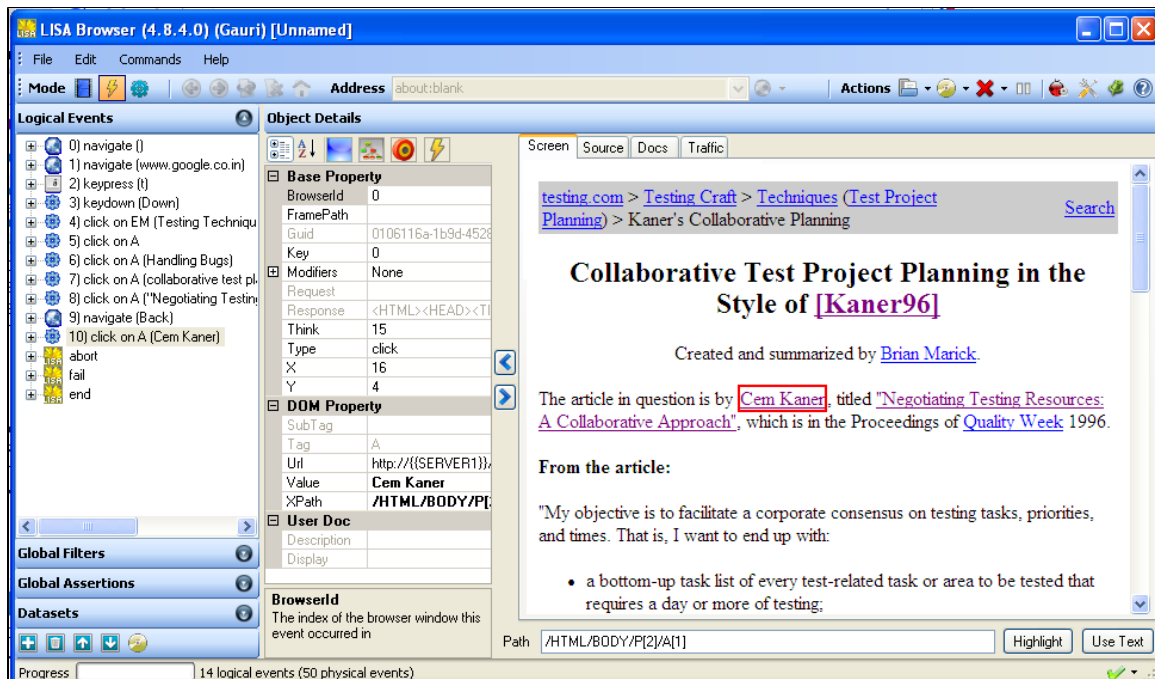
4.2 Logical Events

Once a recording is complete, a LISA test can be generated instantly by **saving** the recording as-is in the Browser window. You can then replay the recording in the Playback mode by reopening the browser window.

However, there are several reasons why it might be desirable **to inspect or modify** events before committing them to a Test Case, or to edit them on an existing Test Case.

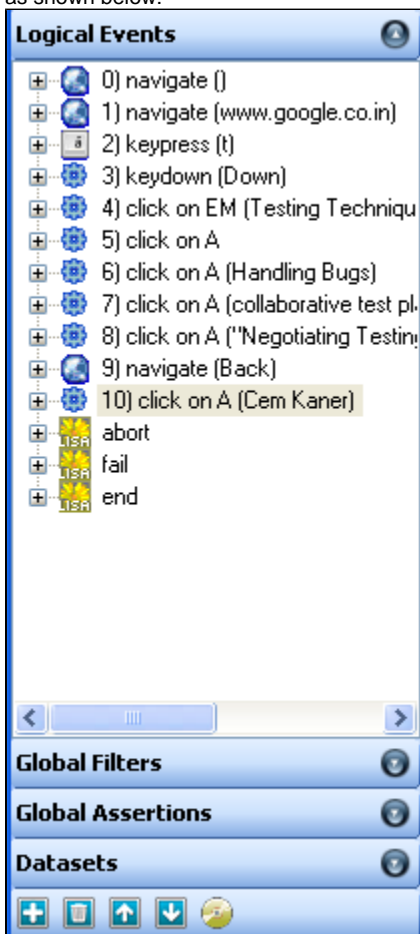
This is the purpose of the Logical Events tab in the browser.

In the Editing mode, you are able to view the logical and physical events of the test case. The Logical Events window in an Editing mode is as seen below:



Logical Events Panel

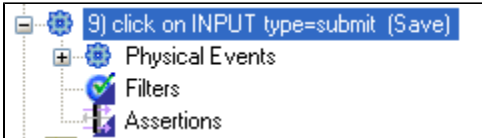
This is on the left of the Browser window. In the **Logical Events panel**, you can see the list of events recorded so far. The logical events panel is as shown below:



To view the Physical events (mouse clicks etc), expand the event by clicking on the "+" sign.

Each of these events is expandable so you can inspect all the physical events in its bucket, all the Filters and all the Assertions attached to that

event as shown below:








You will also see tabs for Global Filters, Assertions and Data Sets at the bottom.



Add/Modify/Delete Events

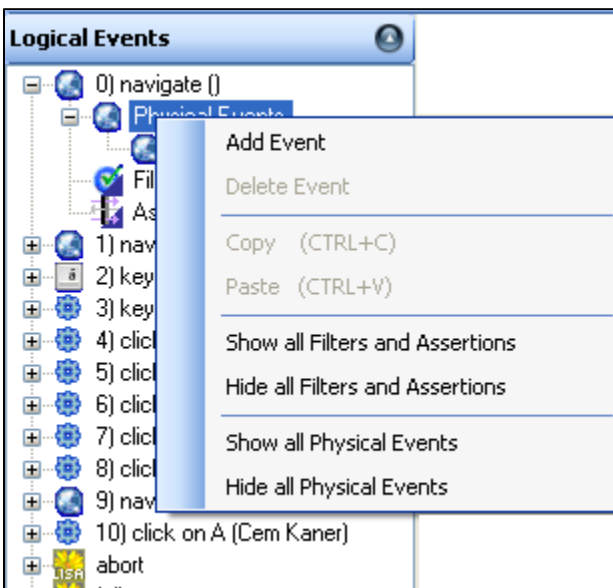
At the bottom, there is a toolbar which allow you to **Add/Delete/Modify an Event/Filter/Assertion** depending on what you select.



| Button | Action | Description |
|---|--------------------------------|---|
|  | Add an event | If a top-level event is selected, a new logical event will be added, otherwise a physical event will be added |
|  | Remove an event | External events can not be removed from the browser, you have to do it in LISA itself |
|  | Save an event | All the changes made to an event in the detail pane (on the right side) are committed |
|  | Move an event up in the list | Use with caution on physical events as results are sometimes hard to predict |
|  | Move an event down in the list | Use with caution on physical events as results are sometimes hard to predict |

In addition to the icons at the bottom of the screen, you can also manipulate events, Filters and Assertions in the left panel:

- Right-click the Step in the Logical Events column to open the screen below which helps to Copy/Paste, Add and Delete an Event



Similarly, you can right click on any filter or assertion to add or delete the same. You can also Physically drag and drop a step from one location to

another.

Configurations

A Configuration is a LISA element that provides a set of properties containing information about the environment or 'system under test'. Configurations can be de-coupled from the Test Case to obtain maximum portability. For detailed information on Configurations, see Using Configurations.

4.3 Object Details

4.3 Object Details

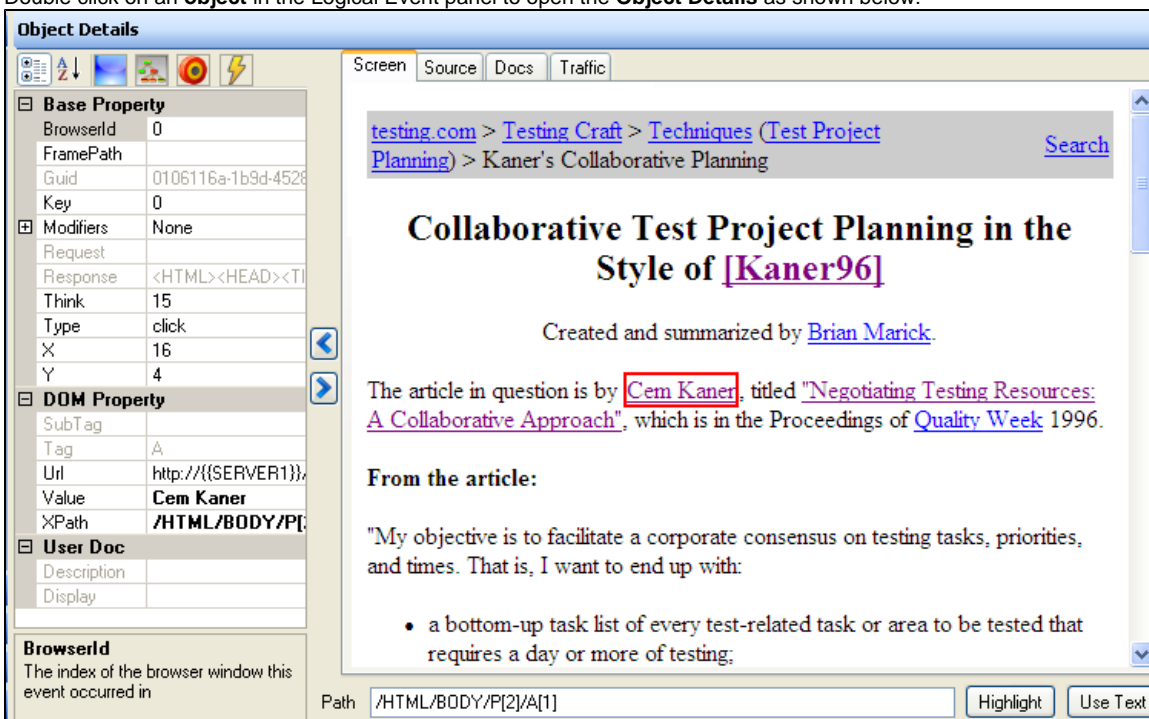
In the Edit mode, you can also see the Object Details Panel. All the details regarding the selected object are listed in this panel.

This panel changes according to the item selected in the left panel.

If you select the Logical events tab, its object details will be seen.

If you select the Global Filters/Assertions/Dataset tabs, their respective editors will be seen.

- Double click on an **object** in the Logical Event panel to open the **Object Details** as shown below:



This panel is again divided into two parts: Left and Right

In the left part, all the properties specific to an object are seen.

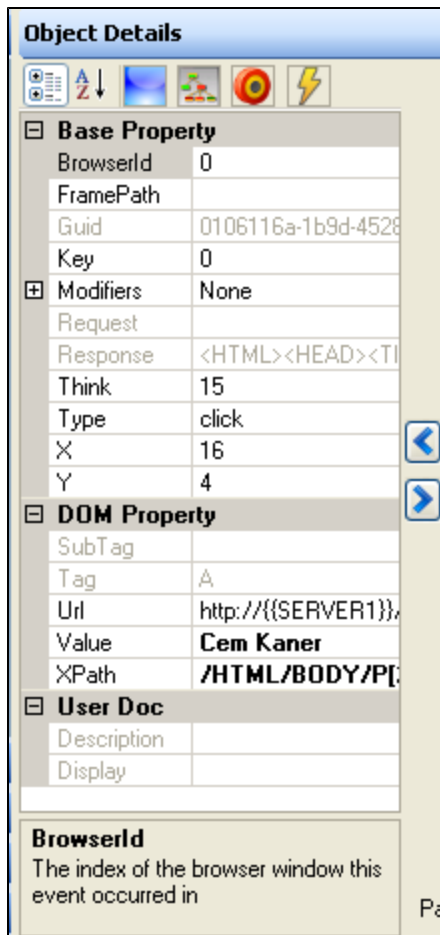
It is divided in 2 sections:

1. General properties (common to all types of objects),
2. Object-Specific properties (i.e. DOM/Applet or Native Properties etc.)

The right part shows the response of the selected object and can be seen only if it is enabled from the  icon.

Let's go over each of these in detail.







Object Details panel



At the top of the Object Details panel, there are buttons which can rearrange the data as required:



Each icon has can be clicked or un-clicked to show or hide data.

- Click to get a categorized view of the data 
- Click to sort the data alphabetically 
- Click to show/hide response pane 
- Click to show/hide hierarchy 
- Click to show/hide Invisible elements 
- Click to enable/disable scripts 

The **Base property tab** is made up of the following:

- **Browser ID:** The index of the browser window this event occurred in
- **Frame Path:** The path to the frame this event occurred in
- **Guid:** The unique event identifier
- **Key:** The keyboard key associated with this event
- **Modifiers:** The mouse or keyboard identifiers in use when this event occurred
- **Request:** The request data associated with this event
- **Response:** The source of the container at the time of this event
- **Think:** The think time of this event
- **Type:** The type of this event
- **X:** The X component of this event relative to its event

- **Y:** The Y component of this event relative to its event

The **DOM property tab** is made up of the following:

- **Sub tag:** The html type attribute of the target element
- **Tag:** The html tag of the target element
- **URL:** The URL of the browser in which the event happened. For Ajax events the URL shown is not the URL shown in the browser, but the URL available to the Server to handle the Ajax calls.
- **Value:** the Value of the DOM element after the event fired, if it makes sense in this context.
- **XPath:** the XPath expression that uniquely identifies the DOM element that was the target of the event.

The **Native properties** are made up of the following:

- **Button:** the label of the button the user used to dismiss the native dialog (such as Open, Save, OK, Cancel, etc.)
- **Selection:** an optional selection value that some dialogs offer (such as a filename)
- **Username:** currently not used
- **Password:** currently not used

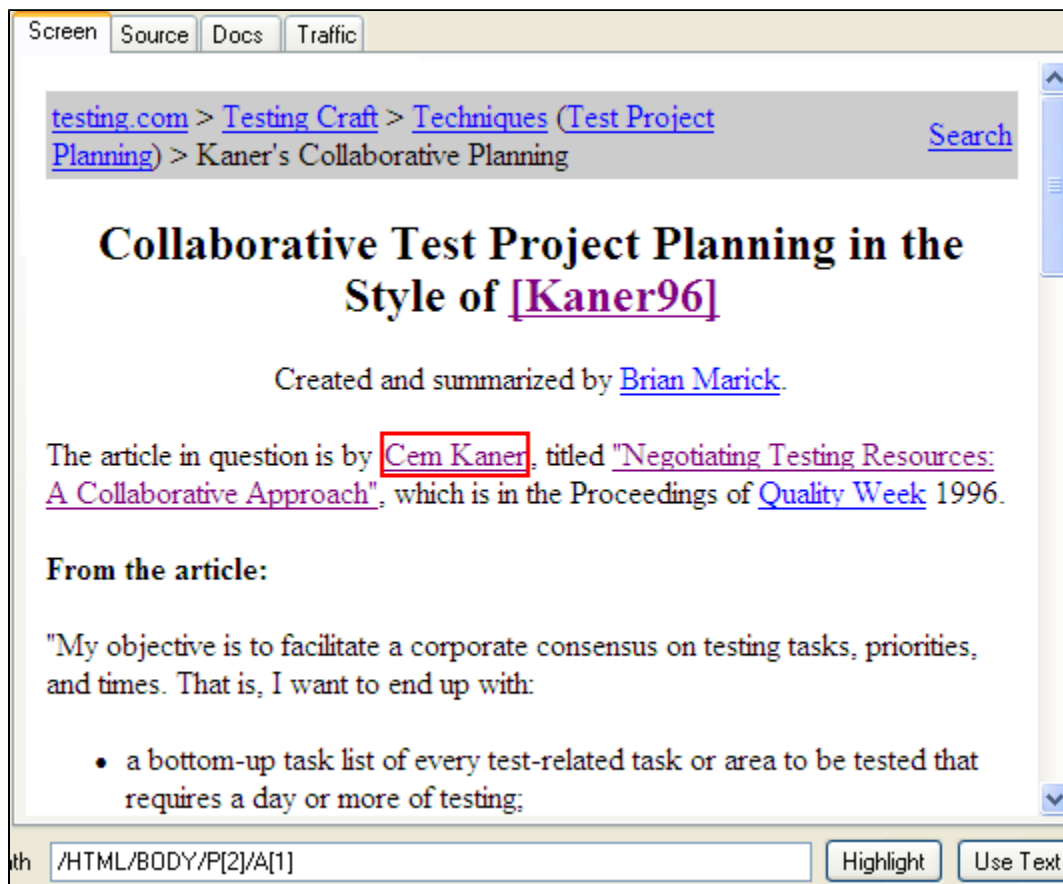
The **Applet properties** are made up of the following:

- **Applet:** the class name of the applet that received the event
- **Path:** the xpath-like expression that uniquely identifies the AWT or Swing component that was the target of the event
- **Class:** the class name of the target component
- **Text:** the text or label of the component after the event was fired

Response Panel

Finally, the Response panel contains the state of the page after the event was fired.

In the case of a DOM event, that translates to an HTML source, in the case of an Applet event, it is an Applet hierarchy. As of now, responses are empty for Native Events.



It has four tabs -

Screen - This shows the actual html page

Source - This shows the html source of the page

Docs - Will show the docs if any

Traffic - This shows the Request and Response headers for the html page.

For an **Ajax request** the Response shows the resulting document that gets loaded.

When you decide to edit an event after a recording is complete, some of the fields are hard to fill correctly, notably the **Path or XPath** fields.

To assist you, there are some Browse Buttons available next to those fields. When you click them, it will bring up a mode dialog window that contains a browser and helps you select those paths.

Below are pictures of these browsers in the case of a DOM event and of an Applet event.

These DOM and Applet browsers have 3 main sections:

- An editable combo box at the top that contains an XPath expression along with a Select and Cancel buttons
- A tree view on the left along with a property grid underneath it
- An actual browser that renders an HTML page or a snapshot of the applet.

You can browse the page or applet by clicking either in the browser or the tree and they will automatically synchronize with each other and the XPath combo box. Typically you would click an element in the browser to generate its xpath, unless it's a hidden element, in which case the tree view is appropriate. When you click an element, either in the tree or the browser, it gets highlighted so you can be certain it is what you think it is. When you are satisfied with the element you chose, you can click the Select button and the Xpath will be transferred to the field that offers the Browse. If you change you mind, Cancel will discard the browser without any changes.

Note that this browser disables all JavaScript or dynamic behavior and can be accessed offline. However it still goes to the Server for css and images if the cache is empty on your disk.

4.4 Filters

4.4 Applying Filters

Authoring or executing a test, steps or events are only half of the equation in LISA.

You need to be able to parameterize inputs and outputs to make the test generic and be able to assert on the results to decide what constitutes success and what constitutes failure.

Filters address the first of these points.

In the context of Web 2.0 tests, you can think of Filters as functions that execute after an event and store the result in a variable that can then later be accessed by other events, Filters or Assertions.

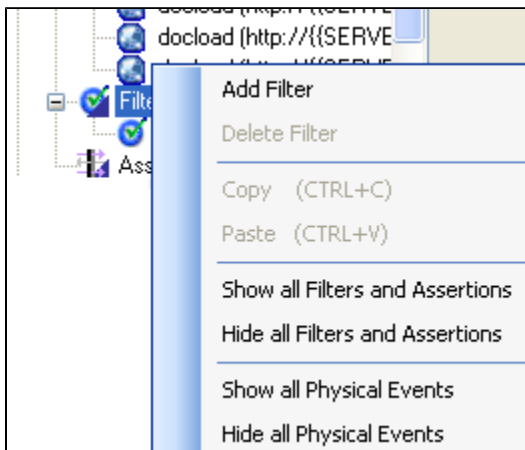
You can inspect, add, remove, or edit Filters in Edit mode of the browser.

These Filters are listed in the Logical Events section. You can see them once you expand the Object tree.

Event Filter

To add an Event Filter

- Select any logical event and click to expand the tree.
- Right click on the Filter node and select **Add Filter** option as shown below:



The filter editor will open in the right panel as shown below:

Object Details

Definition
A filter is a function that executes before or after an event is triggered and stores its result (Filter Value) in a variable (Filter Key).

Filter Key:

☐ Wait up to ms for value

Properties
A text filter retrieves the inner text of a DOM element using a regular expression (first capturing group).

☐ DOM Element

☒ Text

☐ DOM Attribute

☐ Script

☐ Expression

☐ Cache

☐ Capture

☐ Import

☐ Hardware

☐ Code

☐ File

☐ Dataset

☐ Browser ☐ Internet Explorer ☐ Firefox ☐ Safari

☐ Sleep ms

☐ Secure Prompt

Quick Test
Click Evaluate to see what value this filter would return if it were evaluated during the recording.

The Filter object details pane has all the information required to set up a new filter or edit an old one.

After the definition of the Filter, you can see the following:

Filter Key: The key is the name of the variable that is going to receive the Filter result after it executes.

- Or Click on the *Browse* button to open a *Property Browser* as shown.

Property Browser

| Key | Value |
|---------------------------|--------|
| {{instance.name}} | gourik |
| {{instance.number}} | 0 |
| {{event.name}} | |
| {{event.index}} | -1 |
| {{event.type}} | |
| {{event.frame.path}} | |
| {{event.path}} | |
| {{event.response}} | |
| {{event.url}} | |
| {{event.status.code}} | 0 |
| {{event.script.error}} | |
| {{event.bytes.in}} | 0 |
| {{event.bytes.out}} | 0 |
| {{event.bytes.total.in}} | 0 |
| {{event.bytes.total.out}} | 0 |

- Click Select to add.

Filter Properties: the following types are available. Clicking on any of the Filter properties will give its description.

DOM Element: A DOM element Filter extracts an HTML element object from its event's DOM

Text: A text Filter extracts a piece of text from its event's response

DOM Attribute: A DOM attribute Filter extracts an attribute value from its event's response

JavaScript: A JavaScript Filter executes arbitrary JavaScript code in the context of its event's response

Expression: An expression Filter allows you to combine other Filters

Cache: A cache Filter clears the specified browser cache

Capture: A capture Filter saves the current response to the specified location

Import: The import Filter makes the specified file (js or java) available as a library to the running application

Hardware: A hardware Filter executes the selected mouse or keyboard action on the specified desktop

Code: A code Filter executes arbitrary .Net code that has access to all the LISA browser variables

File: A file Filter allows you to execute a variety of operations on file, local or remote

Data Set: A Data Set Filter automatically extracts data out of the specified element into a Data Set according to row and column path rules

Browser: A browser more Filter toggles the state if the playback window to use the selected browsers

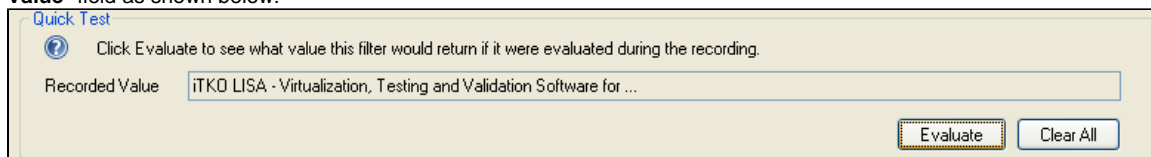
Sleep: A sleep Filter sleeps for a specified amount of time

Secure Prompt: A secure prompt Filter lets you fill in some variable at runtime if you do not want them persisted

Quick Test:

Click to evaluate to see what value this filter will return.

Evaluate: Click on Evaluate to evaluate the filter. The Filter runs with the response it got during recording and displays its result in the "**Recorded Value**" field as shown below:



Quick Test



Click Evaluate to see what value this filter would return if it were evaluated during the recording.

Recorded Value: iTKO LISA - Virtualization, Testing and Validation Software for ...

Evaluate Clear All

Global Filter

To add a global filter,

- Click on the Global Filter button  at the bottom of the Logical Events pane.
- Click the Add button  to add a global filter.

This will open the same filter editor as for event filter.

- Enter the appropriate criterion so that the filter is applied to all the events.

Example

Let's say the page has (and should have) the text "Welcome John" inside some div after a certain DOM event. To extract the value "John" and put it in a variable for later use, you should pick a Text Filter, and a regular expression like "Hello (\w+)". The first capturing group in the regular expression will be used to find John in the page. If you want to be even more specific, or you think there could be more than one match in the page, you could browse for the div that should contain this text and use it as the DOM element.

If you wanted to put the document's title into a variable, you could use a JavaScript Filter with the code: "document.title". If you needed to put the number of rows in a certain table, some JavaScript code like "document.getElementById('mytable').rows.length" would be appropriate.

Let's now say you had 2 tables whose number of rows should always add up to the same value. You could add 2 Filters like the previous one (say "f1" and "f2"), and add another Composite Filter "f3" whose value is "f1 + f2" and use

Once you add some Filters, the resulting variables are available to use almost anywhere else, and the combo boxes for editable fields will then contain those variables.

Finally, there are some intrinsic Filters that are not displayed in the list because they are not editable.

They populate a set of "well-known" variables after each step:

event.type: The last event type
 event.path: The last event path
 event.response: The last event response
 event.raw.response: The last event raw response
 {event.url: The last event URL.

Note: The distinction between response and raw response is that responses may be computed from raw responses and other factors such as previous event responses. In particular most events generate only a raw response that is a diff between various DOM states. The response just

rebuilds the current DOM based on a previous DOM by applying successive diffs.

4.5 Assertions

4.5 Applying Assertions

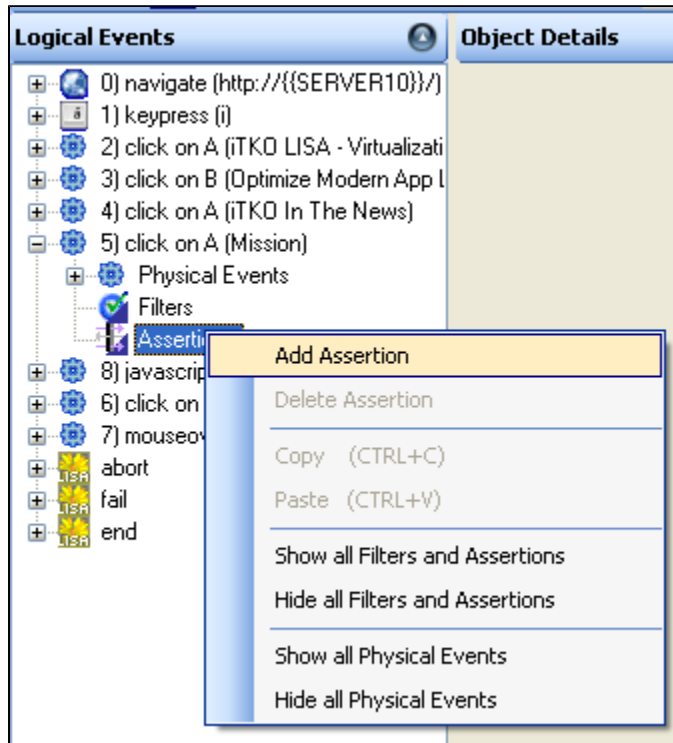
Assertions are also functions that execute after an event is fired, and return a Boolean value. If the return value is true, the test proceeds normally, otherwise what happens depends on the Assertion itself. It is a typical if-then-else scenario.

Practically speaking, you can inspect, add, remove or edit Assertions in the Events tab of the browser.

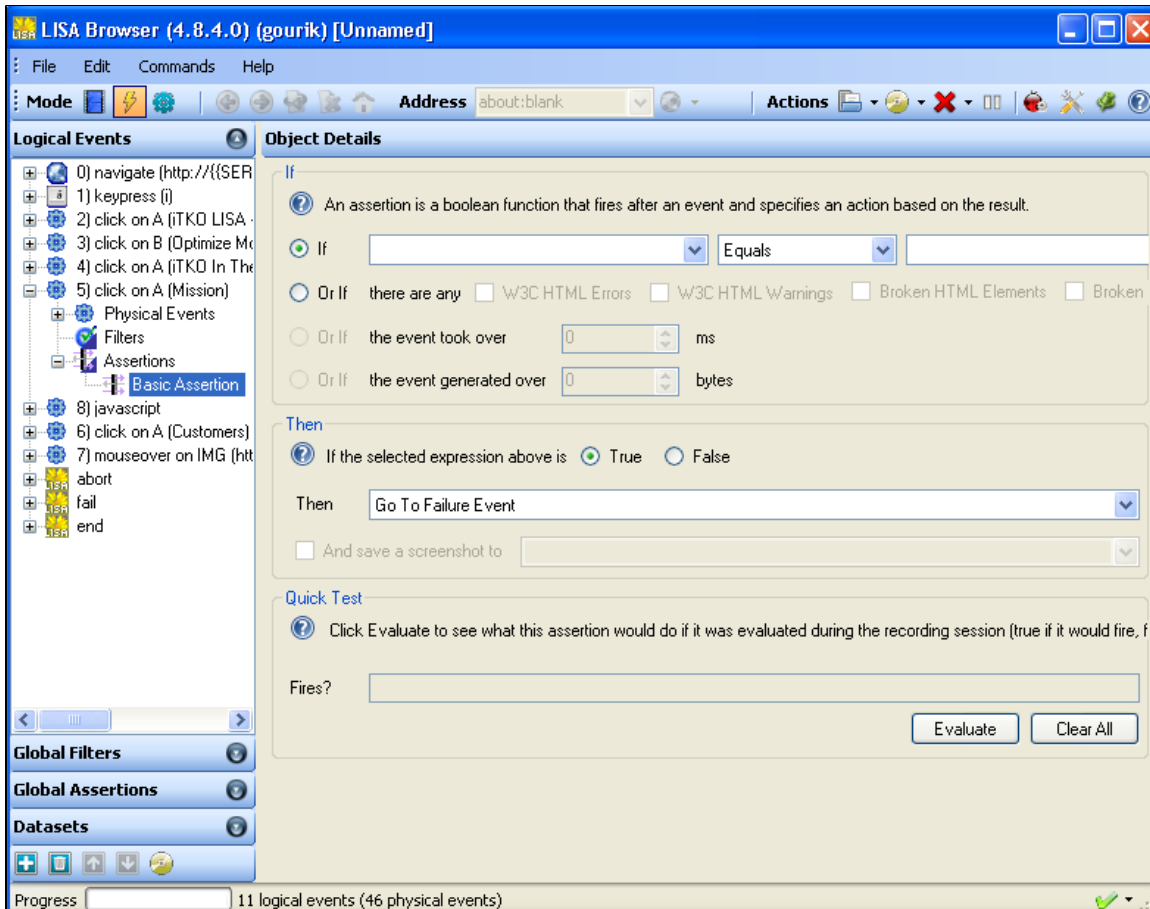
Event Assertion

To add an Event Assertion,

- Click on any logical event to expand the tree.
- Right click the Assertion node and select **Add Assertion** as shown below:



The Assertion Editor will open in the right "Object Details" pane as shown below:



On the Assertion object details pane, after a quick description of the Assertion type that is selected, you can see the following controls.

You need to select two parameters, which are to be compared.

IF - Select the event which you want to compare in the first drop down and select the event, with which you want to compare, in the second drop down.

Select the *type of operation* to be applied in the middle drop down.

Type of operations are -

Equals: those compare the 2 sides of the expression for equality

Not Equals: those compare the 2 sides of the expression for non-equality

Matches: those attempt to match (as regular expression) the left or the right side of the expression against the other side

Not Matches: those attempt not to match (as regular expression) the left or the right side of the expression against the other side

Less Than: those compare the 2 sides of the expression as numeric values for order. Returns false on non-numeric values

More Than: those compare the 2 sides of the expression as numeric values for order. Returns false on non-numeric values

THEN -- Select the step to execute if the expression is TRUE/FALSE

The main point in an Assertion is its expression.

It is this, that determines the success or failure of a test so it is important to be careful in constructing this expression.

There are several types of built-in expressions.

Quick Test



Click to evaluate to see what this expression will do if it was evaluated during the recording session (TRUE if it would fire, FALSE otherwise).

Typically, you will use a variable created by a Filter on the left side and then equal it or match it to a constant value or another variable on the right side. In more advanced cases you can use JavaScript expressions to assert on arbitrary conditions.

Evaluate: Click to evaluate.

Global Assertion

To add a global Assertion,

- Click on the Global Assertion  button at the bottom of the Logical events panel.
- Click the Add button  on the toolbar to add an Assertion.

This will open a editor similar to the Event Assertion.

- Enter the required details to run the assertion on every event.

4.6 Datasets

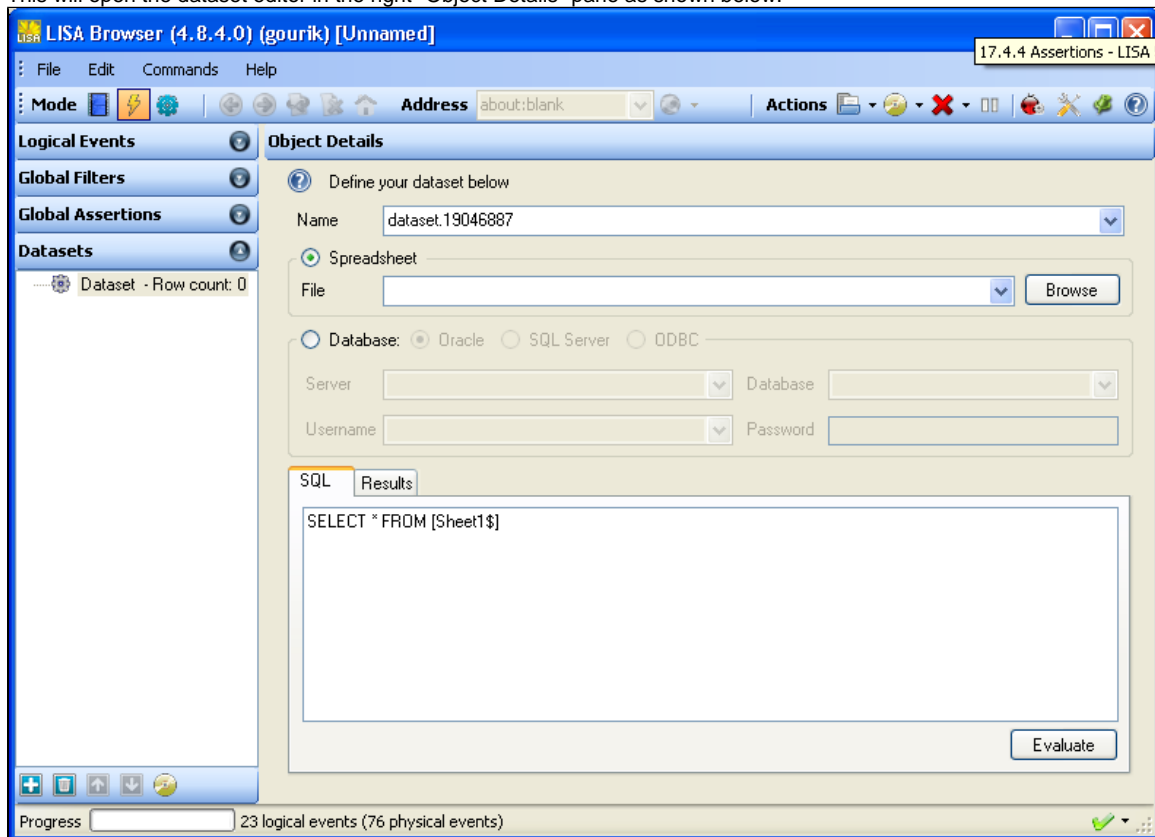
4.6 Applying Datasets

Datasets can only be applied globally, i.e to the entire test cycle.

To add a global dataset,

- Click on the Datasets button  located at the bottom of the Logical events panel.
- Click the Add button  to add a dataset.

This will open the dataset editor in the right "Object Details" pane as shown below:



Define the dataset in the dataset editor.

Name - Enter the appropriate name or accept the default provided by LIS

Data Source - Select the source of the dataset - either from a spreadsheet or from a database.

Spreadsheet - Select this to attach an excel sheet as data input.

- Enter the name of the excel file or browse and select the file.

Database - Select this to select the data source from a database. It will further open the type of databases to choose data from:

- Select one from the Oracle database, SQL database or ODBC database.
- Enter all the details related to the database.
- Enter the SQL query
- Click Evaluate to evaluate the results.

4.7 Editing Steps in Workstation

4.7 Viewing and Editing the Test steps

One of the major strengths of LISA is its outstanding ability to create and run tests that make use of a mix of different technologies (web, j2ee, web services, swing, etc.) as is so often necessary in the enterprise software world.

Web 2.0 tests are no exceptions in this regard and can be mixed with any other type of step. Nothing special is required to achieve this.

To Add steps to an existing test case,

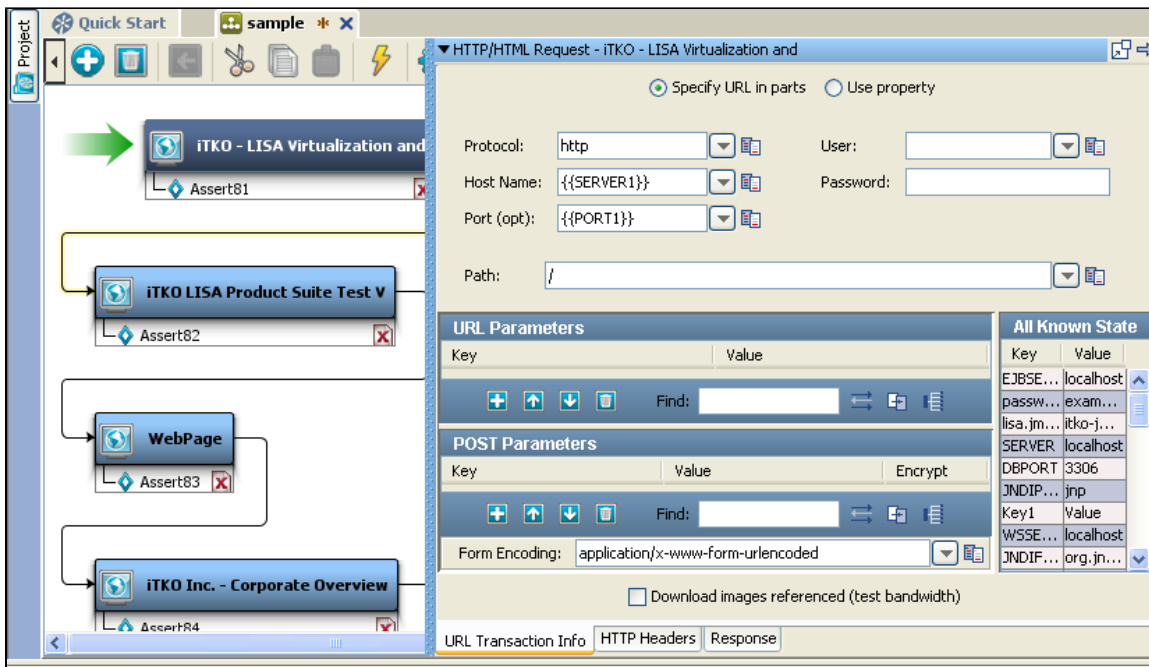
- Open an Existing test case and then start the web 2.0 recording.
- Once done, Save the recording. This will close the web browser.
- This will add all the recorded steps to the already existing test case in the model editor.

To Edit/Add/Delete Web 2.0 steps,

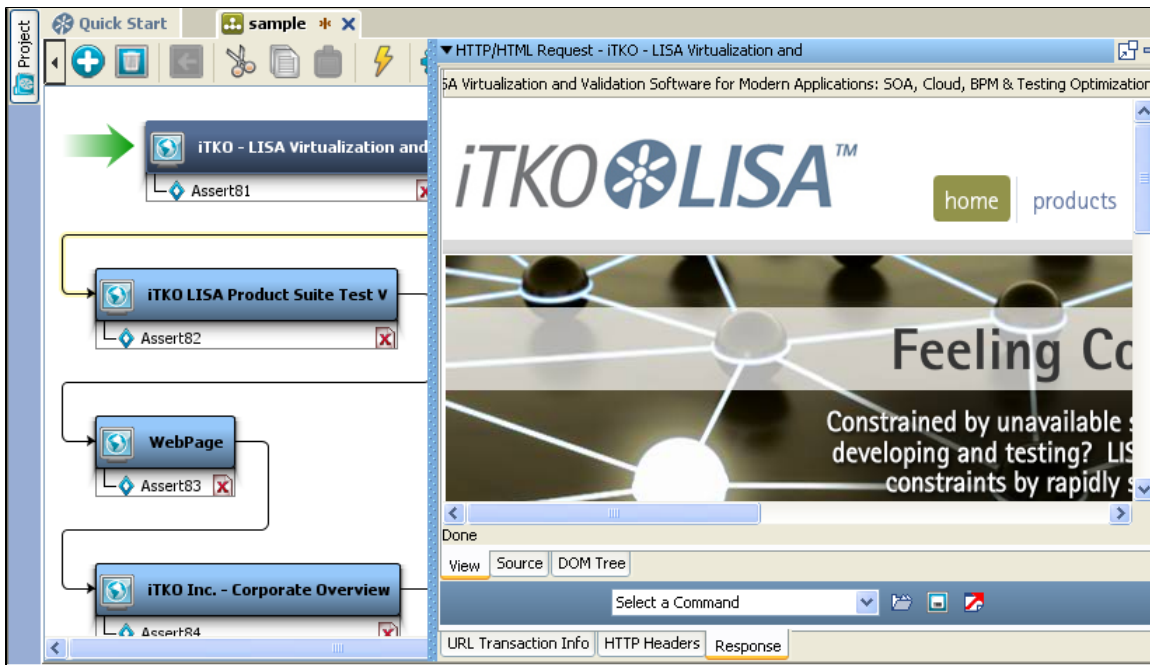
This can be done by requesting a **detailed view** of existing steps.

- Double click the step in the Model editor for which you need a detailed view.
This will launch the editor of the respective test case.

For example, we have chosen to view the detailed view of the HTML type of test step and hence it has opened a **HTTP/HTML Request editor** as seen below:



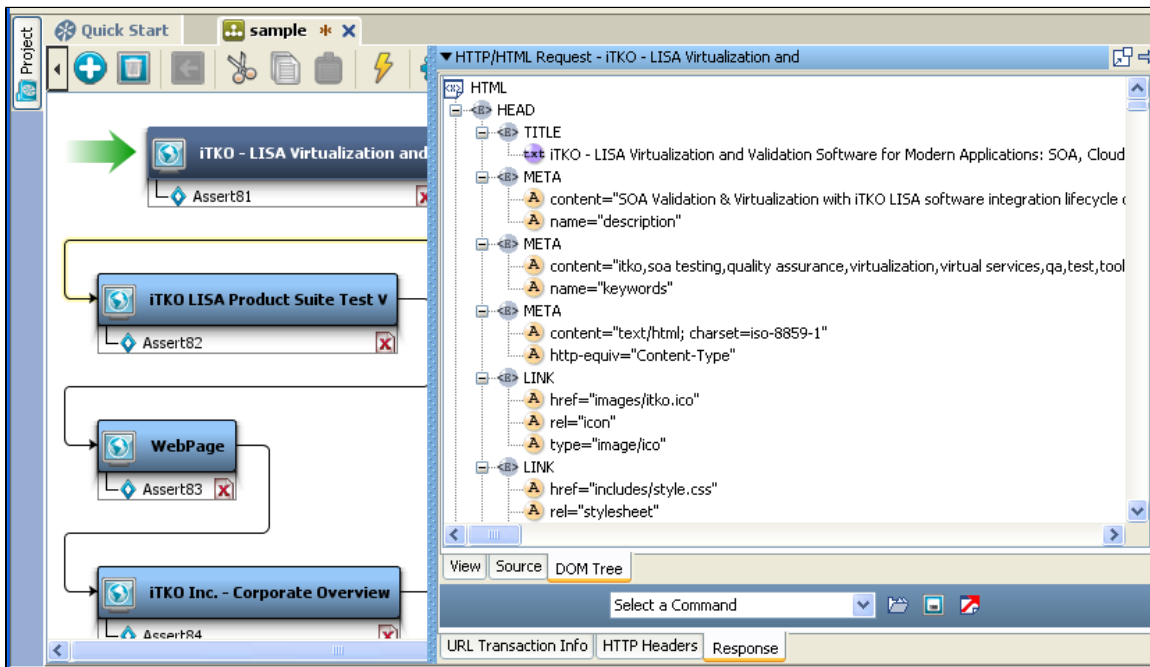
You can also view the response of the Web 2.0 steps exactly the same way you were doing it with regular web steps, by selecting the **Response** Tab in the step's detail pane.



For HTML pages, these responses come in 3 flavors:

- as Headers
- as XML source
- as a DOM tree

Image below shows the DOM tree response:



5. Debugging

5. Debugging

You may need debugging, If you want to debug a code or see the problem area.

Instead of clicking Next repeatedly for every step, you can set one or more breakpoints in the events list and click Play or Replay. Once you have reached the events of interest, you can step through one by one.

If something is not going as you expect during a long test, you can also press **CTRL-C** to stop execution.

Debugging within Browser

In the top toolbar, there is a **Show/Hide Debug Window**  icon.

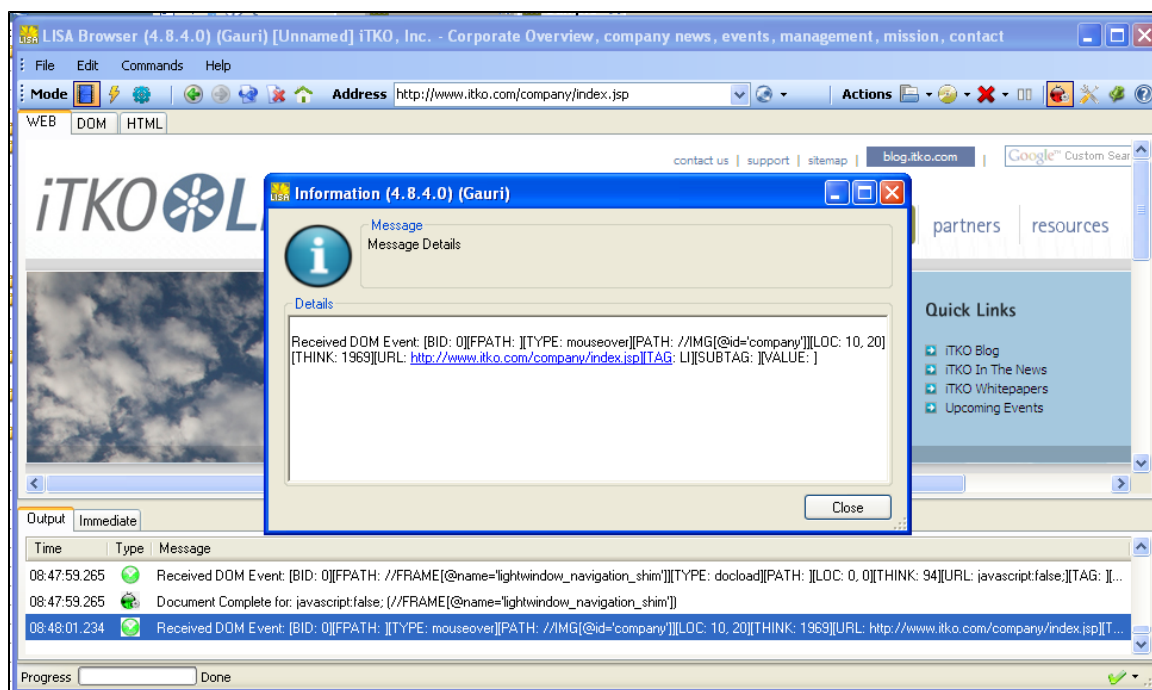
Click to open a debug window at the bottom with a set of tabs.

We have opened the debug window in the Recording mode as below:



The **Output** tab logs information about execution and potential errors or warnings.

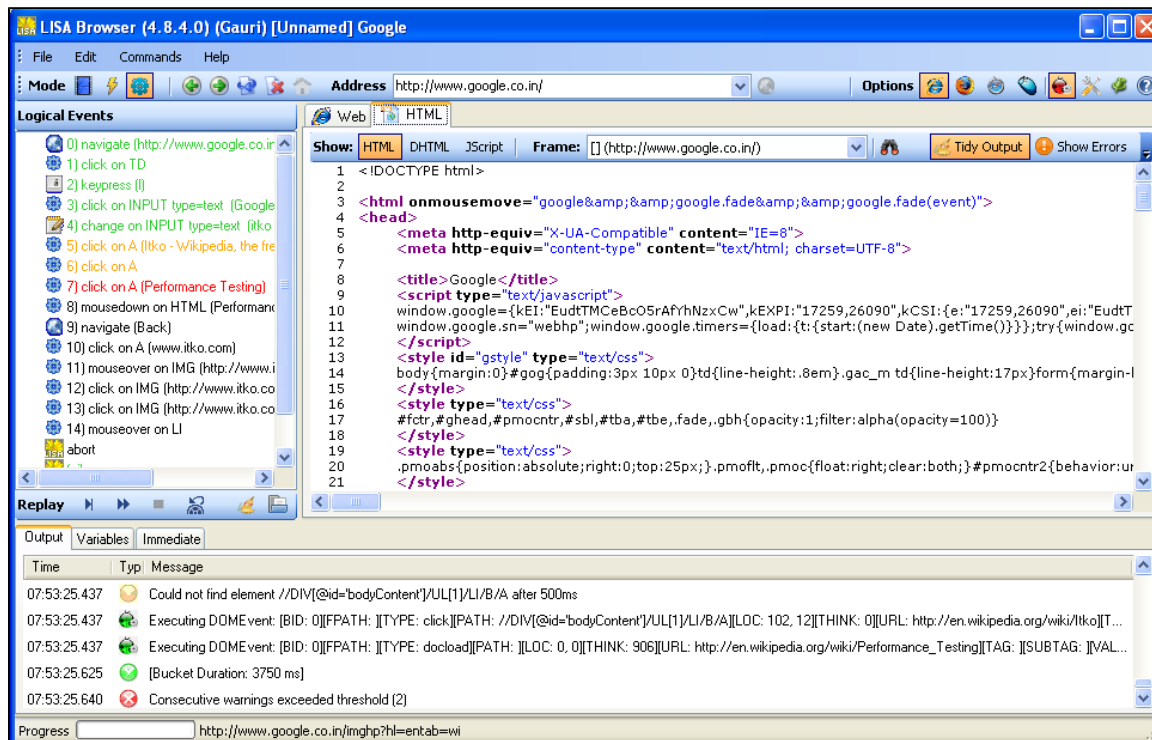
When you double click an **error/message** in the debug window, information about the same is seen.



The **Immediate** tab lets you execute arbitrary JavaScript functions (including using variables).

It is used at design time to debug and evaluate expressions, execute statements, print variable values etc.

We have opened the debug window in the **playback mode** as below:



These tabs are same as in the Recording/Edit mode - with an addition of one Variable tab.

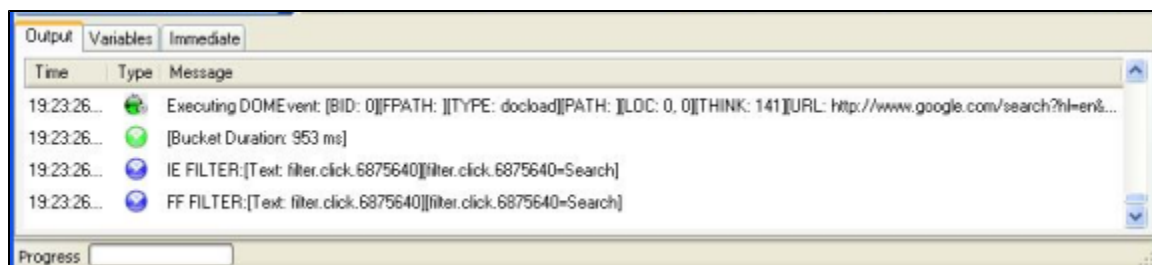
The **Variables** tab shows you all the variable names and values at the current step (the ones highlighted in red are the ones that were potentially modified by the last event)

As a further debugging mechanism, there is an HTML tab at the top that allows you to see the dynamic HTML source of the current page.

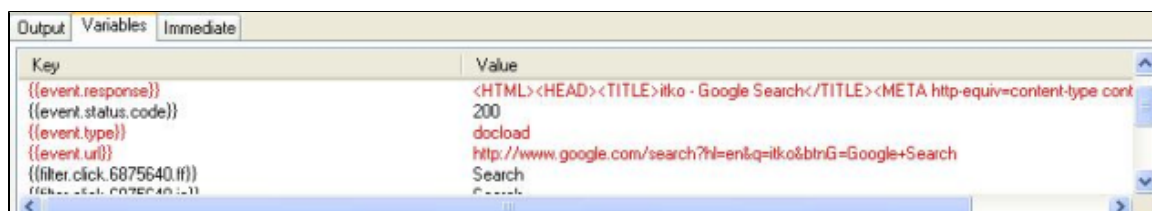
The Split toggle also applies in this source view for further comparison.

Multiple Browsers

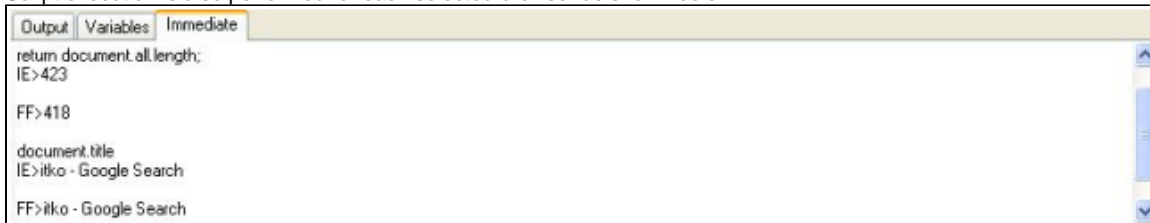
When multiple browsers are selected you get the logging information for each of them as shown below:



Filters also get executed for each browser and the browser abbreviation is appended to the filter key so they can be referenced individually, as shown below:



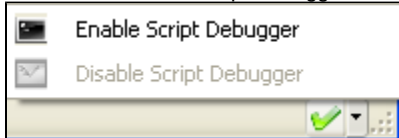
Script execution is also performed for each selected browser as shown below:



Opening a System Debugger

To open a system debugger,

Click on the Enable Script Debugger button at the bottom of the window:



This will open a script debugger if present in the system.

6. Setting up ADF Extensions

6. Setting ADF Extensions

To set up the ADF extensions in the LISA Browser, the browser needs to be updated with the extensions.

To update the extensions,

Open the LISA Browser.

Click on the **Lisa browser -> Help menu -> Extensions Update.**

To update the settings,

Open the LISA Browser.

Click the **LISA Browser > Edit menu > Browser settings.**

This will open the browser settings window.

Click on the **Recording** Tab as shown below:

LISA Settings

General **Recording** Playback Environment

Recording Strategy

All

Ignore ids and names whose value matches:

☐ Externalize recorded text

DOM

Ignore frame names whose value matches:

Ignore text whose value matches:

Use the following attributes (in this order):

| | |
|--|--|
| 1) <input type="text" value="id"/> | 5) <input type="text" value="<none>"/> |
| 2) <input type="text" value="name"/> | 6) <input type="text" value="<none>"/> |
| 3) <input type="text" value="<none>"/> | 7) <input type="text" value="<none>"/> |
| 4) <input type="text" value="<none>"/> | 8) <input type="text" value="index"/> |

Or use the following locator javascript function
☒

☒ Ignore invisible elements

Java

Record using:

| | |
|--|--|
| <input type="checkbox"/> Component names | <input checked="" type="checkbox"/> Deep paths |
| <input checked="" type="checkbox"/> Component text | <input type="checkbox"/> Geometry |

Recording Options

☒ Use context menus for filters and assertions
☐ Write Traffic to Disk
☒ Compress recording files

Capture Level

☐ Capture DOM Level 2
☐ Verbose HTML recording
☐ Ignore HTML responses
☐ Capture HTML changes
☐ Capture Applet snapshots
☐ Capture ActiveX snapshots

Capture Diff Size Bytes
 Capture Max Time ms

Capture DOM Events

| | |
|--|---|
| <input checked="" type="checkbox"/> navigate | <input checked="" type="checkbox"/> docload |
| <input checked="" type="checkbox"/> focus | <input checked="" type="checkbox"/> dblclick |
| <input checked="" type="checkbox"/> mousedown | <input checked="" type="checkbox"/> change |
| <input checked="" type="checkbox"/> mouseup | <input checked="" type="checkbox"/> contextmenu |
| <input checked="" type="checkbox"/> mouseover | <input checked="" type="checkbox"/> drag/drop |
| <input type="checkbox"/> mouseout | <input checked="" type="checkbox"/> mousemove |
| <input checked="" type="checkbox"/> click | <input checked="" type="checkbox"/> keypress |
| <input checked="" type="checkbox"/> open/close | <input type="checkbox"/> ajax callback |

For help click on the ? icon in the top-right corner, then on a setting's label.

Save Cancel

In the "Or use the following locator javascript function" box, click on the check box to enable it.

Enter "\$\$" in the drop down list.

Click **Save** to save these settings.

7. Running Browser Standalone

7. Running Browser Standalone

While taking advantage of the full power of LISA (multi-technology, Data Sets, Companions, etc.) requires running embedded in LISA, doing some quick testing or debugging can be achieved using the LISA browser in standalone mode.

To do this, run the **lisa_browser.exe** executable in the **%LISA_HOME%/bin/browser** directory.


```
lisa_browser.exe -s true [-m <recorder|playback|service>][-f <recording file>]
```

-s or -standalone should be true to run outside of LISA.

-m or -mode can be recorder, playback or service (service allows remote control through a web service interface)

-f or -file optionally specifies a recording file saved previously from the standalone recorder.

8. Troubleshooting

8. Troubleshooting

Here are a few guidelines that will help you and help us assist you in the process of submitting a ticket, from asking a simple question up to reporting a severe bug.

In some cases those tips may dramatically reduce the time to resolution.

- Does the scenario you're testing work in a browser (especially in IE since it is used for recording)? If not there is no chance it will work in the LISA browser.
- Have you read the documentation and made sure your questions are not answered therein? (in particular see [the troubleshooting section](#))?
- Have you familiarized yourself with the different settings in the browser and checked they couldn't address the problem? E.g. Applets not being recorded because applet support was disabled in the settings, or a site that uses mouseover events isn't replaying correctly because mouseover events are unchecked, or timeouts are too low for your applications, etc. (See [Web 2.0 Settings](#))
- Are you sure the test is not replaying correctly because of a lack of parametrization? Specifically, have you looked at the debug output window (or logs) to see what went wrong during replay (e.g. "could not find element "//DIV[@id='gen542348239']" because the id is dynamic)? For example see the How To for [Dynamic Elements](#).
- Are you providing all the information that we're likely to ask for? If the problem triggers an error dialog, you will have an option of "Generating an Error Report", which will contain all this required information (and you can send that to us). Otherwise you can look it up in the Settings dialog (Lisa Browser build number, OS version, IE version, .NET version, JRE version if applicable)
- Can you reproduce the problem consistently? Does it happen only when driven from TestManager or even when using the browser standalone?
- Is there a public URL where you can reproduce the problem? If yes let us know about it. If not, can you package the pages that are giving you trouble and send them to us (usually this can be accomplished by navigating to a browser and going to the File menu and selecting Save As).

If all of this yields nothing useful, we will probably need to contact support@itko.com and have a video conference call (like webex).

NOTE - Please get your system ready to show the wrong behavior to get the correct solutions.

9. Known Limitations

9. Known Issues and Future Work

In no particular order:

- Create an object repository to increase resistance to application change.
- Add the ability to generate test scripts.
- Improve usability, especially around the filters and assertions screens.
- Improve the screenshot algorithm used to capture applets and remote applications images at certain times.
- Ability to use the "Capture HTTP Traffic" mode to replay the tests at the HTTP level if desired (a la web 1.0).
- Full Safari support.
- Various bug fixes

PART 2 - LISA Web 2.0 - How Tos

PART 2 - LISA Web 2.0 - How Tos

1. Introduction
2. Web sites and frameworks
3. How-To: Generate random data (4.5.1.x)
4. How-To: Capture Dynamic HTML for later test editing
5. How-To: Deal with time-sensitive events
6. How-To: Parametrize dynamic data entry in loops
7. How-To: Deal with dynamic elements
8. How-To: Extract complex data from a page
9. How-To: Ajax auto-complete fields
10. How-To: Write custom Web 2.0 steps
11. How-To: Write cross-browser tests
12. How-To: Use Pathfinder integration
13. How-To: Write Java Swing and WebStart tests
14. How-To: Write .NET WinForms tests
15. How-To: Debug a test
16. How-To: Use global filters and global assertions
17. How-To: Interact with external resources
18. How-To: Run Load Tests
19. How-To: Run in a non-privileged account or on 64 bit platforms
20. How-To: Record and replay against non us-english websites
21. How-To: Run in Crash Dump mode

1. Introduction

1. Introduction

This document is a simple list of how-to's that cover typical scenarios and how to deal with them. You should first read [the user guide](#) to familiarize yourself with the basic concepts.

2. Web sites and frameworks

2. Web sites and frameworks

Most web 2.0 sites built today use one or more of many available ajax frameworks, which is where the complexity lies. For a good source of what those frameworks are today, you can consult [ajaxpatterns.org](#). The following frameworks or websites using a framework have been tested during development:

- [Ext](#) (see the demos at [Ext JS](#) and [Ext GWT](#))
- [ICEfaces](#) (see the demos as [Components showcase](#))
- [jQuery](#) (see [Kayak](#))
- [Appcelerator](#) (see the demo at [Appcelerator Web Unit Tests](#))
- [Tibco GI](#) (see the demos at [Xignite](#))
- [Oracle ADF Faces](#)
- [Backbase](#) (see the demos at [Backbase Demos](#))
- [ASP.net Ajax](#) (see the telerik demos at [Telerik](#))
- [Bindows](#) (see the demo on the home page)
- i2 websites (MDM, SCP, etc...) - no public URL available
- and many others...

If there are any websites or frameworks that don't work in the DOM browser, please let me know and I will add support for it. There is nothing that can't be supported (in theory), the DOM browser even works with DOM level 2.

3. How To - Generate random data (4.5.1.x)

3. How To - Generate random data (4.5.1.x)

There are many places in LISA where string generation patterns may be specified. By using a pattern, LISA will create a random string based on the specified pattern during the run of a test model. A string pattern is made up of a mix of pattern and literal characters that will form the final string. The following are the recognized pattern characters:

- D – replace with a random digit (0-9)
- L – replace with a random capital letter
- l – replace with a random lower case letter

A – replace with a random digit or letter of either case
P – replace with a random punctuation character (.,-V)
. – replace with a random printable character

So, for example, the pattern "LDDD" will create a string with a random uppercase letter followed by three random digits. To use a pattern in the LISA browser, the following syntax may be used: =pattern (e.g. =LDDD).

The screenshot shows the LISA browser's Filter Key configuration interface. At the top, there is a 'Filter Key' dropdown menu with 'random' selected and a 'Browse' button. Below this is a 'Wait' checkbox, a 'up to' spinner set to '0', a 'ms for value' dropdown set to 'Equals', and another dropdown. The main section is titled 'Properties' and contains a list of filter types on the left: DOM Element, Text, DOM Attribute, Script, Expression (selected), Cache, Capture, Sleep, and Browser. Each type has a corresponding input field and a 'Browse' button. The 'Expression' field contains the pattern '{{=AAAADDDD}}'. Below the filter types is a 'Sleep' spinner set to '1000' ms and checkboxes for 'Internet Explorer', 'Firefox', and 'Safari'. At the bottom, there is a 'Quick Test' section with a 'Recorded Value' field containing 'STtj0744' and 'Evaluate' and 'Clear All' buttons.

A square bracket expression may be used to randomly select from a specific list. For example, the pattern "[1,2,3]" will create a single character string that is either "1", "2" or "3".

Any character that is not "D", "L", "I", "A", "P", ".", "[", "{", "}" or "*" is taken as a literal. For example, the pattern "BobDDD" will generate 6-character strings, all of which start with "Bob" and end with 3 random digits. If you need one of the "reserved" pattern characters as a literal, prefix it with a back-slash. For example, the pattern "DD\D" will generate strings with two random digits followed by the literal character "D".

With any of the constructs above, there are two types of modifiers that may be specified. A brace expression, "{ }", may be used to specify an exclusion list. For example, the pattern "D{0,2,4,6,8}" may be used to generate a random digit that will only ever be an odd number.

An asterisk expression may be used as shorthand for repeating pattern characters. The asterisk must be followed by one or two numbers surrounded by parentheses. If two numbers are specified, the must be separated by a comma, dash or space.

When the repeater (asterisk) expression specifies only one number, the result of the pattern will be a string of the specified number of characters. For example, the pattern "A*(20)" will generate a 20-character string composed of random alpha-numeric.

When the repeater expression specifies two numbers, the result will be a string that is at least the first number in length but no longer than the second. For example, the pattern "L*(3-5)" will generate strings of random capital letters at least 3 characters, but no more than 5 characters long.

4. How To - Capture Dynamic HTML for later test editing

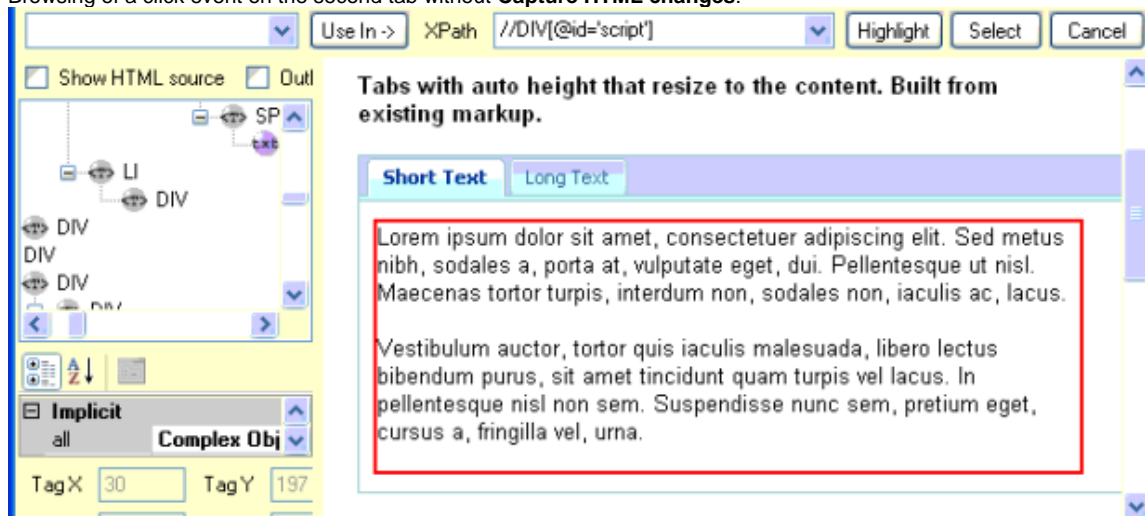
4. How To - Capture Dynamic HTML for later test editing

By default, when you record a test, the response (html document) is captured only when a new document is loaded in the browser, either through a direct navigation or as a result of an ajax load (if ajax callback is checked in the recording settings) so all the events between 2 pages loads will have the same response (as you can see by looking at the events response tab).

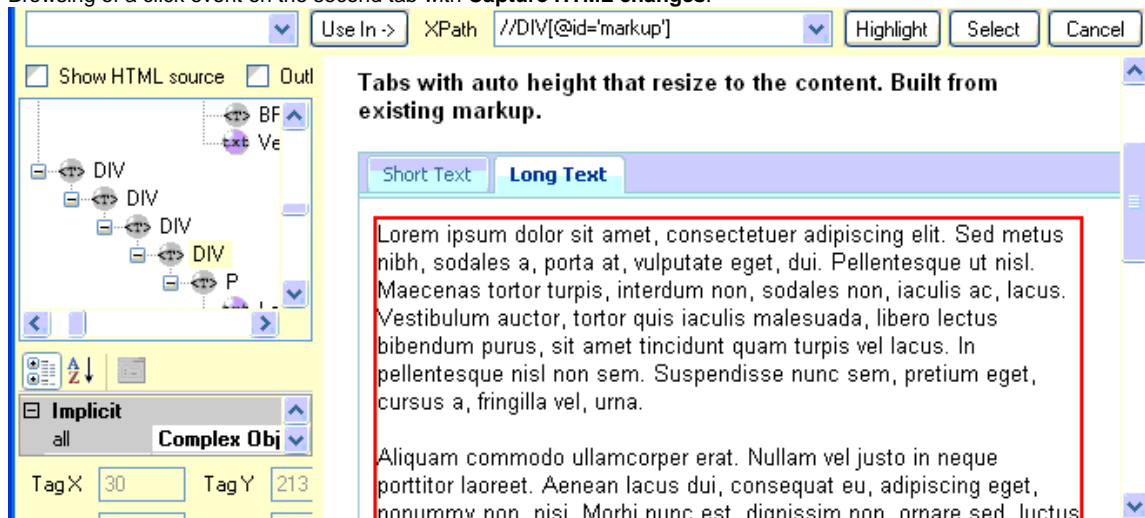
It is normally sufficient and keeps the test size relatively small but for some websites that are very javascript heavy, where the page changes a lot purely on the client as a result of javascript executions, it can make it difficult to edit the test or add filters and assertions through browsing.

For example if you have a page that has tabs (that are preloaded and just change visibility when you click them) and you click on the second tab and want to add a filter for some text on the second tab, if you browse on the events tab you will only ever see the first tab:

Browsing of a click event on the second tab without **Capture HTML changes**:



Browsing of a click event on the second tab with **Capture HTML changes**:



If you add **Capture HTML changes**, what happens is the response is captured on more types of events such as change or clicks, which will make the page as it looked at the time of the event available to you when you browse.

☒ Capture HTML changes

Capture Diff Size Bytes

Capture Max Time ms

Of course if that was the end of the story this would dramatically increase the test size, so to avoid this, there is another setting which is **Capture Diff Size**. What this does is that it compares the size of the change between the page as it is now and the page as it was when it loaded, and if the size of the change is above the number you specify in the settings, it will capture the whole page as it is now, but when the size is below that number it will only keep track of the diff, which is usually quite small and the response for the event is automatically rebuilt for the user from the page load response and the diff.

The reason it doesn't always do the diffs is because above a certain size they become very expensive to compute and would take too long (which is where the second setting, **Capture Max Time** comes in).

The defaults are set to reasonable values and shouldn't need to be changed in general. So if test size is not a concern, it's nice to have this checked to make it really easy to edit the steps/filters later on.

5. How To - Deal with time-sensitive events

5. How To - Deal with time-sensitive events

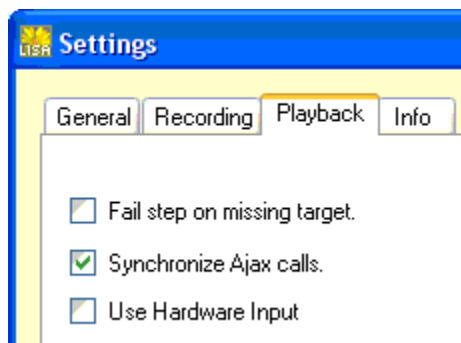
A lot of what makes web 2.0 sites difficult to test is the asynchronous nature of their interactions with the user. In a pure request/response environment, there are no race conditions or multiple threads of execution to worry about (at least from the client's perspective), but web 2.0 environment introduce those difficulties everywhere: frames, set Timeout and particularly asynchronous ajax calls in html pages as well as any java applet code or flash/flex code make heavy use of multiple threads of execution.

The main testability problem it creates is unpredictability. The application may be in a different state as any given event executes during two runs. If you need to capture a value on the page that is updated asynchronously, when do you do the capture? Right after the event, or a fixed amount of time after it? For example, when do you capture the price of a dynamically configured item as in the picture below?



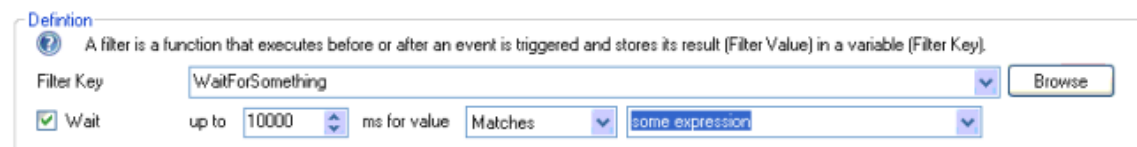
The DOM browser gives you 2 powerful tools to deal with those situations:

- The **Synchronize Ajax calls** setting



This will force all ajax calls made by the application to be executed synchronously so no event, filter, or assertion will execute until the event that triggered the completion returns (note: some websites, that do ajax event pushes through regular polling, are not working well with this setting. If you don't know how the application is coded, just give it a try and see if it works or if it seems to freeze the application from time to time). This is especially useful when there is no visual cue as to when the code completed, as in the example pictured above.

- The **wait for** option of any filter.



This will make the test execution wait until either the wait for condition is met, or the timeout specified expires. You can see an example of how to use this in the [Autocomplete](#) section. Another typical use of this is for pages that have splash screens. For instance, if the splash screen is a div containing the text "Loading..." you would pick a text filter with DOM element the splash div (which you can identify by adding a quick filter while it's shown onscreen for example) add a "wait for" condition with an operator of "Does Not Match" and a value of "Loading...".

6. How To - Parametrize dynamic data entry in loops

6. How To - Parametrize dynamic data entry in loops

To understand this example, you should first be familiar with XPath expressions, as described in [Web 2.0 XPath](#).

Let's look at a couple of typical scenarios to understand how to deal with dynamic data in a loop.

First, let's say you have a list of cities in a table with a checkbox next to them, and you want to check the checkboxes.

| | |
|--|----------------------------------|
| <div><input type="checkbox"/> Austin
<input type="checkbox"/> Dallas
<input type="checkbox"/> Metropolis
<input type="checkbox"/> Gotham</div> | The code for a table row here is |
|--|----------------------------------|

```
<tr>  
  <td><input type=checkbox></td>  
  <td>Metropolis</td>  
</tr>
```

If we want to check them all it is very simple, we record the clicking of just one of them which will result in a change event with an XPath value like `/HTML/BODY/TABLE/TBODY/TR[1]/TD[1]/INPUT`.

So first we define a looping variable. The easiest way to do this is create a filter of type Expression on an event before the change event, let's call it `i` and set its value to 0.

Then we parametrize the XPath of the change event to be something like: `/HTML/BODY/TABLE/TBODY/TR[i]/TD[1]/INPUT`. To increment the value of `i`, we can create another expression filter on the change event with key `i` and value `i + 1`. Each time this filter executes this will increment the value of `i` by 1.

Finally we need to loop onto the change event until `i` is greater than the number of rows (in this example 4, but we could dynamically compute the number of rows first), so we add an assertion on the change event whose expression reads `If i Less Than 5 Then "Go To change event"`. That's it.

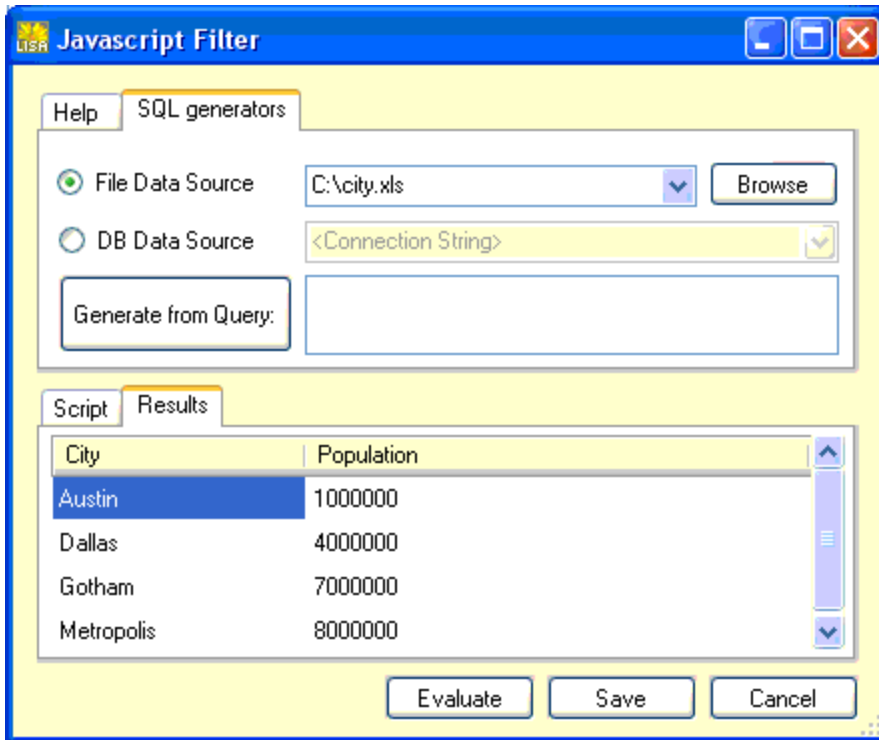
Alternatively, if you are coding inclined, you can simply create a javascript step whose DOM Element is the `/HTML/BODY/TABLE` containing the cities and with the following script: `"for (i=0; i<4; i++) _arg.rows[i].cells(0).childNodes(0).checked=true; return 0;"`.

Ok, now for something a little more complicated, let's say we have a list of cities coming from a Dataset along with population number and we want to input those values on the page:

| <table><thead><tr><th>City</th><th>Population</th></tr></thead><tbody><tr><td><input type="checkbox"/> Austin</td><td><input type="text"/></td></tr><tr><td><input type="checkbox"/> Dallas</td><td><input type="text"/></td></tr><tr><td><input type="checkbox"/> Metropolis</td><td><input type="text"/></td></tr><tr><td><input type="checkbox"/> Gotham</td><td><input type="text"/></td></tr></tbody></table> | City | Population | <input type="checkbox"/> Austin | <input type="text"/> | <input type="checkbox"/> Dallas | <input type="text"/> | <input type="checkbox"/> Metropolis | <input type="text"/> | <input type="checkbox"/> Gotham | <input type="text"/> | The code for a table row here is |
|--|----------------------|------------|---------------------------------|----------------------|---------------------------------|----------------------|-------------------------------------|----------------------|---------------------------------|----------------------|----------------------------------|
| City | Population | | | | | | | | | | |
| <input type="checkbox"/> Austin | <input type="text"/> | | | | | | | | | | |
| <input type="checkbox"/> Dallas | <input type="text"/> | | | | | | | | | | |
| <input type="checkbox"/> Metropolis | <input type="text"/> | | | | | | | | | | |
| <input type="checkbox"/> Gotham | <input type="text"/> | | | | | | | | | | |

```
<tr>  
  <td><input type=checkbox></td>  
  <td>Metropolis</td>  
  <td><input type=text></td>  
</tr>
```

Here the technique is different because instead of iterating based on the elements on the page, we iterate on a dataset and look up the corresponding elements on the page. Let's first create a web 2.0 dataset using a script filter:

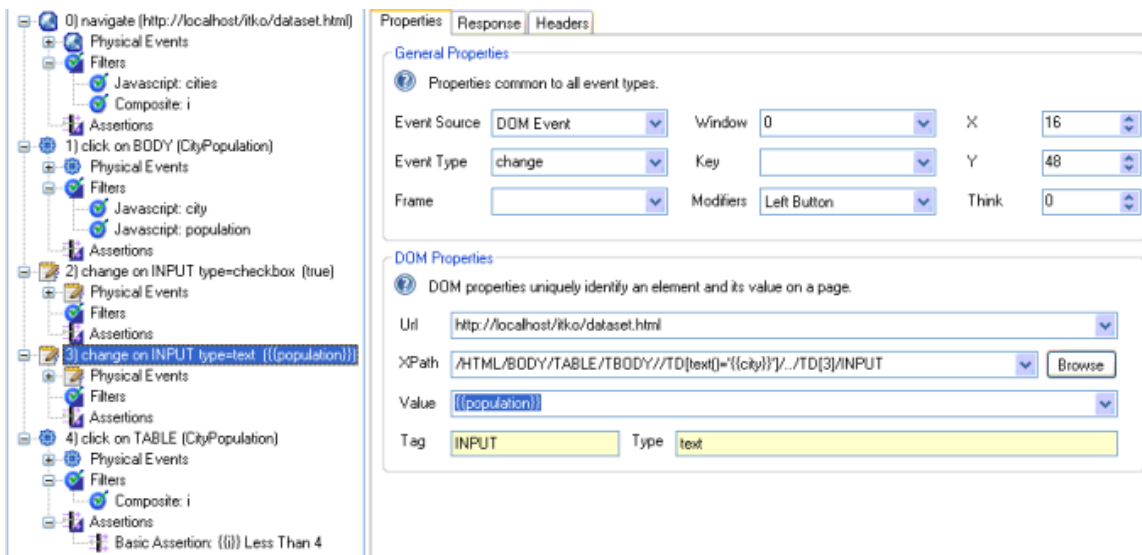


The process is very similar to the one we just followed: first we record checking one checkbox and entering a value in the corresponding text field. It can be any box, the goal is just to generate 2 changes events, one for a checkbox, one for a text field.

Then we create the `i` looping variable before any of the two change events and we parametrize the XPath of those change events. To do this we can no longer rely directly on the index but instead on the values coming from the dataset. To make things clearer, we can define a couple of script filters called `city` and `population` with the following scripts:
`return cities.rows(i).cells(0)` and `return cities.rows(i).cells(1)`.

Now we can use those variables in the XPath on the 2 inputs by first identifying the city name cell: `/HTML/BODY/TABLE/TBODY//TD[text()='city']` and then navigating the DOM from there, which gives us: `/HTML/BODY/TABLE/TBODY//TD[text()='city']/../TD[1]/INPUT` and `/HTML/BODY/TABLE/TBODY//TD[text()='city']/../TD[3]/INPUT`. At the same time, we parametrize the Value of the second change event to use the value coming from the dataset: `population`.

Finally we add the filter that increments `i` and the assertion that loops onto the first change events while there are more rows in the dataset. The screenshot below shows what the test case looks like after adding those filters and assertions.



All of this can also easily be accomplished in a code way as follows:

```
for (i = 0; i < cities.length; i++)
{
```

```

city = cities.rows[i].cells(0);
population = cities.rows[i].cells(1);

lisa.select(document,"/HTML/BODY/TABLE/TBODY//TD[text()='\" + city + "\"]/../TD[1]/INPUT").checked=true;
lisa.select(document,"/HTML/BODY/TABLE/TBODY//TD[text()='\" + city + "\"]/../TD[3]/INPUT").value=population;
}

return 0;

```

7. How To - Deal with dynamic elements

7. How To - Deal with dynamic elements

To understand this example, you should first be familiar with XPath expressions, as described in [Web 2.0 XPath](#).

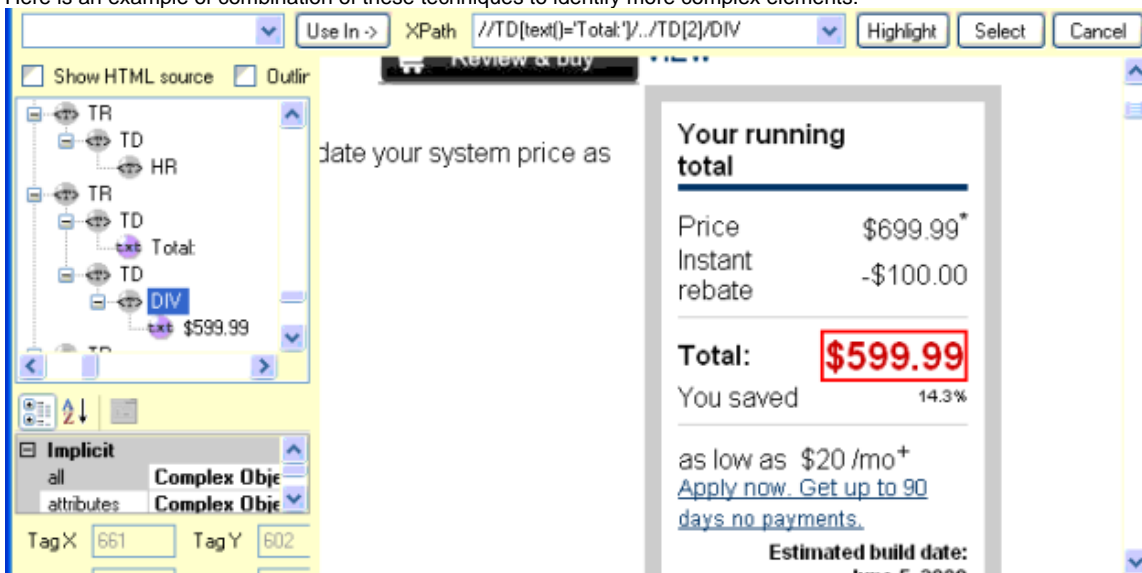
One of the main problems facing recording/replay tools is that of identifying elements on a page or in a control using various properties of the element, such as html attributes, text or value, position in the DOM, position in the page, etc...or any combination of those. Most of these tools boast various algorithms to optimize identification without getting false positives, and the DOM browser is no exception.

The primary means of identifying an html element on the page is through [XPath](#). Those XPath expressions are automatically generated but can also be modified or parametrized to improve reliability in some cases.

In the vast majority of cases, the **id** attribute of html elements is a reliable way to identify them so it will be used whenever possible. However, those ids are sometimes dynamically generated and will change from run to run, sometimes for good reason (as in websites that have dynamic layout), often as a result of poor coding. To avoid having to manually parametrize all these xpaths to make them more reliable, there is a recording setting, **Exclude ids matching**, which will automatically prevent ids whose value matches the supplied regular expression to be used during recording. The rationale is that dynamic ids frequently follow a certain pattern, such as having multiple digits (and the default pattern is indeed 3 digits or more).

Generally, other typical ways to identify elements such as text or value are easily expressible as xpath as well, using the syntax `//TAGNAME[@attributeName='value']` e.g. `//A[@href='www.google.com']`, or `//DIV[text()='Some Value Here']`

Here is an example of combination of these techniques to identify more complex elements:



In the picture above, if the DIV element containing the price has a dynamic id (i.e. it changes from run to run), we don't want to use it. So we Browse and pick the closest element we know doesn't change, in this case the TD whose text is "Total:". Then we navigate up one level using the `..` operator, and then back down a couple of levels using the DOM tree as a guide. This gives us the following xpath: `//TD[text()='Total:']/../TD[2]/DIV`. The Highlight button confirms our guess.

8. How To - Extract complex data from a page

8. How To - Extract complex data from a page

To understand this example, you should first be familiar with XPath expressions, as described in [Web 2.0 XPath](#).

Most interesting assertions rely on data that has been extracted from pages through the use of filters. The vast majority of cases will require no

configuration and will automatically pick up the data you want:

Element filters will select the desired element (to verify it exists for example) and fill the filter value with its xpath. The element will be already chosen for you if you use a filter add during recording (Add Quick Filter or Add and Edit Filter). If you do it post recording, the browse button will easily let you do it visually.

Text and Attribute filters work the same way, there is no work required other than deciding on a regular expression for Text filters (which 95% of the time will be the default (.*) to capture the whole inner text), and as for attributes, the list of available attributes will be prepopulated in the drop-down after an element browse is completed.

Let's take a simple example: we add a user to a list of users and it shows up in a table with ID, Name and Email. We want to make sure that the user has in fact been added and that the email field value is what we think it is:

Wednesday, May 28, 2008

Welcome iTKO !

Add User

Modify User

Delete User

List Users

View Users

| UserID | Name | Email |
|-----------------------------------|-----------------------------|----------------------|
| userC0A80AC8000.. | userC0A80AC80000011A5622E.. | |
| multitier-43784.. | null null | null |
| multitier-64966.. | null null | null |
| User | null null | null |
| multitier-10045.. | null null | null |
| Value | null null | null |
| userC0A80078000.. | null null | null |
| userC0A80078000.. | null null | null |

We can first add a DOM filter, and browse for the row we want in the table. We'll get something like:

`//FORM[@name='edit_users']/TABLE/TBODY/TR[493]/TD[1]/A.`

Use In -> XPath `//FORM[@name='edit_users']/TABLE/TBODY/TR[493]/TD[1]/A` Highlight Select Cancel

Outline Tags

| | | |
|-----------------------------------|-----------------------------|---------------------------|
| hello | null null | null |
| test | null null | null |
| test2 | null null | null |
| idid | jd jd | id@id.com |
| user62326161616.. | user62326161616633312D666.. | |

Of course, this is not reliable because the row number (493 in this instance) could change, so we need to parametrize this expression to use the user name instead. We keep as much of the computed expression as possible, in this case

`//FORM[@name='edit_users']/TABLE/TBODY`

and then we look for the A element whose text is the user name, something like

`A[text()='jdjd']`.

This gives us the following expression: `//FORM[@name='edit_users']/TABLE/TBODY//A[text()='jdjd']`. Note the double // before the A to specify we search children of the TBODY down to any level.

All we have to do now is add an assertion that verifies the value of this filter is not blank, which will indicate the presence of this row in the table. In the same way, to capture the email of this user, we use a text filter where we start with the same expression:

`//FORM[@name='edit_users']/TABLE/TBODY/TR[493]/TD[1]/A`

then we go up two levels to the table row:

`//FORM[@name='edit_users']/TABLE/TBODY/TR[493]/TD[1]/..`

then we pick the anchor in the third cell of this row:

`//FORM[@name='edit_users']/TABLE/TBODY/TR[493]/TD[1]/../TD[3]/A`

At any time we can use the Highlight button to verify we're selecting the right element.

Use In -> XPath `//FORM[@name='edit_users']/TABLE/TBODY//A[text()='jdjd']/../TD[3]/A` Highlight Select Cancel

Outline Tags

| | | |
|-----------------------------------|-----------------------------|---------------------------|
| test | null null | null |
| test2 | null null | null |
| idid | jd jd | id@id.com |
| user62326161616.. | user62326161616633312D666.. | |
| user7F000101000.. | user7F0001010000011A2D5E4.. | |

Now we just have to add an assertion to make sure the value of this filter is indeed the email we think. Of course, all of these values, such as user name, email etc...can be parametrized in the usual way `username`, `email`, etc...

For the cases when you need finer-grained control over the retrieved data, you can use the Script Filters. They give you full control over the DOM (or the applet hierarchy when testing applets). As an illustration, let's reuse the previous example, where we want to extract data from the table.

The first thing you would do in a filter is select the DOM element to be the table (by browsing as usual). This gives you access to the table object in script as the `_arg` variable (see the [_arg](#) scripting object). Then you can easily retrieve all kinds of information:

| | |
|---------------------------------------|--|
| Number of rows: | <code>return _arg.rows.length</code> |
| Text of the 3rd cell of the 12th row: | <code>return _arg.rows(11).cells(2).innerText</code> |
| Text of the last row: | <code>return _arg.rows(_arg.rows.length - 1).innerText</code> |
| Email of the user jdjd: | <code>for (i=0;i<_arg.rows.length;i++) {if (_arg.rows[i].cells(0).innerText=='jdjd') return _arg.rows[i].cells(2).innerText; return null;}</code> |
| Etc... | |

Properties
 A script filter executes a javascript function (that must return a value) on the HTML document of the event response.

☐ DOM Element

☐ Text

☐ DOM Attribute

☒ Script

☐ Expression

☐ Cache

Quick Test
 Click Evaluate to see what value this filter would return if it were evaluated during the recording.

Recorded Value

Finally, some dynamic data is automatically extracted for you, most notably request string parameters and cookies:

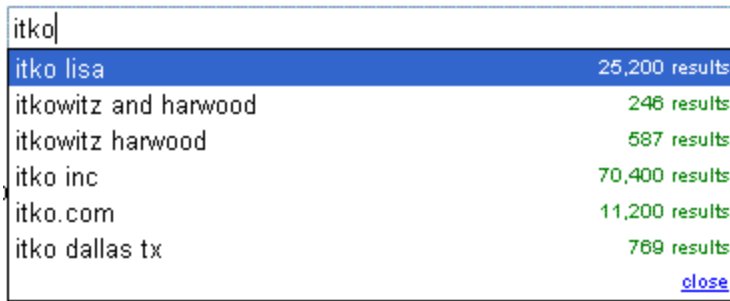
| Key | Value |
|-----------------------------------|---|
| {{event.cookie.NID}} | 15 |
| {{event.cookie.PREF}} | ID |
| {{event.frame.path}} | |
| {{event.param.aq}} | f |
| {{event.param.hl}} | en |
| {{event.param.oq}} | itko |
| {{event.param.q}} | itko |
| {{event.path}} | //DIV[@id='rads']/OL/LI |
| {{event.response.render.time.ie}} | 47 |
| {{event.response.render.time}} | 47 |
| {{event.response.time}} | 15 |
| {{event.response}} | <HTML><HEAD><TITLE>itko - Google Search</TITLE><META http-equiv=content-type content="tex |
| {{event.script.error}} | |
| {{event.status.code}} | 200 |
| {{event.type}} | mouseover |
| {{event.url}} | http://www.google.com/search?hl=en&q=itko&aq=f&oq= |

As shown in the picture above, string parameter names are prefixed with `event.param` and cookie names are prefixed with `event.cookie`.

9. How To - Ajax auto-complete fields

9. How To - Ajax auto-complete fields

One of the most pervasive ajax controls is the so-called auto-complete (or type-ahead) control, which prompts the user with a list of choices even as they are typing letters in a field.



To record and replay this type of interaction, there are a couple of things to be aware of.

First, the keypress event should be enabled for recording (as it is by default) in the settings. In most cases it can be disabled and it leads to leaner test cases but in this case it is needed.

Then, when you replay the test, the drop-down is populated asynchronously, so if you take no precautions, you may execute the next event or a filter, or an assertion before the drop-down is populated. There are 3 ways to deal with that, as explained in the [Time Sensitive](#) section:

- Using the **Synchronize Ajax calls** setting.
- The next way that is very simple is to add a filter that waits for a certain amount of time by specifying a never met condition, thus giving the call enough time to complete.

Definition

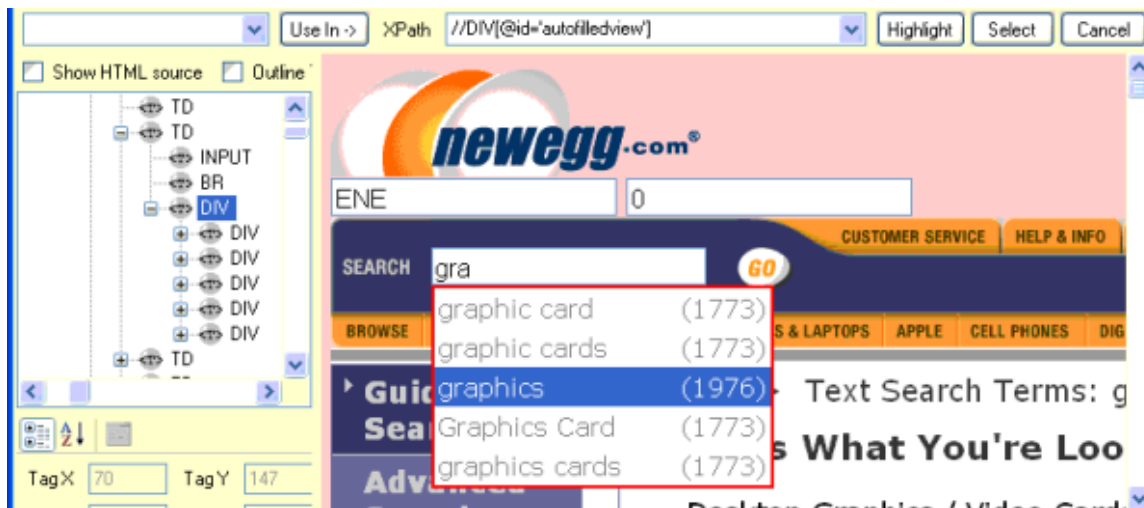
A filter is a function that executes before or after an event is triggered and stores its result (Filter Value) in a variable (Filter Key).

Filter Key:

☒ Wait up to ms for value

- The last way, which will always work but requires a bit more work is add a filter that waits for a specific condition to be fulfilled. In this case, the condition would be that the drop-down is visible and shows a certain number of rows for instance.

Typically, you would add a script filter on the event that triggered the drop-down, then browse for the DOM element and pick the drop-down popup, as illustrated in the picture below (from the search on www.newegg.com).



Then you would specify the script snippet to return something meaningful for this site. For instance, in this case, you can see from the picture above that the drop down is a div tag and each row is made up of a child div tag (easy to see from the DOM tree). So a meaningful filter script might return the number of rows: `return _arg.getElementsByTagName("DIV").length` (see the [_arg](#) scripting object).

Finally, since you expect the filter value to be 5 in this instance, you would specify in the Wait value box of the filter, and the filter screen would look like this:

Definition

☐ A filter is a function that executes before or after an event is triggered and stores its result (Filter Value) in a variable (Filter Key).

Filter Key:

☒ Wait up to ms for value

Properties

☐ A script filter executes a javascript function (that must return a value) on the HTML document of the event response.

☐ DOM Element:

☐ Text:

☐ DOM Attribute:

☒ Script:

☐ Expression:

☐ Cache:

Quick Test

☐ Click Evaluate to see what value this filter would return if it were evaluated during the recording.

Recorded Value:

As you can see from the picture, the Quick Test can be used to tell you if the script you're using is correct and returns the desired result. Also, note that to see the drop down visually in the Browse window, as is pictured above, you will need to enable the **Capture HTML changes** setting in the recording settings as explained in the [Capture section](#).

You are now ready to add more filters for data extraction and assertions for validations. This is more thoroughly covered in the [Data extraction](#) section but in this instance, a very simple filter might be a Text filter with a DOM element set to be the drop down element, which will retrieve all the text contained in the popup:

Properties

☐ A text filter retrieves the inner text of a DOM element using a regular expression (first capturing group).

☐ DOM Element:

☒ Text:

☐ DOM Attribute:

☐ Script:

☐ Expression:

☐ Cache:

Quick Test

☐ Click Evaluate to see what value this filter would return if it were evaluated during the recording.

Recorded Value:

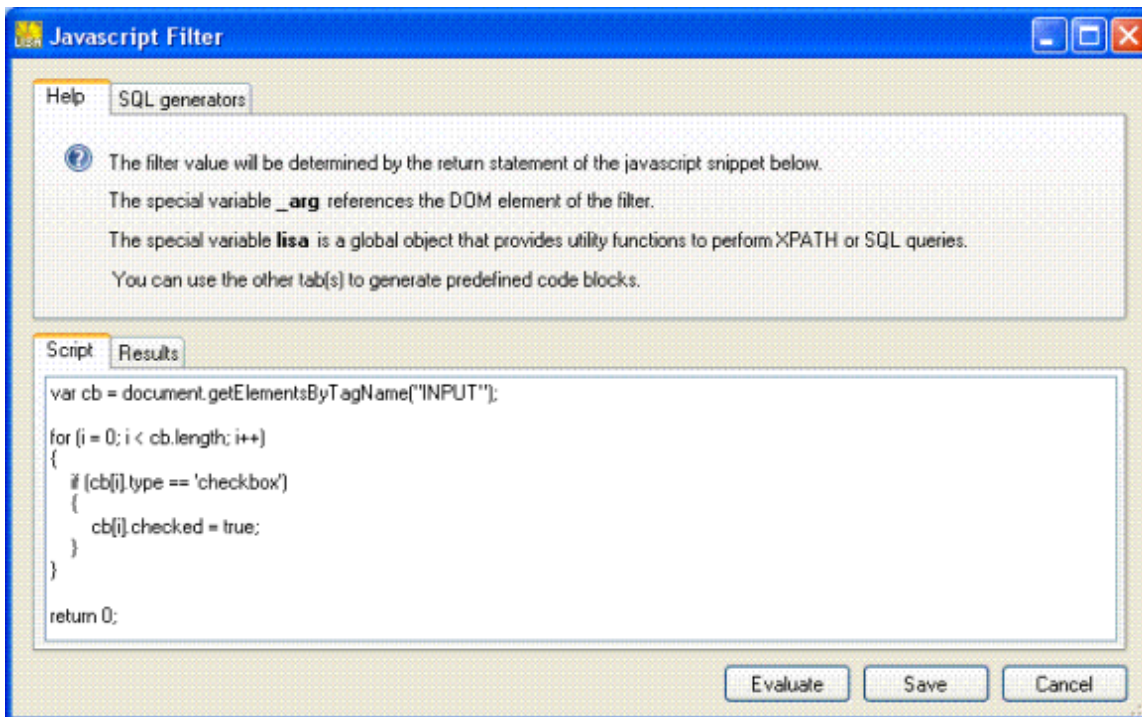
Then all that is left to do is add an assertion that checks some given text is contained in the value of this filter (using a Basic Match assertion).

10. How To - Write custom Web 2.0 steps

10. How To - Write custom Web 2.0 steps

Many record/replay tools let you (or even require you to, if there is no recorder) write scripts to control user actions. There should generally be no need to do this as the recorder is good at capturing all user interactions, but it may still be useful in certain cases.

For instance, if you have a page with many checkboxes that you want to check, the GUI way to do this would be to check one, and then to parametrize it so that it can be embedded in a loop that will visit all the checkboxes with the change event (see [the parametrization](#) section). If you have 2 levels of loops or other constraints it will be quite tedious, but it could be quite easy to do in a script.



When you create a new step in the Events tab, a new event of type script gets created. It does nothing by itself but contains a script filter that can execute arbitrary javascript (or java for applets). The image above shows the script dialog that gets opened when you click the Browse button of a script filter. In this instance, it iterates over all checkboxes on the page and checks them.

11. How To - Write cross-browser tests

11. How To - Write cross-browser tests

The first thing to know about writing tests that run in multiple browsers is that there is nothing special to know. You would author, record and edit the test in the exact same way you normally would a test designed to run in a single browser, and during playback you simply select which browser(s) you want to run the test.

In the screenshot below, note how the Internet Explorer and Firefox buttons are pressed in the toolbar. This will cause both browsers to be used for this test. You can select Internet Explorer and/or Firefox and/or Safari to run a test.



While Internet Explorer is always installed on Windows, other browsers require an initial download. Every time you attempt to use Firefox or Safari, if those browsers have not been installed you will get the following prompts:



There is no difference in how events are executed in multi-browser mode, they are just executed for each browser. Assertions are just executed once since they don't depend on the browser, the only thing to pay attention to is filters because they highly depend on the browser.

First off, we can not execute filters for each browser without any changes because if we did, the filter value coming from one browser would overwrite the filter value coming from an other one since they have the same key.

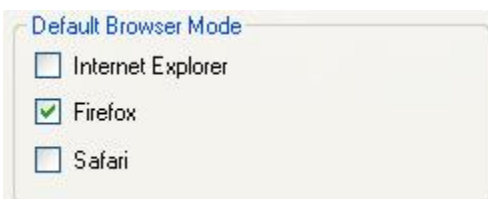
This is why when multiple browsers are used, the keys used in filters are still in use but there are automatically new filter keys being generated, one for each browser, to allow us to distinguish between the filter values coming from different browsers. The convention is to append 2 letters to the filter key: **key.ie** (for Internet Explorer) or **key.ff** (for Firefox) or **key.wk** (for Safari).

The screenshot below shows what happens when we define a javascript filter called allcount with code: "return document.all.length;"

| Key | Value |
|-----------------------|---|
| {{allcount.ff}} | 93 |
| {{allcount.ie}} | 99 |
| {{allcount}} | 93 |
| {{event.path}} | /INPUT[@name='q'] |
| {{event.response}} | <HTML><HEAD><TITLE>Google</TITLE><META http-equiv=content-type content='text/html |
| {{event.status.code}} | 200 |
| {{event.type}} | focusin |

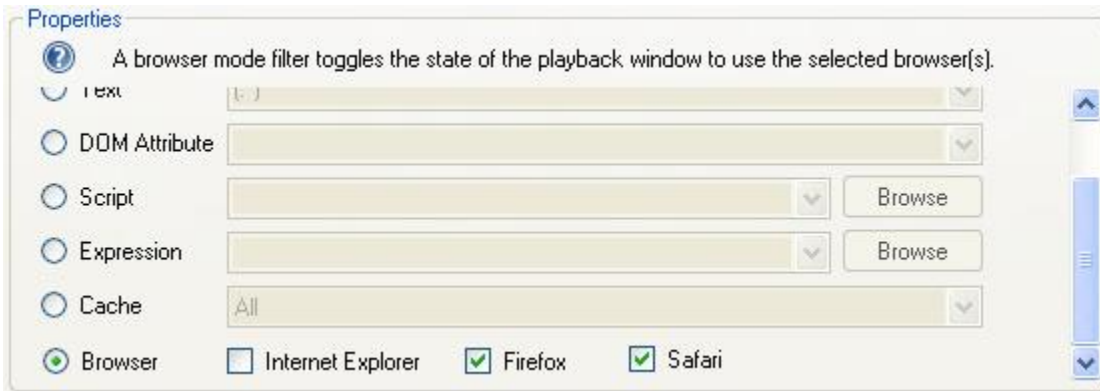
Since the different browser evaluate this javascript expression differently you see 3 values for the allcount variable: one for IE, one for Firefox, and one default one (IE in this instance) so as to not break assertions written for single browser mode. Now we can write assertions that compare allcount.ie and allcount.ff for example. This is particularly useful to compare rendering page properties, by writing filters that return object positions in the DOM and comparing their values in different browsers.

Finally we need to be able to automatically control which browser(s) is (are) selected in general, or for a given test, or even for a given step. There are 3 ways do do that, one for each level of granularity. To control globally which browser(s) is (are) used by default, the "Default Browser Mode" section in the settings should be used.



At the test level it can be overridden using the usual mechanism of defining the following property: `DEFAULT_BROWSER`. It can take the value `NONE`, `IE`, `FF`, `WK` or any pipe-delimited combination of those (like `IE|FF`).

Finally, at the event level, which browser is being used is controlled by the Browser filter:



When a browser filter executes, it selects the browsers specified in the filter to run from that point on. Typically you would use this filter just prior to looping in a test case, so you could run a step of steps in a given browser and then again the same set of steps in an other browser to compare the results.

12. How To - Use Pathfinder integration

12. How To - Use Pathfinder integration

Pathfinder is a server-side LISA component that can be used to make information about what happens on the server available to the client. You can enable it in the settings dialog by checking the "Enable Pathfinder Integration" checkbox. This will automatically turn on the "Capture HTTP traffic" option as well.

The main effect of turning on this option is that several new variables will be automatically made available as can be seen on this screenshot:

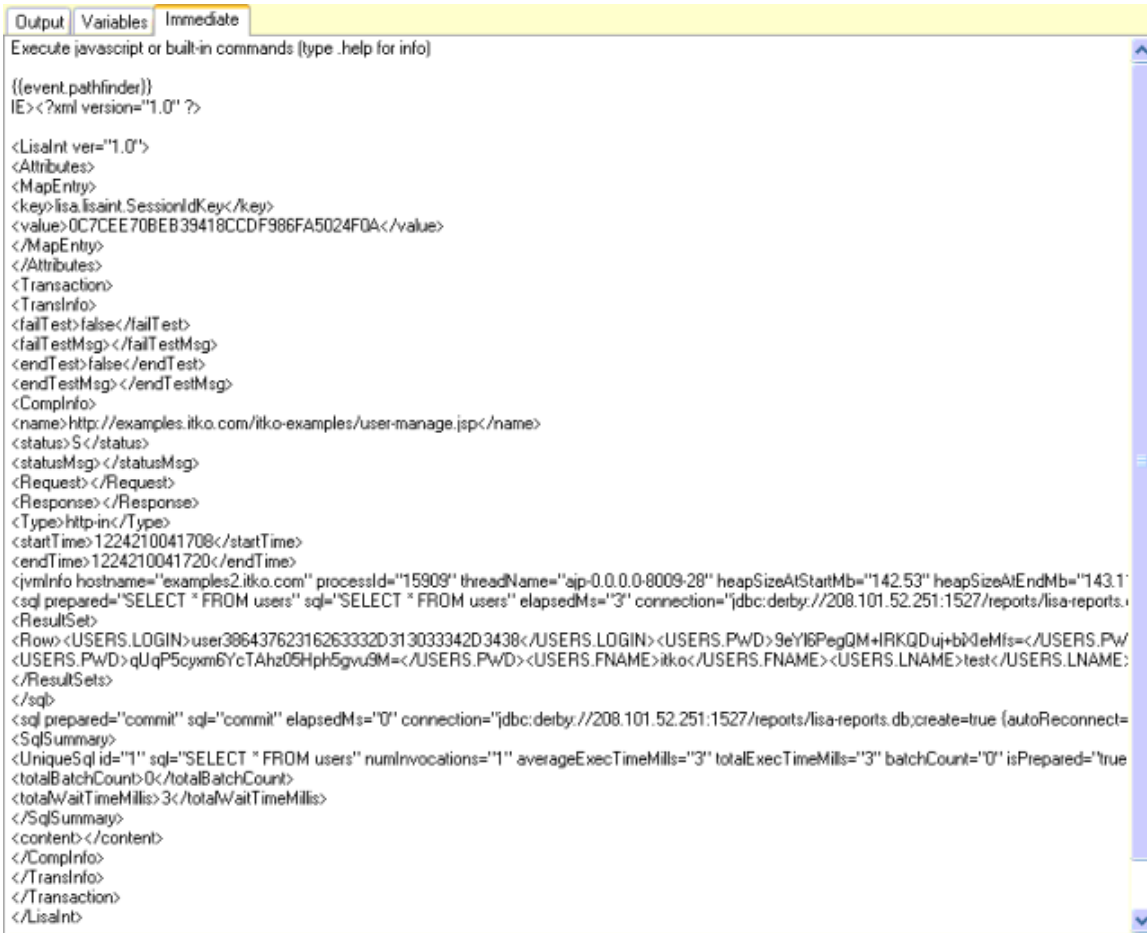
| Key | Value |
|-----------------------------------|---|
| {{event.bytes.in}} | 266669 |
| {{event.bytes.out}} | 10137 |
| {{event.cookie.JSESSIONID}} | 0C7CEE70BEB39418CCDF986FA5024F0A |
| {{event.frame.path}} | |
| {{event.param.cmd}} | list |
| {{event.path}} | |
| {{event.pathfinder}} | <?xml version="1.0" ?> <Lisaint ver="1.0"> <Attributes> <MapEntry> <key>lisa.lisaint.SessionIdK |
| {{event.response.network.time}} | 988 |
| {{event.response.render.time.ie}} | 281 |
| {{event.response.render.time}} | 281 |
| {{event.response.server.time}} | 12 |
| {{event.response.time}} | 1281 |
| {{event.response}} | <HTML><HEAD><TITLE>list users</TITLE><SCRIPT language=javascript// lib.js <site func |
| {{event.status.code}} | 200 |
| {{event.type}} | docload |
| {{event.url}} | http://examples.itko.com/itko-examples/user-manage.jsp?cmd=list |
| {{itko.examples.page}} | examples overview |
| {{itko_examples_page}} | user manage |
| {{lisa.lisaint.SessionIdKey}} | 0C7CEE70BEB39418CCDF986FA5024F0A |
| {{list_prop_command}} | list |
| {{list_size}} | 20 |
| {{list_user_1}} | user38643762316263332D313033342D343879eY16PegQM+IRKQDu+bKleMfs= |

In addition to the variables listed in the User Guide's [Filters section](#), a few more can be seen:

`event.response.server.time` measures the time it took the server to generate the payload received by the client for the last transmission.
`event.response.network.time` measures the time it took the payload to go from the server and reach the client.
 auto variable name are custom variables defined by Pathfinder to represent some variable value on the server and made available for inspection one the client.

Note if those variables are exposed by the LEK they will be available automatically even without Pathfinder integration.

`event.pathfinder` is the most important variable. It is an xml representation of the Pathfinder payload and contains all pathfinder information, that can then be extracted using XPath expressions, with the [lisa.pathfind](#) API.



```
Execute javascript or built-in commands (type .help for info)

{({event.pathfinder})
IE><?xml version="1.0" ?>

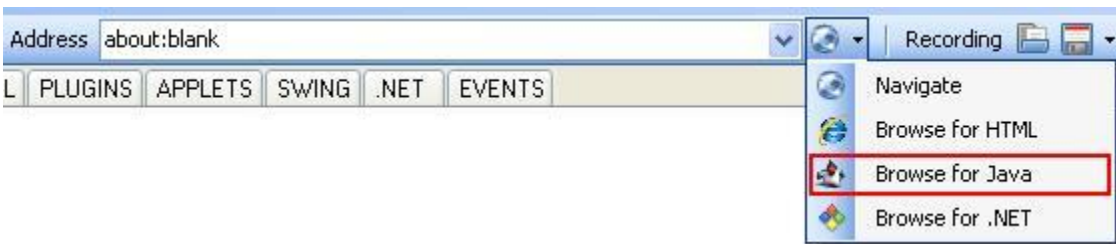
<?xml ver="1.0">
<Attributes>
<MapEntry>
<key>lisa.lisaint.SessionIdKey</key>
<value>0C7CEE708EB39418CCDF986FA5024F0A</value>
</MapEntry>
</Attributes>
<Transaction>
<TransInfo>
<failTest>false</failTest>
<failTestMsg></failTestMsg>
<endTest>false</endTest>
<endTestMsg></endTestMsg>
<ComplInfo>
<name>http://examples.itko.com/itko-examples/user-manage.jsp</name>
<status>S</status>
<statusMsg></statusMsg>
<Request></Request>
<Response></Response>
<Type>http-in</Type>
<startTime>1224210041708</startTime>
<endTime>1224210041720</endTime>
<jvmInfo hostname="examples2.itko.com" processId="15909" threadName="ajp-0.0.0.0-8009-28" heapSizeAtStartMb="142.53" heapSizeAtEndMb="143.1"
<sql prepared="SELECT * FROM users" sql="SELECT * FROM users" elapsedMs="3" connection="jdbc:derby://208.101.52.251:1527/reports/lisa-reports.
</ResultSets>
<Row><USERS.LOGIN>user38643762316263332D313033342D3438</USERS.LOGIN><USERS.PWD>9eY16PegQM+IRKQDuj+b4eMfs</USERS.Pw
<USERS.PWD>qUqP5cyxm6YcTAhz05Hph5gyu9M</USERS.PWD><USERS.FNAME>itko</USERS.FNAME><USERS.LNAME>test</USERS.LNAME>
</ResultSets>
</sql>
<sql prepared="commit" sql="commit" elapsedMs="0" connection="jdbc:derby://208.101.52.251:1527/reports/lisa-reports.db;create=true (autoReconnect=
<SqlSummary>
<UniqueSql id="1" sql="SELECT * FROM users" numInvocations="1" averageExecTimeMills="3" totalExecTimeMills="3" batchCount="0" isPrepared="true
<totalBatchCount>0</totalBatchCount>
<totalWaitTimeMills>3</totalWaitTimeMills>
</SqlSummary>
<content></content>
</ComplInfo>
</TransInfo>
</Transaction>
</?xml>
```

For instance, to get the query string out of the first SQL statement in the payload pictured above, you can use return lisa.pathfind('//sql/attribute::sql'), which will return SELECT * FROM users.

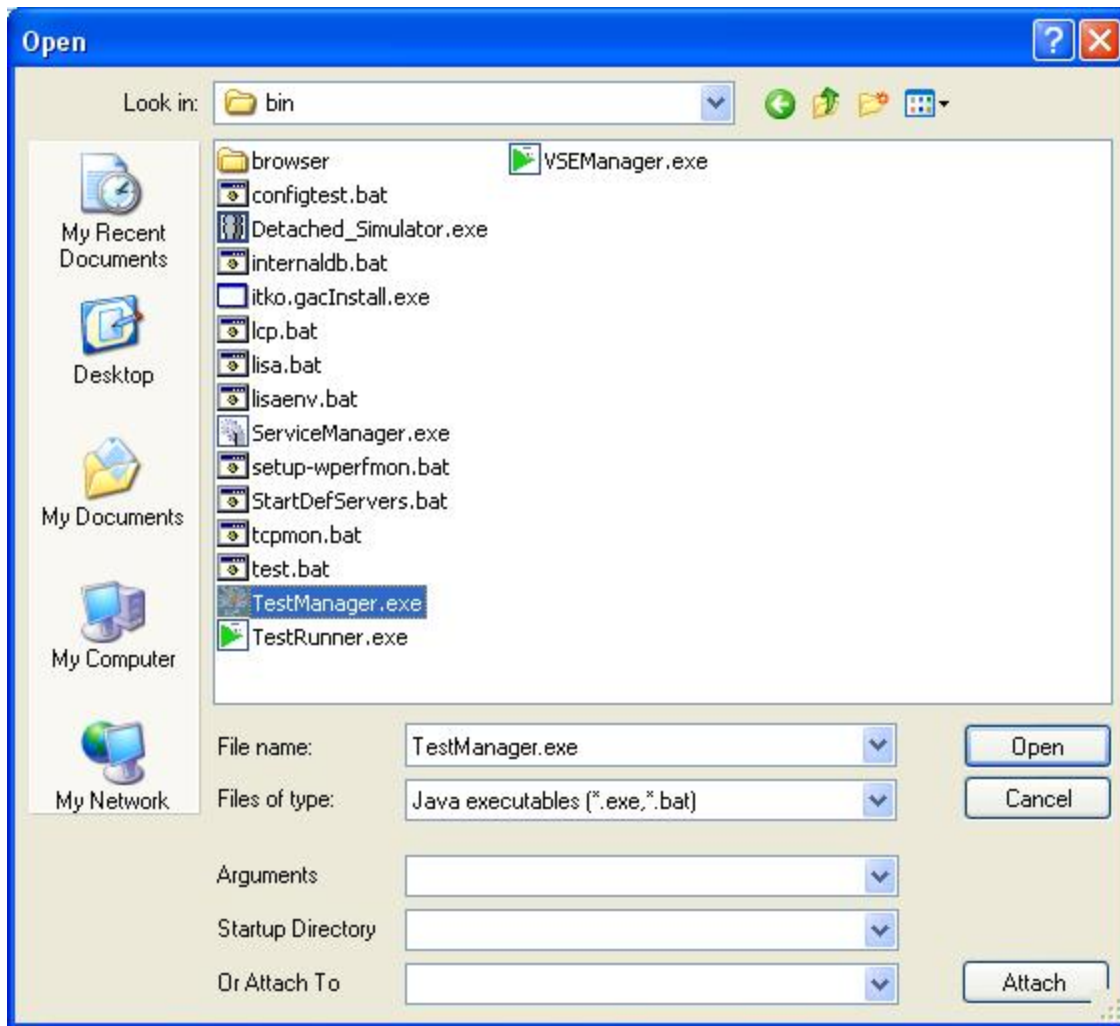
13. How To - Write Java Swing and WebStart tests

13. How To - Write Java Swing and WebStart tests

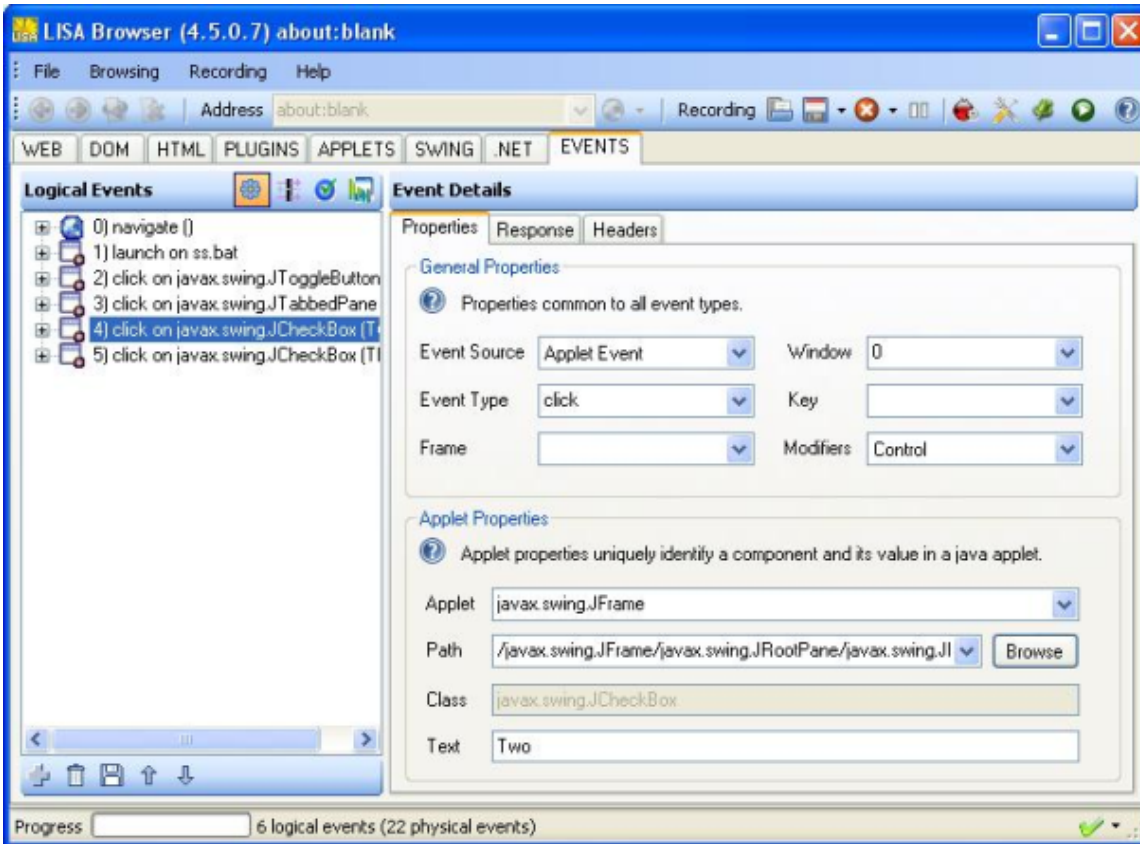
Authoring Swing tests is practically no different from writing applet tests. From a GUI perspective, the only difference is how you launch the application.



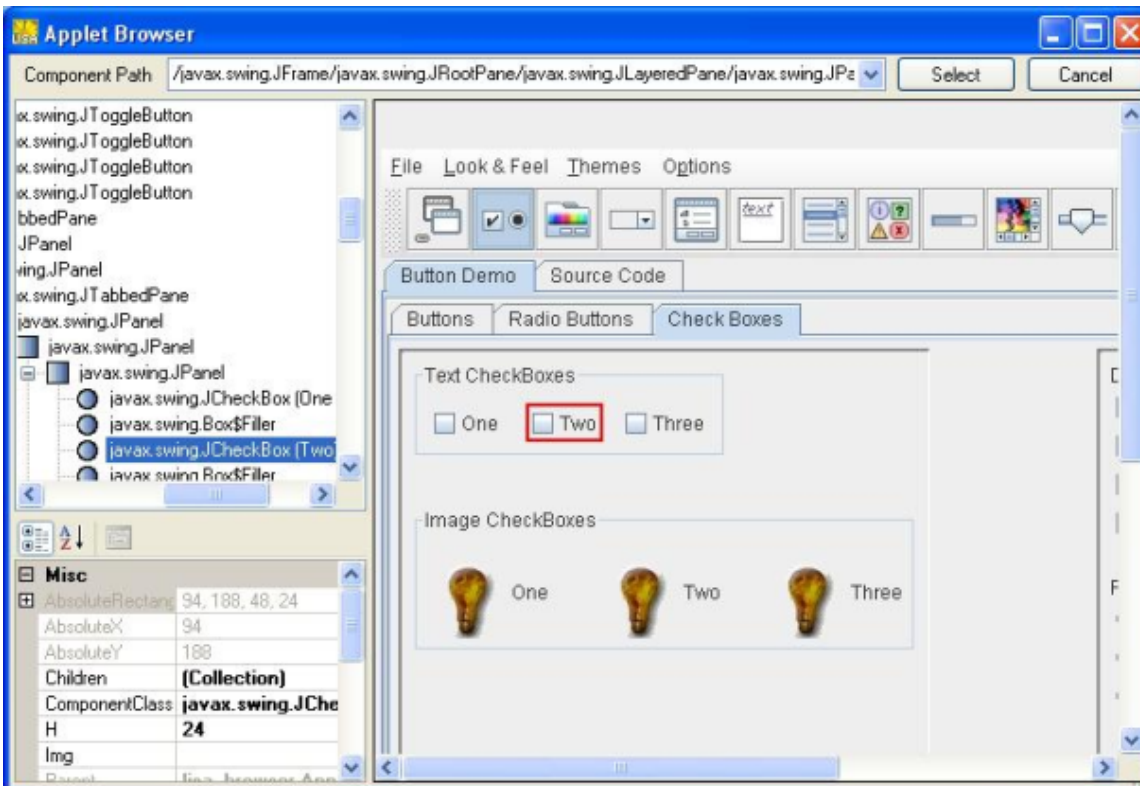
Instead of navigating to a URL, you browse the filesystem for an executable or batch file used to launch the swing application. In most cases the executable is going to be java.exe or javaw.exe and you can specify the command line (including classpath, VM arguments, etc...) in the Open Dialog window, as well as the start directory (optionally). In other cases the application is launched from a pre-packaged executable (as LISA TestManager is for instance) or from a batch script. This is supported too and makes no difference.



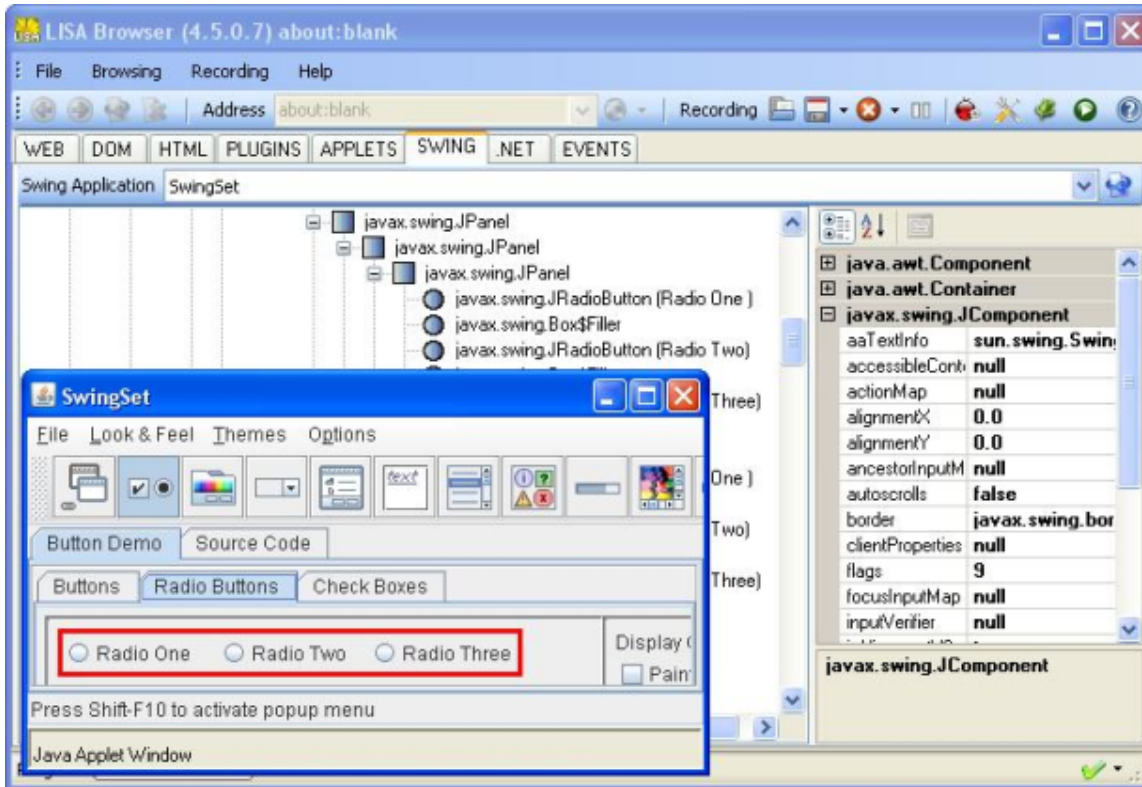
Below is a screenshot of a few events recorded against a Swing application, you will notice it looks almost identical to Applet events, except for the Applet field, which is now a JFrame.



You can similarly browse for offline editing and bring up the Applet/Swing browser to change and/or parametrize event targets (note the red highlight rectangle below is part of the application, not a screenshot artifact):

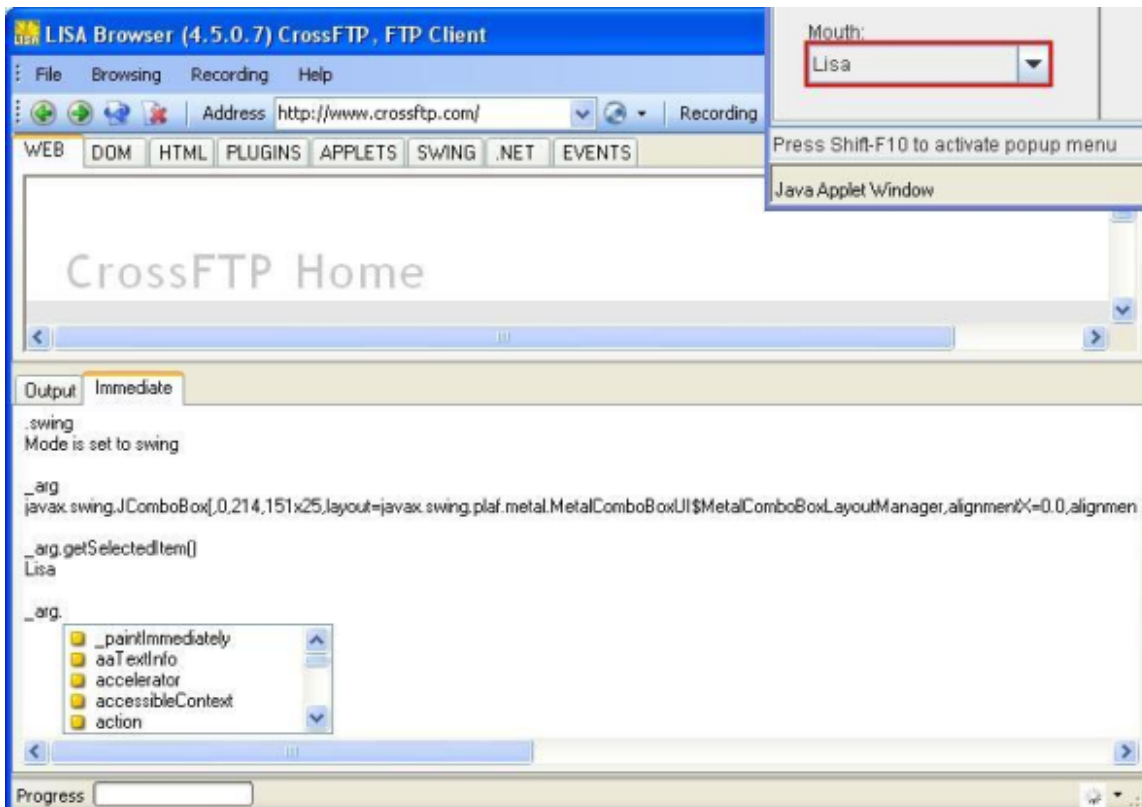


Finally, Swing applications also have their own live instance browser under the SWING tab (in both recorder and playback modes). When selecting a component in the swing hierarchy, this component will get highlighted in the live application so you can easily verify which elements you are targeting (note the red highlight rectangle below is part of the application, not a screenshot artifact):



Note that the exact same things apply to Java WebStart applications except that they can be launched directly from a jnlp link in a browser web page. The test will then automatically mix web steps and swing steps.

Support for arbitrary beanshell expressions is supported just as in applets and the last event target can also be referred to with the special variable `_arg`. You can test your java filters in the debug window by setting the mode to ".swing" and then evaluating the expression:



An example of how this would be useful is when the text filter is not easily able to get some value onscreen, in particular if you have a JTable, you can not get its cell values unless you double click them because the cells are not swing components (until they are double-clicked). With a script

filter, you could simply select the table as the component and then write a script like: `return _arg.getModel().getValueAt(2,3);`

Finally, an interesting point to note is that while this step's primary purpose is to do Swing or AWT testing, there is nothing that prevents you from running against headless or console java applications. You can run "in-container" tests by specifying any scripts you want in java script nodes.

14. How To - Write .NET WinForms tests

14. How To - Write .NET WinForms tests

Coming...

15. How To - Debug a test

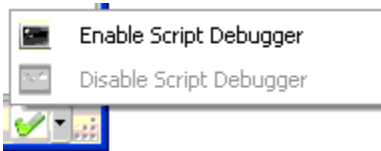
15. How To - Debug a test

Unexpected behavior can occur during recording or replay, and for both cases the best approach is to debug the test directly in the DOM browser, since it acts as the test execution environment, and is just driven by LISA during staging or ITR. The only exception to this rule is if the test must use multiple iterations of a dataset or variables defined in non-web 2.0 steps, because LISA takes care of sending those values to the DOM browser environment.

The main tool at your disposal to understand what is happening in a test is the debugging window, which you can toggle both in recording and playback mode by clicking the red debug icon in the toolbar:

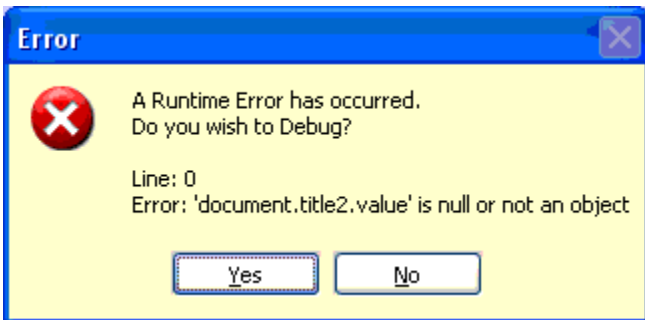


You can also toggle it by clicking the little status icon in the bottom-right corner of the screen:

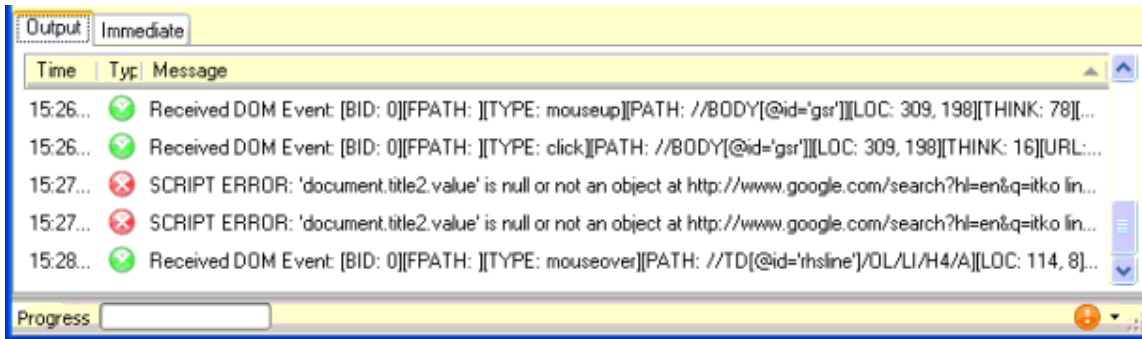


This icon has 3 states: a green checkmark to indicate no errors, a spinning wheel to indicate a document is loading or an orange icon to indicate there have been errors, most of the time script errors, but they could also be non-critical DOM browser errors. Script errors are sometimes difficult to debug given just the error message, hence the little popup menu as seen above to enable or disable a Javascript Debugger.

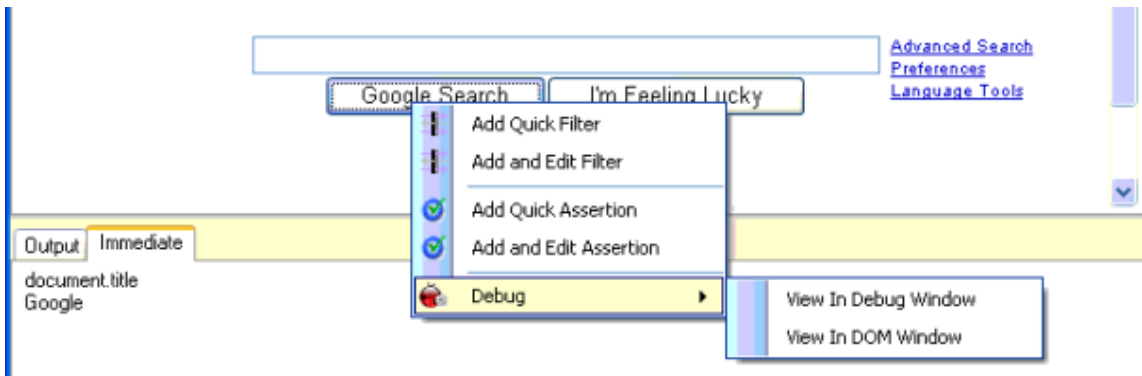
By default all debuggers are disabled to not interfere with the recording or playback, but if you enable it and there is a script error, you will see this error message popup:



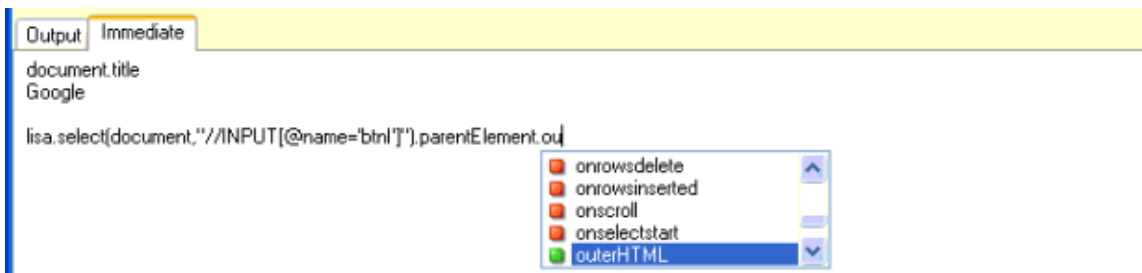
Clicking "Yes" will launch the debugger and show you all the details about the script error. Other than that, just opening the debug window will suffice in most cases:



In recording mode, 2 tabs are available in the debug window, the Output tab as pictured above and the Immediate tab. The Output tab just logs all messages (informational, warnings, debug as controlled by the log settings) and the Immediate lets you do more active debugging by typing in commands. It behaves like an interactive prompt on a web page and supports built-in commands, javascript and java (including the use of variables, as denoted by the usual syntax `variable`). It also supports point-and-click interaction with the currently viewed document as illustrated below:



By right clicking on an element, you get a menu that contains a **Debug** entry, to let you view the element in the DOM view or in the Immediate window. If you choose Immediate, you will be able to evaluate any DOM or javascript expression on the element, helped by intellisense:



This can be helpful in writing script filters but also to inspect javascript event handlers for instance. All built-in commands can be accessed by typing a dot (.) as the first character of the line. You can display what they do using the `.help` command.

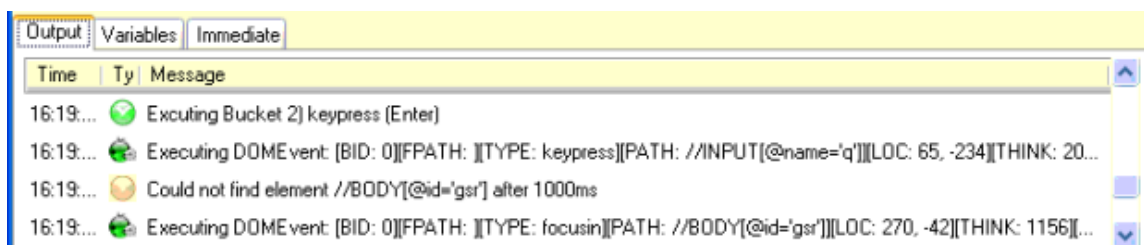


For instance, there are a `.js` and a `.java` entries that allow you to swap between javascript and java syntax for commands. The last element to receive an event can be referenced as the special variable `_arg`, just like in script filters.



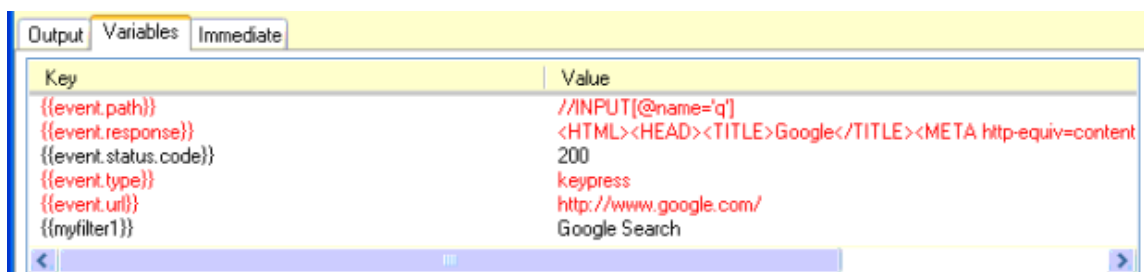
Note that all these javascript and java (beanshell to be exact) expressions can be used in filters so this is a useful way to design your filters during recording.

Most of the debugging will probably take place during playback since it will put to the test both the test design and the application under test. Playback offers the same debugging window with the Output and Immediate tab, and also a Variables tab. Most errors or warning you will see logged in the Output tab will be due to the failure of identifying an element on the page or in a control (as seen in this screenshot).

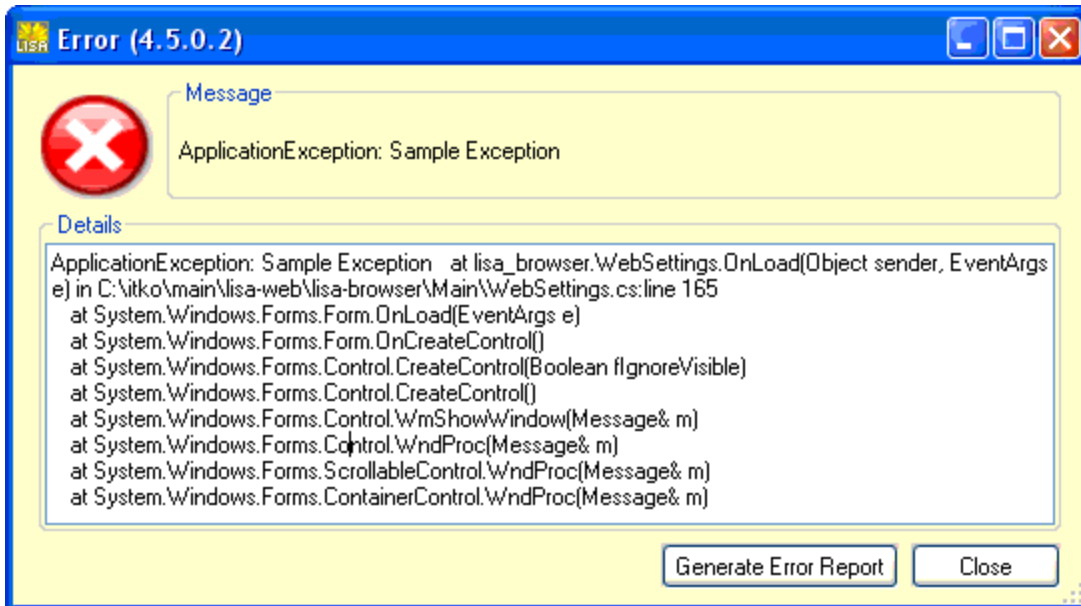


There could be several reasons for this: a genuine bug in the system under test or a failure to adequately parametrize a dynamic element (see the [Dynamic Elements](#) section). You will also see the filters and assertions being executed and their values, so you can quickly see why an assertion fired for example.

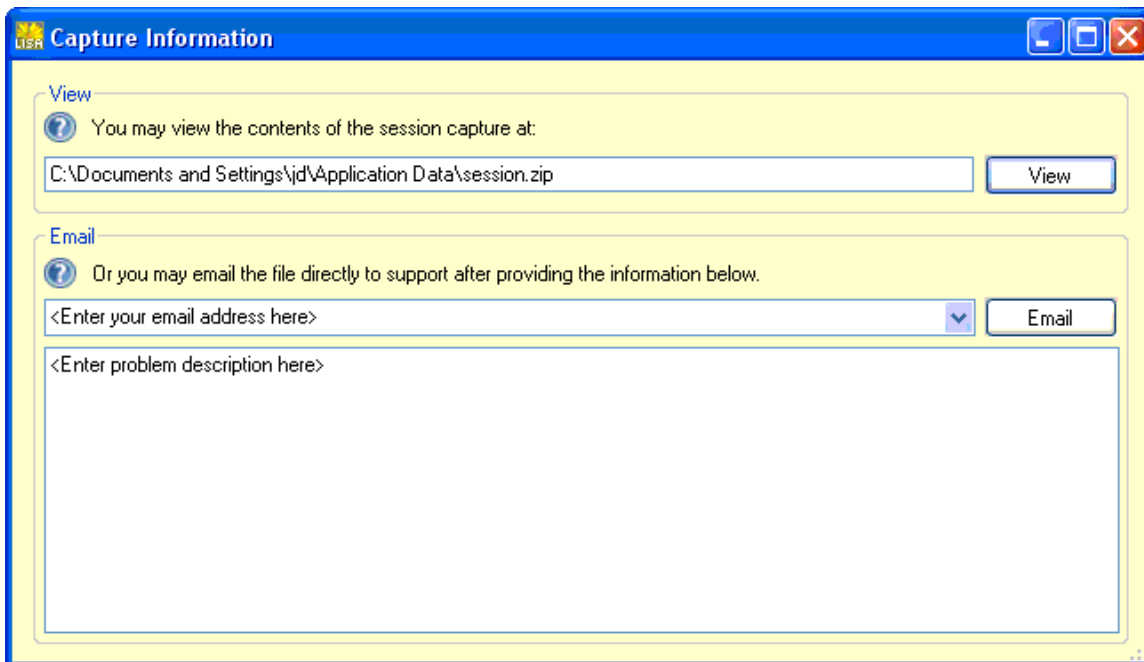
The Immediate tab behaves the same way as in recording mode. The Variables tab will list all the variable values for every step (highlighting in red the ones that were just modified). This is particularly useful when you set breakpoints or step through a test as in a debugger, you can observe the value of all the variables:



Finally, severe errors won't be caught and logged in the Output tab of the debug window, but instead will generate a dialog like:



In most cases (especially during recording) you can close the dialog and proceed but not in all cases. And since it should not happen, you can report it by clicking the **Generate Error Report** button. This will create a zip archive containing information about the environment, the settings, the current recording and playback, all the logs including the exception and a screenshot, and then present the user with the following dialog:



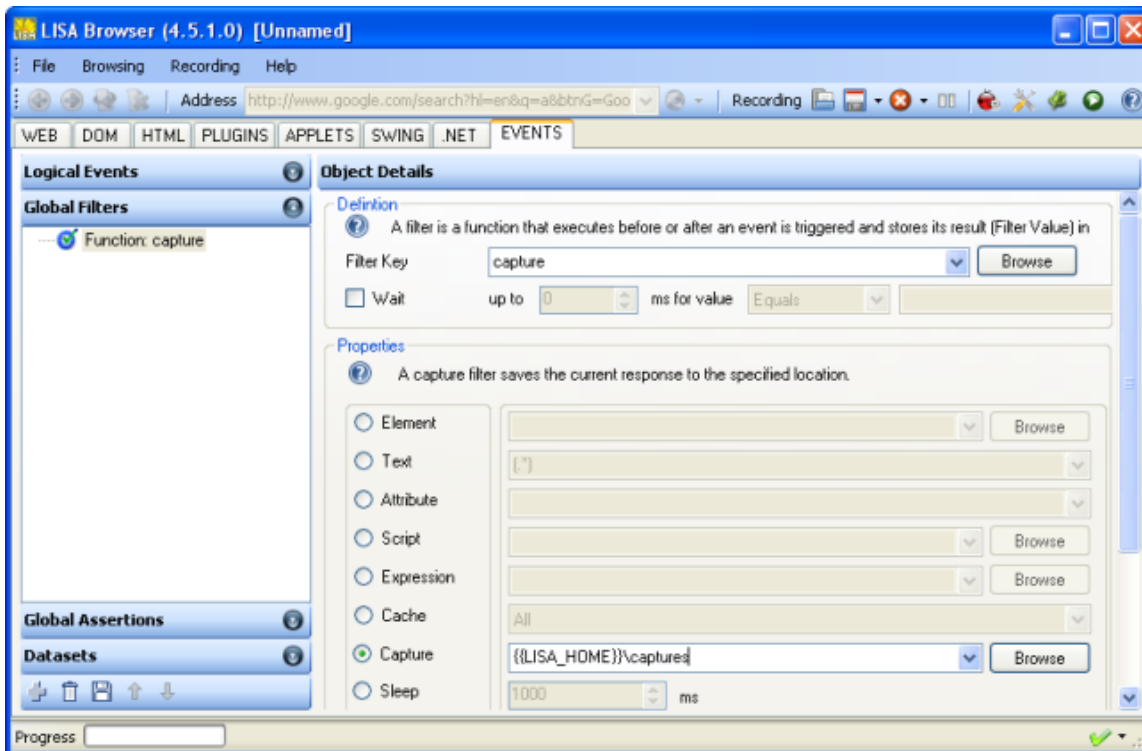
You can open the zip archive by clicking **View** and then email it to support by clicking **Email** after filling the required information for troubleshooting. If there is an SMTP server running on the machine (if IIS is installed for example), the email will be sent automatically, otherwise the default email client will be opened. If there is none, you will have to email the file manually.

16. How To - Use global filters and global assertions

16. How To - Use global filters and global assertions

The web 2.0 browser provides the ability to execute global filters and global assertions, that is filters and assertions that execute on every step. If you have repetitive actions you need to take, rather than defining those manually on every step, you can do define them once, globally and they will execute after every step. Global filters execute before local filters and global assertions execute before local assertions.

Adding a global filter is very similar to adding a normal filter or assertion. You go to the EVENTS tabs and select the "Global Filters" or "Global Assertions" collapsible panels.



This picture, for instance, shows how to capture screenshots on every step and save those in a specified directory for later investigation. Another typically useful usage of a global assertion is depicted below

If

An assertion is a boolean function that fires after an event and specifies an action based on the result.

☒ If

☐ Or If there are any ☐ W3C HTML Errors ☐ W3C HTML Warnings ☐ Broken HTML References

☐ Or If the event took over ms

☐ Or If the event generated over bytes

Then

If the selected expression above is ☒ True ☐ False

Then

☐ And save a screenshot to

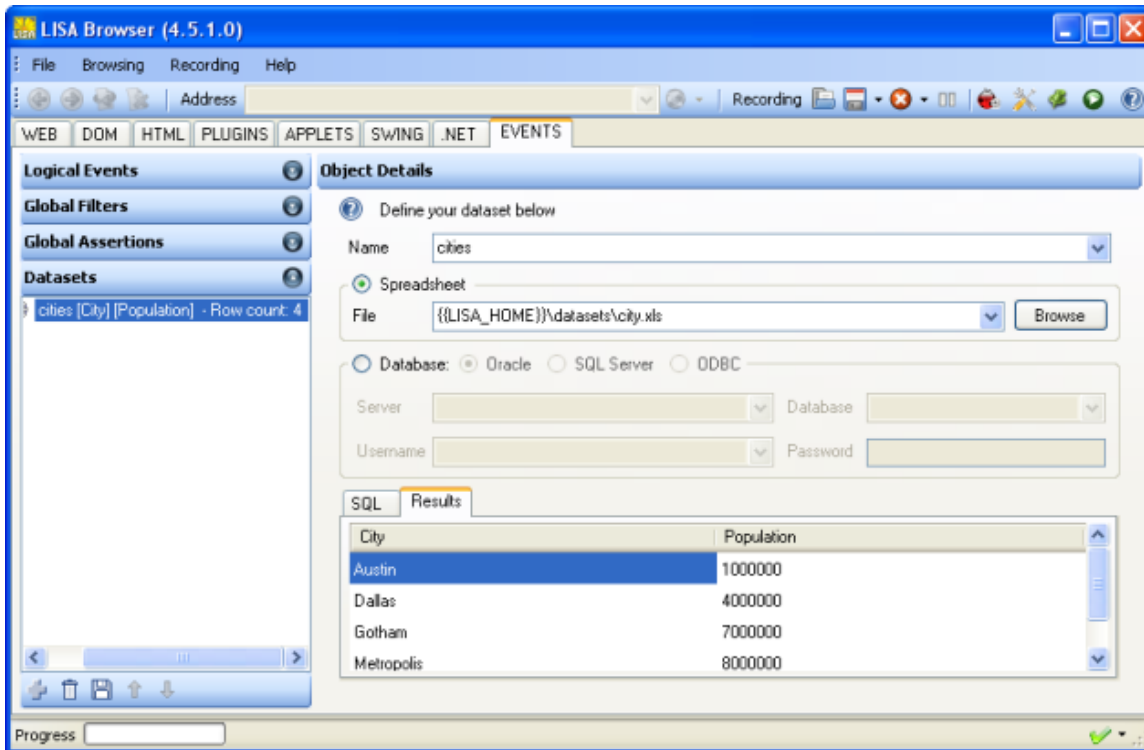
This verifies after each step that the server did not return an internal error code.

17. How To - Interact with external resources

17. How To - Interact with external resources

LISA has many steps to interact with external resources, be they flat files, excel files, databases, etc...and all of these resources will be automatically available to web 2.0 steps when run through LISA. However, if you run standalone, or through LISA but in the debugger, the external LISA steps won't execute and the external data won't be available. That's why there are ways to access external data from the DOM browser too.

First, there is the concept of web 2.0 dataset. Those are defined directly in the browser and don't depend on TestManager. To define them, you go to the EVENTS tab and select the Datasets collapsible panel, then Add a dataset. The details panel lets you specify its name, its data source and makes it easy to test:



Once a dataset is defined, using it in steps or filters is very easy using the following syntax:
dataset_name[row_index][column_name_or_index].

General Properties

Properties common to all event types.

| | | | | | |
|--------------|-----------|-----------|---|-------|------|
| Event Source | DOM Event | Window | 0 | X | -111 |
| Event Type | change | Key | | Y | -27 |
| Frame | | Modifiers | | Think | 0 |

DOM Properties

DOM properties uniquely identify an element and its value on a page.

| | | | | | |
|-------|-----------------------------|--------|------|--|--|
| Url | http://www.google.com/ | | | | |
| XPath | //INPUT[@name='q'] | Browse | | | |
| Value | {{cities[i++][Population]}} | | | | |
| Tag | INPUT | Type | text | | |

This allows you to retrieve data at any specified row and column without having to advance the dataset automatically. You simply need to keep track of an index to get to the desired row. However, if you want to automatically advance or go back, the syntax also supports the operators ++ and -- as pictured above.

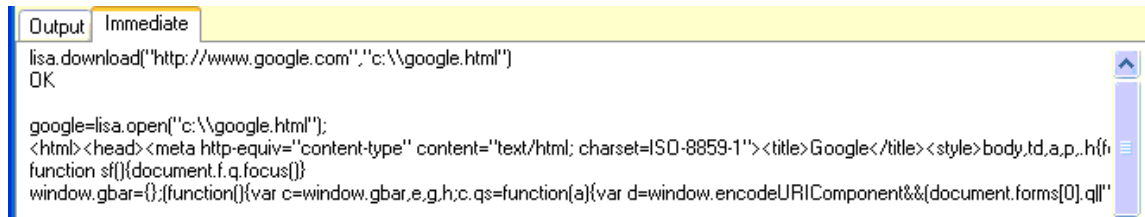
Something like cities[i++][Population] will get the value at the i-th row in the Population column and increase i by one for next time it is used. As a convenience, 1-letter variables used as integers like i will automatically be defined and set to 0 if they haven't been defined before.

Finally, the syntax cities.length or cities.count can be used to determine the number of rows in the cities datasets. This is useful to write exit condition assertions when iterating over a dataset (e.g. If i More Than cities.length Then Go To event xyz).

In addition you can use of the global [lisa](#) scripting object. Of interest here are the following methods: download, open, fileQuery, dbQuery.

- **download**: allows you to download a resource from a given url to the specified path, so you can interact with it, using for example:
- **open**: will read the contents of the specified file and return it as a string.
- **fileQuery**: will execute a SQL query against an Excel file and return a javascript **dataset**.
- **dbQuery**: will execute a SQL query against a Database file and return a javascript **dataset**.

All of these functions can be used from the Immediate debug window or script steps or filters. For example, here is how you would use download and open from the Immediate window:

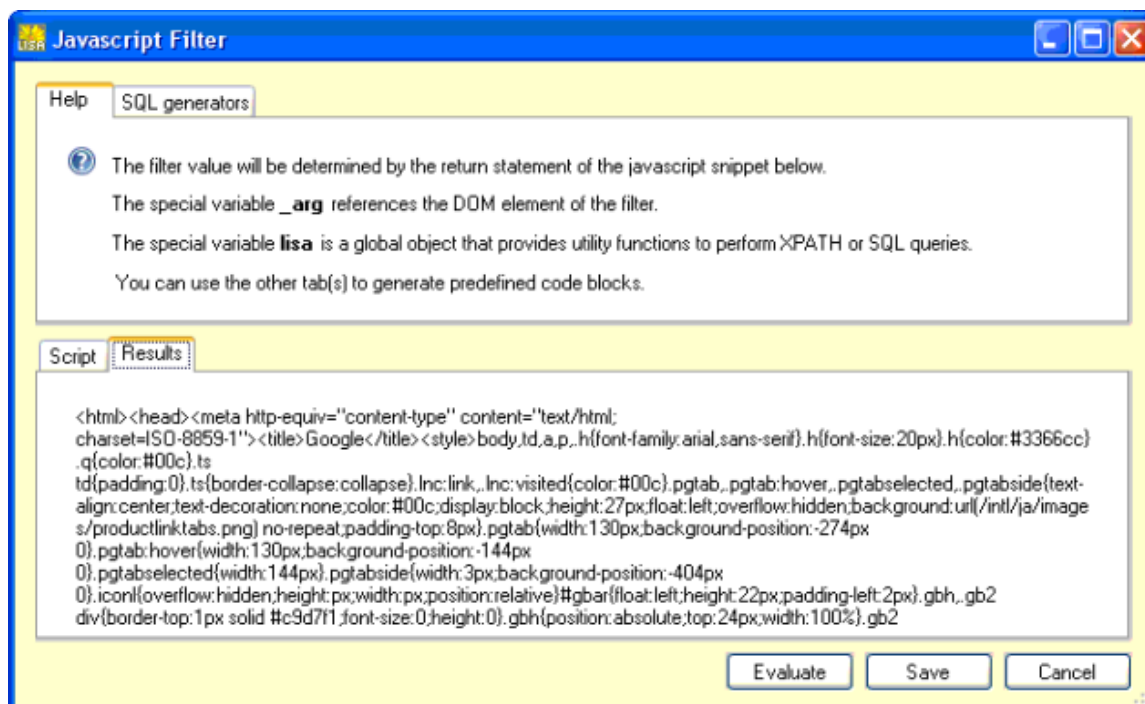


```

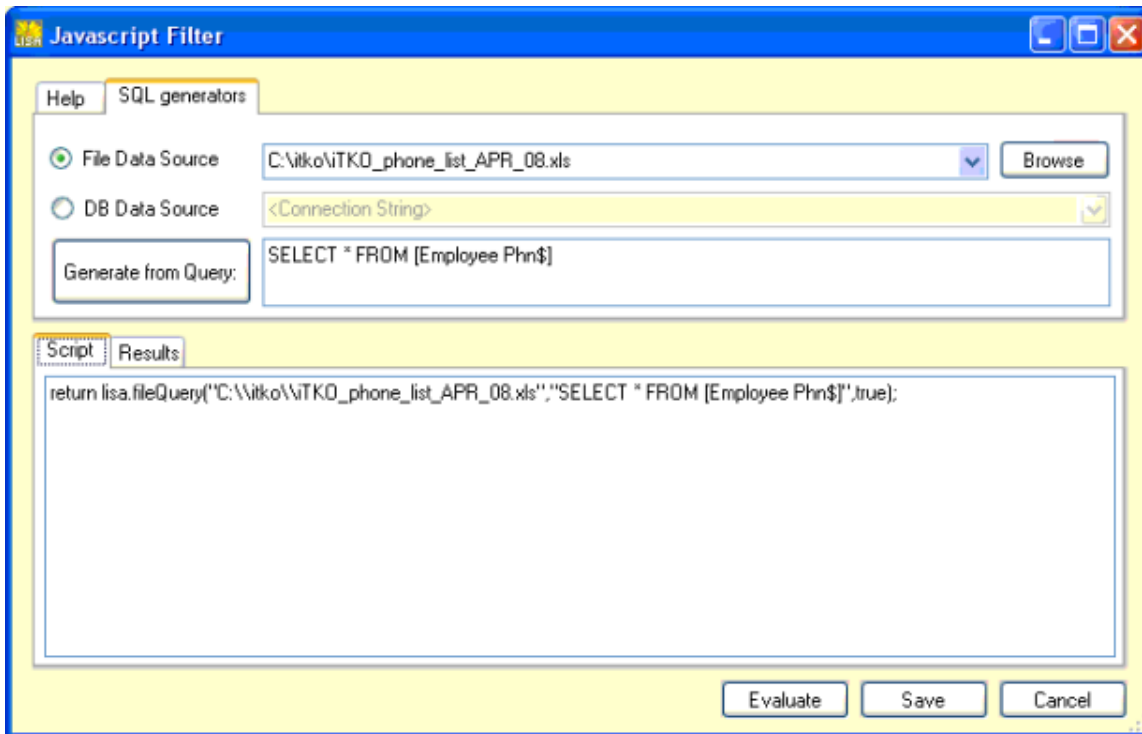
Output Immediate
lisa.download("http://www.google.com","c:\\google.html")
OK

google=lisa.open("c:\\google.html");
<html><head><meta http-equiv="content-type" content="text/html; charset=ISO-8859-1"><title>Google</title><style>body,td,a,p,.h{font-family:arial,sans-serif}.h{font-size:20px}.h{color:#3366cc}.q{color:#00c}.ts{border-collapse:collapse}.lnc:link,.lnc:visited{color:#00c}.pgtab,.pgtab:hover,.pgtab:selected,.pgtab:side{text-align:center;text-decoration:none;color:#00c;display:block;height:27px;float:left;overflow:hidden;background:url(/intl/ja/images/productlinktabs.png) no-repeat;padding-top:8px}.pgtab{width:130px;background-position:-274px 0}.pgtab:hover{width:130px;background-position:-144px 0}.pgtab:selected{width:144px}.pgtab:side{width:3px;background-position:-404px 0}.icon{overflow:hidden;height:px;width:px;position:relative}#gbar{float:left;height:22px;padding-left:2px}.gbh,.gb2{div{border-top:1px solid #c9d7f1;font-size:0;height:0}.gbh{position:absolute;top:24px;width:100%}.gb2
  
```

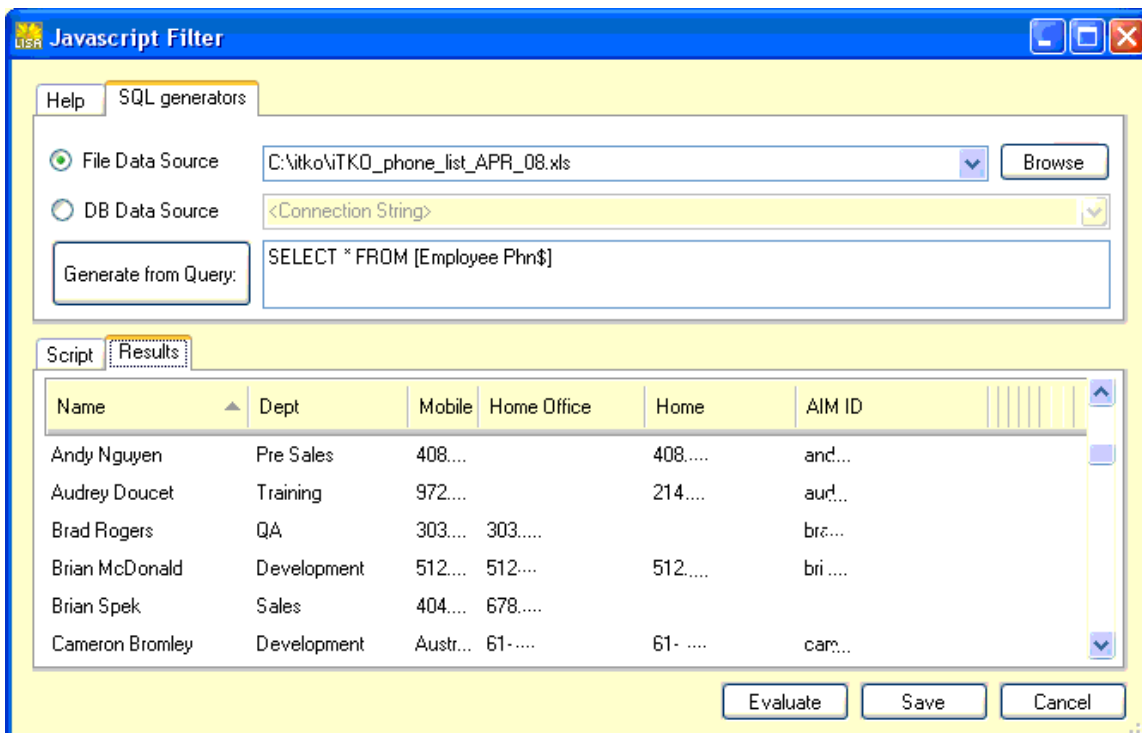
Similarly, if you created a script filter called google, clicked on **Browse** to bring up the script details popup, and clicked the **Evaluate** button after typing the script return `lisa.open("c:\\google.html")`, you would see the evaluation results:



From that point on, this html string is available as `google`. The other 2 functions, `fileQuery` and `dbQuery` can be used similarly, but there is a tab in the script details popup called **SQL Generators** that helps you generate the calls. Filling the properties at the top and clicking **Generate From Query** yields a script snippet:



which in turns evaluates to the following dataset:



Using the [dataset](#) API, you can create other filters and assertions to use the dataset. Let's assume the above dataset is saved in the `employees` filter. For example, all the following expressions are valid to use in a filter script: `return employees.count()`, `return employees.rows(0).cells(0)`, `return employees.rows(0).cells("Name")`, etc...

This makes it extremely easy to parametrize data and loop through datasets. This type of dataset is akin to a Local Lisa dataset since its data is not shared across test instances, but you have explicit access to any row or cell instead of implicitly moving to the next record on a given step.

18. How To - Run Load Tests

How To - 18. Run Load Tests

Starting in LISA 4.6, running web 2.0 load tests is supported in the same way as other steps, by specifying the desired number of virtual users in a staging document. Each virtual user runs a separate browser instance, which consumes a fair number of resources, so there is a hard limit of 40 virtual web 2.0 users per machine.

If your test is purely web-based, by running in dual IE/Firefox mode, you can double that number without using much more resources. Similar considerations apply for running other GUI technologies (Applets, Swing, .NET, Native, etc...). Multiple instances are supported up to point that is mostly determined by available hardware.

Several settings control how multiple instances run (sandboxed or shared, under which user account, etc...). Those can be consulted in the [Web 2.0 architecture](#) section.

If you are using LISA 4.5 or earlier, read on...

Currently web 2.0 steps do not support multiple virtual users running them in parallel in the same JVM (typically it will cause an error dialog to pop up stating there is a socket or listener error if you try to do so). You can still run multiple instances of the DOM browser in parallel, even on the same machine, provided you use one per simulator (i.e. one per JVM).

The other option is to run in headless mode. After a web 2.0 test is recorded and saved back to TestManager, the boolean variable HEADLESS will be added to the configuration (with a value of false by default). If you change that to true, web 2.0 steps will be replayed without the DOM browser, using the pure java javascript engine [Mozilla Rhino](#) and the [htmlunit](#) java library.

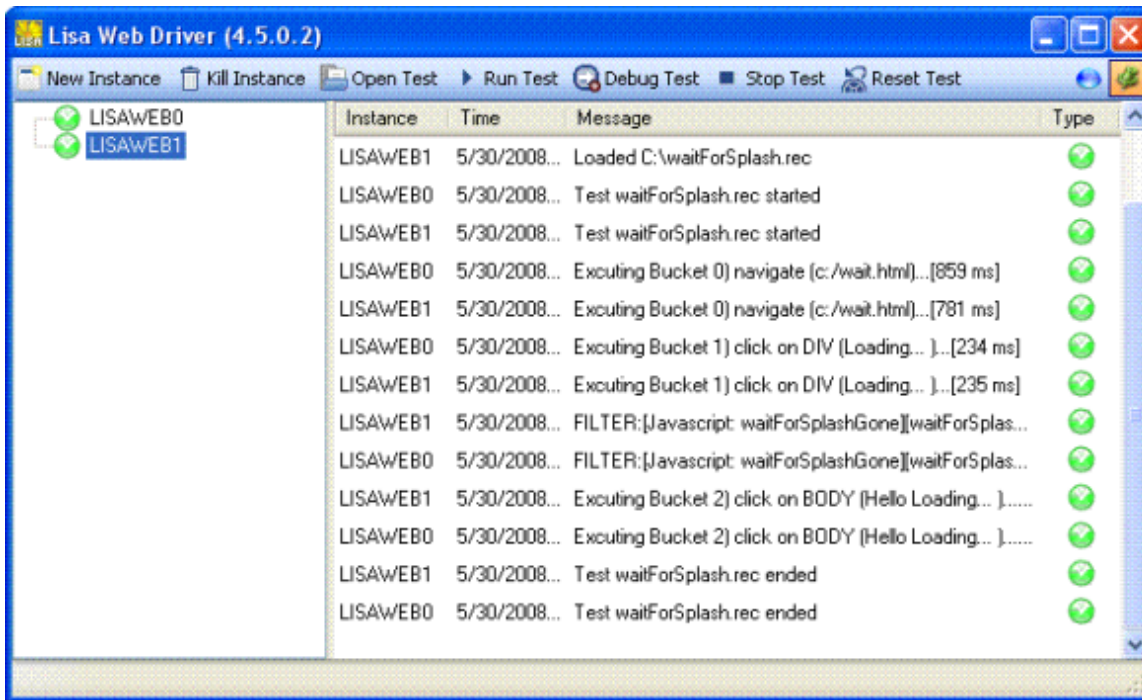
The advantages of this approach are:

- you can run true load tests (multiple vusers per JVM).
- it is platform-independent
- it can simulate multiple browsers (by setting the BROWSER configuration variable).

The disadvantages are:

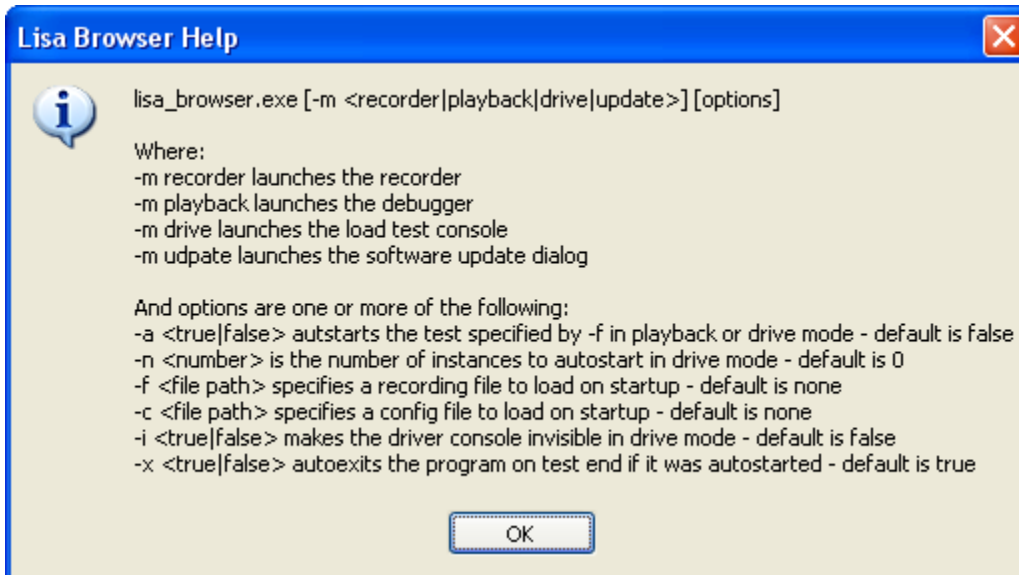
- some sites do not work well with it (the more complex the javascript they use, the more likely it is they may have problems).
- it is relatively heavyweight so don't expect to run hundreds of vusers on the same machine (dozens on a standard desktop is realistic though).
- it doesn't support frame (currently), authentication (currently), applets (currently) or Active X controls (never).

Finally, load testing in GUI mode is going to be available as well, but is only doable in standalone mode at this time. Running `lisa_browser.exe -m drive` will launch the multi-instance driver:



Then you can add as many instances as you want of the DOM browser (in our environment it was able to handle up to about 40 instances), then load a test and run it. All instances will run it in parallel and report the results as they go. If you run in dual mode (IE + Firefox) you can double your number of clients with little overhead (so up to about 80 per machine).

More specifically, you can control through command-line parameters how to start those instances:



There is also a way to specify an xml file that contains a list of instances along with test files (and optionally config files) they need to run. Users can also be specified if you need a test to be run in the context of a given user (typically useful for websites with windows authentication). The syntax of this xml file is:

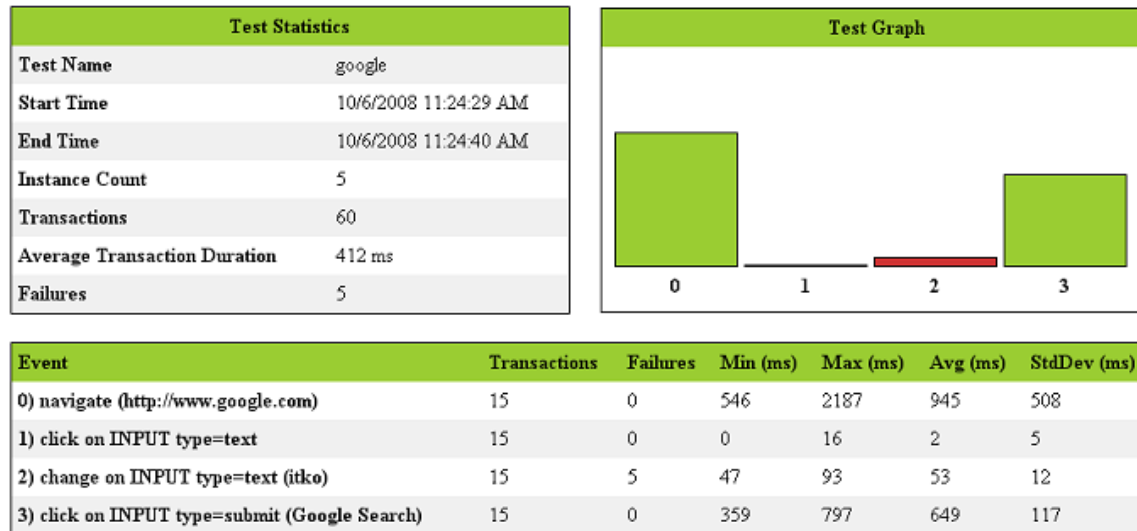
```
<staging>
<instance username="u1" password="p1" file="test1.rec"/>
<instance username="u2" password="p2" file="test2.rec" config="config2.config"/>
<test file="test3.rec" config="config3.config" instances="2"/>
</staging>
```

and it can be invoked as `lisa_browser.exe -m drive -stg stagingfile.xml`

All instances run in a separate windows user account (dynamically created and deleted as needed), so cookies and cache don't overlap between instances. The advantage of this approach is that it supports everything the DOM browser supports in single mode. The top-right icons allow you to pin the driver window on top and to hide all the DOM browsers as they run the test, making it look headless.

Finally, when the running tests are over, xml reports will be generated in the out directory. A default xsl stylesheet is provided to render the report as html but it could be customized to any other rendering as all the data is present in the xml.

LISA Web 2.0 report for [google]



Note: load tests in drive mode currently require you to be logged in as administrator on the computer.

19. How To - Run in a non-privileged account or on 64 bit platforms

19. How To - Run in a non-privileged account or on 64 bit platforms

Q: Can the web 2.0 step be used on non-administrator accounts?

A: Yes, it can be run even in Guest accounts but will require a bit of environment configuration:

1) Port specification

By default, the LISA browser uses dynamic ports to communicate with LISA. If some ports are locked down, you can assign them instead using the lisa.browser.source.port and lisa.browser.target.port properties (in the lisa.properties or local.properties).

2) Add permission for port listening

Some accounts may not have enough privileges to listen on a specific port (typically causing an Access Denied error). To add this permission, use the httpcfg.exe tool (available at

<http://www.microsoft.com/downloads/details.aspx?familyid=49ae8576-9bb9-4126-9761-ba8011fabf38&displaylang=en>) and run the following command: httpcfg.exe set urlacl /u <http://+:8098/> /a D:(A;;GX;;;WD) where you replace 8098 with the port you specified in step 1) as lisa.browser.target.port.

Note: a message box containing these instructions will appear the first time the error is encountered during a LISA run.

3) Register Tidy COM component

The LISA browser uses the w3c Tidy COM component to identify warnings and errors in the HTML code, as well as reformat it. This component is dynamically registered but if the account does not have the sufficient privileges to register a COM component, you should log in as administrator and run the command: regsvr32 %bin%\browser\TidyATL.dll

Note: a message box containing these instructions will appear the first time the error is encountered during a LISA run. If no action is taken the web 2.0 step will still be functional but missing the above functionality.

Q: Can the web 2.0 step run on 64-bit platforms

A: Yes, it can. The only thing to be aware of is that the component used to display the HTML source during test authoring will be disabled, but it should not affect running the test in any way.

20. How To - Record and replay against non us-english websites

20. How To - Record and replay against non us-english websites

Using Web 2.0, there is nothing special you need to do to record and replay against websites that use any language or locale. Of course, to see

the right glyphs on the screen you will need to download the language packs corresponding to the appropriate codepage, but if you can see them correctly in a browser, it means they're already installed.

The following screenshots show the DOM browser and TestManager after it recorded and replayed a test against a japanese website.

Authoring and evaluating a filter during recording:

The screenshot shows two panels from a testing tool. The top panel, titled "Properties", contains a description: "A text filter retrieves the inner text of a DOM element using a regular expression (first capturing group)." Below this are several radio buttons and input fields: "DOM Element" with the value "//TABLE[@id='content-table']/TBODY/TR[1]/TD[2]/TABLE/TBODY/TR/TD/H1" and a "Browse" button; "Text" with the value "[.*)" and a dropdown arrow; "DOM Attribute" with an empty field and a dropdown arrow; "Script" with an empty field and a "Browse" button; "Expression" with an empty field and a "Browse" button; and "Cache" with the value "All" and a dropdown arrow. The bottom panel, titled "Quick Test", contains a description: "Click Evaluate to see what value this filter would return if it were evaluated during the recording." Below this is a "Recorded Value" field containing the text "レノボ・ジャパンについて". At the bottom right of this panel are "Evaluate" and "Clear All" buttons.

Replaying the test in the DOM browser while showing debug events:

The screenshot shows three panels from a testing tool. The top panel, titled "Logical Events", lists a sequence of events: "0) navigate (http://w...", "1) click on A (レノボ...", "2) click on H1 (レノボ...", "3) click on H1 (レノボ...", "continue (quiet)", and "continue". Below the list are navigation buttons. The middle panel, titled "Web HTML", shows a screenshot of the Lenovo Japan website with the text "レノボ・ジャパンについて" visible. The bottom panel, titled "Output", has tabs for "Variables" and "Immediate". It displays a log of events: "13:28:... Executing DOMEvent: [BID: 0][FPATH:][TYPE: click][PATH: //TABLE[@id='content-table']/TBODY/TR[1]/TD[2]...", "13:28:... [Bucket Duration: 1062 ms]", and "13:28:... FILTER:[Text: filter.click.8214062][filter.click.8214062=レノボ・ジャパンについて]".

Replaying the test in the ITR after saving it to TestManager:

Execution History

- 0) navigate (http://{{SERVER0}}/jp/ja/index.html)
- 1) click on A (レノボ・ジャパンについて)
- 2) click on H1 (レノボ・ジャパンについて)
- 3) click on H1 (レノボ・ジャパンについて)
- end

Response Properties Test Events

Initial property values may be changed prior to executing the step. They are read and edited by editing an existing key.

| Key | Value |
|----------------------|------------------------------------|
| AJAX | true |
| BROWSER | Firefox |
| HEADLESS | false |
| LASTRESPONSE | <HTML lang=ja xml:lang="ja" xmlns= |
| LISA_HOST | dude |
| LISA_LAST_STEP | 3) click on H1 (レノボ・ジャパンにつ |
| LISA_TC_PATH | C:\Temp\rand |
| LISA_USER | jd |
| SERVER0 | www.lenovo.com |
| SERVER1 | www-06.ibm.com |
| event.path | //TABLE[@id='content-table']/TBODY |
| event.response | <HTML lang=ja xml:lang="ja" xmlns= |
| event.status.code | 200 |
| event.type | click |
| event.url | http://www-06.ibm.com/jp/pc/lenovo |
| filter.click.8214062 | レノボ・ジャパンについて |

21. How To - Run in Crash Dump mode

21. How To - Run in Crash Dump mode

The LISA browser runs mostly in managed code but due to external native libraries bugs and portions running native code (e.g. ActiveX, Applets, jdglue, etc.), it is possible to sometimes observe crashes in certain environment and under special circumstances (those will usually manifest themselves as AccessViolation exceptions).

To facilitate resolution of those errors, the browser supports running in Crash Dump mode. When these errors occur in Crash Dump mode, the browser will generate a large dump file that can be later analyzed to identify the root cause of the issue. To turn on Crash Dump mode, follow these instructions:

- 1) Download Windows Debugging Tools at the following location: [x86](#) or [x64](#).
- 2) Define the _LISA_CRASH_DUMP environment variable to be the Windows Debugging Tools install directory (e.g. _LISA_CRASH_DUMP=C:\Program Files\Debugging Tools for Windows (x86)).
- 3) Restart LISA

When a crash occurs, it will generate a large (> 100MB) .dmp file in the directory <LISA Install Dir>\bin\browser\out\Crash_Mode_Date_xx-xx-xxxx_Time_xx-xx-xxXX. This is the file that support will need to resolve the issue.

Note: the first time the browser runs, it may fail because it will auto-download a lot of large symbol files from microsoft servers. It should behave normally in subsequent runs (except maybe for a minor slowdown). When the problem is resolved you can unset _LISA_CRASH_DUMP to return to normal operating mode.

PART 3 - LISA Web 2.0 - Reference

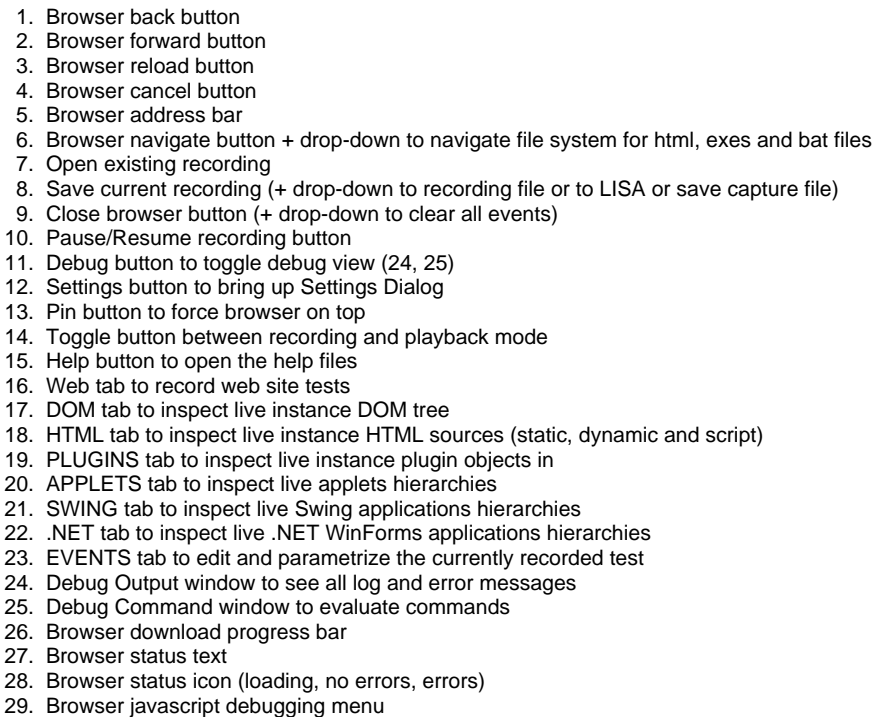
PART 3 - LISA Web 2.0 Reference

The following topics are available.

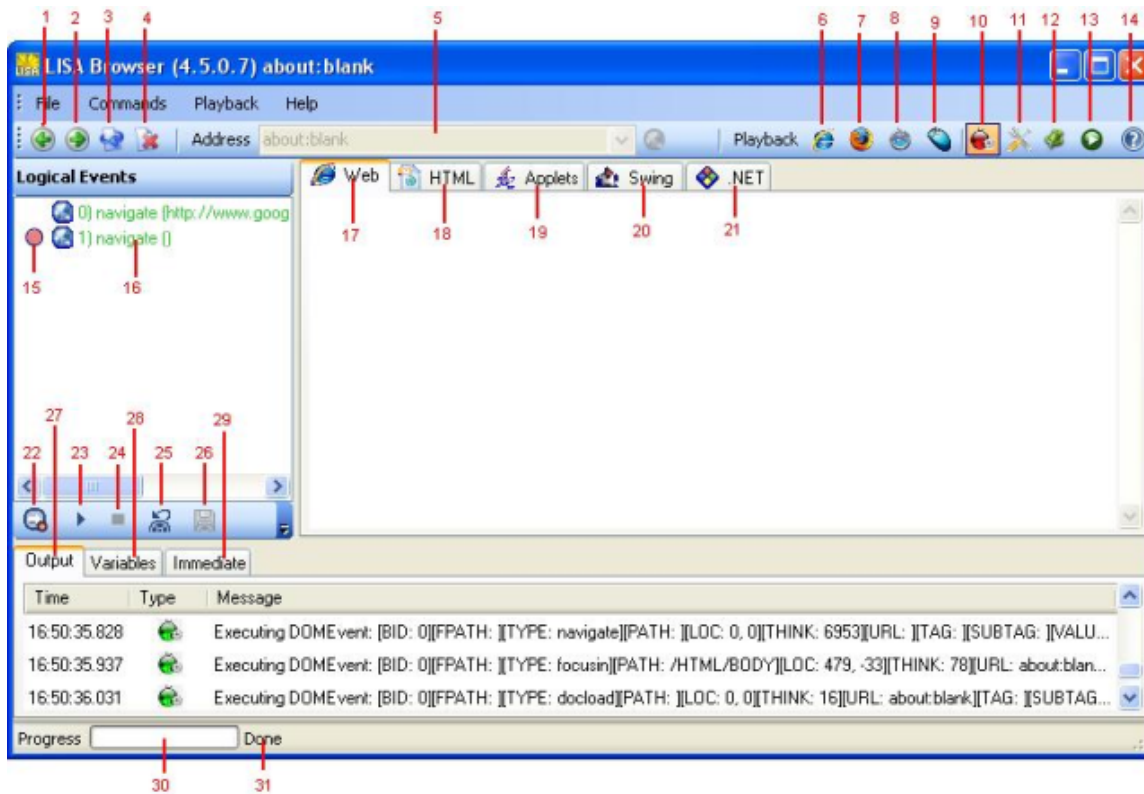
1. Recorder Reference
2. Debugger Reference
3. Settings Reference
4. XPath syntax Reference
5. Scripting Objects Reference
6. Command line Reference

1. Recorder Reference

1. Recorder Reference



2. Debugger Reference




1. Browser back button
2. Browser forward button
3. Browser reload button
4. Browser cancel button
5. Browser address bar (disabled in playback mode)
6. Turn on/off Internet Explorer as a replay environment
7. Turn on/off Firefox as a replay environment
8. Turn on/off Safari as a replay environment
9. Enable/Disable mouse movements during test execution
10. Debug button to toggle debug view (24, 25)
11. Settings button to bring up Settings Dialog
12. Pin button to force browser on top
13. Toggle button between recording and playback mode
14. Help button to open the help files
15. Set/unset Breakpoint
16. Web 2.0 event (LISA step)
17. Browser tab to host web steps execution
18. HTML tab to inspect live instance HTML sources (static, dynamic and script)
19. APPLETs tab to inspect live applets hierarchies
20. SWING tab to inspect live Swing applications hierarchies
21. .NET tab to inspect live .NET WinForms applications hierarchies
22. Execute highlighted step
23. Execute All steps starting at the highlighted one until breakpoint or end is reached
24. Stop current step execution
25. Reset test: blanks out browsers, clears variables, outputs, etc...
26. Save button (disabled now)
27. Debug Output window to see all log and error messages
28. Debug Variables window to see all current variables and their values
29. Debug Command window to evaluate commands
30. Browser download progress bar
31. Browser status text

3. Settings Reference

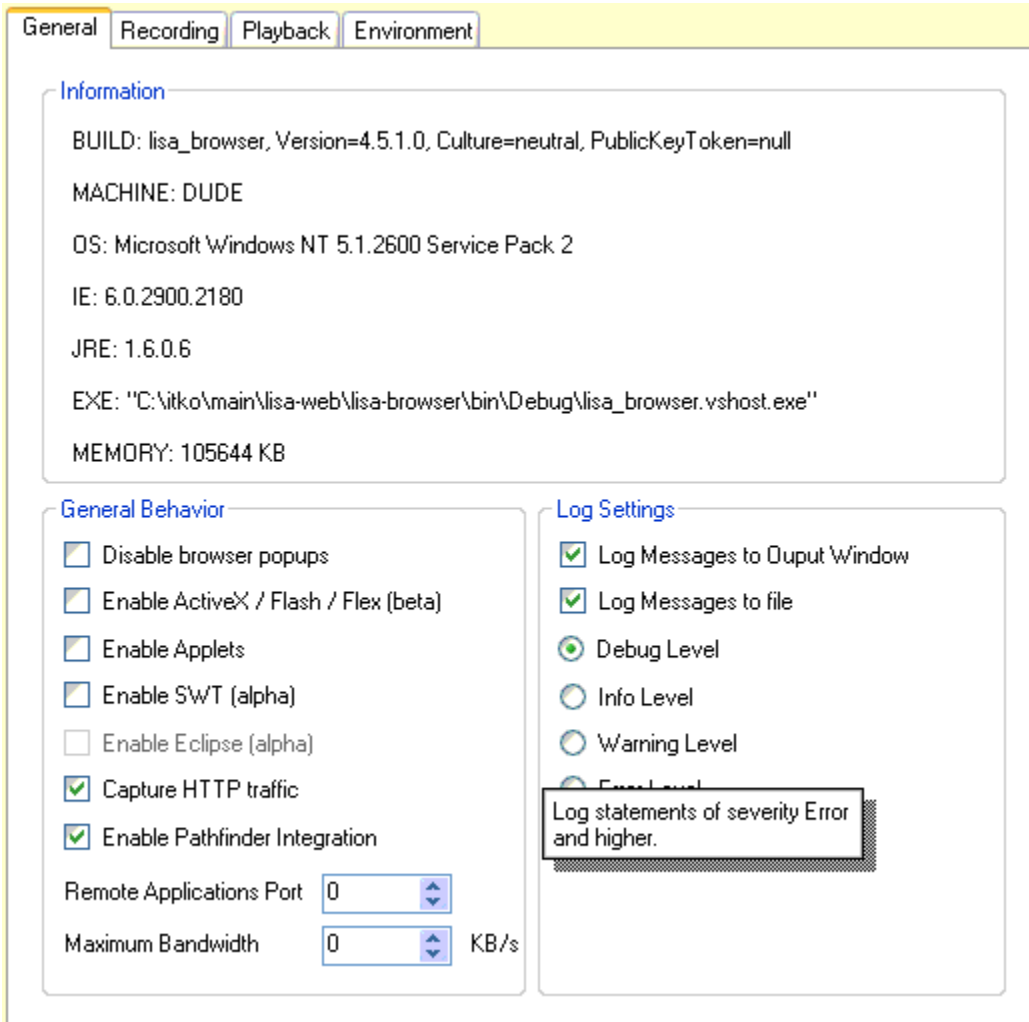
3. Settings Reference

Most of the settings have been mentioned in the documentation or the how-to's but this is the full reference for them.

They are divided in 4 sections: General, Recording, Playback and Environment.

For an online help, click on the  icon in the top right corner of the settings screen and then on one of the fields, to get a tooltip which gives more information about it as shown below:

General Settings:



General Recording Playback Environment

Information

BUILD: lisa_browser, Version=4.5.1.0, Culture=neutral, PublicKeyToken=null

MACHINE: DUDE

OS: Microsoft Windows NT 5.1.2600 Service Pack 2

IE: 6.0.2900.2180

JRE: 1.6.0.6

EXE: "C:\itko\main\lisa-web\lisa-browser\bin\Debug\lisa_browser.vshost.exe"

MEMORY: 105644 KB

General Behavior

☐ Disable browser popups

☐ Enable ActiveX / Flash / Flex (beta)

☐ Enable Applets

☐ Enable SWT (alpha)

☐ Enable Eclipse (alpha)

☒ Capture HTTP traffic

☒ Enable Pathfinder Integration

Remote Applications Port

Maximum Bandwidth KB/s

Log Settings

☒ Log Messages to Output Window

☒ Log Messages to file

☒ Debug Level

☐ Info Level

☐ Warning Level

☐ Error Level

Log statements of severity Error and higher.

- **Disable browser popups:** acts like a popup blocker. Overrideable in a test with `DISABLE_POPUPS`.
- **Suppress Javascript Dialogs:** alert and confirm dialogs will auto-respond during playback `SUPPRESS_DIALOGS`.
- **Enable ActiveX / Flash / Flex:** Turns on support for ActiveX events. Only reason to turn it off might be faster startup time.
- **Enable Applets:** Turns on support for ActiveX events. Only reason to turn it off might be faster startup time or improved stability (the Java Plugin Interface has some known bugs in certain versions of it, notably 1.5.0_01 through 1.5.0_14, that could cause crashes).
- **Enable SWT:** Turns on support for testing non-Eclipse based SWT applications.
- **Capture HTTP traffic:** Turns on a proxy to capture all HTTP(S) traffic and stores all headers in the event requests. Only reason to turn it off might be faster startup time or already having a proxy.
- **Enable Pathfinder integration:** this causes the browser to receive and decrypt Pathfinder payloads for Pathfinder-enabled applications.
- **Remote Applications Port:** specifies the port to use the control remote applications (Swing, SWT or WinForms). The default, 0, picks the port dynamically.
- **Maximum Bandwidth:** throttles request and response speed to simulate a network with the specified throughput. 0 means no limit.
- **Log messages to Output window:** self-explanatory.
- **Log messages to Output file:** self-explanatory. The log files go in the same directory as the DOM browser.
- **Log Level:** the level of log statements required to be logged in the Output window or file if turned on above.

Recording Settings:

General **Recording** Playback Environment

Recording Options

☒ Use context menus for filters and assertions ☐ Capture HTML changes

☐ Capture applet snapshots Capture Diff Size Bytes

☐ Compress recording files Capture Max Time ms

☒ Verbose recording Save Dialog Triggers

Recording Strategy

Use the following attributes (in this order):

| | |
|--|--|
| 1) <input type="text" value="id"/> | 5) <input type="text" value="<none>"/> |
| 2) <input type="text" value="name"/> | 6) <input type="text" value="<none>"/> |
| 3) <input type="text" value="<none>"/> | 7) <input type="text" value="<none>"/> |
| 4) <input type="text" value="<none>"/> | 8) <input type="text" value="index"/> |

Except those whose value matches:

For Java Swing and Applets, record:

☒ Component names ☒ Component text

Capture DOM Events

| | |
|--|---|
| <input type="checkbox"/> navigate | <input checked="" type="checkbox"/> docload |
| <input checked="" type="checkbox"/> focus | <input checked="" type="checkbox"/> dblclick |
| <input checked="" type="checkbox"/> mousedown | <input checked="" type="checkbox"/> change |
| <input checked="" type="checkbox"/> mouseup | <input checked="" type="checkbox"/> contextmenu |
| <input checked="" type="checkbox"/> mouseover | <input checked="" type="checkbox"/> drag/drop |
| <input type="checkbox"/> mouseout | <input type="checkbox"/> mousemove |
| <input checked="" type="checkbox"/> click | <input checked="" type="checkbox"/> keypress |
| <input checked="" type="checkbox"/> open/close | <input type="checkbox"/> ajax callback |

- **Use context menus for filters and assertions:** Override the right-click menu in web pages to popup a custom menu that offers choices about filters, assertions or debugging. Turn it off if your site already uses custom context menus.
- **Capture applet snapshots:** stores in the test the applet screenshots used in applet test editing (browse mode). Turn it off if the tests get too large.
- **Compress recording files:** keep that turned on (used for debugging).
- **Verbose recording:** Records events for which we could not detect an event handler (possibly DOM Level 2). Turn it off for most sites, try to turn it on if you notice some necessary event does not get recorded (happens using ExtJS for instance).
- **Capture HTML changes:** Store the HTML at any click or change event to make it easier for later editing.
- **Capture Diff size:** changes over this size will store the whole response, changes under this size will store the diff.
- **Capture Max time:** the diff above won't be captured if it takes more than this amount of time.
- **Save Dialog Triggers:** for pages with a non text/plain mime type, this decides whether to pop up a "Save As" dialog instead of performing a navigation if the target url matches the regular expression (by default, it does this for Excel, Word, PDF, Text, PowerPoint, Executable and Zip files).
- **Recording strategy:** which HTML attributes to use and in which order when generating XPath expressions.
- **Exclude ids matching:** any element id matching this regular expression won't be used in the auto-generated XPaths.
- **Record component names/text:** use java component names or text (or not) in the XPath generated when recording Java based applications.
- **Capture DOM events:** Turn off any type of DOM event you're not interested in capturing.

Playback Settings:

General Recording **Playback** Environment

Playback Options

☐ Suppress Javascript Dialogs

☐ Fail step on missing target.

☐ Synchronize Ajax calls.

☐ Use Hardware Input

☐ Send Commands Asynchronously

☐ Send Responses back to LISA

Min matching score 1

Default Browser Mode

☐ Internet Explorer

☐ Firefox

☐ Safari

Playback Speed

Wait multiplier: 0x

Timeout Settings

DOM Load Timeout 10000 ms

DOM Lookup Timeout 1000 ms

Applet Load Timeout 5000 ms

Applet Lookup Timeout 3000 ms

ActiveX Load Timeout 10000 ms

ActiveX Lookup Timeout 5000 ms

- **Fail step on missing target:** Generates a failure instead of the default warning when a target can not be found for a step (that includes browser window, frame, element). Overrideable in a test with `FAIL_MISSING`.
- **Synchronize Ajax Calls:** forces all ajax calls to be executed synchronously. Overrideable in a test with `SYNC AJAX`.
- **Use Hardware Input:** Replays tests by controlling keyboard and mouse. Overrideable in a test with `USE_HARDWARE`.
- **Send Responses back to LISA:** By default responses will not be sent back to LISA to improve performance and memory since it's usually not necessary.
- **Min Matching Score:** How many differences are allowed between a recorded XPath value and the best match found during playback. Overrideable in a test with `MIN_BACTRACKING`.
- **Default Browser Mode:** What browsers to enable by default during playback (pipe-delimited combination of IE, FF and WK). Overrideable in a test with `DEFAULT_BROWSER`.
- **Playback Speed:** The default speed to use to replay test as a multiplier of the recorded speed. 0x is usually the best choice. Overrideable in a test with `PLAYBACK_SPEED` (integer between and 10).
- **XXX Load Timeout:** The maximum amount of time waited before a new page/applet/control load before proceeding. Overrideable in a test with `XXX_LOAD_TIMEOUT`.
- **XXX Lookup Timeout:** The maximum amount of time waited to find an element on a page/applet/control before proceeding. Overrideable in a test with `XXX_LOOKUP_TIMEOUT`.

Environment:

| General | Recording | Playback | Environment |
|--|-----------|--|-------------|
| Key | | Value | |
| {{com.itko.lisa.stats.jmx.ITKAgentConnection}} | | LISA_JMX_ITKAGENT | |
| {{com.itko.lisa.stats.jmx.JBossConnection}} | | LISA_JMX_JBOSS3240\ | |
| {{com.itko.lisa.stats.jmx.JSR160RMICConnection}} | | LISA_JMX_JSR160RMI\ | |
| {{com.itko.lisa.stats.jmx.OracleASConnector}} | | LISA_JMX_OC4J\ | |
| {{com.itko.lisa.stats.jmx.Weblogic9Connector}} | | LISA_JMX_WLS9\ | |
| {{com.itko.lisa.stats.jmx.WeblogicConnector}} | | LISA_JMX_WLS6781\ | |
| {{com.itko.lisa.stats.jmx.WebsphereSOAPConnecti... | | LISA_JMX_WASSOAP5\ | |
| {{&Date}} | | LI-DD-DDDD\ | |
| {{&SSN}} | | DDD-DD-DDDD\ | |
| {{&Zip}} | | D*(5) | |
| {{alt.lisa.simulator.webservice.classpath}} | | \ | |
| {{apple.awt.brushMetalLook}} | | false | |
| {{apple.laf.useScreenMenuBar}} | | true | |
| {{BROWSER_HOME}} | | C:\itko\main\lisa-web\lisa-browser\bin\Debug | |
| {{com.apple.macos.smallTabs}} | | true | |
| {{com.apple.mrj.application.growbox.intrudes}} | | true | |
| {{com.itko.lisa.stats.jmx.ITKAgentConnection,com.i... | | | |
| {{com.itko.lisa.stats.jmx.JBossConnection,com.itko.li... | | | |
| {{com.itko.lisa.stats.jmx.WeblogicConnector,com.itk... | | | |
| {{EXAMPLES_HOME}} | | {{LISA_HOME}}/examples | |
| {{file.encoding}} | | UTF-8 | |
| {{Functional Report, com/itko/lisa/report/layout/Fun... | | | |
| {{gui.show.memory.status}} | | false | |
| {{ice.browser.cache.size}} | | 1048576 | |
| {{ice.browser.http.agent}} | | Mozilla/4.0 (compatible; MSIE 6.0; Windows N | |
| {{ice.browser.verbose}} | | false | |
| {{jasper.multi.report}} | | \ | |
| {{jasper.single.report}} | | \ | |
| {{javax.xml.parsers.DocumentBuilderFactory}} | | org.apache.xerces.jaxp.DocumentBuilderFacto | |
| {{javax.xml.parsers.SAXParserFactory}} | | org.apache.xerces.jaxp.SAXParserFactoryImpl | |
| {{javax.xml.transform.TransformerFactory}} | | org.apache.xalan.processor.TransformerFactor | |
| {{laf.default.url}} | | https://license.itko.com | |

This tab shows the list of global environment variables available to the browser, as read from **lisa.properties** and **local.properties**.

LISA Driver Settings:

In addition to the settings above that can be overridden from LISA in a test case or in the local.properties, the following are available:

- **lisa.browser.launch.timeout:** The amount of time allowed for a browser to launch (default is 10,000). Specify in local.properties.
- **lisa.browser.exec.timeout:** The amount of time allowed for a step to execute (default is 300,000). Specify in local.properties.
- **lisa.browser.max.instances:** The maximum number of browser instances per machine (default is 25). Specify in local.properties.
- **lisa.browser.client.user.single:** Whether to run staged browsers using the same user account as the currently logged-in users (default is true). Specify in local.properties.
- **lisa.browser.base.port:** The first port to use in the range of ports available to control web browsers (default is 0 for dynamic value). This normally does not need to be modified except in very secure environments that lock down some local ports. Specify in local.properties.
- **lisa.browser.client.user.<user name>=<encrypted password>:** when lisa.browser.client.user.single is set to false, browser instances will use the specified windows user accounts to run tests. If not enough user accounts are specified in this manner and the currently logged-in user has admin privileges, user accounts will be dynamically created to run the tests (and deleted at the end). To obtain an encrypted password, run the command line: `lisa_browser.exe -m encrypt -in <clear text>`.
- **lisa.browser.share.subprocess.state:** Whether to run a sub-process using the same browser instance as parent test (default is false). Specify in configuration of sub-process.
- **lisa.browser.swing.port:** The first port to use in the range of ports available to control Swing applications (default is 0 for dynamic value). Specify in local.properties.
- **lisa.browser.base.port:** The first port to use in the range of ports available to control web browsers (default is 0 for dynamic value). Specify in local.properties.

4. XPath syntax Reference

4. XPath syntax Reference

The XPath syntax supported to identify html elements in web 2.0 tests is a subset from the standard XPath specification, as described at [XPath 2.0 specification](#). Roughly speaking, the supported vs. not supported functionality is as follows:

- `<Tag1>/<Tag2>` to look for an element child is supported (e.g. `TR[1]/TD[2]`).
- `<Tag1>//<Tag2>` to look for an element descendant is supported (e.g. `TABLE//INPUT`).
- `..` to select the parent of an element is supported (e.g. `TABLE//TD[@id='abc']/../TD[1]`).
- `>>` and `<<` to select the next and previous sibling of an element respectively are supported - the standard axis names following-sibling or preceding-sibling can be used instead with `>>` and `<<` just being shorthand for it - (e.g. `TABLE//TD[@id='abc']/../>>/TD[1]`).
- `<Tag>[@AttributeName='AttributeValue']` to look for an element with the specified tag meeting the `AttributeName='AttributeValue'` condition is supported (e.g. `INPUT[@id='abc']`).
- `<Tag>[text()='innerText']` to look for an element whose inner text equals the specified one is supported (e.g. `TD[text()='Hello']`).
- `<Tag>[matches()='regex']` to look for an element whose inner text matches the supplied regular expression is supported as a proprietary extension (e.g. `TD[matches()='s*Hello*s']`).
- Other XPath axes are not supported.
- Only XPath expressions returning a unique node are supported. Expressions returning potentially multiple nodes will simply return the first one matching the expression.
- XPath functions (in the `fn:` namespace) and operators (arithmetic or boolean) are not supported.

5. Scripting Objects Reference

5. Scripting Objects Reference

Both in its debug window, and in its script filters, the DOM browser supports standard javascript and java (beanshell) syntax, but additionally built-in commands and some useful objects are defined with the following APIs:

_arg: A DOM element, or Java component, or ActiveX accessible element which is specified either as the element in a filter, or is the last recipient of an event in the Immediate debug window while recording or debugging.

lisa: A global javascript object available on every HTML page:

- public object dbQuery(string connectionString, string query, [Optional] bool cache)
-
- Given a database connection string and a SQL query (and optionally a boolean to indicate whether to cache the results), this will return a DataSet object (see reference below).
- public void disableBlur(bool disable)
-
- When passed true, this will disable all the onBlur event handlers on any subsequent page. This can make it easier to debug or add filters on popup menus that would otherwise disappear when losing focus.
- public string download(string url, string path)
-
- Will download the resource at the given url to the specified path. Useful to download files without having to go through the Save As dialogs.
- public object fileQuery(string path, [Optional] string query, [Optional] bool cache)
-
- Generates a DataSetWrapper (see reference below) from an Excel file and optionally a SQL query and a flag to cache the results. If no query is given, the whole first worksheet is returned in the dataset.
- public void fire(HTMLDocument document, string xpath, string evtName)
-
- Manually fires the supplied event on the element identified by the specified xpath.
- public string open(string path)
-
- Returns the contents of the file at the specified path into a string.
- public string pathfind(string xpath)
-
- Returns the node value of the `event.pathfinder.xml` selected by the specified xpath expression.
- public void print(object o)
-
- Prints a textual representation of the argument to the debug Output window.
- public object select(HTMLDocument document, string xpath)
-
- Returns the html element by specified xpath.
- public string showHandlers(HTMLDocument document, string xpath)
-
- Returns a list of all the javascript functions used as event handlers on the specified elements and all its parents.
- public string upload(string url, string path)
-
- Similar to download, but uploads a resource from the specified path to the specified url.
- public void evaluate(string code)
-
- Executes arbitrary C# .NET code. Can be used as last resort if desired functionality is not available.
- public bool compareFiles(string file1, string file2)
-
- Returns whether the contents of file1 and file2 are exactly identical.
- public double diff(string file1, string file2)
-

- Returns a value that indicates how different the files are (based on their length, average value and standard deviation).
- public void jsimport(HTMLDocument document, string jsfile)
-
- Automatically imports a js file into the specified page, making its variables and functions available to subsequent filters on the page. Note that any .js files in the browser directory will be automatically offered as an import in the filter drop-down of DOM event filters.
- public void javainport(string javafile)
-
- Automatically imports a java source file into the currently running java process, making its functions available to subsequent filters in the test. Note that any .java files in the browser directory will be automatically offered as an import in the filter drop-down of java event filters.

DataSetWrapper: A javascript class that encapsulate the result of SQL queries:

- public string asText()
-
- Returns a textual representation of the dataset where each cell is contained in brackets [].
- public int columnCount()
-
- The number of columns in the dataset.
- public string columns(int index)
-
- The name of the column as the specified index.
- public int count() or public int length()
-
- The number of rows in the dataset.
- public object rows(int index)
-
- Returns the DataRowWrapper (see reference below) object at the specified index.
- public string toString()
-
- A textual representation of the dataset.

DataRowWrapper: A javascript class that encapsulate a single row of a DataSetWrapper:

- public object cells(object indexOrName)
-
- Returns the value in the cell at the specified index or in the column with the specified name.
- public string toString()
-
- A textual representation of the row.

6. Command line Reference

6. Command line Reference

lisa_browser.exe supports the following options:

lisa_browser.exe [-m] [options]

Where:

-m recorder launches the recorder
 -m playback launches the debugger
 -m drive launches the load test console
 -m update launches the software update dialog

And options are one or more of the following:

-a autostarts the test specified by -f in playback or drive mode - default is false
 -n is the number of instances to autostart in drive mode - default is 0
 -f specifies a recording file to load on startup - default is none
 -c specifies a config file to load on startup - default is none
 -i makes the driver console invisible in drive mode - default is false
 -x autoexits the program on test end if it was autostarted - default is true
 -stg specifies a path to an xml file that lists instances along with test and config files to use

See [Load testing](#) for the syntax required by the -stg option.

PART 4 - LISA Web 2.0 - Videos

PART 4 - LISA Web 2.0 - Videos

http://www.itko.com/download/release/lisa_browser/docs/web20videos.html

The other information available at this location is not up-to-date. Please refer to the videos only.

NOTE: This site requires a valid login. If you have any questions, please contact your iTKO Sales Representative. Thank you.

PART 5 - LISA Web 2.0 - Repository

PART 5 - LISA Web 2.0 - Repository

The following topics are available...

1. Instructions
2. Always update
3. Update with major revisions changes
4. First time update (i.e. only if you're missing these files)

1. Instructions

1. Instructions

If you can not use the browser update mechanism for policy or connectivity reasons, you can download the updated files manually from this page. There are no installations steps.

Simply download the following files to the directory: <LISA install dir>\bin\browser. The only exceptions are: web20bridge.jar and jdbridge.jar that should go into the <LISA install dir>\lib directory
djbridge.dll and jdglue.dll that should go into the <LISA install dir>\bin directory

You usually have to download only a few files as indicated below. If you are unsure which ones to download, you can always download them all. Using manual download, you are responsible for backing up the files you overwrite in case you need to revert later.

2. Always update

2. Always update

- [lisa_browser.exe](#)
- [appletcallback.jar](#)
- [swingcallback.jar](#)
- [web20bridge.jar](#)

3. Update with major revisions changes

3. Update with major revisions changes

- [applet-monitor.dll](#)
- [dotnet-callback.dll](#)
- [dotnet-monitor.dll](#)
- [injector.dll](#)
- [lisa_browser.XmlSerializers.dll](#)
- [global-hook.dll](#)
- [jdbridge.jar](#)
- [djbridge.dll](#)
- [jdglue.dll](#)

4. First time update (i.e. only if you're missing these files)

4. First time update (i.e. only if you're missing these files)

- [Interop.SHDocVw.dll](#)
- [Interop.TidyATL.dll](#)
- [Interop.WebKit.dll](#)
- [Microsoft.mshtml.dll](#)

- [SciLexer.dll](#)
- [ScintillaNET.dll](#)
- [TidyATL.dll](#)
- [Microsoft.Office.Interop.Excel.dll](#)
- [Office.dll](#)
- [swt.jar](#)

PART 6 - LISA Web 2.0 - FAQ

Q: Can I test Flash and Flex applications?

A: The answer is "it depends".

First let's clarify Flash vs. Flex: Flex is an extra layer on top of Flash that helps programmers render certain layouts and controls more easily within a Flash VM. These days, non-Flex Flash applications are used almost only for video and/or animations so they are unusual in enterprise applications, so I'll focus on Flex in the remainder of the discussion.

Generally speaking there are 2 ways to test or automate Flex applications:

- using the so-called MSAA (accessibility interface) which provides ActiveX control clients with some level of visibility inside them. Support for the MSAA is pretty variable even within Flex applications, depending how they're coded and compiled. This means, in the best case that you can have visibility into every label, button, text field, drop down, etc...or in the worst case that the whole component (or large parts of it) are simply seen as graphics blobs. In the latter case all automation has to revert to coordinate based events and testing becomes very brittle, data can't be extracted - other than using OCR (optical character recognition) which some vendors do, with little success. To get an idea of how much detail is available in a given Flex app, you can download and use the `accexplorer32.exe` Microsoft tool.

This is the approach employed by the LISA browser: a Flash control (and thus a Flex one too) is embedded in a web browser as an ActiveX control or a plugin, so the the LISA browser has visibility into it inasmuch as it can see ActiveX controls and their subcontrols. This generally (but not always) is sufficient for small or simple controls embedded in HTML pages but rarely so for full-fledged apps that use the whole page or complex controls. Unfortunately there is no way to tell in advance and you'll have to try it to find out (or use `accexplorer32.exe` to have an idea).

The advantage of this technique is that it requires no code changes or recompilation of the Flex app to work, so it's been favored by many vendors in the past (including QTP) but its limitations are starting to push some vendors toward the second approach:

- using a Flex agent that allows clients to communicate directly with the Flash VM and its controls, variables and functions. This is the approach taken by vendors like GorillaLogic's FlexMonkey (or to a lesser extent Flex Selenium, which uses `ExternalInterface` objects to access the VM through javascript). When this approach is possible, it is much more robust and much preferable to the previous one, but as of Flex 3 it still required instrumentation and/or recompilation of the Flex application. This may no longer be needed now or in the future but I am not sure about the details.

We will probably invest into this alternative at some point but there is no ETA on this yet.