DevTest Solutions - 10.4

DevTest Solutions - Home

Date: 12-Nov-2018

A Broadcom Company
technologies

This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the “Documentation”) is for your informational purposes only and is subject to change or withdrawal by CA at any time. This Documentation is proprietary information of CA and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2018 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Table of Contents

DocOps Search Tips 38

How do I search in DocOps? 38

How do I refine my search? 39

Can I search on exact phrases? 40

Can I search for content with special characters? 41

Why can I not find what I am looking for? 41

Release Notes 42

Release Comparison 42

New Features and Enhancements 46

- Identity and Access Manager (IAM) 46
- Authentication for Enterprise Dashboard 47
- Virtual Service Catalog 47
- OpenAPI 3.0 48
- Jenkins Plug-in 48
- Apache Kafka 48
- Collection of Customer Experience Improvement Data 48
- Improved Testing Execution Reports 49

Preview Features 49

- Enable and Disable Features 49

Resolved Issues 50

- Application Test 51
  - Stop Test or Kill Test from Portal 51
  - Can't Specify Concurrent Capacity in MarInfo 51
  - List for CVS Monitored Events Badly Sorted 51
  - ANDROID_HOME Error is Appearing in Logs 51
  - BLOB Data Type is not Supported in JDBC Step 51
  - JDBC Assets Getting Changed in Drop Down for a Different project.config File 51
  - Ctrl+s Save Icon in the Workstation are Not Working When Trying to Create a New Suite 52
- Service Virtualization 52
  - VS from WSDL Issue 52
  - Inspection View Recent Requests Has Two Entries and Incorrect NO MATCH FOUND 52
  - Workstation Search not Functioning as Expected for Transaction IDs and Tags or Values in the Request/Response 52
  - vsBuilderSessions File Corruption Prevents VSE From Starting 52
  - Create VS from PCAP Not Working as Expected 52
  - VS Not Getting Filtered Using Group Tag Name 53

JSON Response, Invalid Property ID ... 53
Sample Match Script Failing ... 53
Getting Null Pointer Exception Error using getStateValue in VS ResponseName ... 53
Version compatibility issues in DevTest 10.2 ... 53
Unable to Set msg.putDateTime in Response Meta Data ... 53
Unable to Kill/Stop Tests from Portal ... 53
UUID Fields are Ignored During Virtual Service Creation Process with Swagger 2.0 ... 54
General/Common Components ... 54
Console Does Not Retain the Service Name of the Service When Downloading the MAR ... 54
High Open File Descriptor Count in Portal ... 54
Excessive Memory Requirement Using Portal > Monitoring > Tests ... 54
Java Client Can't Obtain Kerberos Authentication When Passing Through DevTest ... 54
lisa-virtualize-invoke Does Not Work and Gives Authorization Fails ... 55
Unable to List Virtual Services in Virtual Service Catalog ... 55
Consolidated Elasticsearch ... 55
SSL Configuration ... 55
Reporting Period End Dates in Export Usage Audit Data ... 55
Upgrade Schema V15 for DevTest 10.3 Fails for Dradis ... 56
Known Issues ... 56
SSL Support on AIX ... 56
Cumulative Transactions in Advanced Metrics Graph in Enterprise Dashboard ... 56
Virtual Service Environment Detail Table in Enterprise Dashboard Shows Transaction Count Inconsistency ... 57
Bar Graphs in The Enterprise Dashboard ... 57
Viewing Virtual Service Catalog and Identity and Access Manager on Internet Explorer ... 57
Ampersand Character in Resource Names ... 58
Searching for Long Strings in a Virtual Service ... 58
Data-Driven Virtual Services ... 58
Oracle New Install Issue ... 59
dradis.properties File Required ... 59
Demo Server Does Not Work on Solaris ... 59
Enabling or Disabling the Enhanced VS Recorder Can Cause Errors ... 59
Perfmon Restrictions ... 59
Server Component Installation Restrictions ... 59
Windows: ... 59
Solaris: ... 60
OS X: ... 60
Using Data Files in DevTest Portal ... 60
Testing Android Applications on OS X ... 60
Incorrect Command in Readme file ... 60
Enterprise Dashboard Time ... 61
Standalone Demo Server ... 61
End of Service Announcement ... 61
End of Life Features ... 61

End of Life Feature as of August 1, 2017 ... 61
End of Life Features as of DevTest 10.0 ... 61
End of Life Features as of DevTest 9.5 ... 61
End of Life Features as of DevTest 9.0 ... 62
End of Life Features as of DevTest 8.5 ... 62
End of Life Features as of DevTest Solutions 8.2 ... 62
End of Life Features as of DevTest Solutions 8.0.2 ... 62
End of Life Features as of DevTest Solutions 8.0 ... 62
End of Life Features as of LISA 7.5 ... 63
End of Life Features as of LISA 7.1 ... 63
End of Life Features as of LISA 6.0.9 ... 63
DevTest Portal Functionality ... 64
CA Application Test ... 64
CA Service Virtualization ... 64
DevTest Common Components ... 65
Third Party Acknowledgments ... 65
Product Accessibility Features ... 77

Installing ... 78
Preinstallation ... 78
System Requirements ... 79
Operating System Requirements ... 79
Supplying Your Own JVM ... 81
DevTest Server Requirements ... 84
DevTest Workstation for CA Application Test Requirements ... 85
CA Service Virtualization Requirements ... 85
Application Insight Requirements ... 85
Identity and Access Manager Requirements ... 85
Virtual Service Catalog Requirements ... 86
Communication Requirements ... 86
Database Requirements ... 86
Supported Browsers ... 89
Supported Languages ... 89
Application Insight Support Matrix ... 90
DevTest Versions ... 90
Application Insight Agents ... 91
Java SE and EE JVMs ... 91
Application Servers ... 91
Visible Transaction Protocols ... 92
Virtualization Features ... 93
Baseline Features ... 95
DevTest Server Components ... 95
DevTest Process Relationships ... 98
DevTest Solutions Architecture ... 101

Architecture for Testing ... 101
Architecture for Service Virtualization ... 103
Architecture for Application Insight ... 103
Enterprise Dashboard Architecture ... 104
Virtual Service Catalog ... 104
Data Flow in DevTest Server for CA Application Test and CA Service Virtualization ... 105
Download DevTest Solutions Installers ... 107
DevTest Installation Overview ... 108
Post-Installation Steps ... 109
Installation Options ... 110
Destination Directory ... 110
Typical Configurations ... 111
Options for Installing Required Processes (Windows) ... 112
How License Activation Works ... 112
Installing and Configuring DevTest Server ... 113
Install DevTest Server on Windows ... 113
Install DevTest Server on UNIX ... 117
Install DevTest Server on OS X ... 121
Activate the Registries ... 124
Verify Registry Activation ... 125
Post-Installation ... 126
Using an HTTP/S Proxy Server - DevTest Server ... 126
Running Components on Different Systems ... 127
Calculate Simulator Instances ... 128
Load and Performance Server Sizing ... 128
Sample Performance Benchmarks for Basic Service Virtualization ... 129
Sample Performance Benchmarks for IBM MQ Service Virtualization ... 133
Using DevTest Workstation with Your Java Environment ... 136
Change the Default Project Home ... 136
Project Directory Structure ... 137
Uninstall DevTest Server ... 137
Installing DevTest Workstation ... 138
Install DevTest Workstation on Windows ... 138
Install DevTest Workstation on UNIX ... 140
Install DevTest Workstation on OS X ... 142
Using an HTTP/S Proxy Server - DevTest Workstation ... 144
Environment Settings ... 146
Installing the Demo Server ... 146
Running Forward Cars ... 148
Verifying the DevTest Installation ... 149
Start the DevTest Processes or Services ... 150
Start Order Sequence ... 150
Ways You Can Start DevTest Server ... 151
Log in as the Standard Super User ... 151
Create a User with the Super User Role ... 151

Access the User Interfaces ... 152
Enterprise Dashboard ... 152
Portal ... 152
Workstation ... 153
Installing Virtual Service Catalog ... 153
Install Virtual Service Catalog on Windows ... 153
Install Virtual Service Catalog on UNIX ... 155
Install Virtual Service Catalog on OS X ... 157
Installing Integration Tools ... 158
Install Performance Monitor (Perfmon) ... 158
Install and Configure SNMP ... 159
SNMP support on UNIX ... 159
SNMP support on Windows ... 160
Run TCPMon ... 161
Using TCPMon as an Explicit Intermediary ... 161
Using TCPMon as a Request Sender ... 162
Install DevTest ALM Plug-in ... 162
Prerequisites ... 162
Installation ... 163
Post Installation Files ... 163
InstallationUtility Command-Line ... 164
Using the HP ALM Client to Run Test Cases ... 165
Set Up the SAP System Landscape Directory ... 166
Manually Register DevTest with SLD ... 167
Import the DevTest SLD XML File ... 167
Configure CA Agile Central ... 168
Installing DevTest Solutions with a Silent Install ... 169
Sample Response File (Windows) ... 171
Sample Response File (Linux) ... 172
Docker Containers ... 173
Docker Prerequisites ... 173
Connecting to Oracle Databases in Docker Containers ... 173
Build Baseline Docker Images ... 174
Docker Images ... 174
Example ... 175
Docker Commands to Start DevTest Components in Their Own Containers ... 175
Starting Enterprise Dashboard Container ... 175
Starting Registry Container ... 176
Starting Broker Container ... 176
Starting Coordinator Container ... 176
Starting Simulator Container ... 176
Starting VSE Container ... 176
Starting Portal Container ... 176
Preparing Your DevTest Upgrade ... 177
Upgrading the Authentication Component ... 177

Migrating Authentication Information ... 178
What Gets Migrated? ... 179
Reports ... 179
Video ... 179
Upgrade DevTest ... 180
After You Upgrade ... 181
Verify the Authentication Migration ... 181
Patches Folder ... 182
Verify Services and Applications ... 182
Open DevTest Workstation ... 182
Additional Upgrade Details ... 182
Product Changes ... 183
Licensing ... 183
Enterprise Dashboard and Registry ... 183
Using DDL Scripts to Upgrade Enterprise Dashboard ... 184
Mixing Component Versions ... 185
Property Files ... 185
Custom Folders ... 185
Custom JAR Files ... 185
Memory Allocation Settings ... 186
Database Safeguards ... 186
Migrating the SDK ... 186
Upgrading SDK Components ... 186
SDK Documentation ... 187
Java Version Notes ... 187
Assertion Updates ... 188
8.X to 9.X ... 188
9.X to 10.X ... 188
Companion Updates ... 188
8.X to 9.X ... 188
9.X to 10.X ... 188
Filter Updates ... 188
8.X to 9.X ... 188
9.X to 10.X ... 188
Step Updates ... 189
8.X to 9.X ... 189
9.X to 10.X ... 189
Data Protocol Updates ... 189
8.X to 9.X ... 189
9.X to 10.X ... 189
Transport Protocol Updates ... 189
8.X to 9.X ... 189
9.X to 10.X ... 189
Body Carrier Updates ... 190
8.X to 9.X ... 190
9.X to 10.X ... 190

Request Updates 190
8.X to 9.X 190
9.X to 10.X 190
Response Updates 190
8.X to 9.X 190
9.X to 10.X 190
Transient Response Updates 191
8.X to 9.X 191
9.X to 10.X 191
TCP Delimiter Updates 191
8.X to 9.X 191
9.X to 10.X 191
API Notes 191
8.X to 9.X 191
9.X to 10.X 191

Getting Started 192
CA Application Test 192
CA Service Virtualization 192
Role in the Continuous Delivery Ecosystem 192
DevTest Portal 193
Open DevTest Portal 193
Navigating DevTest Portal 194
Navigation Menu 194
Home Page 196
All Resources Window 197
Manage Projects 198
Manage Tabs 199
Customize the Home Page Dashboard 200
Update Preferences 200
Change Your Password 200
Enable and Disable Help Tips 201
Select Default Language 201
CA Application Test Tutorials 201
Tutorial 1 - Projects, Test Cases, and Properties 202
Step 1 - Start DevTest Workstation 203
Step 2 - Create a Project 203
Step 3 - Create a Test Case 203
Step 4 - Add a Property to the Project Configuration 204
Step 5 - Add a Test Step 204
Step 6 - Add a Log Message 204
Step 7 - Add a Second Log Message 205
Step 8 - Run the My Output Log Message Step 206
Step 9 - Observe Property Values 206

Step 10 - Run the Output Log Message Step 206
Tutorial 1 - Review 207
Tutorial 2 - Data Sets 207
Step 1 - Create a Data Set 208
Step 2 - Create a Test Case 208
Step 3 - Add a Property to the Project Configuration 208
Step 4 - Add a Test Step for Output Log Message 209
Step 5 - Create Another Output Log Message Step 209
Step 6 - Execute the Test 210
Step 7 - Add the Data Set 210
Step 8 - Change the Data Set Behavior 211
Tutorial 2 - Review 212
Tutorial 3 - Filters and Assertions 212
Step 1 - Create a Test Case from an Existing Test Case 212
Step 2 - Change Action of Test Step 213
Step 3 - Add an Assertion 213
Step 4 - Test the Assertion 215
Step 5 - Add a Filter 216
Step 6 - Test the Filter 217
Tutorial 3 - Review 218
Tutorial 4 - Manipulate Java Objects (POJOs) 219
Step 1 - Create a Test Case 219
Step 2 - Create a Dynamic Java Execution Test Step 219
Step 3 - Make a Call on the Java Object 222
Step 4 - Add an Inline Filter 225
Step 5 - Verify the Property Created 226
Tutorial 4 - Review 227
Tutorial 5 - Run a Demo Server Web Application 227
Step 1 - Start the Web Application 228
Step 2 - Log in to the Web Application 228
Step 3 - Create an Account 229
Step 4 - Close an Account 231
Step 5 - Log Out from the Web Application 232
Tutorial 5 - Review 232
Tutorial 6 - Test a Website 232
Tutorial 6 - Part A - Record the Test Case 233
Tutorial 6 - Part B - Run the Test Case 238
Tutorial 6 - Part C - Modify Request Test Steps 239
Tutorial 7 - Test an Enterprise JavaBean (EJB) 245
Step 1 - Create a Test Case 245
Step 2 - Create a Configuration 246
Step 3 - Add an EJB Test Step 246
Step 4 - Connect to the Server 247
Step 5 - Locate the EJB Interface 247
Step 6 - Configure the EJB 249
Step 7 - Add an Assertion 251

Step 8 - Verify the Method Execution ... 252
Step 9 - Add Another EJB Test Step ... 253
Tutorial 7 - Review ... 254
Tutorial 8 - Test a Web Service ... 255
Step 1 - Create a Test Case ... 255
Step 2 - Add a Web Service Execution (XML) Test Step ... 255
Step 3 - Create a Web Service Client ... 256
Step 4 - Execute the Web Service Request ... 258
Step 5 - View the Request and Response ... 258
Tutorial 8 - Review ... 259
Tutorial 9 - Examine and Test a Database ... 259
Step 1 - Create a Test Case ... 260
Step 2 - Add Database Properties to the Configuration ... 260
Step 3 - Add a SQL Database Execution (JDBC) Test Step ... 261
Step 4 - Connect to the Database ... 262
Step 5 - Execute a SQL Query ... 263
Step 6 - Add an Assertion ... 264
Step 7 - Run the Test Case ... 266
Step 8 - Change the Assertion ... 266
Step 9 - Add a Filter ... 267
Step 10 - Test the Filter and Assertion ... 268
Tutorial 9 - Review ... 269
Tutorial 10 - Stage a Quick Test ... 270
Step 1 - Open the Test Case ... 270
Step 2 - Review the Test Case ... 270
Step 2 - Part A - Run a Quick Test ... 271
Step 2 - Part B - View Generated Reports ... 273
Tutorial 10 - Review ... 274
Tutorial 11: Run the Forward Cars Demo Application ... 274
Step 1 - Start Forward Cars ... 275
Step 2 - Perform an Inventory Search ... 275
Step 3 - Change the Enterprise Service Bus Location ... 276
Step 4 - Add a Car ... 277
Step 5 - Edit a Car’s Information ... 278
Step 6 - Delete a Car ... 278
Step 7 - Look up a Car’s History Report Through the VIN ... 279
Step 8 - Log in to Forward Cars ... 279
Step 9 - Apply for an Auto Loan ... 280
Step 10 - Check on Your Loan Status ... 280
Step 11 - Log Out from Forward Cars ... 280
Tutorial 11 - Review ... 280
CA Service Virtualization Tutorials ... 281
Tutorial 1: REST-based Virtual Service ... 281
Step 1 - Start DevTest Portal ... 281
Step 2 - Start VSE ... 282
Step 3 - Start Demo Server ... 282

Step 4 - Deploy a REST-based Virtual Service 283
Step 5 - Monitor the REST-based Virtual Service 283
Step 6 - Run a Baseline Test Against the REST-based Virtual Service 283
Step 7 - Run a Baseline REST-based Test Against the Live Forward Cars Application 284
Tutorial 2: JMS Virtual Service Using DevTest Portal 284
Step 1 - Start DevTest Portal 284
Step 2 - Start VSE 285
Step 3 - Copy JAR Files 285
Step 4 – Configure Forward Cars to Use Proxy JMS Queues 285
Step 5 - Deploy a Loan Application Virtual Service 286
Step 6 - Monitor the Virtual Service 286
Step 7 – Apply for a Loan Application in Forward Cars 287
Step 8 - Check on Your Loan Status 287
Tutorial 3: JMS Virtual Service Using DevTest Workstation 287
Step 1 – Copy JAR Files 288
Step 2 – Configure Forward Cars to Use Proxy JMS Queues 288
Step 3 - Open DevTest Workstation 289
Step 4 - Record a JMS Session to Apply for a Loan in Forward Cars 289
Step 5 – Apply for a Loan Application in Forward Cars 290
Step 6 - Complete the JMS Recording in DevTest Workstation 290
Step 7 - Change the Forward Cars Application ID to a Magic String 291
Tutorial 4: HTTP/S Virtual Service 291
Prerequisites 292
Process for Creating and Testing a VSI 292
Step 1 - Start DevTest Workstation and Portal 292
Step 2 - Start VSE 293
Step 3 - Start Demo Server 293
Step 4 - Run a Test Case 293
Step 5 - Create a Config File 294
Step 6 - Activate Config File 295
Step 7 - Configure the VSE Recorder 295
Step 8 - Record the Test Case 297
Step 9 - Deploy the VSM 298
Step 10 - Test Against the VSM 299
Review the Tutorial 300
Tutorial 5: Virtual Service from WSDL 300
Step 1 - Start DevTest Portal 300
Step 2 - Start VSE 301
Step 3 - Create a Credit Inquiry Virtual Service 301
Step 4 - View the Virtual Service 302
Step 5 - Configure Forward Cars to Point to Virtual Service 302
Step 6 - Run the Baseline Test 303
Step 7 – Apply for a Loan Application in Forward Cars 303
Step 8 - Reset the Configuration File 304
Tutorial 6: Virtual Service from Swagger (OpenAPI) 304
Step 1 - Start DevTest Workstation 304

Step 2 - Start DevTest Portal ... 305
Step 3 - Start the Virtual Service Environment ... 305
Step 4 - Create a Virtual Service from Swagger ... 305
Step 5 - Access the Monitor in DevTest Portal ... 307
Step 6 - Configure Forward Cars to Point to Virtual Service ... 307
Step 7 - Perform Actions in Forward Cars ... 308
Review ... 308
Application Insight Tutorial ... 309
Prerequisites ... 309
Step 1 - Configure the Capture Levels ... 309
Step 2 - Generate Transactions from the Demo Application ... 310
Step 3 - View the Transactions in the Analyze Transactions Window ... 311
Glossary ... 312
assertion ... 312
asset ... 312
audit document ... 312
baseline ... 312
companion ... 313
configuration ... 313
conversation tree ... 313
coordinator ... 313
data protocol ... 313
data set ... 313
de-identify ... 314
event ... 314
filter ... 314
group ... 314
Interactive Test Run (ITR) ... 314
magic date ... 314
magic string ... 314
match tolerance ... 315
metrics ... 315
Model Archive (MAR) ... 315
Model Archive (MAR) Info ... 315
navigation tolerance ... 315
node ... 315
path ... 315
path graph ... 315
project ... 316
property ... 316
quick test ... 316
registry ... 316
service image ... 316
simulator ... 316
staging document ... 316
stitching ... 316

subprocess ... 317
test case ... 317
test step ... 317
test suite ... 317
think time ... 317
transaction frame ... 317
virtual service model (VSM) ... 317
virtual service environment (VSE) ... 317

Using ... 318

Using CA Application Test ... 319
The Registry ... 320
Start the Registry ... 321
Create a Named Registry ... 321
Change the Registry ... 321
Coordinator Server ... 322
Create a Coordinator Server ... 322
Monitor Coordinator Servers ... 323
Simulator Server ... 323
Create a Simulator Server ... 323
Monitor Simulator Servers ... 324
Command-Line Utilities ... 325
Using DevTest Portal with CA Application Test ... 325
Create an API Test ... 326
Monitor Tests ... 329
Monitor with CVS (Continuous Validation Service) ... 335
Manage Test Artifacts ... 343
Testing Reports ... 347
Using DevTest Workstation with CA Application Test ... 371
Open DevTest Workstation ... 372
Building Test Cases ... 390
Building Documents ... 502
Working with Model Archives (MARs) ... 530
Running Test Cases and Suites ... 538
Reports ... 588
Recorders and Test Generators ... 591
Using BeanShell in DevTest ... 595
Class Loader Sandbox Example ... 598
Generating DDLs ... 598
Using CA Service Virtualization ... 599
Service Virtualization Overview ... 599
Product Configurations ... 600
SV Installation and Configuration ... 600
System Requirements for CA Service Virtualization ... 600

Software Directory Structure and Files ... 601
Install CA Service Virtualization ... 601
Configure CA Service Virtualization ... 602
Install the Database Simulator ... 603
DDLs for Major Databases ... 605
How To Run Multiple VSE Services on a Single Server ... 610
CA Service Virtualization Concepts ... 611
Basic Workflow ... 611
CA Service Virtualization Components ... 612
Live Traffic Recording ... 613
Request/Response Pairs ... 616
Engineering Specifications ... 617
Virtual Service Models ... 618
Service Images ... 618
How Virtualization Works ... 619
VSE Transactions ... 620
Match Tolerance ... 626
Magic Strings and Dates ... 628
Parsing URIs in HTTP Requests ... 633
Using DevTest Portal with CA Service Virtualization ... 636
Create Virtual Services by Recording ... 637
Create Virtual Services from Request/Response Pairs ... 656
Create Virtual Services from Swagger (OpenAPI) ... 657
Create Virtual Services from WSDL ... 660
View and Edit VSE Transactions ... 661
Select Data Protocols ... 664
Configure Data Protocols ... 665
Select and Configure Data Protocols - Enhanced VS Recorder ... 668
Save a Virtual Service ... 669
Create Data-Driven Virtual Services ... 671
Virtualize JDBC Traffic ... 679
REST Data Protocol Handler ... 687
JSON Data Protocol Handler ... 692
Edit Virtual Services ... 693
Deploy Virtual Services ... 711
Monitor Virtual Services ... 713
Create Copybook Bundles ... 724
Manage Copybook Bundles ... 731
Create Baseline from Virtual Service ... 732
Virtual Service Metrics Reports ... 733
Using DevTest Workstation with CA Service Virtualization ... 741
Creating Service Images ... 742
Editing Service Images ... 881
Editing a VSM ... 906
De-identifying Data ... 939
Virtualizing a Service ... 940

Track Transactions ... 943
VSE Manager - Manage and Deploy Virtual Services ... 943
Install VSE Manager ... 944
Configure VSE Manager ... 944
Import a LISA Project ... 945
Use the VSE Manager View ... 945
Properties View Integration ... 946
VSE Commands ... 947
VSE Manager Command - Manage Virtual Service Environments ... 947
ServiceManager Command - Manage Services ... 949
ServiceImageManager Command - Manage Service Images ... 949
VirtualServiceEnvironment Commands ... 954
Java Agent VSE Properties ... 955
Using Application Insight ... 956
Getting Started with Application Insight ... 957
Application Insight Prerequisites ... 957
Application Insight Supported Platforms ... 958
Download DevTest Java Agent Files ... 959
Business Transactions and Transaction Frames ... 960
SAP NetWeaver Notes for Application Insight ... 961
Agent Configuration ... 961
CICS Agents ... 962
Agent Status Indicator ... 963
VSE Mode Indicator ... 963
Filter Agents ... 964
Stop and Start Agent Transaction Capture ... 965
View Performance Data ... 965
View CICS Virtualized Programs ... 966
Configure Capture Levels for Agents ... 967
View Agent and Broker Information ... 968
View Broker Notifications ... 968
View and Update Properties ... 969
Configure Agent Logging ... 970
Manage Java Intercepts ... 971
De-identifying Data in Application Insight ... 971
Upgrade an Agent in DevTest Portal ... 972
Upgrade an Agent Manually ... 973
Delete an Offline Agent ... 973
Create and Manage Agent Groups ... 973
Analyzing Business Transactions ... 976
Graphical View and List View ... 976
Search and Filter Transactions ... 977
Exclude Data from Agent Capture ... 985
View Transaction Details ... 986
Pin and Unpin Transactions ... 993
Share Point of Interest ... 994

Share a Link ... 994
Label the Links Between Transaction Frames ... 995
Merge Repeated Paths ... 995
View Usage Counts for Identical and Partial Transaction Paths ... 997
View Producer and Consumer Relationship ... 998
Transaction Stitching Rules ... 1000
Delete Transactions ... 1001
Export and Import Paths ... 1003
Export Path Diagram to PDF ... 1005
Shelve Transactions ... 1005
Generate Artifacts for an Agent ... 1008
Work with Defects ... 1010
Using the Shelf ... 1011
Shelf Dialog ... 1012
Shelf Popup ... 1012
Work with Transactions in the Shelf ... 1013
Create and Manage Tickets ... 1016
Disabling the Tickets Feature ... 1016
Create Tickets in an Application ... 1017
Email Settings for New Tickets ... 1018
Send Tickets to HP ALM - Quality Center ... 1019
Manage a Ticket ... 1020
Configure Browsers to Support Screenshots ... 1025
Creating Virtual Services ... 1025
Consolidation of Transactions When Creating Virtual Services ... 1026
Create Virtual Services from CICS Transactions ... 1027
Create Virtual Services from EJB Transactions ... 1028
Create Virtual Services from Java Transactions ... 1030
Create Virtual Services from JCA Transactions ... 1032
Create Virtual Services from JDBC Transactions ... 1033
Create Virtual Services from JMS Transactions ... 1034
Create Virtual Services from REST Transactions ... 1036
Create Virtual Services from RMI Transactions ... 1037
Create Virtual Services from SAP ERPConnect Transactions ... 1038
Create Virtual Services from SAP IDoc Transactions ... 1041
Create Virtual Services from SAP JCo Transactions ... 1043
Create Virtual Services from TIBCO ActiveMatrix BusinessWorks Transactions ... 1047
Create Virtual Services from Web HTTP Transactions ... 1050
Create Virtual Services from Web Service Transactions ... 1051
Create Virtual Services from webMethods Transactions ... 1053
Create Virtual Services from WebSphere MQ Transactions ... 1055
VRS Files in Application Insight ... 1056
Creating Baselines ... 1056
Baseline Creation Approaches ... 1056
Workflow ... 1057
Baseline Types ... 1057

EJB Baselines 1059
JMS Baselines 1065
REST Baselines 1067
TIBCO BusinessWorks Baselines 1072
TIBCO Enterprise Message Service Baselines 1074
Web HTTP Baselines 1074
Web Service Baselines 1078
webMethods Baselines 1082
WebSphere MQ Baselines 1085
Generic Baselines 1089
Database Validation 1092
Test Templates 1095
Run a Baseline 1097
Consolidated Baseline Tab 1097
Baseline Results Panel 1099
Dynamic Responses and Transaction Frame Steps 1100
Create Data Sets 1100
Create Request/Response Pairs from Transaction Frames 1102
Document Transactions 1104
Document a Transaction for a Manual Test 1104
Troubleshooting for Documenting Transactions 1113
Native MQ Application Insight 1113
WebSphere MQ User Privileges for Native MQ Application Insight 1115
Create the ITKO_PATHFINDER Queue 1115
Install the API Exit for Native MQ Application Insight 1115
Configure the API Exit for Native MQ Application Insight 1116
Configure the Agent Properties for Native MQ Application Insight 1117
Generate Transactions from the ITKO_PATHFINDER Queue 1119
Stop and Resume Message Mirroring 1120
Troubleshooting Native MQ Application Insight 1121
Application Insight Command-Line Tool 1122
Search Criteria 1123
Querying and Display 1124
Export and Import 1125
Baselines 1125
Virtualization Artifacts 1127
Agent Condition 1127
Capture Levels 1128
Miscellaneous 1128
Example: Display Root Transaction Frames 1129
Example: Display Transaction Frame Hierarchy 1129
Example: Generate a Baseline Suite 1129
Example: Generate a Stateful Baseline 1129
Example: Generate a Baseline with a Template 1129
Example: Generate Virtualization Artifacts 1129
Example: Check the Agent Condition 1130

Example: Configure the Capture Level ... 1130
Example: Search Transaction Frame ... 1130
Example: Generate Large Data Set File ... 1130
Example: Import Recorded Transactions ... 1130
VS Traffic to CA Application Insight Companion ... 1130
DevTest Extension for Google Chrome ... 1131
Installing and Configuring the DevTest Extension for Google Chrome ... 1132
Using the DevTest Extension for Google Chrome ... 1135
CAI Top N and Metrics Reports ... 1140
CAI Inventory Reports ... 1140
CAI Top N Reports ... 1141
Application Insight for CICS Walkthrough ... 1141
Configure the Mainframe Bridge Properties ... 1142
Start the DevTest Components ... 1142
Enable Transaction Capture for CICS Agents ... 1142
Verify Enablement ... 1143
Run the CICS Transaction ... 1143
View Transaction Data in DevTest Portal ... 1144
Create a Virtual Service ... 1145
Best Practices for Application Insight for CICS Agents ... 1146
Integrate with CA API Management ... 1148
Configure and Deploy Policy ... 1148
Capture Request and Response Data ... 1149
Using CA Application Trace Kit ... 1150
Start the Application Trace Kit ... 1150
Application Trace Kit Layout ... 1151
Live Paths ... 1152
Saved Paths ... 1154
Configure the Amount of Data Captured ... 1154
Configure Agent Properties ... 1155
View and Configure Logging ... 1155
View and Manage Classes ... 1156
Manage Files ... 1157
Execute Remote Commands from the Terminal ... 1157
CPU Profiling ... 1157
Reinvoke a Transaction ... 1158
CA Application Trace Kit Database Tools ... 1159
Explore the DevTest Database ... 1159
Explore Databases Connected to Agents ... 1160
Creating Java Agent Extensions ... 1160
Create, Build, and Upload the Extension ... 1161
Using Extensions to Reduce Noise ... 1161
Live Instances and OQL Queries ... 1162
Live Instances ... 1162
Query the Heap with OQL ... 1162
Using Virtual Service Catalog ... 1163

Start Virtual Service Catalog 1163
View Virtual Services 1164
View Virtual Service Details 1166
Configure Virtual Service Catalog 1167
Configure the Identity and Access Manager Instance 1168
Configure Connection Settings 1168
Configure Custom Ports 1169
Configure Logging Output to a File 1170
Using the APIs 1170
LISA Invoke API 1171
Run Test Cases with LISA Invoke 1171
Run Test Suites with LISA Invoke 1173
Run Model Archives with LISA Invoke 1173
Invoke the Callback Service with LISA Invoke 1174
LISA Invoke Responses 1174
REST Invoke API 1175
Explore the REST Invoke API from the Web Interface 1175
Change the Execution Mode Using the REST Invoke API 1176
Create a Virtual Service from R/R Pairs Using the REST Invoke API 1177
Test Invoke API 1179
Overview 1180
Execute a Test 1180
Retrieve an Indicated Test Run by Providing "Run Unique Id" 1181
Delete a Test Run by Providing "Run Unique Id" 1182
Retrieve all Test Runs 1182
Execute a Suite 1183
Retrieve an Indicated Suite Run by Providing "Run Unique Id" 1184
Delete a Run Suite by providing "Run Unique Id" 1184
Retrieve all Suite Runs 1184
Retrieve Test Reports 1185
Retrieve a Test Report for a Given "Test Run Unique Id" 1190
Retrieve all the Cycles for a Test Report for a Given "Test Run Unique Id" 1192
Retrieve a Cycle for a Test Report for a Given "Test Run Unique Id" and "Cycle Unique Id" .. 1193
Retrieve Test Reports 1199
Retrieve Suite Reports 1200
Retrieve a Suite Report for a Given "Suite Run Unique Id" 1203
Retrieve all Tests for a Suite Report Given "Suite Run Unique Id" 1204
Retrieve a Test Given "Test Run Unique Id" and for Suite Report Given "Suite Run Unique Id" 1208
Retrieve all Cycles for a Test Given "Test Run Unique Id" and for a Suite Report Given "Suite Run Unique Id" 1211
Retrieve a Given Cycle for "Cycle Unique Id" , for a Test Given "Test Run Unique Id" , and for a Suite Report Given "Suite Run Unique Id" 1215
Virtual Service Invoke API 1222
Overview 1223

Create a Virtual Service By Recording ... 1225
Add Transactions to an Existing Virtual Service Through Recording ... 1230
Create a Virtual Service from RR Pairs or a Specification Document (WSDL) ... 1233
Add Transactions to an Existing Virtual Service from RR Pairs or a Specification Document (WSDL) ... 1242
Export a Virtual Service as MAR File ... 1250
Using the SDK ... 1252
Examples and API Documentation ... 1252
The Integration API ... 1252
Integration API Concepts ... 1252
Integration Flow ... 1253
The Integration Process ... 1253
Integrating Components ... 1253
Integrate Server-Side Components ... 1254
Collect Transaction Information ... 1256
Integrators ... 1257
Handle Integrated Output ... 1257
Testing Integrated Components ... 1258
Integration Filters ... 1258
Integration Assertions ... 1260
Extending DevTest Solutions ... 1262
Extension Concepts ... 1262
Extending Test Steps ... 1267
Custom Java Test Steps ... 1268
Native Test Steps ... 1272
Extending Assertions ... 1273
Create an Assertion ... 1273
Deploy an Assertion ... 1275
Define and Test an Assertion ... 1276
Extending Filters ... 1276
Create a Filter ... 1276
Deploy a Filter ... 1279
Define and Test a Filter ... 1279
Custom Report Generators ... 1279
Create a Report Generator ... 1279
Deploy a Report Generator ... 1280
Use a Report Generator ... 1280
Custom Report Metrics ... 1281
Create a Report Metric ... 1281
Deploy a Report Metric ... 1281
Custom Companions ... 1282
Create a Companion ... 1282
Deploy a Companion ... 1283
Using Hooks ... 1284
Create a Hook ... 1284
Deploy a Hook ... 1285

Custom Data Sets 1285
Data Set Characteristics 1285
Create a New Data Set 1285
Deploy a New Data Set 1287
Using Jenkins Plugin for CA Service Virtualization 1287
Prerequisites 1288
Install 1288
Configure 1288
Features 1289
Add a new build step 1289
Configure existing build steps 1289
Use a custom registry endpoint 1289
Deploy Virtual Services 1290
Deploy Test Cases and Test Suites 1291
Start, Stop, or Undeploy Virtual Services 1291
Pipeline Support 1292
Deploy Virtual Service 1292
Deploy and Run Test 1292
Stop Virtual Services 1293
Start Virtual Services 1293
Undeploy Virtual Services 1293

Agents 1294

DevTest Java Agent 1294
Java Agent Architecture 1295
Java Agent Components 1295
Java Agent Data Flow 1296
Java Agent Database Schema 1298
Java Agent Data Categories 1299
Java Agent Installation 1299
Install the Pure Java Agent 1300
Install the Native Agent 1301
Options for Agent Parameters String 1302
Java Agent Install Assistant 1303
Platform-Specific Agent Configuration 1306
Wily Introscope Agent 1314
Start the Broker 1314
rules.xml File 1315
When Updates Are Applied 1316
Directives 1316
Managing Agents in Groups 1317
Protocol Weight Configuration 1318
Signature Specification 1319
Instrumentation Rules for VSE 1320

Configure a Database Sink 1321
Category Settings 1321
Add a Class for Tracking 1323
JNDI Priority 1323
Adding a Method to Intercept 1324
Excluding from Interception and Virtualization 1324
Excluding Methods, Classes, and Packages 1325
Excluding URLs 1325
Java Agent Auto-Response Generation 1326
Example Auto-Response Generation for EJB Calls 1327
Java Agent REST Application Programming Interfaces 1328
REST API Categories 1328
View the REST APIs 1328
Example: Retrieve Performance Data 1329
Developing Against the Java Agent 1329
Agent General APIs 1330
Agent Discovery APIs 1332
Agent Transaction APIs 1334
Agent VSE APIs 1338
Agent API Examples 1340
Java Agent Extensions 1341
Extending the Agent 1341
Adding Tags to Transaction Frames 1345
Extending the Broker 1345
Extending Java VSE 1347
Load Balancers and Native Web Servers 1347
Java Agent Security 1348
Extensions 1348
Remote Code Invocation 1349
Configure the Java Agent to Use SSL 1349
Java Agent Log Files 1351
Transaction Dispatchers 1351
Troubleshooting the Java Agent 1352
Error at startup: Error occurred during initialization of VM. Could not find agent library in absolute path... 1353
Agent exits immediately (or shortly after starting the process). 1354
Agent exits with the message "GetEnv on jvmdi returned -3 (JNI_EVERSION)". 1354
Agent exits with the message "UTF ERROR" ["../../../src/solaris/instrument /EncodingSupport_md.c":66]: ..." 1354
Agent hangs or throws numerous exceptions at startup (LinkageErrors, CircularityErrors, and so on). 1354
Agent throws java.lang.VerifyErrors or hot swapping has no effect. 1355
Agent starts but the consoles or broker cannot see the agent. 1355
Agent causes some operations to time out. 1355
java.lang.NoClassDefFoundError on com/itko/lisa/remote/transactions /TransactionDispatcher.class 1355

Security-related exceptions are thrown when the agent is enabled. 1356
Abnormal resource consumption (CPU, memory, file handles, ...). 1356
I can see the agent started, but Application Insight data is missing or incomplete. 1356
I turned on Java VSE recording or playback, but VSE does not receive any requests from the agent. 1357
Other issues with Java VSE. 1357
Ticket functionality is not working. 1357
DevTest is configured to use an IBM DB2 database. In DevTest Portal, I selected a JDBC node in a path graph. The Statements and Connection Url columns are blank. 1357
Received the following exception: com.itko.lisa.remote.transactions. TransactionSizeExceededException: could not XStream ... Serialization for this frame will be disabled. 1358
The VSE log includes the following error: com.itko.lisa.vse.stateful.protocol.java.listen. DefaultJavaPlaybackCallback - Timed out waiting on response. 1358
DevTest CICS Agent 1358
Install the CICS Agent 1359
CICS Agent Storage Requirements 1365
CICS Exit Considerations 1366
Coexistence with Other Exits 1366
DevTest CICS Agent Exit Behavior Summary 1366
CICS Agent User Exits 1367
ITKOUEX1 - DevTest CICS Agent Initialization Exit 1367
ITKOUEX2 - DevTest CICS Agent Termination Exit 1368
ITKOUEX3 - LISA CICS COMMAREA Consolidation Exit 1369
CICS Agent Operation 1371
Starting the DevTest CICS Agent and Exits 1371
Stopping the DevTest CICS Agent and Exits 1371
Impact on Virtualized CICS LINK Commands 1371
Impact on Virtualized CICS DTP/MRO Commands 1372
Monitor Transaction (TKOM) 1372
CICS Agent Messages 1373
DevTest CICS Agent Message Descriptions 1376
DevTest LPAR Agent 1399
Install the LPAR Agent 1399
LPAR Agent Storage Requirements 1399
LPAR Agent Installation 1400
LPAR Agent Operation 1403
Starting the DevTest LPAR Agent 1403
Stopping the DevTest LPAR Agent 1403
Monitoring and Managing the LPAR Agent 1404
LPAR Agent Messages 1405
DevTest Mainframe Bridge 1405
DevTest Mainframe Bridge Properties 1406
DevTest Mainframe Bridge Rules 1407

Administering 1408

General Administration 1408
- Default Port Numbers 1408
  - DevTest Server Default Port Numbers 1409
  - DevTest Workstation Default Port Numbers 1410
  - Demo Server Default Port Numbers 1410
  - Change Port Number for DevTest Portal 1410
- Separating the Data Directory from the Installation Directory 1411
  - Adding User-specific JAR and ZIP Files to the Workstation Classpath 1412
- Directory Structure 1413
  - DevTest Workstation Directories 1414
  - DevTest Server Directories 1415
- Running Server Components as Services 1417
- Running DevTest with Ant and JUnit 1418
  - Run DevTest Tests as JUnit Tests 1419
  - DevTest Solutions Ant Tasks 1419
  - Ant and JUnit Usage Examples 1422
  - JUnit Usage Examples 1422
  - Integration with CruiseControl 1424
  - More Example Build Files 1424
- Memory Settings 1425
  - Change the Java Heap Size on Windows and UNIX 1425
  - Change the Java Heap Size on OS X 1426
  - Adjust the Memory for Individual Processes 1426
- Third-Party File Requirements 1427
  - ActiveMQ File Requirements 1428
  - Apache Kafka 1428
  - JCAPS File Requirements 1428
  - JMS Messaging File Requirements 1429
  - Oracle OC4J File Requirements 1429
  - RabbitMQ Requirements 1429
  - SAP File Requirements 1429
  - SAP IDoc Virtualization Requirements 1429
  - SAP RFC Virtualization Requirements 1430
  - SonicMQ File Requirements 1430
  - TIBCO File Requirements 1430
  - WebLogic File Requirements 1431
  - webMethods File Requirements 1432
  - WebSphere MQ File Requirements 1432
- Enabling Additional Scripting Languages 1434

Database Administration 1434
- External Registry Database Configuration 1434
  - Configure DevTest to Use DB2 1435
  - Configure DevTest to Use MySQL 1436
  - Configure DevTest to Use Oracle 1438

Configure DevTest to Use SQL Server ... 1439
External Java Agent Database Configuration ... 1441
Scenario 1: Same Schema for Registry and Agent ... 1441
Scenario 2: Different Schema for Registry and Agent ... 1441
DDL Statements for Creating and Deleting the Schema ... 1441
External Enterprise Dashboard Database Configuration ... 1442
Configure Enterprise Dashboard to Use DB2 ... 1442
Configure Enterprise Dashboard to Use MySQL ... 1443
Configure Enterprise Dashboard to Use Oracle ... 1444
Configure Enterprise Dashboard to Use SQL Server ... 1445
Database Maintenance ... 1446
Automatic Reporting Maintenance ... 1446
Automatic Deletion of Audit Log Entries ... 1447
Automatic Deletion of Transactions ... 1447
Automatic Deletion of Tickets ... 1448
License Administration ... 1448
License Requirements ... 1448
How to Request a New License File ... 1449
Track Reporting and Consumption ... 1450
DevTest Solutions Usage Audit Report ... 1450
Overview ... 1450
User Type Chart ... 1451
SV for Performance Chart ... 1452
Historical Usage Tab ... 1453
Component by User ... 1453
Count Calculation Example ... 1454
User Types ... 1455
Usage Audit Report FAQs ... 1457
Security ... 1461
Identity and Access Manager ... 1461
IAM Administration ... 1462
Using the Lightweight Directory Access Protocol (LDAP) with DevTest Solutions. ... 1462
Authentication ... 1463
Authorization ... 1463
User Sessions ... 1464
IAM Database ... 1464
Define Realm Settings ... 1464
Add SSL Certificates to the Trust Store ... 1464
IAM and JAVA_HOME ... 1465
Start Identity and Access Manager ... 1465
Configure User Federation (LDAP) ... 1466
Manage Roles and Users ... 1469
Manage Sessions and Events ... 1475
Update the IAM Database ... 1476
Configure IAM Logging ... 1476
Access Control (ACL) ... 1476

ACL Administration 1476
Authentication 1477
Authorization 1477
Permission Types 1477
Standard Permissions 1478
View User Information from DevTest Workstation 1483
Manage Role Permissions 1483
Resource Groups 1485
Resources 1487
Using SSL to Secure Communication 1488
SSL Certificates 1489
Use Your Own Self-Signed Certificate 1490
Use SSL with Multiple Certificates 1491
Mutual (Two-Way) Authentication 1492
SSL/TLS Protocol Configuration 1492
Supported SSL/TLS Protocols and the Equivalent Values Expected by the JVM 1492
Using HTTPS Communication with the invoke APIs 1493
(Optional) Generate a New Key Pair and Certificate 1493
Configure the Properties Files 1495
Using Kerberos Authentication 1496
DevTest Support for Principal + Password Authentication 1497
Sample krb5.conf File 1497
Sample krb5.conf File with Active Directory as KDC 1497
Sample login.config File 1497
Logging 1498
Main Log Files 1498
Demo Server Log Files 1499
Logging Properties File 1499
Status Messages for Server Components 1501
Automatic Thread Dumps 1501
Test Step Logger 1502
VSE Logging 1502
VSE_Matches File 1502
VSE.log File 1503
Monitoring 1503
Use the Server Health Monitor 1503
Use the ServiceManager 1505
Service Manager Options 1505
Service Manager Examples 1507
Use the Registry Monitor 1508
Registry Monitor - Test Tab 1508
Registry Monitor - Simulators Tab 1509
Registry Monitor - Coordinator Servers Tab 1509
Registry Monitor - Virtual Environments Tab 1509
Use the Enterprise Dashboard 1509
Features and Functionality of the Enterprise Dashboard 1509

Reactivate a Registry or Enterprise Dashboard 1516
Maintain Registries 1517
Export Dashboard Data 1518
Purge Dashboard Data 1520
Change Dashboard Language 1520
Return On Investment Calculator 1520
Telemetry 1521
Benefits of Telemetry 1522
Enabling or Disabling Telemetry 1522
Telemetry Data 1523

Reference 1525

Assertion Descriptions 1525
HTTP Assertions 1525
Highlight HTML Content for Comparison 1526
Check HTML for Properties in Page 1527
Ensure HTTP Header Contains Expression 1529
Check HTTP Response Code 1529
Simple Web Assertion 1530
Check Links on Web Responses 1530
Database Assertions 1531
Ensure Result Set Size 1531
Ensure Result Set Contains Expression 1532
XML Assertions 1532
Highlight Text for Comparison 1532
Ensure Result Contains String 1535
Ensure Step Response Time 1535
Graphical XML Side-by-Side Comparison 1536
XML XPath Assertion 1540
Ensure XML Validation 1541
JSON Assertions 1543
Ensure Result Equals 1543
Ensure Result Contains 1544
Ensure JSON Schema 1545
Virtual Service Environment Assertion 1547
Assert on Execution Mode 1547
Other Assertions 1547
Highlight Text Content for Comparison 1548
Ensure Non-Empty Result Assertion 1550
Ensure Result Contains String Assertion 1551
Ensure Result Contains Expression Assertion 1551
Ensure Property Matches Expression Assertion 1551
Ensure Step Response Time Assertion 1552
Scripted Assertion 1552

Ensure Properties Are Equal Assertion 1553
Assert on Invocation Exception Assertion 1554
File Watcher Assertion 1554
Check Content of Collection Object Assertion 1555
WS-I Basic Profile 1.1 Assertion 1556
Messaging VSE Workflow Assertion 1558
Validate SWIFT Message Assertion 1558
Asset Descriptions 1560
Apache Kafka Assets 1560
Kafka Connection 1560
Kafka Topic 1561
Kafka Consumer and Producer 1563
Email Connection Asset 1563
IBM MQ Native Assets 1564
JDBC Connection Assets 1566
JMS Client Assets 1567
Connection Factories 1568
Connections 1570
Sessions 1570
Producers 1571
Consumers 1571
Destinations 1571
JNDI Initial Context Asset 1572
RabbitMQ Assets 1572
Connection 1572
Queue 1573
Exchange 1574
Temp Queue 1575
Channel 1576
SAP JCo Destination Assets 1576
SSL Assets 1578
Basic Scenarios 1579
Advanced Scenarios 1581
SSL Context Parameters 1583
Companion Descriptions 1583
Web Browser Simulation Companion 1584
Browser Bandwidth Simulation Companion 1585
HTTP Connection Pool Companion 1585
Configure DevTest to Use a Web Proxy Companion 1586
Set Up a Synchronization Point Companion 1587
Set Up an Aggregate Step Companion 1587
Execute Event-driven Subprocess Companion 1588
Observed System VSE Companion 1588
Configuration Information 1589
Data Set Source Example 1590
Observed System Data Provider (Wily) Source 1594

Run Time Priorities ... 1595
VSE Think Scale Companion ... 1596
Batch Response Think Time Companion ... 1598
Recurring Period Think Time Companion ... 1599
Messaging Retry Policy Companion ... 1600
Counting Retry ... 1601
Simple Delay ... 1601
Exponential Backoff ... 1602
Create a Sandbox Class Loader for Each Test Companion ... 1602
Set Final Step to Execute Companion ... 1603
Negative Testing Companion ... 1603
Fail Test Case Companion ... 1603
XML Diff Ignored Nodes Companion ... 1603
JSON Ignored Nodes Companion ... 1604
Correlation Schemes ... 1605
Overview of Correlation Schemes ... 1605
Correlation ID ... 1606
Message ID to Correlation ID ... 1606
Payload ... 1607
ReplyTo and Temporary Response Queues ... 1607
Data Set Descriptions ... 1608
Read Rows from a Delimited Data File Data Set ... 1608
Create Your Own Data Sheet Data Set ... 1609
Create Your Own Set of Large Data Data Set ... 1612
Convert to Comma-Separated Value (CSV) File ... 1613
Read Rows from a JDBC Table Data Set ... 1613
Create a Numeric Counting Data Set ... 1614
Read Rows from Excel File Data Set ... 1615
Read DTOs from Excel File Data Set ... 1617
Building the Excel spreadsheet ... 1618
Unique Code Generator Data Set ... 1621
Random Code Generator Data Set ... 1622
Message-Correlation ID Generator Data Set ... 1623
Load a Set of File Names Data Set ... 1624
XML Data Set ... 1624
Basic Settings Tab ... 1625
Advanced Settings Tab ... 1626
Record Editing Panel ... 1627
Visual XML Tab ... 1628
Raw XML Tab ... 1629
Scripted DataSet ... 1629
Event Descriptions ... 1630
Filter Descriptions ... 1634
Utility Filters ... 1635
Create Property Based on Surrounding Values ... 1635

Store Step Response 1638
Override Last Response Property 1638
Save Property Value to File 1639
Parse Property Value As Argument String 1639
Save Property from One Key to Another 1640
Time Stamp Filter 1640
Database Filters 1641
Extract Value from JDBC Result Set 1641
Size of JDBC Result Set 1643
Set Size of a Result Set to a Property 1643
Get Value For Another Value in a Result Set Row 1644
Messaging/ESB Filters 1646
Extract Payload and Properties from Messages 1646
Convert an MQ Message to a VSE Request 1647
Convert a JMS Message to a VSE Request 1647
HTTP/HTML Filters 1648
Create Resultset from HTML Table Rows 1648
Parse Web Page for Properties 1650
Parse HTML/XML Result for the Value of a Specific Tag or Attribute 1653
Parse HTML Result for Specific Value and Parse It 1655
Parse HTML Result for the Child Text of a Tag 1656
Parse HTML Result for HTTP Header Value 1656
Parse HTML Result for the Value of an Attribute 1657
Parse HTML Result for DevTest Tag 1657
Choose Random HTML Attribute 1658
Parse HTML into List of Attributes 1658
Parse HTTP Header Cookies 1659
Dynamic Form Filter 1660
Parse HTML Result by Searching Tag Attribute Values 1660
Inject HTTP Header 1661
XML Filters 1662
Parse Text from XML 1662
Read Attribute from XML Tag 1664
Parse XML Result for DevTest Tag 1666
Choose Random XML Attribute 1666
XML XPath Filter 1668
JSON Filters 1668
JSON Path Filter 1668
Java Filters 1671
Java Override Last Response Property Filter 1671
Java Store Step Response Filter 1671
Java Save Property Value to File Filter 1672
VSE Filters 1672
Data Protocol Filter 1672
CAI Filters 1673
DevTest Integration Support for CA CAI 1673

DevTest Integration Support for webMethods Integration Server 1674
Copybook Filters 1674
Copybook Filter 1674
Metrics Descriptions 1675
DevTest Whole Test Metrics 1675
DevTest Test Event Metrics 1676
SNMP Metrics 1677
JMX Metrics 1678
Enable JMX Metrics for Tomcat 1680
TIBCO Hawk Metrics 1681
Windows Perfmon Metrics 1682
UNIX Metrics Via SSH 1683
Property Descriptions 1685
DevTest Property File (lisa.properties) 1685
Enterprise Dashboard Properties 1686
Comma-separated List of Paths for Javadoc and Source Code 1686
System Properties 1687
Server Properties 1687
OS X Properties 1687
Update Notifications 1688
Basic Defaults 1688
HTTP Header Keys Properties 1689
HTTP Field Editor Properties 1690
Test Case Execution Parameters 1690
TestEvent Handling Customizations 1691
Test Manager/Editor Properties 1692
J2EE Server Parameters 1694
Native Browser Information to Use for Internal Rendering 1695
Test Manager/Monitor Properties 1695
Built-In String-Generator Patterns 1696
JMX Information 1696
Test Manager/Interactive Test Run Properties 1697
External Command Shells 1698
Testing Parameters 1698
Properties for Use by StdSchedulerFactory to Create a Quartz Scheduler Instance 1698
VSE Properties 1700
Network Port Properties 1707
Application Insight Properties 1708
Database Properties 1712
Mainframe Properties 1714
Selenium Integration Properties 1714
VSEasy Properties 1716
Solr Properties 1716
Application Insight Stateful Baseline Parameterization 1716
Application Insight Shelve Frames 1717
CA Agile Central Integration Properties 1717

SAP Properties 1717
Community Edition Properties 1717
Identity and Access Manager Properties 1717
VS Recorder 1718
Custom Property Files 1718
Local Properties File 1719
Site Properties File 1736
logging.properties 1740
Enterprise Dashboard Properties (dradis.properties) 1745
Portal Properties File (phoenix.properties) 1747
Test Step Descriptions 1747
Standard Options in Test Steps 1748
Abort the Test 1748
End the Test 1748
Fail the Test 1748
Web-Web Services Steps 1749
HTTP-HTML Request Step 1749
REST Step 1756
Web Service Execution (XML) Step 1757
WSDL Validation 1791
Base64 Encoder 1792
Multipart MIME Step 1792
SAML Assertion Query 1793
Java-J2EE Steps 1797
Dynamic Java Execution 1798
RMI Server Execution 1801
Enterprise JavaBean Execution 1805
Other Transaction Steps 1808
SQL Database Execution (JDBC) 1808
SQL Database Execution (JDBC with Asset) 1811
CORBA Execution 1813
Utilities Steps 1814
Save Property as Last Response 1814
Output Log Message 1815
Write to Delimited File 1815
Read Properties from a File 1817
Do-Nothing Step 1818
Parse Text as Response 1818
Audit Step 1818
Base64 Encoder Step 1819
Checksum Step 1819
Convert XML to Element Object 1820
Compare Strings for Response Lookup 1821
Compare Strings for Next Step Lookup 1823
Send Email 1824
External-Subprocess Steps 1825

Execute External Command 1825
File System Snapshot 1827
Execute Subprocess 1828
Execute JUnit Test Case-Suite 1829
Read a File (Disk URL or Classpath) 1829
External - FTP Step 1830
JMS Messaging Steps 1832
JMS Messaging (JNDI) (Deprecated) 1832
JMS Messaging - Message Consumer (Deprecated) 1839
JMS Send Receive Step 1842
Tutorial - Send and Receive a JMS Message 1846
Apache Kafka Steps 1848
Kafka Send Receive Step 1848
Kafka Advanced Use Cases 1851
BEA Steps 1859
Message Consumer 1859
Read a File 1859
Web Service Execution (XML) 1859
FTP Step 1859
WebLogic JMS (JNDI) (Deprecated) 1859
Sun JCAPS Steps 1864
JCAPS Messaging (Native) (Deprecated) 1865
JCAPS Messaging (JNDI) (Deprecated) 1867
Oracle Steps 1868
Oracle OC4J (JNDI) (Deprecated) 1868
Oracle AQ (JMS) (Deprecated) 1869
TIBCO Steps 1873
TIBCO Rendezvous Messaging 1873
TIBCO EMS Messaging (Deprecated) 1878
TIBCO Direct JMS (Deprecated) 1878
Sonic Steps 1879
SonicMQ Messaging (Native) (Deprecated) 1879
SonicMQ Messaging (JNDI) (Deprecated) 1880
webMethods Steps 1881
webMethods Broker 1881
webMethods Integration Server Services 1885
IBM Steps 1888
IBM WebSphere MQ Step (Deprecated) 1889
IBM MQ Native Send Receive Step 1894
RabbitMQ Steps 1897
RabbitMQ Send Receive Step 1898
SAP Steps 1902
SAP RFC Execution 1902
SAP IDoc Sender 1904
SAP IDoc Status Retriever 1905
Selenium Integration Steps 1907

Create and Export a Selenium Builder Recording 1907
Import a Selenium Builder Recording 1908
Export a Test Case with Selenium Test Steps to a JSON Script 1913
Virtual Service Environment Steps 1914
CAI Steps 1914
Execute Transaction Frame 1915
Execute webMethods Transaction Frame 1917
Custom Extension Steps 1917
Custom Test Step Execution 1918
Java Script Step (deprecated) 1918
Execute Script (JSR-223) Step 1919
Opaque Data Listener Step 1920

Connect 1923
User Community 1923
Support 1923
Webcasts, Success Stories, White Papers, Demos, and More 1923
DevTest Solutions Education and Training 1923
Enable Your Teams with CA Education 1923
Learning Paths 1924
CA Learning Subscription 1924
Free Training 1925

DevTest Solutions - Home

DocOps Search Tips

This article answers common CA Technologies DocOps search questions.

> **Note:** DocOps is powered by Atlassian Confluence. For detailed information about the Confluence search engine and advanced search tips, see the Atlassian Documentation.

- How do I search in DocOps?
- How do I refine my search?
- Can I search on exact phrases?
- Can I search for content with special characters?
- Why can I not find what I am looking for?

How do I search in DocOps?

To search for topics, enter your search criteria or key words in the **Search this product** field at the top of the navigation tree.

Exact matches to topics that are listed in the navigation tree display in a list. Click the topic that you want to view.

If you don't see the topic that you're looking for, press **Enter** or click the search icon to view a more complete search results page.

How do I refine my search?

To refine your search, use Boolean and Lucene search methods.

- **Boolean** allows you to refine your search by combining terms and showing a relationship between the words. To refine the search, use the Boolean modifiers OR, AND, and NOT (capitalized).

- **Lucene** searches on single terms or phrases. It does not support wildcards at the beginning of strings, however, as a workaround, you can use regular expressions as shown in the following example:

  - **. (dot)**
    Matches any single character
    Example: /.et/
    Matches get, set, let, yet.

  - *** (asterisk)**
    Matches the preceding element zero or more times.
    Example: /.*gram/
    Matches program, diagram, kilogram.

  - **[] (square brackets)**
    Matches one of the characters that the square brackets contain.
    Example: /[hc]at/
    Matches hat or cat.

  - **| (vertical bar)**
    Matches either the expression before or the expression after the operator.
    Example: /CAVH(LOAD|JCL|SAMP)/
    Matches CAVHLOAD, CAVHJCL and CAVHSAMP.

  - **{x} (curly brackets)**
    Matches the preceding element exactly x times.
    Example: /[0-9]{5}/
    Matches a number, exactly 5 digits in length.

  - **{x,} (curly brackets)**
    Matches the preceding element at least x times.
    Example: /[A-Z]{3,}/
    Matches one word consisting of capital letters, at least 3 characters in length.

  - **{x,y} (curly brackets)**
    Matches the preceding element a number of times. The number is in the range x to y.
    Example: /[0-9A-F]{4,8}/
    Matches any hexadecimal string of 4 to 8 characters: 3456, 3425FCB2.

Can I search on exact phrases?

To search for an exact phrase, use quotation marks at the beginning and end of the phrase. The search feature uses word stemming, which means that searched words with a common root yield the same search results. You still might get more results than you expect. For example, the following searches would yield the same results:

- "initialize"
- "initialized"
- "initialization"

The search engine treats certain words as "reserved words". Reserved words (or "stop words") may be ignored. If your search term includes any reserved words, the search might produce results that are different from the exact term, or might produce no results at all. For example, a search for "CONTROL FILE HAS NOT BEEN INITIALIZED" may produce no results because "not" is a reserved word, or the search result may be reduced to "CONTROL FILE HAS BEEN INITIALIZED."

Can I search for content with special characters?

Yes. If the search does not return an exact match, it displays a list of matches on the “Search results” page.

Why can I not find what I am looking for?

Your search request may not produce results for various reasons. The following list includes the most common reasons:

- You are not signed in.
- You did not type a valid search string in the Search field.
- You may need to refine your search, using Boolean, Lucene query syntax, or Confluence advanced search methods.
- Your search string may contain a stop word. For more information, see “How do I refine my search?”.

More Information

For more information about advanced search methods, see the Atlassian Confluence Documentation.

Release Notes

The Release Notes section outlines the new features, enhancements, and resolved issues for DevTest Solutions. Use the release notes as a starting point when you first install or upgrade to a new version of DevTest Solutions. This section includes the following pages:

- New Features and Enhancements
- Preview Features
- Resolved Issues
- Known Issues
- End of Service Announcement
- End of Life Features
- DevTest Portal Functionality
- Third Party Acknowledgments
- Product Accessibility Features

Note: To upgrade from DevTest 10.3 or earlier, you must request a new license file. For more information, see License Administration.

Release Comparison

This table compares the new features in the latest DevTest Solutions releases.

The CA Support site provides information on Release and Support Lifecycle Dates.

Key Features	Release 10.4	Release 10.3	Release 10.2	Release 10.1	Release 10.0	Release 9.5	Release 9.1	Release 9.0
Application Test Enhancements								
Support for Apache Kafka	**yes**	no	no	no	no	no	no	no
Create baseline tests from OpenAPI 3.0	**yes**	no	no	no	no	no	no	no
REST step supports the HEAD, PATCH, and OPTIONS methods	**yes**	**yes**	**yes**	no	no	no	no	no
Support for Microsoft SQL Server 2016	**yes**	**yes**	**yes**	no	no	no	no	no
Create baseline tests from Swagger 2.0	**yes**	**yes**	**yes**	**yes**	**yes**	no	no	no
Create baseline tests from virtual services	**yes**	**yes**	**yes**	**yes**	**yes**	no	no	no
Convert large data data set to comma-separated value (CSV) file	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no

HP ALM Plugin for DevTest 64-Bit	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
CA Agile Central report generator	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
JMS support for ActiveMQ direct connection factory	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
Failed to Stage status added to Monitoring Tests window	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no
Selenium support for Firefox 43	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no
Certified Selenium tests on Linux	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no
RabbitMQ Send Receive step	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**
CVS Dashboard moved to DevTest Portal	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**
Service Virtualization Enhancements								
Support for Apache Kafka	**yes**	no	no	no	no	no	no	no
Create from OpenAPI 3.0	**yes**	no	no	no	no	no	no	no
Virtual Service Catalog	**yes**	**yes**	no	no	no	no	no	no
Can use web proxy in gateway recorder	**yes**	**yes**	no	no	no	no	no	no
Create from Swagger in DevTest Portal	**yes**	**yes**	**yes**	**yes**	no	no	no	no
Create from WSDL in DevTest Portal	**yes**	**yes**	**yes**	**yes**	**yes**	no	no	no
Support for Swagger references	**yes**	**yes**	**yes**	**yes**	**yes**	no	no	no
Inspection View enhancement	**yes**	**yes**	**yes**	**yes**	**yes**	no	no	no
Record CICS virtual services in DevTest Portal	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
Manage copybook bundles in DevTest Portal	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
JSON data protocol handler in DevTest Portal	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no
ODP transport protocol in DevTest Portal	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no
Bind address option in ServiceImageManager tool	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no
RabbitMQ transport protocol	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**
Perform model healing in DevTest Portal	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**
	yes	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**

Generate baseline tests from virtual services								
Virtual Service Metrics reporting	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**
Data-driven virtual services	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**
Find and replace text in virtual services in DevTest Portal	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**
Record TCP virtual services in DevTest Portal	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**
General Enhancements								
Jenkins Plug-in	**yes**	no	no	no	no	no	no	no
Identity and Access Manager for DevTest authentication	**yes**	no	no	no	no	no	no	no
Authentication for Enterprise Dashboard	**yes**	no	no	no	no	no	no	no
Collection of customer experience improvement data	**yes**	no	no	no	no	no	no	no
Enhanced API Functionality in Test Execution and Reporting	**yes**	**yes**	no	no	no	no	no	no
Improved Workstation installation process	**yes**	**yes**	**yes**	no	no	no	no	no
Improved visibility of key data in the Enterprise Dashboard	**yes**	**yes**	**yes**	no	no	no	no	no
Use the Return On Investment Calculator	**yes**	**yes**	**yes**	no	no	no	no	no
Database Safeguards	**yes**	**yes**	**yes**	no	no	no	no	no
Messaging Retry Policy companion	**yes**	**yes**	**yes**	**yes**	no	no	no	no
Support for CICS TS Version 5.4	**yes**	**yes**	**yes**	**yes**	**yes**	no	no	no
Self-administration of password	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
Support for Microsoft SQL Server 2014	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
ACL administration moved to DevTest Portal	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
New charts added to Usage Audit Report	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
Coordinator and simulator view moved to DevTest Portal	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
	yes	**yes**	**yes**	**yes**	**yes**	**yes**	no	no

CICS Agent configuration enhancements								
Use a user-specified account when installing Windows services	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
Added historical concurrent user counts to the Enterprise Dashboard	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
Server Health Monitor in DevTest Portal	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
Modify keys or values in configuration file when staging a test	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
Added more report information in headings and in additional page for some reports	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no
Enterprise Dashboard updates	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no
Forward Cars demo application	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**
Monitor virtual services and recordings in DevTest Portal	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**
Paging capabilities in Test Monitoring Details screen in DevTest Portal	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**
Application Insight Enhancements								
CA API Management integration	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
Streamlined creation of virtual services for JMS and WebSphere MQ	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
Support for parameterization in REST and Web HTTP data-driven baselines	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
Configure mainframe agents in DevTest Portal	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
Create request/response pairs from CICS transactions in the shelf	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
Use a VRS file when creating a virtual service for a CICS transaction	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
Create stitching rules for Java agent in DevTest Portal	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no	no
	yes	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no

Broker notifications in DevTest Portal								
Support for parameterization in web HTTP stateful baselines	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no
Delete transactions from Analyze Transactions window	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no
Configure Java agent logging in DevTest Portal	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	no
Perform bulk delete of recordings in Document Transactions window	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**
Create virtual services from RMI transactions	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**
Native MQ CAI message mirroring	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**
View producer and consumer relationship	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**
Download agent from portal	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**

New Features and Enhancements

- Identity and Access Manager (IAM)
- Authentication for Enterprise Dashboard
- Virtual Service Catalog
- OpenAPI 3.0
- Jenkins Plug-in
- Apache Kafka
- Collection of Customer Experience Improvement Data
- Improved Testing Execution Reports

Identity and Access Manager (IAM)

Enhanced the Identity and Access Manager feature so that it is now part of the larger DevTest Solutions install and provides authentication for the Enterprise Dashboard.

- IAM is now responsible for all user and role creation for DevTest.
- Role permissions are still defined in DevTest Portal.
- Added support for running IAM as a service.
- Deprecated the adduser command-line utility.

Case Number: F67691

Updated the Following Pages:

- Identity and Access Manager
- Start Identity and Access Manager
- Configure User Federation (LDAP)
- Manage Roles and Users
- Manage Sessions and Events
- Update the IAM Database
- Access Control (ACL)
- Manage Role Permissions

Authentication for Enterprise Dashboard

Added authentication to the Enterprise Dashboard. Users are now required to log in before viewing and using the Enterprise Dashboard. This authentication is handled by the new Identity and Access Manager feature.

Case Number: F67691

Updated the Following Pages:

- Features and Functionality of the Enterprise Dashboard

Virtual Service Catalog

Enhancements to Virtual Service Catalog:

- Separated the Virtual Service Catalog installation from Identity and Access Manager. Virtual Service Catalog is now its own independent install. Identity and Access Manager is now part of the DevTest Solutions install.
- Added support for running Virtual Service Catalog as a service.
- Added ability to view Header details of a response when viewing the details of a virtual service.
- Replaced the vmoptions file with an application.yml file for configuring Virtual Service Catalog.

Case Number: F60776

Updated the Following Pages:

- Installing Virtual Service Catalog
- Using Virtual Service Catalog

- View Virtual Services
- Configure Virtual Service Catalog

OpenAPI 3.0

Added support for creating virtual services and tests from OpenAPI 3.0 within the portal.

Case Number: F71204

Updated the Following Pages:

- Quick Start Window
- Create Virtual Services from Swagger (OpenAPI)

Jenkins Plug-in

Description

Case Number:

Added the Following Page:

- Using Jenkins Plugin for CA Service Virtualization

Apache Kafka

Added support for Apache Kafka for the creation of tests and virtual services.

Case Number: F68268

Updated or Added the Following Pages:

- Third-Party File Requirements
- Apache Kafka Assets
- Apache Kafka Steps
- Apache Kafka Transport Protocol

Collection of Customer Experience Improvement Data

Enhanced the telemetry feature to increase usability. Telemetry can now be enabled directly from the DevTest Solutions installer or from the dradis.properties file. Also added the ability to capture product version and operating system at the registry level.

Case Number: F61992

Updated the Following Pages:

- Telemetry
- Telemetry Data

Improved Testing Execution Reports

Ability to export a report that includes all drill down details into a single PDF file.

Case Number: F65642

Updated the Following Pages:

- Testing Execution Reports

Preview Features

This page describes the features that are available in the Sandbox for this release. We are considering adding these features as core product functionality. We are making these features available for you in the Sandbox to try and explore. Depending on your feedback and other factors, these features can change or be removed in subsequent versions at any time.

The following preview features are available in this release:

Feature	Description	Documentation
DevTest Chrome Extension	Lets you analyze calls and transactions started from the browser.	DevTest Chrome Extension
Create From JDBC	Lets you virtualize JDBC-based database traffic.	Virtualize JDBC Traffic
Enhanced VS Recorder	Provides a new workflow for recording a virtual service, using version 2 of the DevTest API. The new version of the API is also used in creating virtual services with R/R pairs and specifications when this preview feature is enabled.	Virtualize Services by Recording (HTTP or HTTP/S) - Enhanced VS Recorder Select and Configure Data Protocols - Enhanced VS Recorder

Enable and Disable Features

You can quickly enable and disable the features in the Sandbox.

Follow these steps:

1. Open DevTest Portal.
2. Click the down arrow to the right of your user name and select **The Sandbox**.
3. To enable a feature, click the gray **Enabled** switch.
4. To disable a feature, click the blue **Enabled** switch.

Resolved Issues

- Application Test
  - Stop Test or Kill Test from Portal
  - Can't Specify Concurrent Capacity in MarInfo
  - List for CVS Monitored Events Badly Sorted
  - ANDROID_HOME Error is Appearing in Logs
  - BLOB Data Type is not Supported in JDBC Step
  - JDBC Assets Getting Changed in Drop Down for a Different project.config File
  - Ctrl+s Save Icon in the Workstation are Not Working When Trying to Create a New Suite
- Service Virtualization
  - VS from WSDL Issue
  - Inspection View Recent Requests Has Two Entries and Incorrect NO MATCH FOUND
  - Workstation Search not Functioning as Expected for Transaction IDs and Tags or Values in the Request/Response
  - vsBuilderSessions File Corruption Prevents VSE From Starting
  - Create VS from PCAP Not Working as Expected
  - VS Not Getting Filtered Using Group Tag Name
  - JSON Response, Invalid Property ID
  - Sample Match Script Failing
  - Getting Null Pointer Exception Error using getStateValue in VS ResponseName
  - Version compatibility issues in DevTest 10.2
  - Unable to Set msg.putDateTime in Response Meta Data
  - Unable to Kill/Stop Tests from Portal
  - UUID Fields are Ignored During Virtual Service Creation Process with Swagger 2.0
- General/Common Components
  - Console Does Not Retain the Service Name of the Service When Downloading the MAR
  - High Open File Descriptor Count in Portal
  - Excessive Memory Requirement Using Portal > Monitoring > Tests
  - Java Client Can't Obtain Kerberos Authentication When Passing Through DevTest
  - lisa-virtualize-invoke Does Not Work and Gives Authorization Fails
  - Unable to List Virtual Services in Virtual Service Catalog
  - Consolidated Elasticsearch
  - SSL Configuration
  - Reporting Period End Dates in Export Usage Audit Data
  - Upgrade Schema V15 for DevTest 10.3 Fails for Dradis

Application Test

Stop Test or Kill Test from Portal

If you requested Stop or Kill multiple times in DevTest Portal, you would see multiple entries in the Monitor > Tests report for the same test when it finally stopped. This change resolves that so you now only see one result per test.

Defect: DE304883

Support Case: 00784851

Can't Specify Concurrent Capacity in MarInfo

Fixed specifying concurrent capacity in MarInfo.

Defect: DE345898

Support Case: 00962078

List for CVS Monitored Events Badly Sorted

Corrected filtering error on display.

Defect: DE346040

Support Case: 00961686

ANDROID_HOME Error is Appearing in Logs

Fixed an issue that caused ANDROID_HOME to erroneously appear in the logs through Null ointer Exceptions.

Defect: DE 354666

Support Case: 00979118

BLOB Data Type is not Supported in JDBC Step

Enhanced the JDBC step to support the BLOB data type.

Defect: DE 353786

Support Case: 00985806

JDBC Assets Getting Changed in Drop Down for a Different project.config File

This change fixes the Asset Selector drop down selected item changing when the active config changes.

Defect: DE 358343

Support Case: 01010378

Ctrl+s Save Icon in the Workstation are Not Working When Trying to Create a New Suite

Change was made to clear the dirty bit after saving a Suite document in Workstation. Logs showed file was saved but dirty bit was not cleared.

Defect: DE360808

Support Case: 0069560

Service Virtualization

VS from WSDL Issue

Fixed a defect that prevented the successful generation of a virtual service image from a WSDL when that WSDL contained multiple “xsd:schema” in a single file.

Defect: DE138278

Inspection View Recent Requests Has Two Entries and Incorrect NO MATCH FOUND

Fixed a defect in which Inspection View 'Recent Requests' incorrectly shows 2 entries per transaction of which incorrectly shows 'NO MATCH FOUND'.

Defect: DE329475

Support Case: 00897298

Workstation Search not Functioning as Expected for Transaction IDs and Tags or Values in the Request/Response

Corrected an issue with the Find field in the Transactions tab of DevTest Workstation. This field now allows you to search specific transactions, META transactions, and searches both the request and response.

Defect: DE336080

Support Case: 00922252

vsBuilderSessions File Corruption Prevents VSE From Starting

Added error message for the case when vsBuilderSession contains corrupted files.

Defect: DE346109

Support Case: 00963004

Create VS from PCAP Not Working as Expected

Fixed parsing of PCAP requests with 100 Continue header.

Defect: DE351352

Support Case: 00980726

VS Not Getting Filtered Using Group Tag Name

Fixed filtering on a group tag name with hyphen in Portal.

Defect: DE356757

Support Case: 01003355

JSON Response, Invalid Property ID

Corrected an issue that caused a validation error when using a JSON 2.0 virtual service with a LISA property expression.

Defect: DE359792

Support Case: 01004743

Sample Match Script Failing

Fixed an error that caused the sample match scrip to fail with the following error, "Cannot access field: getOperation, on object: Request".

Defect: DE359908

Support Case: 01067584

Getting Null Pointer Exception Error using getStateValue in VS ResponseName

Fixed null pointer errors in the vse.log.

Defect: DE360219

Support Case: 01012744

Version compatibility issues in DevTest 10.2

Fixed null pointer errors that were generated when the value of an associated resource was missing or null.

Defect: DE361163

Support Case: 01073038

Unable to Set msg.putDateTime in Response Meta Data

Added support for {{...}} property references in the in the msg.correalationid meta-data property.

Defect: DE362044

Support Case: 01077773

Unable to Kill/Stop Tests from Portal

Fixed an issue that prevented users from being able to successfully stop a running test from DevTest Portal.

Defect: DE364762, DE337468

Support Case: 01085995, 00927278

UUID Fields are Ignored During Virtual Service Creation Process with Swagger 2.0

Added support for UUID property fields where the values are treated as strings. The default minimum length is 5. The default maximum length is 15. However, these defaults can be modified by specifying min/max length of the property in the swagger doc. Previously these fields were skipped and did not show up in the VSI.

Defect: DE368820

Support Case: 01104971

General/Common Components

Console Does Not Retain the Service Name of the Service When Downloading the MAR

When user downloaded a MAR file, the name for the file was replaced by an id.

Defect: DE337903

Support Case: 00927632

High Open File Descriptor Count in Portal

Updated asset-pipeline .jar files to prevent excessive file handle consumption in DevTest Portal.

Defect: DE343649

Support Case: 00948570

Excessive Memory Requirement Using Portal > Monitoring > Tests

Added additional filter capabilities to Portal Monitor Tests window.

Defect: DE349630

Support Case: 00975707

Updated the Following Page:

- Monitor Tests

Java Client Can't Obtain Kerberos Authentication When Passing Through DevTest

Added a local.property to allow HTTP authentication.

Defect: DE355567

Support Case: 00999033

Updated the Following Page:

- Local Properties File

lisa-virtualize-invoke Does Not Work and Gives Authorization Fails

There was an issue with LDAP authentication with the invoke v2 APIs and the swagger-ui API would occassionaly result in Out Of Memory exceptions and long load times. This resolved those issues.

Defect: DE357218

Support Case: 01006219

Unable to List Virtual Services in Virtual Service Catalog

Virtual Service Catalog did not find virtual services that were running on Service Virtualization components on non-default ports.

Defect: DE359593

Support Case: 01063492

Updated Page:

- Configure Virtual Service Catalog

Consolidated Elasticsearch

Upgraded Elasticsearch to version 1.6.1 to address a known security vulnerability.

Defect: DE361466, DE358861

Support Case: 01074462, 01063202

SSL Configuration

Added support for property values that contain a "%" special character in dradis.properties.

Defect: DE364112

Support Case: 00985090

Updated Page:

- Using HTTPS Communication with the invoke APIs

Reporting Period End Dates in Export Usage Audit Data

Modified the Export Usage Audit Data feature in Enterprise Dashboard so that when users select a date range for their data export, the dates selected are inclusive.

Defect: DE367827

Support Case: 01105548

Upgrade Schema V15 for DevTest 10.3 Fails for Dradis

Change was made to remove the hard-coded default schema for the telemetrics tables when using MS SQL as your database.

Defect: DE369041

Support Case: 01110421

Known Issues

- SSL Support on AIX
- Cumulative Transactions in Advanced Metrics Graph in Enterprise Dashboard
- Virtual Service Environment Detail Table in Enterprise Dashboard Shows Transaction Count Inconsistency
- Bar Graphs in The Enterprise Dashboard
- Viewing Virtual Service Catalog and Identity and Access Manager on Internet Explorer
- Ampersand Character in Resource Names
- Searching for Long Strings in a Virtual Service
- Data-Driven Virtual Services
- Oracle New Install Issue
- dradis.properties File Required
- Demo Server Does Not Work on Solaris
- Enabling or Disabling the Enhanced VS Recorder Can Cause Errors
- Perfmon Restrictions
- Server Component Installation Restrictions
- Using Data Files in DevTest Portal
- Testing Android Applications on OS X
- Incorrect Command in Readme file
- Enterprise Dashboard Time
- Standalone Demo Server

SSL Support on AIX

SSL support does not work on AIX.

Cumulative Transactions in Advanced Metrics Graph in Enterprise Dashboard

In the Enterprise Dashboard, cumulative transactions in the Advanced Metrics graph do not show the cumulative amount across months. The cumulative usage is shown for a given month and then begins again in the next month. For example, if you select January 1 to February 15 you will see a drop in cumulative transactions around the 1 February. The data is accurate but if you are trying to calculate total transactions across two months you need to take the cumulative amount from one month and add it to the next.

Virtual Service Environment Detail Table in Enterprise Dashboard Shows Transaction Count Inconsistency

In the enterprise dashboard, if you click on the specific VSE in the far left column of the VSE Detail table, another window opens showing Virtual Service details running on that VSE. The total transaction count on this screen may not match exactly with the total transaction count for that VSE on the VSE Detail table.

This issue is due to a slight synchronization issue that sometimes occurs. The total transaction count on the VSE Detail table is the most accurate.

Bar Graphs in The Enterprise Dashboard

Due to limitations in the library that is used for the graphs in the Enterprise Dashboard, some bars may overlap with other bars in the graph.

Viewing Virtual Service Catalog and Identity and Access Manager on Internet Explorer

A known issue can sometimes prevent the Virtual Service Catalog and Identity and Access Manager UIs from displaying correctly. This issue occurs in specific versions of Internet Explorer when combined with specific Windows operating systems. If you are having issues viewing either application in Internet Explorer, you can resolve this issue by modifying your Local Internet security options in your browser settings.

To modify your browser settings:

1. Click the **Tools** icon and select **Internet options**.
2. Click the **Security** tab in the **Internet Options** window.
3. Click **Local intranet** as the zone to view or change security settings.
4. Click **Sites**.
5. Clear the **Automatically detect intranet network** check box and all child check boxes underneath it.
6. Click **Advanced**.
7. If present, remove the URL for Virtual Service Catalog and Identity and Access Manager from the zone and click **Close**.
8. Click **OK** to close the Local Intranet window.
9. Click **OK** to close the Internet Options window.
10. Refresh the page.

Case Number: DE353202, DE350625, DE350641

Ampersand Character in Resource Names

When you create any of the following resources, do not include an ampersand character (&) in the name:

- Virtual service model
- Virtual service image
- Test case
- Suite
- Configuration
- Data set
- Staging document
- Keystore
- Recording session definitions
- Builder session name

Including the ampersand character can cause unexpected behavior. For example, a **SAXParseException** error can be generated.

This issue applies to both DevTest Workstation and DevTest Portal.

Case Number: DE254855

Searching for Long Strings in a Virtual Service

The virtual service editor in DevTest Portal includes a search field. If a string in the search field has more than 35 characters, the string cannot be found. The *total* search query can exceed 35 characters, but each string must be fewer than 35 characters.

As a work-around, you can do a whole word search or can use a regular expression (for example, **mySuperLongSearchTerm.***).

Data-Driven Virtual Services

If you use a Microsoft Excel file for an external data set, we recommend that you limit the size of the file. For example, if the heap size is 1 GB, avoid using Excel files that are larger than 3 MB.

Because of the Java APIs that CA Service Virtualization uses to read the Excel files, high memory usage can occur.

As an alternative, you can export the Excel file to a comma-separated value (CSV) file and use that instead.

Oracle New Install Issue

When using an Oracle database for Enterprise Dashboard, the DDLs must be run for a user with All/Admin privileges. Tables will not get created for a user with limited privileges.

dradis.properties File Required

When running Enterprise Dashboard with any database other than Derby, you must add the dradis.properties file, that has the database connection information, in the DevTest home folder before starting the Enterprise Dashboard. If you only add the connection properties to site.properties, a new dradis.properties file is created automatically, but you must restart the dashboard.

Demo Server Does Not Work on Solaris

The DevTest Demo Server does not work on Solaris. While initializing, it gets the following error:

```
WARN  [ServiceController] Problem starting service jboss:service=CorbaTransaction
```

Enabling or Disabling the Enhanced VS Recorder Can Cause Errors

When you have a recorder or any other Create Virtual Service tab open, and you switch on The Sandbox flag (Use Enhanced Recorder), the previously opened recorders start throwing errors and produce unexpected results. If the enhanced recorder setting is on, and recorder tabs are open, turning off the flag also produces errors.

When the Enhanced VS Recorder is enabled, you must close the recorder to return to The Sandbox.

Perfmon Restrictions

Windows Perfmon Metrics does not work on Windows operating systems that have Microsoft .NET 1.1+ installed.

Case Number: DE281792

Server Component Installation Restrictions

The following issues exist for installations where the data directory is different from the installation directory. This behavior is the same as with earlier releases with a shared directory. We do not recommend splitting the data directory from the installation directory, unless you are upgrading from an existing shared installation.

Windows:

- Services are not created on a shared install.

- You can not start Enterprise Dashboard as non-root user.
- The registry will start, but springframework has problems because it cannot write files to LISA_HOME.

Solaris:

- You can not start Enterprise Dashboard as non-root user.
- The registry starts, but springframework has problems because it cannot write files to LISA_HOME.
- The broker starts, but gives errors about not being able to write rules.xml, rules.xml.sample, and a directory for solr data.
- The Portal seems to start, has a reshub problem where the projects directory points to Projects under LISA_HOME, and there is no way to change it without editing a file in LISA_HOME.
- "Error while retrieving list of Projects. (500 AccessDeniedException /opt/CA/DevTest /Projects/.rhProjectInfo)"

OS X:

- You cannot stage a test on OS X with a shared install.

Using Data Files in DevTest Portal

If you run a test case that includes a data file, which references an external resource relative to the project structure, the reference of that resource in the test case will not work when the test runs. For such scenarios you must create a MARI in DevTest Workstation and manually add a reference to the external resource into it. Run the test case using this MARI in DevTest Workstation.

Testing Android Applications on OS X

To test an Android application on OS X, you must unzip the **appium-mac.itko.zip** file in the *DevTest_HOME*\addons directory. The contents of this zip file must reside in the DevTest_HOME\addons directory.

Incorrect Command in Readme file

The Docker examples readme file has the command:

```
docker build -t devtest-x11vnc-mysql:latest
```

It should be:

```
docker build -t dradis-licensed-mysql
```

Enterprise Dashboard Time

The timestamps shown on the Transaction Count and VS Count of Enterprise Dashboard are times taken from the server time zone, not the browser time zone.

Standalone Demo Server

You can not run a standalone demo server; that is, a demo server without having DevTest installed.

End of Service Announcement

In order to focus resources on the most up-to-date technology and continue to innovate on the Application Insight feature, we will be ending support for agent-based JDBC virtualization for Teradata on September 10, 2016. We will not support this feature starting with DevTest 8.5.

CA Technologies will continue to support agent-based JDBC virtualization for Teradata with older versions of CA LISA and DevTest under the CA Maintenance contract until those versions are no longer supported by CA Technologies.

For detailed information about the service contract for each version of DevTest Solutions, see DevTest Solutions Release and Support Lifecycle Dates.

End of Life Features

End of Life for a feature is defined as no future development, upgrades, or fixes are provided for that feature. Support for that feature ceases.

End of Life Feature as of August 1, 2017

Effective August 1, 2017 we will be deprecating the DevTest Mobile Testing feature. CA will continue to support current functionality for up to 1 calendar year after the End-of-Life announcement.

End of Life Features as of DevTest 10.0

The following features have reached End of Life status as of DevTest 10.0:

- Support has ended for importing paths from Splunk.

End of Life Features as of DevTest 9.5

The following features have reached End of Life status as of DevTest 9.5.

- Support has ended for DevTest support for IBM Rational Quality Manager.
- Support has ended for the Oracle AQ (JMS) step.
- Support has ended for DRDA virtualization.

End of Life Features as of DevTest 9.0

The following features have reached End of Life status as of DevTest 9.0.

- You can no longer configure registries through the Enterprise Dashboard UI. All supported registries must be configured through the properties files.
- You can no longer use the CA APM agents to see the health of the registry, coordinator, simulator, and VSE Server.
- Support has ended for the Oracle AQ (JPub) step.

End of Life Features as of DevTest 8.5

The following features have reached End of Life status as of DevTest 8.5.

- Support has ended for the Java .NET Bridge.

End of Life Features as of DevTest Solutions 8.2

The following features have reached End of Life status as of DevTest 8.2.

- Support has ended for Apple Safari browser on Windows.
- Support has ended for In-Container Testing (ICT).

End of Life Features as of DevTest Solutions 8.0.2

The following features have reached End of Life status as of DevTest 8.0.2.

- Support has ended for driver-based JDBC virtualization.

End of Life Features as of DevTest Solutions 8.0

The following features have reached End of Life status as of DevTest 8.0.

- The URL http://localhost:1505/reporting has been deprecated in 8.0. Accepted alternatives for navigating to the Reporting Console are:
  - http://localhost:1505/index.html?lisaPortal=reporting
  - http://localhost:1505, then navigate to reporting from there.

- APPC virtualization has been removed in CA Service Virtualization 8.0.
- Support has ended for the following:
  - Windows Vista
  - Windows Server 2008
  - Fedora 8
  - OS X 10.8
  - Ubuntu 9

  CA Technologies will continue to provide support for **Windows Vista, Windows Server 2008, OS X 10.8, Fedora 8, and Ubuntu 9** with older versions of CA LISA under CA Maintenance contract until those versions are no longer supported by CA Technologies or the operating system is no longer supported by Microsoft (whichever is sooner).

End of Life Features as of LISA 7.5

The following features have reached End of Life status as of LISA 7.5.

- The following operating systems are no longer supported as of LISA 7.5:
  - Windows XP
  - Windows Server 2000
- Legacy messaging steps are supported in LISA 7.5, but we plan to remove these steps in a future release. A new set of messaging steps that were introduced in LISA 7.5 replace these steps. For more information, see the *Migration Guide*.
- Driver-based JDBC virtualization is supported in LISA 7.5, but we plan to remove this feature in a future release. Agent-based JDBC virtualization is the replacement for this feature. For more information, see the *Migration Guide*.

End of Life Features as of LISA 7.1

The following features have reached End of Life status as of LISA 7.1.

- Web 2.0 is no longer supported.

End of Life Features as of LISA 6.0.9

The following features have reached End of Life status as of LISA 6.0.9.

- In-container testing support for JBoss and WebLogic through the deployment of a LISA EJB. This affects the Dynamic Java Execution step with the **Remote** option.
- Swing testing: The step is now in the custom extension.

- VSE WSDL generation that was developed in LISA 5 or earlier versions with the Axis client was replaced by the Quick Start option to create a VSI from WSDL.
- LISAINT 1.0 by including of .jar or EJB co-deployment. If a LISAINT tag is written, it is respected for filters and assertions.
- Web 2.0 only supports HTML and HTML 5 applications. We are dropping support for swing, Flex, and non-HTML testing.
- Web Service Execution (Legacy) testing web service (with Axis) is deprecated in favor of the Web Service Execution (XML) step. The step was moved to custom steps, and there is no new development. Customers should actively start to move to the Web Service Execution (XML) step.
- The legacy Raw SOAP request step is deprecated in favor of the Web Service Execution (XML) step. The step was moved to custom steps, and there is no new development. Customers should actively start to move to the Web Service Execution (XML) step.

DevTest Portal Functionality

This page identifies the specific features and functions that are available from DevTest Portal in the current release. Features not identified in the following table are available from DevTest Workstation.

CA Application Test

- Create an API test case with request/response pairs
- Manage API tests, created in DevTest Portal
- Manage test cases, created in DevTest Workstation
- Manage test suites, created in DevTest Workstation
- Manage projects
- Monitor tests
- Choose a configuration and staging document while running tests and test suites

CA Service Virtualization

- Virtualize a website (HTTP/S)
- Virtualize a website (TCP)
- Virtualize with Opaque Data Processing (ODP)
- Virtualize CICS activity
- Virtualize using request/response pairs

- Virtualize using WSDL
- Virtualize using Swagger
- Virtualize JDBC
- Data protocol handlers: REST, JSON
- Create and manage copybook bundles
- Manage virtual services
- Manage copybook bundles
- Monitor virtual services

DevTest Common Components

- Access Control (ACL) Administration
- Server Health Monitor

More Information:

- DevTest Portal
- Using DevTest Portal with CA Application Test
- Using Workstation with CA Application Test
- Using DevTest Portal with CA Service Virtualization
- Using Workstation with CA Service Virtualization
- Using Application Insight
- Access Control (ACL)

Third Party Acknowledgments

The attached zip file contains all third party acknowledgment information for the following products:

- Ace Editor 1.1.6
- ActiveMQ 5.4.2
- aeonbits owner 1.0.6
- AlivePDF 0.1.5RC

- amoebacode
- android studio ddms 22.0
- Angular 5.2.0
- Angular-file-upload 1.6.12
- Angular-tree-control 0.2.8
- Angular-chosen-localytics 1.0.6
- Angular-dashboard-framework 0.8.0
- Angular-dynamic-locale 0.1.27
- Angular-elastic 2.3.5
- Angular-http-auth 1.2.1
- Angular-jquery-dialog-service 2013-10-25 “fix-remove-dialog-from-dom-on-close”
- AngularJS 1.2.20
- Angularjs-slider 6.2.3
- Angular-material 5.0.4
- Angular-router-extras 0.0.10
- Angular-selection-model 0.7.0
- Angular-translate 2.2.0
- Angular-typeahead 0.2.0
- Angular-ui-codemirror 0.1.6
- Angular-ui-grid 3.0.0
- Angular-ui-layout
- Angular-ui-router 0.2.10
- Angular-ui-sortable 0.12.8
- Angular-utf8-base64 0.0.5
- Angular-xeditable 0.1.8
- annotation-detector 3.0.3
- annotation-detector 3.0.4
- Ant 1.8.1
- Ant 1.9.3

- Antlr 2.7.6
- Apache FOP 2.2
- Apache Solr 5.1.0
- Apache Wink 1.1.1
- Appium 1.4
- AppleJavaExtensions 1.0.0
- ASM 4.1
- ASM 5.0.1
- aspectjrt 1.8.9
- aspectjweaver 1.8.9
- asset-pipeline 3.0.0
- asset-pipeline-grails 2.8.2
- asset-pipeline-grails 2.10.2
- Automaton 1.0.0
- Avalon Framework 4.3.1
- Axiom 1.2.8
- axiom-impl 1.2.8
- Axis 1.4
- Balloon Tip for Java 1.2.4.1
- Batik 1.9
- Bean shell (bsh) 2.1.8
- bitbar 2.0
- Bootstrap 3.2.0
- Bouncy Castle 1.59
- Bower 1.7.9
- c3po 0.9.5
- cb2xml 0.94
- cglib-nodep 2.2.2
- Chart.js 2.7.1

- Codemirror 4.3.0
- Commons beanutils 1.9.2
- Commons Cli 1.4
- Commons Codec 1.10
- Commons Collections 3.2.2
- Commons Collections 4.1
- Commons DBCP 1.4
- Commons Digester 1.8
- Commons Discovery 0.2
- Commons Email 1.4
- Commons FileUpload 1.3.2
- Commons FileUpload 1.3.3
- Commons IO 2.4
- Commons Lang 2.6
- Commons Lang 3.4
- Commons Logging 1.2
- Commons Net 3.3
- Commons Pool 1.5.4
- Commons Validator 1.4.0
- Commons Validator 1.6.0
- Commons Launcher 1.1
- Concurrent Utilities 1.3.4
- converter-gson 2.3.0
- converter-scalars 2.3.0
- core-js 2.5.3
- d3 3.5.17
- d3-tip 0.6.4
- darcula
- Derby 10.12.1.1

- Derby 10.14.1.0
- DiffMatchPatch
- dom4j 1.6.1
- Dumbster 1.6
- EclipseLink 2.3.2
- elasticsearch 1.6.1
- Expression Layout 1.0
- EZMorph 1.0.6
- fastclick 1.0.2
- FlexLib 2.5
- flywaydb 3.1
- Font Awesome 4.7.0
- FreeMarker 2.3.20
- GeckoFX 1.9.1.0
- Generex 1.0
- geronimo-jms_1.1_spec-1.1.1.jar
- GlazedLists 1.5.0
- Google opensans version 8
- google-gson 2.8.2
- Gradle-node-plugin 0.13
- grails 3.1.1
- grails 3.2.9
- Grails Quartz Plugin 2.0.13
- grails-ziplet .3
- Groovy 2.4.7
- Guava 18.0
- Guava 19.0
- guice 3.0
- H2 1.4.178

- Hamcrest 1.3
- highlight.js 8.2
- HTMLDecoder 1.6.1
- HttpAsyncClient 4.0.2
- HttpBuilder 0.7.2
- HttpClient 4.3.6
- HttpClient 4.5.5
- HttpComponents 4.3.4
- HttpCore 4.3.3
- HttpCore 4.4.9
- IBM DB2 Driver for JDBC and SQLJ 10.5
- ICU4J 49.1
- intl 1.2.5
- intl.js 1.2.5
- jackson 2.4.5
- jackson-annotations 2.8.11
- jackson-core 2.8.11
- jackson-databind 2.8.11.1
- jackson-databind 2.8.11.2
- JACOB - Java COM Bridge 1.9.1
- Jakarta Byte Code Engineering Library 5
- Jakarta Commons Exec 1.1
- Jakarta Regexp 1.2
- Jasper Reports 6.1.0
- java persistence api 1
- java persistence api 2.0.3
- java tablelayout jdk 1.5
- Java UUID Generator 3.1.3
- javaFX 1.7.0_71

- Javassist 3.18.2-GA
- javax.annotation-api 1.2
- javax el 3.0.1-b08
- javax.ejb-api 3.2
- javax.servlet (Java API for Servlets) 3.1.0
- javax.ws.rs-api 2.0.1
- javax-inject 1
- javax-inject 2.2.0
- Jaxen 1.1.3
- Jaxen 1.1.4
- jax-rs-hateoas 0.4.5
- JBoss Application Server 4.2.3
- JBoss Logging 3.3.1
- JDBC 11.2.0.3
- jdbi 2.51
- JDOM 2.0.6
- Jenkins JUnit 1.21
- Jersey 1.17.1
- Jersey 2.5.1
- jersey-guice 1.17.1
- jersey-json 1.17.1
- jersey-multipart 1.17.1
- jersey-servlet 1.17.1
- Jetty 9.3.5
- Jetty 9.3.11
- JFreeChart 1.0.13
- JGoodies Forms 1.0.7
- Jhrome 2.0
- Jit 2.0.1

- jjwt 0.9.0
- jnetpcap 1.3.0
- joda-convert 1.2
- Joda-time 2.8.2
- Jode 0.9.1.2
- jQuery 2.1.1
- jQuery qTip2 Plugin 2.2.1
- jQuery UI 1.10.4
- JS Beautifier 1.5.5
- JSON 1.0
- JSON Diff Patch 0.1.22
- JSON Path 1.2.0
- JSON Smart 2.2
- JSONassert 1.2.3
- JSON-lib 2.4
- JSON-schema-validator 2.2.5
- JSON-schema-validator 2.2.6
- Jsoup 1.8.1
- JSQLParser 0.9.3
- jsr311-api 1.0
- JSW (Java Service Wrapper) 2.2.9
- JUnit 4.11
- JUnit 4.12
- Keycloak 3.4.3
- Keycloak-authz-client 3.4.3
- Lodash 4.17.4
- Log4J 1.2.13
- Lunr 0.5.4
- mapdb 2.0-beta 7

- MC4J 1.2.2
- metrics 3.0.1
- micrometer-core 1.0.3
- mimepull 1.9.3
- Mockrunner 0.4
- moment-timezone 0.5.11
- Nb-darcula 1.5
- nekohtml 1.9.21
- NETBeans IDE 6.5 subset
- Ng2-ace-editor 0.3.4
- Ng2-bootstrap 1.0.0
- Ng2-charts 1.6.0
- Ng-grid 2.0.13
- Ng-idle 0.3.5
- Ng-table 0.3.2
- Ng-table-resizable-columns alpha
- Ngrx 4.1.1
- Ngrx/core 1.2.0
- Ngx-translate/core 8.0.0
- Ngx-translate/http-loader 2.0.1
- node.js 0.10.25
- node.js 6.9.1
- not-yet-commons-ssl 0.3.11
- NPM 3.10.8
- nvd3 1.8.6
- ocLazyLoad 0.3.8
- ojdbc7 12.1.0.2
- okhttp 2.5.0
- okhttp 3.9.1

- Open SAML - Java 1.1b
- Open SAML 2.2.2.3
- OpenCover 4.5.1403
- OpenNLP 1.5.3
- Oracle Java Runtime Environment (JRE) 1.8.0_102
- ORO 2.0.8
- p6spy 1.1
- palominolabs metrics-guice 3.0.1
- Picketbox 5.0.2
- Pipeline API 2.20
- Pipeline Basic Steps 2.6
- Pipeline Groovy 2.39
- Pipeline Job 2.11.2
- Pipeline Step API 2.12
- Plain Credentials 1.1
- POI 3.17
- primeng 4.2.2
- primeng 5.2.0
- Procyon Java Decompiler 0.5.26
- pygments 1.6.1
- qdox 1.6.2
- Quartz 1.6.5
- RAML_Parser 0.9
- relaxng datatype 2011.1
- resteasy-client 3.1.4
- retrofit 1.9.0
- retrofit 2.3.0
- Rhino 1.7r3
- Rhino 1.7r4

- RSyntaxTextArea 2.5.3
- RxJS 5.5.6
- SAAJ 1.4
- ScintillaNET 2.2
- Scrollable Bar
- Select2 3.4.8
- Selenium 2.53.0
- Selenium Builder Interpreter Beta 8
- SIGAR - System Information Gatherer and Reporter 1.6.4
- Silk Icons 1.3
- Simple Logging Facade for Java (SLF4J) 1.7.5
- Slf4j-api 1.7.25
- SNMP4J 1.11.3
- Solr 6.6.2
- Spin.js 2.0.1
- Splunk Java SDK 1.5
- Spring Framework 4.1.6
- Spring Boot 2.0.0
- spring-data-solr 3.0.2
- Spring Security 4.0.1
- spring-security-config 5.0.0
- spring-security-core 5.0.0
- spring-security-data 5.0.0
- spring-security-web 5.0.0
- Stax2-api 4.0.0
- StringSearch 1.2
- structs 1.7
- Struts 1.2.9
- Swagger 2.0.1

- Swagger-jaxrs 1.5.8
- Swagger-parser 2.0.1
- SwaggerUI 2.14
- swing-layout 1.0.3
- swingx 1.6
- The Grinder - J2EE Performance Testing
- ThreeTen-Backport 1.3.6
- tidy 4aug00
- Tomcat Juli 9.0.1
- Trilead SSH Build 213
- UDDI4j 2.0.5
- UI Bootstrap 0.11.0
- ui-ace 0.2.3
- Velocity 1.7
- W3C Schemas
- WADL Java Parser 1.0
- Web-animations-js 2.3.1
- WIFE 7.3
- Woodstox 3.2.9
- Workflow Durable Task Step Plugin for Jenkins 2.13
- WSDL4J 1.6.2
- wss4j 1.5.4
- Xalan 2.7.2
- xdb6 12.1.0.1
- Xerces-J 2.11.0
- xml-apis 1.4.01
- XML Beans 2.6.0
- xml-resolver 1.2
- XML Schema 1.4.6

- xml-sec 1.4.2
- XML Unit 1
- XOM 1.2.10
- xpp3_min 1.1.4c
- xsom 20110809
- XStream 1.3.1
- XStream 1.4.8
- XULRunner 1.9.0.11
- Zip.js 4c93974
- Zip4j 1.3.1
- Zone.js 0.8.20

Product Accessibility Features

CA Technologies is committed to making sure that all customers, regardless of ability, can successfully use its products and supporting documentation to accomplish vital business tasks.

Installing

This section describes how to install DevTest Solutions and all required components. This section also provides information about upgrading from an earlier version.

- Preinstallation
- DevTest Installation Overview
- Installing and Configuring DevTest Server
- Installing DevTest Workstation
- Installing the Demo Server
- Verifying the DevTest Installation
- Installing Virtual Service Catalog
- Installing Integration Tools
- Installing DevTest Solutions with a Silent Install
- Docker Containers
- Preparing Your DevTest Upgrade
- Migrating the SDK

Preinstallation

Before you start the installation process, be sure to perform planning activities.

Verify that your system meets the system requirements.

Consider the following items:

- Is the installation targeted for workstation or server use?
- Which enterprise database do you plan to use?
  If the installation is a permanent installation or you intend to use it in an environment that is generally available, the supplied Derby database is not supported.

The **devtestlic.xml** license file must be available and present on your Enterprise Dashboard machine.

> **Note**: To upgrade from DevTest 10.3 or earlier, you must request a new license file. For more information, see License Administration.

Other considerations regarding the ideal layout of the various services and processes within the target systems include:

- The Enterprise Dashboard must be reachable by the registry, but it does not need to be local to the registry.
- The Enterprise Dashboard requires its own database instance.
- The simulators should be on the same network segment as the coordinator.

- The registry, coordinator, simulators, and virtual service environments require high-performance database access. Performance data is recorded directly to the database by these components. We recommend that the database be located in the same data center. Databases that are hosted within a virtual machine are not recommended for general availability use.

Part of planning a DevTest system is deciding how many DevTest Servers to install and what processes or services to run on each host computer containing a DevTest Server. The following pages can help you with these decisions:

- DevTest Server Components depicts the suite products with product-specific UIs and processes and the common UI and processes shared by all products. It also describes the purpose of each process and indicates where you can find more information.

- DevTest Process Relationships introduces major concepts concerning what processes you can run on each host in a distributed system.

The platform-specific installers and the demo server are available from CA Support Online. Installing the demo server is optional.

System Requirements

This page describes the system requirements for the operating system, database, browser, and more. Review this page before you install DevTest Solutions.

Operating System Requirements

We do not support DevTest on operating systems for which the software vendor has discontinued support.

The following operating systems are supported:

- Microsoft Windows
  - Windows Server 2012
  - Windows Server 2012 R2
  - Windows Server 2016
  - Windows 7
  - Windows 8.1
  - Windows 10

> Depending on the Windows settings are for Windows 7, 8.1, 9 and 10, you might be required to start the installer executable as “Run as Administrator.”

- Linux and UNIX
  - Red Hat Enterprise Linux 6.9, 7.3, 7.4

  - Ubuntu 16.04.x LTS
  - SUSE Linux 11 SP4, 12 SP3 (server component support only)
  - Oracle Solaris 11 (server component support only)
  - IBM AIX 7.1 (server component support only)
- OS X 10.12, 10.13

We recommend a 64-bit operating system and JRE, especially for DevTest Server.

DevTest Server JVM System Requirements

We recommend a 64-bit Java 8 for DevTest Server when requiring heap sizes above 1.3 GB.

Java JDK

Products composing DevTest Solutions are Java applications. An Oracle JRE is included with each operating system-specific installer (1.8.0_102 JRE including a **tools.jar** from the JDK) other than the generic UNIX installer. IBM JRE Version 8 is also supported if you supply your own JVM.

The minimum supported Java version for DevTest Workstation, DevTest Server, and VSE is Java 8 update 60.

This requirement is a DevTest-side requirement only. DevTest running in a Java 8 virtual machine (VM) can be used to test applications on application servers running older or newer JREs.

The following table lists the support that is provided for various JDKs:

	DevTest Workstation	DevTest Server - Coordinator, Simulator, Registry	DevTest Server - VSE	DevTest Java Agent (Application Insight)	Identity and Access Manager	Virtual Service Catalog
JDK 1.5	Not supported	Not supported	Not supported	Supported	Not supported	Not supported
JDK 1.6	Not supported	Not supported	Not supported	Supported	Not supported	Not supported
JDK 1.7	Not supported	Not supported	Not supported	Supported	Not supported	Not supported
JDK 1.8	Required for run time	Required for run time	Required for run time	Supported (WildFly 8.2 only)	Required for run time	Required for run time

Note: DevTest does not support IBM JRE Version 7, Oracle JRE 1.7, or OpenJDK.

Kerberos authentication is not supported on IBM JREs.

DevTest Workstation cannot test web services and messaging backbones on different versions of Java.

The following information provides general guidance on JDK support in the DevTest Java Agent (Application Insight):

- Application Insight supports systems that are based on JDK 1.5, JDK 1.6, JDK 1.7, and JDK 1.8 (WildFly 8.2 only).
- Only JDKs from Oracle are fully supported.

/tmp Partition

When you install DevTest on Linux and UNIX operating systems, ensure that the **/tmp** partition has at least 2 GB of available free space.

This requirement is necessary because the installer unpacks the install in the **/tmp** partition.

If the partition does not have enough free space, you might receive the following message:

```
java.io.IOException: No space left on device
```

DevTest Locks Folder

The following folder in the LISA_HOME directory requires read/write permissions:

- locks

In general, the rest of the LISA_HOME directory can be restricted with read-only permissions.

The lisatmp Folder

The following folder in the user home directory (on Linux, UNIX, and OS X) or Documents and Settings (on Windows) requires read/write permissions:

- lisatmp_x.x (if it exists)

DevTest CICS Agent

The DevTest CICS Agent supports the following CICS versions: 4.2, 5.1, 5.2, 5.3 and 5.4 running under z/OS 2.1.x, 2.2.x or 2.3.x

You can get the DevTest CICS Agent by opening a case with CA Support.

DevTest LPAR Agent

The DevTest LPAR Agent supports the following z/OS versions: 2.1.x, 2.2.x and 2.3.x

You can get the DevTest LPAR Agent by opening a case with CA Support.

Supplying Your Own JVM

The generic UNIX installer does not include a JRE. If you are using the generic UNIX installer, you *must* supply your own JVM.

Optionally, you can use this procedure with an installer for another platform to override the included JRE. See Using DevTest Workstation with Your Java Environment.

DevTest supports IBM JRE Version 8. DevTest does not support IBM JRE Version 7, Oracle JRE 1.7, or OpenJDK.

> If you use a Continuous Integration Server; for example, Jenkins or TeamCity, double check that the JVM you have specified for your build has the proper JCE library installed. Some CIs use their own configuration of JVM location and do not use the **JAVA_HOME** system variable.

Installing and Configuring the Oracle JVM

Follow these steps:

1. Download the Java SE Development Kit (JDK) 8 package for your platform from the Oracle website.

2. Download the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 8 from the Oracle website.

3. Install JDK 8 for your operating system on the computer where you plan to install DevTest. If there is no **java** directory, create one (mkdir java).
   For example, install JDK 8 in \usr\java, which creates the usr\java\jdk1.8.0_102 directory (JDK_HOME).

4. After you copy the jdk-8u65-*platform*. tar.gz file, enter this command:

   ```
   tar zxvf jdk-8u65-platform-x64.tar.gz
   ```

5. Set environment variables:

   - Set the **JDK_HOME** environment variable to the directory where you installed JDK 8.

   - Set the **JAVA_HOME** environment variable to point to the **JDK_HOME** directory.

   For example:

   ```
   cd \usr\java\jdk1.8.0_102
   pwd
   export JAVA_HOME=$PWD
   export JDK_HOME=$PWD
   ```

6. Extract the UnlimitedJCEPolicyJDK8 folder from the jce_policy-8.zip file. Move the following JAR files from this folder to the **JDK_HOME\jre\lib\security** directory:

   - local_policy.jar

   - US_export_policy.jar

   This action replaces the existing JAR files with the same names.

7. Copy the **tools.jar** file from the **JDK_HOME\lib** directory to the **JDK_HOME\jre\lib\ext** directory.

   ```
   cd $JDK_HOME\jre\lib\ext
   cp $JDK_HOME\lib\tools.jar .
   ```

8. If you are overriding the included JRE, perform the following steps after you install DevTest:

    a. Set the **LISA_JAVA_HOME** environment variable. For example:

    ```
    cd \usr\java\jdk1.8.0_102
    pwd
    export LISA_JAVA_HOME=$PWD
    ```

    b. Rename the **jre** directory under the DevTest installation directory to jre_default as described in Using DevTest Workstation with Your Java Environment.

Important: You may get the following error when starting Enterprise Dashboard:

JVMJ9VM007E Command-line option unrecognized: ${LISA_MORE_VM_PROPS}

The JVM could not be started. The maximum heap size (-Xmx) might be too large or an antivirus or firewall tool could block the execution.

To correct the error, set the system environment variable **LISA_MORE_VM_PROPS** to **-Xmx512m**

Installing and Configuring the IBM JVM

Note: The IBM JRE has been certified only on the AIX operating system.

Follow these steps:

1. Download the JVM for your platform from the IBM website.

2. Install JDK 8 for your operating system on the computer where you plan to install DevTest. If there is no **java** directory, create one (mkdir java).
   For example, install JDK 8 in \usr\java, into the \usr\java\ibm_sdk80 directory (JDK_HOME).

3. Copy the **tools.jar** file from the **JDK_HOME\lib** directory to the **JDK_HOME\jre\lib\ext** directory.

    ```
    cd $JDK_HOME\jre\lib\ext
    cp $JDK_HOME\lib\tools.jar .
    ```

4. If you are overriding the included JRE, perform the following steps after you install DevTest:

    a. Set the **LISA_JAVA_HOME** environment variable. For example:

    ```
    cd $JDK_HOME
    pwd
    export LISA_JAVA_HOME=$PWD
    ```

    b. Rename the **jre** directory under the DevTest installation directory to jre_default as described in Using DevTest Workstation with Your Java Environment.

Important: You may get the following error when starting Enterprise Dashboard:

JVMJ9VM007E Command-line option unrecognized: ${LISA_MORE_VM_PROPS}

The JVM could not be started. The maximum heap size (-Xmx) might be too large or an antivirus or firewall tool could block the execution.

To correct the error, set the system environment variable **LISA_MORE_VM_PROPS** to **-Xmx512m**

DevTest Server Requirements

CA Application Test and CA Service Virtualization require the registry in DevTest Server.

The minimum requirements for DevTest Server are:

- **CPU:** 2 GHz or faster, 8 cores minimum
- **RAM:** 8 GB available
- **Disk Space:** 50 GB
- **Database:** See Database Requirements. The registry database can reside on a different system and must have at least 500 GB of available storage.

For load and performance testing, the following resources are recommended:

- 250 virtual users for each simulator
- 1 processor core and 2-GB available RAM for each simulator

Example for 4000 concurrent virtual users: 16 simulators; 16 processor core; 32-GB available RAM (for DevTest)

For each data center:

- 1 test registry and 1 coordinator
- 1 processor core/process = 2 processor cores
- 2 GB available RAM/each = 4 GB available RAM (for DevTest)

A basic DevTest Server configuration has one enterprise dashboard, one registry, and one portal, which are required for all products. CA Application Test requires one coordinator server and one simulator server. CA Service Virtualization requires one virtual server environment. The Application Insight feature requires one broker. Only one enterprise dashboard server is required for any given configuration.

Note: The requirements that are listed here are intended as a guideline. For large load environments, we recommend contacting Professional Services to assist with developing a load generation environment to suit your needs.

DevTest Workstation for CA Application Test Requirements

The minimum requirements for DevTest Workstation are:

- **CPU:** 2 GHz or faster, 2 cores minimum
- **RAM:** 4 GB available
- **Disk Space:** 4 GB free space

CA Service Virtualization Requirements

The CA Service Virtualization component, VSE, is required for maintaining a virtualization environment. VSE is a server-level service and can coexist with a registry that has a coordinator and a simulator that is attached to it. The simulator and coordinator are not mandatory to run VSE.

The following requirements are baseline requirements only:

- **CPU:** 2 GHz or faster, two cores minimum
- **RAM:** 2 GB available RAM for VSE, in addition to the RAM requirement for DevTest Workstation, DevTest Server, or Application Insight
- **Disk Space:** 50 GB of free space

By default, the DevTest Java Agent uses the same database schema as the registry. However, you can configure the agent and the registry to use different database schemas.

Application Insight Requirements

The minimum requirements for Application Insight are as follows:

- **CPU:** 2 GHz or faster, 8 cores minimum
- **RAM:** 8 GB available
- **Disk Space:** 50 GB of local available disk storage

Identity and Access Manager Requirements

The minimum requirements for Identity and Access Manager are as follows. These requirements are in addition to the requirements for DevTest Workstation, DevTest Server, or Application Insight.

- **CPU:** 2 GHz or faster, 8 cores minimum
- **RAM:** 512 MB available RAM
- **Disk Space:** 1 GB of free space
- **Database:** Requires its own unique database and must have at least 25 GB of storage

Virtual Service Catalog Requirements

The minimum requirements for Virtual Service Catalog are as follows. These requirements are in addition to the requirements for DevTest Workstation, DevTest Server, or Application Insight

- **CPU:** 2 GHz or faster, 4 cores minimum
- **RAM:** 4GB available RAM
- **Disk Space:** 15 GB of free space

> **Note:** More disk space is required if you are indexing a large number of virtual services or virtual services with large operations.

Communication Requirements

Ensure that your firewall allows DevTest Solutions to send and receive network transmissions. The functionality that DevTest Solutions provides requires access to network resources and will not work properly if blocked by a firewall. Authorize DevTest Solutions applications.

All DevTest server components must have access to the External Database the Registry is connecting to. Each DevTest server component communicates directly with the database to record their actions. Any restriction to the flow of this data has adverse effects.

> **Note:** To implement secure communication, see Using SSL to Secure Communication and Using HTTPS Communication with the invoke APIs.

Communications to and from the DevTest ports must be opened in any relevant firewall. See the following pages:

- DevTest Server Default Port Numbers
- DevTest Workstation Default Port Numbers
- Demo Server Default Port Numbers

Database Requirements

The following components store information in a database:

- **DevTest Server:** The database is used for report results, which can be exported to other formats as needed. The database is also used for access control (ACL).
- **VSE:** The database is used for usage counts and legacy service images.
- **Application Insight:** The database is used for paths, including request and response data, SQL statements, and application logs. The database is also used for tickets.

- **Enterprise Dashboard:** The database is used for the DevTest Solutions Usage Audit Report data, other registry information, historical event logs, and metrics.

> **Important!** Enterprise Dashboard requires its own unique large database. The database can reside on a different system and must have at least 50 GB of storage. All DevTest server components must have access to the External Database the Registry is connecting to. Each DevTest server component communicates directly with the database to record their actions. Any restriction to the flow of this data has adverse effects.

By default, these components use an Apache Derby database that is included with DevTest. This database is adequate only for small deployments that do not require load and performance testing, and is not supported. For all other scenarios, configure DevTest to use an external database.

To ensure correct performance when using an external database, the database server and DevTest server should have high network bandwidth and low latency.

When run in a distributed configuration, DevTest depends on server components having a high-bandwidth, low-latency connection to a well-maintained, enterprise-class database.

All DevTest server components communicate directly with the database to record their actions. Any restriction to the flow of this data has adverse effects.

To ensure that your DevTest functions correctly, no DevTest server components should have a Round Trip Time (RTT) of greater than 20 ms to the database host. If the network latency exceeds this 20 ms value, you can expect performance problems.

> **Important!** If you are upgrading from an older version of DevTest and reusing the database, the database user must have permissions to modify the schema in order to make any schema and data changes necessary when migrating between the old and new versions.

The following external databases are supported:

- **IBM DB2 9.8, 10.1, and 10.5**
  The code page of the database must be 1208. In addition, the page size must be at least 8 KB for the DevTest registry and 16 KB for Enterprise Dashboard. Note that IBM DB2 9.8 is supported as a result of being integrated into IBM DB2 10.1.

- **MySQL 5.5, 5.6, and 5.7**
  The MySQL database must provide collation and characters set supporting UTF-8; double-byte characters are stored in the ACL and reporting tables. The default code page for the database must be UTF-8; it is not enough only to define your database as UTF-8.

- **Oracle 11g Release 2 and 12c Release 2**
  The character set must be Unicode set. For the initial creation of the database, the Oracle user must have the **CREATE VIEW** system privilege. Oracle must be installed with UTF-8 encoding. UTF-8 is the only encoding we have tested and is the only encoding we support.

- **Microsoft SQL Server 2008 R2, 2012, 2014, and 2016**

The schema is automatically created in the external database when the registry starts for the first time. Before the schema is created, ensure that the DevTest user has DBA privileges. After the schema is created, you can remove the DBA privileges from the user.

If your security policy does not permit this approach, the database administrator can manually create the schema. The DDL files in the **LISA_HOME\database** directory contain SQL statements that can serve as the basis for manually creating the schema. Provide this information to the database administrator.

Note: For more information about the configuration of an external database, see Database Administration.

Important! Registries and Enterprise Dashboards must each have a unique schema. Do not point multiple registries at the same database schema.

For load and performance testing, tune the external database to ensure that it can support the amount of data storage that DevTest requires.

The registry, coordinator, simulators, and any virtual service environments require high-performance database access. Performance data is recorded directly to the database by these components. CA recommends that the database be present within the same data center. Databases that are hosted within a virtual machine are not recommended for general availability use.

DDL Files

The **LISA_HOME\database** directory contains the following DDL files for DevTest:

- db2.ddl
- derby.ddl
- mysql.ddl
- oracle.ddl
- sqlserver.ddl

The **LISA_HOME\database** directory contains the following DDL files for the Enterprise Dashboard:

- db2_enterprisedashboard.ddl
- derby_enterprisedashboard.ddl
- mysql_enterprisedashboard.ddl
- oracle_enterprisedashboard.ddl
- sqlserver_enterprisedashboard.ddl

For information about Application Insight, see Java Agent Database Schema.

Supported Browsers

DevTest includes the following web-based portals, consoles, and dashboards:

- Enterprise Dashboard (http://*hostname*:1506/)
- DevTest Portal (http://*hostname*:1507/devtest)
- Forward Cars Demo Server (/*hostname*:3434/cars-app)
- Demo Server (http://*hostname*:8080/lisabank/)
- Identity and Access Manager (https://*hostname*:51111)
- Virtual Service Catalog (https://*hostname*:51110)

DevTest Portal requires HTML 5. Therefore, the latest internet browser technology is required to use the new web-based UI. The following internet browser versions support HTML 5:

- Google Chrome 64
- Mozilla Firefox 58
- Apple Safari 11 (Mac)
- Microsoft Internet Explorer 11
  Some versions of Internet Explorer 11 handle cookies in a way that is not compatible with the DevTest Enterprise Dashboard. If you experience issues accessing the Enterprise Dashboard, upgrade Internet Explorer to the latest version or use another browser.

DevTest Portal provides content to browsers with WebSockets. Any browser that you use must support WebSockets to be able to successfully use DevTest Portal. Any firewall or reverse-proxy that you use must also permit the use of WebSockets. If they do not, we recommend that you access DevTest Portal with HTTPS to allow WebSocket connectivity.

The recommended screen resolution for viewing DevTest Portal is 1024 x 768.

Note: Browser support for recording Selenium Integration tests is limited to Mozilla Firefox 42 through 47. After you import these tests to DevTest, running the test cases has been verified on Google Chrome, Mozilla Firefox 42 and 43, Apple Safari 7.0 (Mac), and Internet Explorer 10 and 11. Safari is not supported on Windows.

DevTest uses Selenium Library 2.53.0, which might not be fully compatible with the latest driver. Use a driver version that is compatible with Selenium API 2.53.0 (usually an older release version). Some newer web browsers might be unsupported.

Supported Languages

DevTest Solutions provides support for the following languages:

- English
- Japanese
- French

Note: The use of Korean characters is not supported.

Application Insight Support Matrix

This page provides a reference to various Application Insight components and features and when they are supported.

- DevTest Versions
- Application Insight Agents
- Java SE and EE JVMs
- Application Servers
- Visible Transaction Protocols
- Virtualization Features
- Baseline Features

DevTest Versions

The following table indicates when each DevTest version was released:

Product	2015	2016	2017	2018
DevTest10.4	No	No	No	Yes (from September)
DevTest 10.3	No	No	No	Yes (from March)
DevTest 10.2	No	No	Yes (from December)	Yes
DevTest 10.1	No	No	Yes (from April)	Yes
DevTest 10.0	No	Yes (from December)	Yes	Yes (through December)
DevTest 9.5	No	Yes (from June)	Yes	Yes (through June)
DevTest 9.1	No	Yes (from February)	Yes	Yes (through February)
DevTest 9.0	Yes (from November)	Yes	Yes (through October)	No
DevTest 8.5	Yes (from October)	Yes	Yes (through October)	No
DevTest 8.4	Yes (from September)	Yes	Yes (through July)	No

Product	2015	2016	2017	2018
DevTest 8.3	Yes (from August)	Yes	Yes (through July)	No
DevTest 8.2	Yes (from July)	Yes	Yes (through July)	No
DevTest 8.1	Yes (from June)	Yes	Yes (through July)	No
DevTest 8.0.x	Yes (from first half)	Yes	Yes (through March)	No
DevTest 8.0	Yes	Yes	Yes (through March)	No

Application Insight Agents

The following table indicates when each agent was released:

Agent	2015	2016	2017	2018
Pure Java Agent	Yes	Yes	Yes	Yes
Native Java Agent	Yes	Yes	Yes	Yes
.NET Agent (ERPConnect only)	Yes	Yes	Yes	Yes

Java SE and EE JVMs

The following table describes the support for various Java Virtual Machines:

JVM	2015	2016	2017	2018
Oracle 1.8*	Yes (DevTest +8.2)	Yes (DevTest +8.2)	Yes (DevTest +8.2)	Yes (DevTest +8.2)
Oracle 1.7	Yes	Yes	Yes	Yes
Oracle JRockit 1.6	Yes (DevTest 8.x)	Yes (DevTest 8.x)	Yes (DevTest 8.x)	No
Sun 1.6	Yes	Yes	Yes	Yes
Sun 1.5	Yes	Yes	Yes	Yes
Sun 1.4	Yes (LISA 7.x)	No	No	No
IBM 1.6	Yes	Yes	Yes	Yes
IBM 1.5	Yes (LISA 7.x)	Yes (LISA 7.x)	No	No

* Limited to WildFly 8.2.

OpenJDK is not supported.

Application Servers

The following table describes the application server support:

Application Server	2015	2016	2017	2018
IBM WebSphere 7.0	Yes	Yes	Yes	Yes
IBM WebSphere 8.5	Yes (DevTest 8.x)	Yes	Yes	Yes
JBoss AS 4.2	Yes	Yes	Yes	Yes
JBoss AS 7.3	Yes (DevTest 8.x)	Yes	Yes	Yes
JBoss EAP 6.2	Yes (DevTest 8.x)	Yes	Yes	Yes
WildFly 8.2	Yes (DevTest +8.1)	Yes	Yes	Yes
JBoss Fuse 6.2	No	Yes (DevTest +9.1)	Yes (DevTest +9.1)	Yes
Jetty 8	Yes (DevTest 8.x)	Yes	Yes	Yes
Jetty 9.1	Yes (DevTest 8.x)	Yes	Yes	Yes
Jetty 9.2	Yes (DevTest +8.3)	Yes	Yes	Yes
Oracle WebLogic 10.3	Yes	Yes	Yes	Yes
Oracle WebLogic 12.1.1	Yes (DevTest 8.x)	Yes	Yes	Yes
TIBCO BW 5.x	Yes	Yes	Yes	Yes
TIBCO EMS	Yes (DevTest 8.x)	Yes	Yes	Yes
webMethods Integration Server 9.6	Yes (DevTest +8.3)	Yes (DevTest +8.3)	Yes (DevTest +8.3)	Yes
webMethods Integration Server 9.5	Yes	Yes	Yes	Yes
webMethods Integration Server 9.0	Yes	Yes	Yes	Yes
webMethods Integration Server 8.2	Yes (LISA 7.x)	Yes (LISA 7.5.2)	No	No
webMethods Integration Server 7.1	Yes (LISA 7.x)	Yes (LISA 7.5.2)	No	No
SAP NetWeaver 7.3 and 7.4	Yes (LISA 7.5.2*, DevTest +8.0.1*)	Yes (LISA 7.5.2*, DevTest +8.0.1*)	Yes (DevTest +8.0.1*)	Yes

* Limited to the following protocols: EJB, JMS, REST, and Web service.

Visible Transaction Protocols

The following table indicates when support for capturing and viewing protocols was added:

Feature	2015	2016	2017	2018
Exception	Yes	Yes	Yes	Yes
Logs	Yes	Yes	Yes	Yes
HTTP	Yes	Yes	Yes	Yes
Web service	Yes	Yes	Yes	Yes
REST	Yes	Yes	Yes	Yes
RMI	Yes	Yes	Yes	Yes
EJB	Yes	Yes	Yes	Yes
JDBC	Yes	Yes	Yes	Yes
JCA	Yes	Yes	Yes	Yes
JMS	Yes	Yes	Yes	Yes
WebSphere MQ	Yes	Yes	Yes	Yes
webMethods	Yes	Yes	Yes	Yes
TIBCO	Yes	Yes	Yes	Yes
CICS	Yes	Yes	Yes	Yes
SAP JCo	Yes	Yes	Yes	Yes
SAP IDoc	Yes (LISA +7.5)	Yes	Yes	Yes
ERPConnect 3.0	Yes	Yes	Yes	Yes
Java	Yes (DevTest +8.2)	Yes (DevTest +8.2)	Yes (DevTest +8.2)	Yes

Virtualization Features

The following table indicates the support for various virtualization features:

Feature	2015	2016	2017	2018
Microsoft JDBC Driver 4.0 to Microsoft SQL Server 2012 database (sqljdbc4.jar)	Yes (DevTest +8.0)	Yes (DevTest +8.0)	Yes (DevTest +8.0)	Yes
jTDS 1.3.1 JDBC driver to Microsoft SQL Server 2012 database (sqljdbc4.jar)	Yes (DevTest +8.0)	Yes (DevTest +8.0)	Yes (DevTest +8.0)	Yes
Microsoft JDBC Driver 4.0 to Microsoft SQL Server 2008 R2 database (sqljdbc4.jar)	Yes (DevTest +8.0)	Yes (DevTest +8.0)	Yes (DevTest +8.0)	Yes
jTDS 1.3.1 JDBC driver to Microsoft SQL Server 2008 R2 database (sqljdbc4.jar)	Yes (DevTest +8.0)	Yes (DevTest +8.0)	Yes (DevTest +8.0)	Yes
	Yes (DevTest +8.0)	Yes (DevTest +8.0)	Yes (DevTest +8.0)	Yes

Feature	2015	2016	2017	2018
IBM DB2 9.5 JDBC driver to IBM DB2 10.5 database (db2jcc.jar 3.57.82 and db2jcc4.jar)				
IBM DB2 9.5 JDBC driver to IBM DB2 9.8 database (db2jcc.jar 3.57.82 and db2jcc4.jar)	Yes (DevTest +8.0)	Yes (DevTest +8.0)	Yes (DevTest +8.0)	Yes
IBM DB2 9.5 JDBC driver to IBM DB2 9.5 database (db2jcc.jar 3.57.82 and db2jcc4.jar)	Yes (LISA +7.5.1)	Yes	Yes	Yes
Oracle 11g JDBC driver to Oracle 11g database (ojdbc7.jar, ojdbc6-11.1.0.2.jar, and ojdbc5.jar)	Yes	Yes	Yes	Yes
Oracle 11g JDBC driver to Oracle 12c database (ojdbc7.jar, ojdbc6-11.1.0.2.jar, and ojdbc5.jar)	Yes (DevTest +8.0)	Yes (DevTest +8.0)	Yes (DevTest +8.0)	Yes
Progress DataDirect Connect for JDBC 5.1 to Oracle 11g database (oracle.jar)	No	Yes (DevTest +9.1)	Yes (DevTest +9.1)	Yes
Informatica DataDirect JDBC Driver 5.1.2 to Oracle 11g database (com.informatica.datadirect-dworacle-5.1.2.HF1.jar)	No	Yes (DevTest +9.1)	Yes (DevTest +9.1)	Yes
Apache Derby 10.8.3.0 JDBC driver to Apache Derby database (derbyclient-10.10.2.0.jar)	Yes	Yes	Yes	Yes
Teradata 14.10	Yes (LISA 7.5 through DevTest 8.4)	Yes (LISA 7.5 through DevTest 8.4)	Yes (LISA 7.5 through DevTest 8.4)	No
Web HTTP	Yes	Yes	Yes	Yes
REST	Yes	Yes	Yes	Yes
Web services	Yes	Yes	Yes	Yes
EJB	Yes	Yes	Yes	Yes
JCA	Yes	Yes	Yes	Yes
JMS	Yes	Yes	Yes	Yes
WebSphere MQ	Yes	Yes	Yes	Yes
webMethods	Yes	Yes	Yes	Yes
TIBCO BW	Yes	Yes	Yes	Yes
CICS	Yes	Yes	Yes	Yes
SAP JCo	Yes	Yes	Yes	Yes

Feature	2015	2016	2017	2018
SAP IDoc	Yes (LISA +7.5)	Yes	Yes	Yes
ERPConnect 3.0	Yes	Yes	Yes	Yes
Java	Yes (DevTest +8.2)	Yes (DevTest +8.2)	Yes (DevTest +8.2)	Yes
RMI	Yes (DevTest +9.0)	Yes (DevTest +9.0)	Yes (DevTest +9.0)	Yes

Baseline Features

The following table indicates the support for various baseline features:

Feature	2015	2016	2017	2018
Web HTTP	Yes	Yes	Yes	Yes
Web Services	Yes	Yes	Yes	Yes
REST	Yes	Yes	Yes	Yes
REST (stateful)	Yes	Yes	Yes	Yes
RMI	Yes	Yes	Yes	Yes
EJB	Yes	Yes	Yes	Yes
EJB (stateful)	Yes	Yes	Yes	Yes
JMS	Yes	Yes	Yes	Yes
WebSphere MQ	Yes	Yes	Yes	Yes
webMethods	Yes	Yes	Yes	Yes
TIBCO BW and EMS	Yes	Yes	Yes	Yes

DevTest Server Components

When you install DevTest Server, the following products are installed:

- CA Application Test
- CA Service Virtualization

These products use various components. The products share some components; other components are specific to a product.

The following graphic shows the products and components. The components are divided into executables and user interfaces.

The following table lists the executables and indicates which products use them.

Name	Required By	Required To	More Information
Identity and Access Manager	Enterprise Dashboard	Provides user authentication for all DevTestcomponents.	Identity and Access Manager

Name	Required By	Required To	More Information
	Registries (All DevTest Solutions products)		
Enterprise Dashboard	Registries (All DevTest Solutions products)	Monitor enterprise activity. Generate the Usage Audit Report.	Use the Enterprise Dashboard
Registry	All DevTest Solutions products	Register DevTest Server and DevTest Workstation components. The registry is the central hub or engine for all processes. Collect usage data for the Usage Audit Report.	The Registry
Portal	All DevTest Solutions products	Start DevTest Portal.	DevTest Portal
Coordinator	CA Application Test	Coordinate tests run on simulators.	Coordinator Server
Simulator	CA Application Test	Run tests.	Simulator Server
Virtual Service Environment (VSE)	CA Service Virtualization	Deploy virtual services.	Virtualizing a Service
Broker	Application Insight	Coordinate Java agents.	Java Agent Components

The registry, coordinator server, simulator server, and VSE are "headless" Java applications that run in separate virtual machines.

A minimal DevTest Server configuration for CA Application Test and CA Service Virtualization must include at least one of each of these components. There can be as many instances of each type as is needed for a specific testing environment.

Typically, a DevTest Server configuration for CA Application Test has one registry, one coordinator server, and multiple simulator servers.

VSE is a server-level service. The service can coexist with a registry that has a coordinator and a simulator that are attached to it. The simulator and coordinator are not mandatory to run VSE.

You start DevTest Server by starting a series of processes or services. The components that you start first (Enterprise Dashboard, registry, and portal) support DevTest Solutions. The other components are product-specific. For startup details, see Start the Server Components.

DevTest Workstation must be installed on desktop computers for users who author test and virtual model assets. Any number of workstations can attach to the registry and can share the server environment.

*Multiple Coordinators, Simulators, and VSEs on the same machine will have different port numbers.

DevTest Process Relationships

When you are installing and configuring DevTest Solutions, the installer is typically run on multiple computers with different selections. Consider the following example:

- Computer 1
  Identity and Access Manager, Enterprise Dashboard, and server components, where the embedded workstation may be unused.

- Computer 2
  Server where the registry component links to Identity and Access manager and Enterprise Dashboard on Computer 1. Typically, only one registry is needed for a distributed system. This second registry is added here to demonstrate that multiple registries are supported.

- Computer 3
  Workstation and Demo Server with links to the registry on Computer 1. This computer represents user machines.

- Computer 4
  Can also represent user machines, particularly users who need only portal access.

As you set up your environment, it is important to have a clear understanding of your authentication and authorization needs. The registry is at the center of all DevTest systems. Identity and Access Manager (IAM) is the component that provides user authentication for all registries and other components that require authentication. IAM lets you manage users, roles, sessions, and events. It also lets you define your realm settings, define your authentication database, and federate external user databases. However, it is important to understand that, although you create and manage users and roles in IAM, the permissions for the individual roles are defined at the registry level through DevTest Portal.

When you are installing DevTest Solutions, you have the option to install an Identity and Access Manager instance or point to an existing IAM. Typically, you would create a single IAM that controls authentication for your entire enterprise. Each Enterprise Dashboard and each registry in your enterprise can then point to that single IAM. The advantage of a single IAM is that it provides centralized management of users and roles across your enterprise. If you are managing your users through an LDAP server, this also provides a single connection to that server for your entire DevTest environment.

The following diagram illustrates a distributed environment with a single IAM and multiple registries.

The limitation of a single IAM, is that you cannot manage multiple registries separately. If you have multiple registries in you environment, and you need to manage each registry separately, you have to install a separate IAM for each registry. This is especially true if users from one registry cannot have rights to another. For example, assume the following:

- User 1 needs SV Power User permissions for Registry A.
- User 1 needs Runtime permissions for Registry B.
- User 2 needs Runtime permissions for Registry A, but cannot access Registry B.

If you have a single IAM, and you assign SV Power User permissions *and* Runtime permissions to User 1, that user then has *both* sets of permissions for *both* registries. If you assign Runtime permissions to User 2, User 2 then has Runtime permissions for *both* registries, rather than

permissions for Registry B only. If you have multiple registries with complex authentication scenarios that cannot be accomplished through the assignment of permissions to a given role, then you should install a separate IAM for each registry.

The following diagram illustrates an environment with multiple IAMs.

Also, you can install the DevTest server on a computer to run only one product-specific service, such as the simulator executable or service.

This example system shares the following characteristics with any DevTest system:

- Each DevTest system has one or more IAM instances.

  Note: You must have one IAM and one Enterprise Dashboard per accessible network. If you have closed networks (networks that cannot reach each other), you need an IAM and an Enterprise Dashboard for each network where registries are running.

- Each IAM is connected to one or more registries.
- Each DevTest Solutions system has one Enterprise Dashboard.
- Each Enterprise Dashboard is connected to one IAM and to one or more registries. Typically, one registry is sufficient.
- Each DevTest Server installation installs one registry, the component that audits all user activity.
- Each DevTest Server installation installs one local workstation. This embedded workstation is used in a standalone installation. When the DevTest Server is installed in a distributed environment, the local workstation may be unused.
- Each DevTest Server registry in a distributed system can connect to one or more remote workstations.

- A Web browser can access DevTest Portal by browsing to a DevTest Server on port 1507 where the URL ends with **devtest**. The portal can be accessed from a Web browser without needing any DevTest Solutions software to be installed on the computer.

Each registry and each Enterprise Dashboard must be connected to an IAM before they can start.

To get a license, the registry must connect to an Enterprise Dashboard. Once connected, the registry must forward use counts to an Enterprise Dashboard. A dashboard can run with no registries. The first time a registry starts, it must connect to an Enterprise Dashboard to download the license. For subsequent startups, the registry uses a cached version of the license, so it does not need to be connected to an Enterprise Dashboard.

If the connection between a registry and Enterprise Dashboard is lost, the registry caches the use counts locally. When the connection is re established, the registry forwards the use counts to the Enterprise Dashboard.

More Information:

- DevTest Server Components
- Installing and Configuring DevTest Server
- Identity and Access Manager
- Access Control (ACL)

DevTest Solutions Architecture

DevTest can be described by looking at the architecture of each of its functional areas.

- Architecture for Testing
- Architecture for Service Virtualization
- Architecture for Application Insight
- Enterprise Dashboard Architecture
- Virtual Service Catalog

Architecture for Testing

CA Application Test uses the following DevTest Server components:

- Registry
- DevTest Portal
- DevTest Workstation
- Coordinator server
- Simulator server

DevTest Portal and DevTest Workstation are used to create and monitor the tests, but the test cases are run in the DevTest Server environment. A coordinator server and a simulator server are embedded in DevTest Workstation.

The following diagram shows the CA Application Test architecture.

All tests 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.

An embedded instance of DevTest Workstation runs on the same computer that the DevTest Server is running on. You can also configure standalone DevTest Workstations to run on separate computers in a large distributed environment. Your testing requirements dictate the server architecture to be used. DevTest Solutions can be scaled for large testing environments by distributing the different components onto different hardware/operating systems.

DevTest products include CA Application Test, CA Service Virtualization, and Application Insight, each of which connects to a central DevTest registry. CA Application Test users use an embedded DevTest Workstation with its optional coordinator server and simulator servers and optionally and all remotely installed DevTest Workstations. Components of all DevTest products connect to a central registry.

Architecture for Service Virtualization

The following diagram shows the architecture for service virtualization.

*Multiple Coordinators, Simulators, and VSEs on the same machine will have different port numbers.

Architecture for Application Insight

The following diagram shows the architecture for Application Insight.

*Multiple VSEs on the same machine will have different port numbers.

Enterprise Dashboard Architecture

The following diagram shows the Enterprise Dashboard architecture.

Virtual Service Catalog

The following diagram shows the Virtual Service Catalog architecture.

More Information:

- DevTest Server Components
- Data Flow in DevTest Server for CA Application Test and CA Service Virtualization

Data Flow in DevTest Server for CA Application Test and CA Service Virtualization

The following graphic shows the data flow among the registry, DevTest Workstation, coordinator server, simulator servers, and database.

The coordinator server sends test cases to one or more simulator servers in the form of Model Archives (MARs).

The simulator servers interact with the system under test. The types of data that are exchanged between these components can vary enormously. The data could be simple HTTP requests with HTML responses, web service calls, database calls, and so on.

The various components send metrics and reporting information to the database.

The registry hosts the reporting portal, so it communicates with the database to retrieve reporting data.

The following graphic shows the data flow among the registry, VSE, DevTest Portal, and database.

Download DevTest Solutions Installers

The Download Center on CA Support Online lets you download one or more platform-specific installers and the optional Demo Server.

Follow these steps:

1. Go to https://support.ca.com.
2. Log in with your CA Support Online user name and password.
3. Click **Download Management** (below the cloud icon) or from the **Menu** option.
   The Download Management page opens.
4. In the **Search by Product Name** box, enter **Service Virtualization** and select the drop-down option for **CA Service Virtualization (DevTest / LISA / VSE / Application Test)**.
5. On the **Product Downloads** tab, scroll down to see the product results.
   The Solutions Downloads tab is not enabled for DevTest products.
6. Click the product name in blue.
   The **Product Downloads** page for that product appears.
7. Click the **Release** drop-down at the upper right of the page to select the product release.

8. Find the installer that matches the OS and processor needs. The Installation wizard contains includes DevTest Server and DevTest Workstation. To download, click the **Download Now** icon to the right of the file name.

9. If you have not selected a download preference, you are prompted to select one here. Select **HTTP Via Internet Browser**.

> For this download, you must disable pop-up blockers on this page.

10. Save the ZIP file to your desired location.

11. We strongly recommend that you download the DevTest Demo Server.

> **Note:** To use Virtual Service Catalog, you must also download the Virtual Service Catalog installer. For more information about installing Virtual Service Catalog, see Installing Virtual Service Catalog.

DevTest Installation Overview

Use the following guidelines to install DevTest Solutions for the first time:

1. Download DevTest Solutions installers.

2. Store the **devtestlic.xml** license file on the target system.

> **Note:** Record the location of the license file for future use by CA Support.

> **Note:** To upgrade from DevTest 10.3 or earlier, you must request a new license file. For more information, see License Administration.

3. If you are using the generic UNIX installer, supply your own JVM.
4. Install DevTest Solutions on as many systems as you need. When you install the first DevTest Server, select **New Enterprise Dashboard** and browse to the **devtestlic.xml** license file.

Post-Installation Steps

1. Configure Identity and Access Manager.
   - If you are upgrading from 10.3 or earlier, also see Upgrading the Authentication Component.
2. Configure external databases.
   - Configure an external database for registries.
   - Configure an external database for the Enterprise Dashboard.

   Note: There is no migration path from the Apache Derby database that is installed with DevTest Solutions to an enterprise-grade database. If you use Derby, you must re-enter the mandatory user authentication (role) data when your system becomes generally available, where an enterprise-grade database is required.
3. Activate the Enterprise Dashboard and all registries.
   a. Activate the registries.

   > **Note:** This process also activates the Enterprise Dashboard with a product key.

   b. Verify registry activations. (New or upgrade)
4. Perform post-installation tasks.
5. (Optional) Install more workstations with the Demo Server for use with each registry.
6. Verify the installation.
7. To prevent unauthorized use, change the passwords for standard users.
8. Install the integration tools that you need.

When the installation is complete, see Administering for details on setting up ACL-enforced security and honoring your license agreement.

Installation Options

You must execute the installer as administrator on Windows to ensure that you have the appropriate permissions for all installer actions, such as creating desktop links, registry entries, and installing services.

CA Application Test and CA Service Virtualization are installed together as one solution set.

- Destination Directory
- Typical Configurations
- Options for Installing Required Processes (Windows)

Destination Directory

When you are installing DevTest Solutions, you are prompted to select a destination directory for the installation. This section outlines some additional considerations as you select your installation destination and data directory locations.

Installation Destination Directory

The default directory is:

- Windows: C:\Program Files\CA\DevTest.
- Unix: /opt/CA/DevTest
- OS X: /opt/CA/DevTest

If you choose to install in a different location, browsing to a path and entering the name of a new folder automatically creates the folder if it does not exist.

if you want to keep the current release separate from older releases, consider a name that includes the release identifier. For example, enter: C:\DevTestServer_10.4.

Data directory

If you install both Workstation and Server, we recommend that the data directory is the same as the installation folder.

If you install Workstation only, the default data directory is a subfolder named *DevTest* within the user's home folder.

If you use the silent installer, you must specify a data directory (such as the user's home folder) if you want to separate the data directory from the installation directory.

- Windows example: %userprofile%/DevTest
- Mac/Linux/UNIX example: ~/DevTest

Using the %userprofile% (or '~') user home variables might be useful for system administrators who are installing this product on behalf of other users. These variables resolve to the user's home folder of whatever user is using the product.

If the data directory is different than the installation directory, DevTest creates a lisa.user.properties file in both the installation directory and in the user's home folder. The **lisa.data.dir** property in this property file tells the product where the data directory is located.

The user must have write permissions to the specified data directory. If the data directory does not exist, the installer creates it and test to be sure it is writable.

The components included in the data directory are listed below:

- Workstation Components: _local.properties, _logging.properties, _de-identify.xml, hotdeploy, library for customizations, projects with assets, examples folders.
- Server Components: hotdeploy, CVS Monitor, VSE Deploy.

On initial start-up of DevTest Workstation, Projects and Examples are copies or refreshed from the install directory to the data directory.

> If your DevTest Workstations need more memory, you can customize the memory allocation settings either by editing the .vmoptions file in the installation directory or by setting the system environment variable LISA_MORE_VM_PROPS=-Xms?m -Xmx?m on the machine where Workstation runs.

Typical Configurations

Standalone

If you are installing DevTest Solutions on one computer, make the following selections:

- Server
- New Enterprise Dashboard
- Install demo server

Distributed

If you are installing a distributed DevTest Solutions system on multiple computers, consider the following approach:

1. For the *first installation*, from a server that meets system requirements, make the following selections:
   - Server
   - Install or upgrade Identity and Access Manager
   - New Enterprise Dashboard
2. For *more installations of servers*, make the following selections:
   - Server
   - Connect to an existing Identity and Access Manager

- Existing Enterprise Dashboard Server

> **Note:** You install the server even if you plan to use only one component from it. For example, if you are collecting many metrics or requesting many reports, you can run a coordinator server on a computer with no other service running.

3. To install components on individual hosts for CA Application Test and CA Service Virtualization users, make the following selections:

   - Workstation
     When users log in to the workstation, they link to the registry installed with one of the servers.

   - Install demo server
     New users could benefit from the tutorials.

Options for Installing Required Processes (Windows)

Up to seven processes must be started to use DevTest Solutions. The installation wizard lets you select how to perform this task:

- Create a Start menu folder

- Install Services

You can select one of these options, both options, or neither option. If you select neither, you can start the process executables from the **bin** folder under the DevTest server installation directory (**LISA_HOME\bin**) or from a command prompt.

How License Activation Works

One DevTest Solutions license (**devtestlic.xml**) is issued for each enterprise. This file unlocks all functionality of DevTest Solutions.

> **Note:** To upgrade from DevTest 10.3 or earlier, you must request a new license file. For more information, see License Administration.

When you install your first DevTest Server with the DevTest Solutions Setup wizard, you navigate to the license file. The Setup process then installs the Enterprise Dashboard and places that file in the DevTest Server installation directory (LISA_HOME). When you install more DevTest registry servers, you provide the URL to the Enterprise Dashboard Server.

> **Note:** To change the location of the Enterprise Dashboard database, you must rerun the installer while retaining the current database schema. Back up customized properties files before you reinstall. These include local.properties, site.properties, and the various vmoptions files.

Important! If the URL to the Enterprise Dashboard changes, you must update the **site.properties** file in the LISA_HOME directory for each registry server that reports to the Enterprise Dashboard, setting the **devtest.enterprisedashboard.host**=*somehost* property with the new information.

When you start the Enterprise Dashboard process and each registry process, the Enterprise Dashboard process reads the registry settings and it activates the registries. The activated registries are displayed on the Enterprise Dashboard UI for your verification.

Important! If the host name or port of a registry changes, you must restart the registry so that the Enterprise Dashboard can reactivate it.

Note: See Reactivate a Registry or Enterprise Dashboard.

Installing and Configuring DevTest Server

This section describes how to install and configure DevTest Server.

- Install DevTest Server on Windows
- Install DevTest Server on UNIX
- Install DevTest Server on OS X

Note: If you are upgrading from an earlier version of DevTest Solutions, carefully consider the impacts of the upgrade to your current authentication scheme. For more information, see Upgrading the Authentication Component and Identity and Access Manager.

Install DevTest Server on Windows

Before you start the installation procedure, download the following files from the Download Center:

- The installer for your platform
- (Optional) The demo server zip file

> **Note:** Typically, the Demo Server is installed with the Server components only in a standalone system. In a distributed system, the Demo Server is usually installed with DevTest Workstation for users who are new to DevTest Solutions. The Demo Server is used for tutorials and for many of the artifacts in the Examples project.

Follow these steps:

1. Run the installer file; for example, devtest_win_x64.exe.
   The Welcome to the DevTest Solutions Setup Wizard step opens.

2. Click **Next**.
   The CA End User License Agreement step opens.

3. Read the license agreement, scrolling to the end, select the **I accept the terms of the License Agreement** option, and click **Next**.
   The Select Components step opens.

4. Ensure that the **Server** check box is selected. An embedded Workstation is installed with the Server. Click **Next**.
   The Select Destination Directory step opens.

5. Enter the path and folder name for the installation directory (LISA_HOME).

   - Consider a name that includes the release identifier, if you want to keep the current release separate from older releases. For example, enter: C:\DevTestServer_10.4. Or, accept the default (C:\Program Files\CA\DevTest).

   - If you browse to a path, and you enter the name of a new folder, the installation wizard creates the folder.

   - For more information about selecting the appropriate destination and data directory, see Installation Options.

6. If you are upgrading from a shared install, enter the path and folder name for the data directory.

   - If you are not upgrading from a shared install, this option does not appear.

   - Enter the path and folder name for the data directory. The default is the data directory detected from the previous shared installation.

   - The specified data directory must exist and the user must have write permissions to it. If it does not exist, the installer will create it and will test to be sure it is writable.

   - If the data directory is different than the installation directory, DevTest creates a lisa.user.properties file in both the installation directory and in the user's home folder. The **lisa.data.dir** property in this property file tells the product where the data directory is located.

   - The data directory will include hotdeploy for server components. Other server component files will be in the installation directory.

7. Click **Next**.
   The Identity and Access Manager step opens. Identity and Access Manager is required for a DevTest Solutions installation.

8. Select one of the following options and click **Next**.

   - **Install or upgrade Identity and Access Manager**
     If this is the first time you are installing IAM, or if you want to upgrade your existing IAM, select this option.

   - **Connect to an existing Identity and Access Manager**
     If you want to connect to an existing IAM in your environment, select this option. Enter the URL for the existing IAM. This URL is the host and port where Identity and Access Manager was installed. Clicking **Validate** verifies that you have entered a valid URL, but does not attempt to connect to IAM.

     Note: A single Identity and Access Manager instance for your entire enterprise allows component access to be managed in a single location. However, if you have multiple registries and need to manage access to each registry separately, you need to install a separate IAM for each registry. This is especially true if users from one registry cannot have rights to another. For more information, see DevTest Process Relationships.

   The Identity and Access Manager Configure Database step opens.

9. Select the type of database you want to use for your user management.

   - If you select the default value, **Embedded Database**, no additional configuration is required.

   - If you select another option, enter the details for your database and click **Validate**.

   Note: Using the embedded database is not recommended as an enterprise solution. This database is not intended for use in a production environment.

10. Click **Next**.
    The Enterprise Dashboard step opens.

11. Specify one of the following options, and click **Next**.

    - **New Enterprise Dashboard (Specify location of license file)**
      If this is the first Server you are installing, click **Browse**, navigate to the location of your license file, **devtestlic.xml**. The installer copies the **devtestlic.xml** file to the specified installation directory (LISA_HOME) on the local host. The new Enterprise Dashboard process is installed in the LISA_HOME\bin directory. This option specifies that all registries are to connect to the new Enterprise Dashboard.

      Note: To upgrade from DevTest 10.3 or earlier, you must request a new license file. For more information, see License Administration

    - **Existing Enterprise Dashboard Service**

If this is not the first Server that is installed in this network, enter the location of the existing Enterprise Dashboard. This option specifies that the registry installed with this Server is to connect to the existing Enterprise Dashboard.

```
localhost:1506
```

> To find the URL for an existing Enterprise Dashboard, navigate to the DevTest home directory and locate the **site.properties** file. The **devtest.enterprisedashboard.host** and **devtest.enterprisedashboard.port** properties define the Enterprise Dashboard location.

12. Click **Next**.
    The CA Customer Experience Improvement step opens.

13. Select whether you want to send diagnostic and usage data to CA Technologies to help us improve our products and services.

    - This step only appears if you are installing a new Enterprise Dashboard. If you are connecting to an existing Enterprise Dashboard, this step does not display.

    - For more information about this option and the data that is collected, see Telemetry and Telemetry Data.

14. Click **Next**.
    The Demo Server step opens.

15. If you want the installer to unzip the DevTestDemoServer.zip into the LISA_HOME directory, select the **Install demo server** option. Then, accept the default path (the Downloads directory) or specify another fully qualified path.

    > **Note:** In a distributed system where Workstations are installed on user computers, the demo server is typically installed with the Workstations for new users; not with the Server.

16. Click **Next**.
    The Select Start Menu Folder step opens.

17. (Optional) Specify whether the DevTest Solutions processes can be started from the Start menu. (You can start the executable for each process manually from the bin directory under the installation directory.) The advantage of starting the executables, as opposed to the associated services, is that you can monitor messages that are displayed in the command-line interface (CLI).

    - To create a Start menu folder with shortcuts for all users, accept all defaults. Optionally, enter a new folder name.

    - To have no Start menu folder, clear the **Create a Start Menu folder** check box. To create a Start menu folder and restrict the shortcut display to your Start menu:

      - Accept the selection of **Create a Start Menu folder**.

- Accept the default name or enter another name.
- Clear the **Create shortcuts for all users** check box.

18. Click **Next**.
    The Desktop Icons step opens.

19. (Optional) If you do not want to create desktop icons for DevTest Enterprise Dashboard UI, DevTest Portal UI, and DevTest Workstation, clear the check box. Click **Next**.
    The Windows Services step opens.

20. (Optional) Select **Install Services** to create the following Windows services.

    - DevTest Broker Service (for Application Insight)
    - DevTest Coordinator Service
    - DevTest Enterprise Dashboard Service
    - DevTest Identity and Access Manager Service
    - DevTest Portal Service
    - DevTest Registry Service
    - DevTest Simulator Service
    - DevTest VSE Service

    This selection adds the services to Administrative Tools, Component Services, Services. The advantage to starting services as opposed to the associated executables is that you reduce the number of icons that are displayed on the system tray. If you want the Startup type to be defined as Automatic so that the services start when the host computer is restarted, select the **Start on bootup** check box. You can configure Automatic startup within Services, if you do not select this check box here.
    The services on Windows run under a specific user account, click **Optional Service Account** . To have the services run under the system account, select **System Account**. To have the services run under another account,enter the account name and password in the **Account Name** and **Account Password** fields.

21. Click **Next**.
    The Select File Associations step opens. All associations are selected by default.

22. Leave all associations selected or clear the file extensions that you do not want to associate with DevTest Solutions.

23. Click **Install** to start the installation.
    The Installing step opens. When the installation finishes, the Information step opens.

24. Read the information and click **Next**.

25. The Completing the DevTest Solutions Setup Wizard step opens.

26. Click **Finish**.

Install DevTest Server on UNIX

Before you start the installation procedure, download the following files from the Download Center:

- The installer for your platform
- (Optional) The demo server zip file

> **Note:** Typically, the Demo Server is installed with the Server components only in a standalone system. In a distributed system, the Demo Server is usually installed with DevTest Workstation for users who are new to DevTest Solutions.

The following procedure is based on the graphical version of the installer. To use the command-line version of the installer, add the **-c** option. For example:

```
./devtest_linux_x64.sh -c
```

If a graphical front end is not available on your computer, you must use the command-line version.

> **Note:** If you downloaded the generic UNIX installer, ensure that a Java virtual machine (JVM) is on the same computer. If the installer cannot find a JVM, the installer displays a message and exits. For more information, see the section on supplying your own JVM in System Requirements.

Follow these steps:

1. In a terminal window, navigate to the directory where the installer file is located.
2. Ensure that the installer file has the execute permission. For example:

   ```
   chmod 777 devtest_linux_x64.sh
   ```

   This command gives the read, write, and execute permissions to each class of user.
3. Run the installer file. For example:

   ```
   ./devtest_linux_x64.sh
   ```

   > On Ubuntu systems, if you install as root user, you can not display DevTest icons on the desktop.

   The Welcome to the DevTest Solutions Setup Wizard opens.
4. Click **Next**.
   The CA End User License Agreement step opens.
5. Read the license agreement, scrolling to the end, select the **I accept the terms of the License Agreement** option, and click **Next**.
   The Select Components step opens.

6. Ensure that the **Server** check box is selected; an embedded Workstation is installed with the Server. Click **Next**.
   The Select Destination Directory step opens.

7. Specify the directory where you want to install DevTest Solutions, for example, **/opt/CA/DevTest**. Do not use a path with a directory name that contains spaces.

   - For more information about selecting the appropriate destination and data directory, see Installation Options.

8. If you are upgrading from a shared install, enter the path and folder name for the data directory.

   - If you are not upgrading from a shared install, this option does not appear.

   - Enter the path and folder name for the data directory. The default data directory is the same as the installation folder.

   - The specified data directory must exist and the user must have write permissions to it. If it does not exist, the installer will create it and will test to be sure it is writable.

   - If the data directory is different than the installation directory, then DevTest will create a lisa.user.properties file in both the installation directory and in the user's home folder. The **lisa.data.dir** property in this property file tells the product where the data directory is located.

   - The data directory will include hotdeploy for server components. Other server component files will be in the installation directory.

9. Click **Next**.
   The Identity and Access Manager step opens. Identity and Access Manager is required for a DevTest Solutions installation.

10. Select one of the following options and click **Next**.

    - **Install or upgrade Identity and Access Manager**
      If this is the first time you are installing IAM, or if you want to upgrade your existing IAM, select this option.

    - **Connect to an existing Identity and Access Manager**
      If you want to connect to an existing IAM in your environment, select this option. Enter the URL for the existing IAM. This URL is the host and port where Identity and Access Manager was installed. Clicking **Validate** verifies that you have entered a valid URL, but does not attempt to connect to IAM.

      > **Note:** A single Identity and Access Manager instance for your entire enterprise allows component access to be managed in a single location. However, if you have multiple registries and need to manage access to each registry separately, you need to install a separate IAM for each registry. This is especially true if users from one registry cannot have rights to another. For more information, see DevTest Process Relationships.

    The Identity and Access Manager Configure Database step opens.

11. Select the type of database you want to use for your user management.

- If you select the default value, **Embedded Database**, no additional configuration is required.
- If you select another option, enter the details for your database and click **Validate**.

> **Note:** Using the embedded database is not recommended as an enterprise solution. This database is not intended for use in a production environment.

12. Click **Next**.
    The Enterprise Dashboard step opens.
13. Specify one of the following options:
    - **New Enterprise Dashboard (Specify location of license file)**
      If this is the first Server you are installing, click **Browse**, navigate to the location of the license file, select **devtestlic.xml**, and click **Open**. The installer copies the **devtestlic.xml** file to the specified installation directory (LISA_HOME) on the local host. The new Enterprise Dashboard process is installed in the LISA_HOME/bin directory.

      > Note: To upgrade from DevTest 10.3 or earlier, you must request a new license file. For more information, see License Administration.

    - **Existing Enterprise Dashboard Service**
      If the Registry in this DevTest Server connects to the Enterprise Dashboard installed with the first DevTest Server you installed, enter the location of the existing Enterprise Dashboard (ED).
14. Click **Next**.
    The CA Customer Experience Improvement step opens.
15. Select whether you want to send diagnostic and usage data to CA Technologies to help us improve our products and services.
    - This step only appears if you are installing a new Enterprise Dashboard. If you are connecting to an existing Enterprise Dashboard, this step does not display.
    - For more information about this option and the data that is collected, see Telemetry and Telemetry Data.
16. Click **Next**.
    The Demo Server step opens.
17. If you want the installer to unzip the demo server into the same directory where you are installing the DevTest Server, select the **Install demo server** check box and browse to the DevTestDemoServer.zip file.
18. Click **Next**.
    The Select Directory for Symlinks step opens.

19. Click **Browse** and navigate to the directory where DevTest creates symbolic links to the executable files. The default is **/usr/local/bin**. You must have the required permissions to write to the directory. If you do not want symbolic links to be created, clear the **Create symlinks** check box.

20. Click **Next**.
    The Desktop Icons step opens.

21. (Optional) If you do not want to create desktop icons for DevTest Workstation, DevTest Portal, and Enterprise Dashboard, clear the check box.

22. Click **Install** to start the installation.
    The Installing step opens. When the installation finishes, the Information step opens.

23. Read the information and click **Next**.
    The Completing the DevTest Solutions Setup Wizard step opens.

24. Click **Finish**.

Install DevTest Server on OS X

Before you start the installation procedure, download the following files from the Download Center:

- The installer for your platform
- (Optional) The demo server zip file

> **Note:** Typically, the Demo Server is installed with the Server components only in a standalone system. In a distributed system, the Demo Server is usually installed with DevTest Workstation for users who are new to DevTest Solutions. The Demo Server is used for tutorials.

Follow these steps:

1. Run the installer file, for example, devtest_osx_x64.dmg.
   The Welcome to the DevTest Solutions Setup Wizard step opens.

2. Click **Next**.
   The CA End User License Agreement step opens.

3. Read the license agreement, scrolling to the end, select the **I accept the terms of the License Agreement** option, and click **Next**.
   The Select Components step opens.

4. Ensure that the **Server** check box is selected; an embedded Workstation is installed with the Server. Click **Next**.
   The Select Destination Directory step opens.

5. Specify the folder where you want to install one or more components of the DevTest Solutions. If you specify a folder that does not exist, the Setup wizard creates it.

- For more information about selecting the appropriate destination and data directory, see Installation Options.

6. If you are upgrading from a shared install, enter the path and folder name for the data directory.

   - If you are not upgrading from a shared install, this option does not appear.

   - Enter the path and folder name for the data directory. The default data directory is the same as the installation folder.

   - The specified data directory must exist and the user must have write permissions to it. If it does not exist, the installer will create it and will test to be sure it is writable.

   - If the data directory is different than the installation directory, then DevTest will create a lisa.user.properties file in both the installation directory and in the user's home folder. The **lisa.data.dir** property in this property file tells the product where the data directory is located.

   - The data directory will include hotdeploy for server components. Other server component files will be in the installation directory.

7. Click **Next**.
   The Identity and Access Manager step opens. Identity and Access Manager is required for a DevTest Solutions installation.

8. Select one of the following options and click **Next**.

   - **Install or upgrade Identity and Access Manager**
     If this is the first time you are installing IAM, or if you want to upgrade your existing IAM, select this option.

   - **Connect to an existing Identity and Access Manager**
     If you want to connect to an existing IAM in your environment, select this option. Enter the URL for the existing IAM. This URL is the host and port where Identity and Access Manager was installed. Clicking **Validate** verifies that you have entered a valid URL, but does not attempt to connect to IAM.

     Note: A single Identity and Access Manager instance for your entire enterprise allows component access to be managed in a single location. However, if you have multiple registries and need to manage access to each registry separately, you need to install a separate IAM for each registry. This is especially true if users from one registry cannot have rights to another. For more information, see DevTest Process Relationships.

   The Identity and Access Manager Configure Database step opens.

9. Select the type of database you want to use for your user management.

   - If you select the default value, **Embedded Database**, no additional configuration is required.

   - If you select another option, enter the details for your database and click **Validate**.

> **Note:** Using the embedded database is not recommended as an enterprise solution. This database is not intended for use in a production environment.

10. Click **Next**.
    The Enterprise Dashboard step opens.

11. Specify one of the following options:

    - **New Enterprise Dashboard (Specify location of license file)**
      If this is the first Server you are installing, click **Browse**, navigate to the location of the license file, select **devtestlic.xml**, and click **Open**. The installer copies the **devtestlic.xml** file to the specified installation directory (LISA_HOME) on the local host. The new Enterprise Dashboard process is installed in the LISA_HOME/bin directory. This option specifies that the registry installed with this Server is to connect to the new Enterprise Dashboard.

      > **Note:** To upgrade from DevTest 10.3 or earlier, you must request a new license file. For more information, see License Administration.

    - **Existing Enterprise Dashboard Service**
      If this is not the first Server that is installed in this network, enter the location of the existing Enterprise Dashboard. This option specifies that the registry installed with this Server is to connect to the existing Enterprise Dashboard.

      ```
      localhost:1506
      ```

12. Click **Next**.
    The CA Customer Experience Improvement step opens.

13. Select whether you want to send diagnostic and usage data to CA Technologies to help us improve our products and services.

    - This step only appears if you are installing a new Enterprise Dashboard. If you are connecting to an existing Enterprise Dashboard, this step does not display.

    - For more information about this option and the data that is collected, see Telemetry and Telemetry Data.

14. Click **Next**.
    The Demo Server step opens.

15. If you want the installer to unzip the demo server into the LISA_HOME directory, select the **Install demo server** option. Then, accept the default path (the Downloads directory) or specify another fully qualified path.

    > **Note:** In a distributed system where Workstations are installed on user computers, the demo server is typically installed with the Workstations for new users; not with the Server.

16. Click **Next**.
    The Desktop Icons step opens.

17. (Optional) If you do not want to create desktop icons for DevTest Enterprise Dashboard, DevTest Portal UI, and DevTest Workstation, clear the check box. Click **Next**.

18. Click **Install** to start the installation.
    The Installing step opens. When the installation finishes, the Information step opens.

19. Read the information and click **Next**.
    The Completing the DevTest Solutions Setup Wizard step opens.

20. Click **Finish**.

More Information:

- Activate the Registries
- Verify Registry Activation
- Post-Installation
- Uninstall DevTest Server

Activate the Registries

After you install all instances of DevTest Server, you start Identity and Access Manager and the Enterprise Dashboard process on the first DevTest Server you installed. Then you start the Registry process on that DevTest Server and every other DevTest Server. The activation of the registries does not occur immediately. For example, if you restart the registries between 10:15 and 10:30, the activation occurs at 11:00.

You should run the Enterprise Dashboard once, before the first time you run the Registry. The Registry needs something to verify the license before running the first time. The verification is done through Enterprise Dashboard. After that, the Registry has the information that the license has been verified. Once the Registry takes note of that, you never need Enterprise Dashboard running to run the Registry.

Verify that a registry from each new and existing DevTest Server displays on the Enterprise Dashboard.

Follow these steps:

1. Start Identity and Access Manager.

    a. Log on to the host where Identity and Access Manager is installed.

    b. Start Identity and Access Manager.
    Windows users start the registry from the Start menu (under DevTest Solutions) or from the **IdentityAccessManager** executable in the IdentityAccessManager/bin directory in the installation directory.

2. Start the Enterprise Dashboard Server.

    a. Log on to the host where the Enterprise Dashboard is installed.

    b. Start the Enterprise Dashboard Server.
    Windows users can select Enterprise Dashboard Server from the Enterprise Dashboard option in the Start menu. Alternatively, navigate to the LISA_HOME\bin directory and launch the **EnterpriseDashboard** executable.

3. Start each registry.

    a. Log on to the host where a DevTest Server is installed.

    b. Start the registry.
    Windows users start the registry from the Start menu (under DevTest Solutions) or from the **Registry** executable in the bin directory under the DevTest Server installation directory.

    c. Repeat these steps for each registry.

    > **Note:** After this property is set, the registry does not push data to the Enterprise Dashboard until the top of the next hour. For example, if you set this property at 1:35pm, the registry data does not appear until 2:00pm.

4. (Optional) If you use a remote registry for DevTest Portal, set the **lisa.registryName** property in the **local.properties** file to the URL of the registry.For example: You have two machines. Your registry is running on a machine with IP address 10.155.12.796 but you want to start DevTest Portal on a different machine. On the second machine, add **lisa.registryName=tcp://10.155.12.796:2010/Registry** to **local.properties**. To connect using SSL between DevTest Portal and registry, use **lisa.registryName=ssl://10.155.12.796:2010/Registry** instead. If you use a custom port, it would go in place of the 2010 in that URL. Machine names and domain names can be used instead of IP addresses assuming they can be resolved by the network.You must also set the **registry.host** property in **phoenix.properties**. If you use a custom port, **registry.host.port** is also required.

    Set **lisaAutoConnect=tcp://<HOST>:2010/Registry** in the **local.properties** file.

Verify Registry Activation

After starting each registry, wait until the beginning of the next hour before confirming registry activation.

Follow these steps:

1. Open the Enterprise Dashboard UI in one of the following ways:

    - Browse to the Enterprise Dashboard. Specify the IP address or host name if installed remotely or specify **localhost** if installed locally.

      `http://hostname:1506`

- From the computer where the Enterprise Dashboard is installed, Windows users select the Start menu option, **Enterprise Dashboard**, **Enterprise Dashboard UI**.

The Enterprise Dashboard opens.

2. Examine the registry configurations on the Enterprise Dashboard.
   A list of your registries appears.

3. Verify that a registry from each new and existing DevTest Server displays on the Enterprise Dashboard.

Post-Installation

This section describes the post-installation tasks that are required to configure DevTest Server.

- Using an HTTP/S Proxy Server - DevTest Server
- Running Components on Different Systems
- Calculate Simulator Instances
- Load and Performance Server Sizing
- Sample Performance Benchmarks for Basic Service Virtualization
- Sample Performance Benchmarks for IBM MQ Service Virtualization
- Using DevTest Workstation with Your Java Environment
- Change the Default Project Home
- Project Directory Structure

Using an HTTP/S Proxy Server - DevTest Server

If you are using a plain HTTP proxy server or an SSL-secured HTTP proxy server, define that proxy server and any hosts to exclude in the **local.properties** file in the LISA_HOME directory.

Follow these steps:

1. Log on to the host where the DevTest Server is installed.

2. Navigate to LISA_HOME.

3. If local.properties does not exist, copy **_local.properties** and save the copy as local.properties (without the underscore).

4. Open local.properties for edit and locate the section header for either HTTP Proxy Server or HTTPS Proxy Server, depending on whether your proxy server uses plain HTTP or is SSL-secured.

5. Identify your proxy server by FQDN or IP address and port:

   - For an HTTP server, use the lisa.http.webProxy.host and lisa.http.webProxy.port properties

   - For an HTTPS server, use the lisa.http.webProxy.ssl.host and lisa.http.webProxy.ssl.port properties

6. Identify any hosts to exclude from going through your proxy server:

- For an HTTP server, use the lisa.http.webProxy.nonProxyHosts property
- For an HTTPS server, use the lisa.http.webProxy.ssl.nonProxyHosts property

7. Verify that you removed the comment symbols in front of the properties.
8. Save the file and exit.

Example:

The first two lines of the following example specify that the URL for the HTTP proxy server is **http://192.168.24.242:49185**.

The third line specifies that the hosts that will not go through this proxy include the loopback address for the localhost (127.0.0.1) and IP addresses in the range 192.168.32.0 through 192.168.32.255. Notice that the pipe symbol (|) is used as the delimiter between the IP addresses to exclude. Notice also that the wildcard (*) represents any valid value, where valid values for an IP address node range from zero to 255. The wildcard character can also be used with FQDNs and host names, if hosts to exclude share a standard naming convention.

```
lisa.http.webProxy.host=192.168.24.242
lisa.http.webProxy.port=49185
lisa.http.webProxy.nonProxyHosts=127.0.0.1|192.168.32.*
```

HTTP/S Proxy Server settings in local.properties

```
## ==============================================
## HTTP Proxy Server
## ==============================================
#lisa.http.webProxy.host=<machine name or ip>
##list of excluded machine names or ip addresses delimited by pipes, * wildcard ac
cepted <machine name or ip>[|<machine name or ip>]*
lisa.http.webProxy.nonProxyHosts=127.0.0.1
#lisa.http.webProxy.port=

## ==============================================
## HTTPS Proxy Server
## ==============================================
#lisa.http.webProxy.ssl.host=<machine name or ip>
##list of excluded machine names or ip addresses delimited by pipes, * wildcard ac
cepted <machine name or ip>[|<machine name or ip>]*
lisa.http.webProxy.ssl.nonProxyHosts=127.0.0.1
#lisa.http.webProxy.ssl.port=

## === Leave blank to use integrated NTLM authentication
#lisa.http.webProxy.host.domain= used for NTLM authentication
#lisa.http.webProxy.host.account=
#lisa.http.webProxy.host.credential=

## === Exclude simple host names from proxy use - default value is true
#lisa.http.webProxy.nonProxyHosts.excludeSimple=false

## === Preemptively send authorization information rather than waiting for a chall
enge
## ===== valid values are basic or ntlm
#lisa.http.webProxy.preemptiveAuthenticationType=ntlm
```

Running Components on Different Systems

If the server components are on different systems, be sure to use the following properties correctly:

- The registry uses the **lisa.registryName** property to name itself to something other than the default.
- The nonregistry server components use the **lisa.registry.url** property as the locator.

In the **local.properties** file for the nonregistry server component, you specify the registry with the **lisa.registry.url** property.

```
lisa.registry.url=tcp://registry-hostname-or-ip:port/registry-name
```

For example:

```
lisa.registry.url=tcp://myserver.example.com:2010/Registry
```

Do not use the **lisa.registryName** property for this purpose.

Another option for specifying the registry is to pass **-m** as an argument when starting the nonregistry server component.

```
./CoordinatorServer -m tcp://registry-hostname-or-ip:port/registry-name
```

For example:

```
./CoordinatorServer -m tcp://myserver.example.com:2010/Registry
```

Calculate Simulator Instances

To calculate the number of instances for a specific simulator, do the following analysis:

1. Start DevTest Workstation, select the registry, and log in. Note the memory usage from **Help**, **DevTest Runtime Info**.
2. Run the test suite locally and note the memory usage from **Help**, **DevTest Runtime Info**.
3. Take the difference between the memory usage in step 2 and step 1.
4. Multiply your available RAM by 60 percent.
5. Divide the available RAM in step 4 by the memory usage in step 3.

The result of step 5 is a good starting estimate of the number of virtual users (instances) that you need configured in your simulator server.

If the coordinator server and the registry both run on the same server as the simulator server, then multiply available RAM by 40 percent. Use 40 percent instead of 60 percent because the coordinator server collects all reports and metrics and therefore consumes RAM.

This technique provides a starting point. To get to the correct number of instances for each simulator, use several iterations and other intuitive methods.

> **Note:** You can set the number of concurrent instances for a simulator with a command-line option. Open a command prompt, navigate to the LISA_HOME\bin directory, and type **simulator --help** for details.

Load and Performance Server Sizing

This section discusses the considerations in sizing a server to perform load and performance testing.

It is not easy to calculate how many simulation servers are needed for a specific load test. How many servers are required depends on many factors, including:

- Server host configuration (number of CPUs, amount of RAM)
- Test case footprint (number of test steps, type of test steps)
- Other test requirements (number of reports, size of data sets)

We recommend making several test runs of your performance test. These test runs allow you to collect data that can be helpful in determining the configuration of your DevTest Server environment. Collecting metrics and monitoring memory and CPU usage is invaluable for estimating the number of virtual users you can use on a given simulator server.

The registry is lightweight and requires few computing resources. The registry can be run from virtually any computer in your network.

The coordinator server requires resources. Although the coordinator server does not require its own computer, installing it on a separate computer is a common practice. Follow this practice if you are collecting many metrics, requesting many reports, or both.

Simulator servers are used to simulate thousands of virtual users. We recommend running one simulator server per physical server. Technically, a single simulator server can be started with as many instances as you want. However, server memory size and speed typically limit the number of instances for each simulator. A good upper limit is around 250 virtual users.

Vertical or horizontal scaling can be used for sizing the server. In vertical scaling, you increase CPU speed and available memory, which are typically limited. In horizontal scaling, you add more servers. To increase the number of virtual users, horizontal scaling is recommended.

The number of instances per simulator depends on many factors. A simple rule is not available for calculating the maximum number of instances.

Network latency impacts load and performance. We recommend that the database is housed on a server within the same data center as the major DevTest components.

Sample Performance Benchmarks for Basic Service Virtualization

The DevTest development team performed testing to show benchmarks for performance of service virtualization. You can use these benchmarks to help tune the performance of your system.

The following data represents a simple scenario to identify the behavior of VSE under heavy load. The variation of the independent variable has been limited to the payload size. Performance data has been collected for the following payload sizes: 1 KB, 10 KB, and 100 KB.

Environment Details

The following graphic pictures the topology of the Performance Lab.

System Specifications for All Machines Except Load Generator

OS	Windows 2012 R2 64-bit VMs
vCPU	4 individual cores **assigned** and **reserved**
Memory	16 GB **reserved**
Hard Drive	100 GB
Processor	Intel® Xeon® CPU ES-2680 v3 @ 2.5GHz, 1 Core

DevTest version: 10.1.0.108

JVM Arguments

- -Xmx2048m
- -XX:+UseConcMarkSweepGC

DevTest Properties

- lisa.vse.performance.enabled=true
- lisa.pathfinder.on=false
- vse.socket.reuse=true
- lisa.net.timeout.ms =60000
- lisa.net . xstream=false
- lisa.dcm.maxExpandPerLoadChange=50

- lisa.jpm.MaxInstances=5000
- lisa.jpm.unlimited=false
- lisa.overloadThreshold=30000
- lisa.eventPool.maxQueueSize=65535
- lisa.eclipselink.cache.timeout.ms =3600000
- lisa.CycleExecHistory.buffer.size=2

Virtual Service configuration

Execution Mode	Most Efficient
Think Scale	0
Capacity	20
Transaction type	Stateless
Match Style	Signature
Log Level	**VSE**: OFF **Other components**: WARN

A single virtual service was deployed to the VSE. Each HTTP call produces a single transaction count in the VSE. There is no search and replacement activity involved in this transaction. The service returns an exact match response. Further processing is required neither at the requests nor the response level.

Test Results

JSON Response Payload Size	Average TPS	Max TPS	Total Transactions per Run	Memory Utilization (VSE Server)	CPU Utilization (VSE Server)	Data Protocol
1 KB	10,822	11,560	39,202,897	13%	93%	REST Data Protocol
10 KB	8,405	8,808	29,888,786	18%	90%	REST Data Protocol
100 KB	2,974	3176	10,635,062	13%	90%	REST Data Protocol

Duration of each test run: 60 minutes during off-peak hours

The data that are provided in the previous table represents the number of transactions that are handled by the VSE during each 60-minute test run.

System Resources Utilization

The following charts represent data that were captured from the VSE server machine during the execution of each test run. The machines hosting the database utilized a very modest amount of resources. The machine hosting the registry consumed limited CPU time (2 to 3%) and approximately 40% of the available RAM. Disk activity of the registry ranged from 50 to 100 KB /sec. This demonstrates that the machines supporting the VSE machine do not represent a bottleneck during the test runs.

Legend

- RED: CPU Utilization
- GREEN: Memory Available in GB (out of 16 GB)

Test Run With Payload of 1 KB

Available Memory: 13.5 GB out of total of 16 GB

Test Run With Payload of 10 KB

Available Memory: 13.4 GB out of total of 16 GB

Test Run With Payload of 100 KB

Sample Performance Benchmarks for IBM MQ Service Virtualization

The DevTest development team performed testing to show benchmarks for performance of service virtualization of IBM MQ. You can use these benchmarks to help tune the performance of your system.

Environment Details

Payload

Size	XML Elements
100B	4
2 kB	75
10 kB	116
100 kB	2802

Machine Settings

Settings	
CPU	Intel Xeon E5-2680 2.5 GHz
Virtual cores	16
Memory	16 GB
OS	Windows Server 2012 R2
MQ Server	8.0.0.5
CA Service Virtualization Version	10.1

- **DevTest Settings**
  - lisa.pathfinder.on=false
  - lisa.vse.performance.enabled=true
  - vse.socket.reuse=true
  - lisa.net.timeout.ms=60000
  - lisa.net.xstream=false
  - activemq.msgCompression=true
  - lisa.dcm.maxExpandPerLoadChange=50
  - lisa.jpm.MaxInstances=5000
  - lisa.jpm.unlimited=false
  - lisa.overloadThreshold=30000
  - lisa.eventPool.maxQueueSize=65535
  - lisa.eclipselink.cache.timeout.ms=3600000
  - lisa.CycleExecHistory.buffer.size=2
  - lisa.vse.metrics.sample.interval=1m
- **JVM Settings (vmoptions)**

  - -Xmx2048m
  - -XX:+UseConcMarkSweepGC
  - -Dorg.apache.xml.dtm.DTMManager=org.apache.xml.dtm.ref.DTMManagerDefault This is optimization to improve performance for the Generic XML Payload Parser.
- Scope of MQ request/response queues is set to *model.*
- Concurrent capacity increased to 30.
- Think time set to 0.

Test Results

Payload	Number of transactions in VSI	Data Protocol Filter	SV [T /s]	CPU % for the SV machine	Memory % for the SV machine
100 B	300	Generic XML Payload Parser	6500 T /s	75%	70%
3 kB	300	Generic XML Payload Parser	5500 T /s	80%	70%
10 kB	300	Generic XML Payload Parser	5000 T /s	80%	70%
100 kB	300	Generic XML Payload Parser	1200 T /s	80%	70%
100 B	3000	Generic XML Payload Parser	5500 T /s	80%	70%
3 kB	3000	Generic XML Payload Parser	5300 T /s	80%	70%
10 kB	3000	Generic XML Payload Parser	5000 T /s	80%	70%
100 kB	3000	Generic XML Payload Parser	1100 T /s	80%	70%
100B	3000	XML Data Protocol	5600 T /s	99%	65%

The Generic XML Payload Parser always selects one XML element as a key.

Test Results Summary

The deprecated IBM MQSeries transport protocol has poor performance and should not be used in any performance tests. Use the IBM MQ Native (with queues defined as assets) instead.

Performance depends significantly on the number of arguments that are used for requests comparison. So the Generic XML Payload is better for larger payloads if a limited set of arguments is used for response selection. XML Data Protocol is not so efficient if the payload is large, as it automatically includes every XML element as an argument for matching. This results in a much larger set of match criteria, which takes longer to evaluate against incoming requests.

This applies for every protocol where these DPHs are used.

Using DevTest Workstation with Your Java Environment

You can replace the default JRE used by the DevTest installation with your own Java environment.

Warning! Replacing the default JRE means that you are migrating away from an officially certified JRE/SDK. We discourage you from replacing the default JRE, unless it is to support specific JRE/JDK specific functionality. Using a different JDK/JRE from the one that is included in your DevTest environment may not be supported, depending on the impact of the specific JRE/JDK installed.

If you do this, it is important to understand how DevTest Workstation selects the JRE to use.

The following priority is used to select what Java VM to use when starting DevTest Workstation:

1. The DevTest Workstation-installed JRE in the LISA_HOME\jre directory
2. LISA_JAVA_HOME environment variable
3. JAVA_HOME environment variable
4. JDK_HOME environment variable

Follow these steps:

1. Rename the LISA_HOME\jre directory (for example, rename jre to jre_default).
2. Point the LISA_JAVA_HOME environment variable to your Java installation directory.

More Information:

- Supplying Your Own JVM

Change the Default Project Home

You can change the default project home directory. By default, projects are saved in the **LISA_HOME\Projects** directory.

This procedure describes how to change the default location for DevTest Portal. You cannot change the default location for DevTest Workstation.

Follow these steps:

1. Navigate to the **LISA_HOME** directory.
2. Create a text file with the name **res-hub-config.properties**.
3. Add the **resHub.projects.dir** property to the file. Be sure to use forward slashes in the directory path, even on Windows platforms. For example:

```
resHub.projects.dir=C:/MyNewProjectHome
```

4. Save and close the file.

5. If the portal server component is running, restart it.

Project Directory Structure

As a best practice, make test assets (for example, projects) available to the server components using them.

To manage access to the test assets, the requirements are as follows:

- Use naming standards. Multiple teams can use the same server environment. To differentiate ownership and purpose and to maintain order, use naming standards.
- The project names must be unique. On the server environment, if two deployed projects have the same name, unexpected things can happen.

Uninstall DevTest Server

You can use the uninstall application to remove DevTest Server. The uninstall application is located in the **LISA_HOME** directory.

The following videos illustrate this procedure.

Follow these steps:

1. Stop DevTest Solutions.

    a. Verify that all users have logged off DevTest workstations, the DevTest Portal, and the Enterprise Dashboard.

    b. Verify that no DevTest command-line utilities are running.

    c. Close the executables or stop the services in this order: Simulator, Coordinator, VSE, Broker, Portal, Registry, and Enterprise Dashboard.

2. (Optional) Remove the directory where the main log files are located.

3. Ensure that the **lisa.user.properties** file is deleted. The default location is the **USER_HOME** directory for the user who installed DevTest.

    Important! If you do not delete this file, future installations where the data directory is the same as the install directory will not install correctly.

4. (Optional) Remove the DevTest schema objects from the external database. If necessary, you can use the DDL statements in the **LISA_HOME\database\dropSchema** directory to perform this action.

5. Run the uninstall executable that is located in your installation directory.
   The DevTest Solutions Uninstall step opens.

6. Click **Next**.

7. To delete folders for the database, hotDeploy, temp folder, data directory, and related user preferences, select the **Delete all files** check box.

   If you select **Delete all files**, the entire data directory and installation directory will be deleted along with all files that are located inside the folders. Before you uninstall, we recommend that you copy these directories.

   If a system administrator chooses to "Remove All Files," and the data directory is specified using a user home variable (%userprofile% or ~), the uninstaller will be unable to remove the data files for a user other than the system administrator.

8. Click **Next**.
   If you did not choose to delete all files, a step opens with the list of files that could not be deleted.

9. Click **Finish**.

Installing DevTest Workstation

You can install DevTest Server with an embedded DevTest Workstation or you can install DevTest Workstation as a standalone application.

This section describes how to install and configure DevTest Workstation as a standalone application.

After you install and configure DevTest Workstation, you can log in by following the steps in Open DevTest Workstation. If your computer has an installation of DevTest Workstation without DevTest Server, specify a registry that is running on a remote computer. DevTest Workstation installations do not include a local registry.

- Install DevTest Workstation on Windows
- Install DevTest Workstation on UNIX
- Install DevTest Workstation on OS X

Install DevTest Workstation on Windows

Before you start the installation procedure, download the following files from the Download Center:

- The installer for your platform

- (Optional) The demo server zip file

The following video illustrates this procedure.

Follow these steps:

1. Run the installer file; for example, devtest_win_x64.exe.
   The Welcome to the DevTest Solutions Setup Wizard opens.

2. Click **Next**.
   The CA End User License Agreement step opens.

3. Read the license agreement, select the **I accept the terms of the License Agreement** option, and click **Next**.
   The Select Components step opens.

4. Clear the **Server** check box, ensure that the **Workstation** check box is selected, and click **Next**.
   The Select Destination Directory step opens.

5. Enter the path and folder name for the installation directory (LISA_HOME). Consider a name that includes the release identifier, if you want to keep the current release separate from older releases. For example, enter: C:\DevTestServer_10.0. Or, accept the default (C:\Program Files\CA\DevTest). If you browse to a path and you enter the name of a new folder, the installation wizard creates the folder.

6. Enter the path and folder name for the data directory.

   > The Workstation components in the data directory include _local.properties, _logging.properties, _de-identify.xml, hotdeploy, library for customizations, projects with assets, and examples folders. On initial start-up of DevTest Workstation, Projects and Examples will be copied or refreshed from the install directory to the data directory.

   When installing, the default data directory is a subfolder named *DevTest* within the user's home folder. Example %userprofile%/DevTest
   Using the %userprofile% (or '~') user home variables might be useful for system administrators who are installing this product on behalf of other users. These variables resolve to the user's home folder of whatever user is using the product.The specified data directory must exist and the user must have write permissions to it. If it does not exist, the installer will create it and will test to be sure it is writable.
   If the data directory is different than the installation directory, then DevTest creates a lisa.user.properties file in both the installation directory and in the user's home folder. The **lisa.data.dir** property in this property file tells the product where the data directory is located.

7. Click **Next**.
   The Demo Server step opens.

8. If you want the installer to unzip the demo server into the LISA_HOME directory, select the **Install demo server** option and specify the fully qualified path of the demo server zip file.

9. Click **Next**.
   The Select Start Menu Folder step opens.

10. Specify whether to add DevTest Workstation to your Start menu and if so, whether to add DevTest Workstation to the Start menu of other users.

    - To create a Start menu folder with shortcuts for all users, accept all defaults. Optionally, enter a new folder name.

    - To have no Start menu folder, clear the **Create a Start Menu folder** check box.

    - To create a Start menu folder and restrict the shortcut display to your Start menu:

      - Accept the selection of **Create a Start Menu folder**.

      - Accept the default name or enter another name.

      - Clear the **Create shortcuts for all users** check box.

11. Click **Next**.
    The Desktop Icons step opens.

12. (Optional) If you do not want to create desktop icons for DevTest Enterprise Dashboard UI, DevTest Portal UI, and DevTest Workstation, clear the check box. Click **Next**.
    The Select File Associations step opens. All associations are selected by default.

13. Leave all associations selected or clear the file extensions that you do not want to associate with DevTest.

14. Click **Install** to start the installation.
    The Installing step opens. When the installation finishes, the Information step opens.

15. Read the information and click **Next**.
    The Completing the DevTest Solutions Setup Wizard step opens.

16. Click **Finish**.

Install DevTest Workstation on UNIX

Before you start the installation procedure, download the following files from the Download Center:

- The installer for your platform

- (Optional) The demo server zip file

The following procedure is based on the graphical version of the installer. To use the command-line version of the installer, add the **-c** option. For example:

```
./devtest_linux_x64.sh -c
```

If a graphical front end is not available on your computer, you must use the command-line version.

Note: If you downloaded the generic UNIX installer, ensure that a Java virtual machine (JVM) is on the same computer. If the installer cannot find a JVM, the installer displays a message and exits. For more information, see the section on supplying your own JVM in System Requirements.

Follow these steps:

1. In a terminal window, navigate to the directory where the installer file is located.

2. Ensure that the installer file has the execute permission. For example:

   ```
   chmod 777 devtest_linux_x64.sh
   ```

   This command gives the read, write, and execute permissions to each class of user.

3. Run the installer file. Double-click the icon or enter a command similar to the following command from a terminal window:

   ```
   ./devtest_linux_x64.sh
   ```

   On Ubuntu systems, if you install as root user, you can not display DevTest icons on the desktop.

   The Welcome to the DevTest Solutions Setup Wizard opens.

4. Click **Next**.
   The CA End User License Agreement step opens.

5. Read the license agreement, select the **I accept the terms of the License Agreement** check box, and click **Next**.
   The Select Components step opens.

6. Clear the **Server** check box, ensure that the **Workstation** check box is selected, and click **Next**.
   The Select Destination Directory step opens.

7. Enter the path and folder name for the installation directory (LISA_HOME). Do not use a directory that contains spaces. Consider a name that includes the release identifier, if you want to keep the current release separate from older releases. Or, accept the default (/opt/CA/DevTest). If you browse to a path and you enter the name of a new folder, the installation wizard creates the folder.

8. Enter the path and folder name for the data directory.

   The Workstation components in the data directory include _local.properties, _logging.properties, _de-identify.xml, hotdeploy, library for customizations, projects with assets, and examples folders. On initial start-up of DevTest Workstation, Projects and Examples will be copied or refreshed from the install directory to the data directory.

When installing, the default data directory is a subfolder named *DevTest* within the user's home folder. Example: ~/DevTest
Using the %userprofile% (or '~') user home variables might be useful for system administrators who are installing this product on behalf of other users. These variables resolve to the user's home folder of whatever user is using the product.The specified data directory must exist and the user must have write permissions to it. If it does not exist, the installer will create it and will test to be sure it is writable.
If the data directory is different than the installation directory, then DevTest creates a lisa.user.properties file in both the installation directory and in the user's home folder. The **lisa.data.dir** property in this property file tells the product where the data directory is located.

9. Click **Next**.

10. The Demo Server step opens.

11. If you want the installer to unzip the demo server into the **LISA_HOME** directory, select the **Install demo server** check box and specify the fully qualified path of the demo server zip file.

12. Click **Next**.

13. The Select Directory for Symlinks step opens.

14. Click **Browse** and navigate to the directory where DevTest creates symbolic links to the executable files. The default is /usr/local/bin. You must have the required permissions to write to the directory. If you do not want symbolic links to be created, clear the **Create symlinks** check box.

15. Click **Next**.

    The Desktop Icons step opens.

16. (Optional) If you do not want to create a desktop icon for DevTest Workstation, clear the check box.

17. Click **Install**.

    The Installing step opens. When the installation finishes, the Information step opens.

18. Read the information and click **Next**.

19. The Completing the DevTest Solutions Setup Wizard step opens.

20. Click **Finish**.

Install DevTest Workstation on OS X

Before you start the installation procedure, download the following files from the Download Center:

- The installer for your platform
- (Optional) The demo server zip file

Follow these steps:

1. Run the installer file, for example, devtest_osx_x64.dmg.
   The Welcome to the DevTest Solutions Setup Wizard opens.

2. Click **Next**.
   The CA End User License Agreement step opens.

3. Read the license agreement, select the **I accept the terms of the License Agreement** option, and click **Next**.
   The Select Components step opens.

4. Clear the **Server** check box, ensure that the **Workstation** check box is selected, and click **Next**.
   The Select Destination Directory step opens.

5. Enter the path and folder name for the installation directory (LISA_HOME). Do not use a directory that contains spaces. Consider a name that includes the release identifier, if you want to keep the current release separate from older releases. Or, accept the default (/opt/CA/DevTest). If you browse to a path and you enter the name of a new folder, the installation wizard creates the folder.

6. Enter the path and folder name for the data directory.

   > The Workstation components in the data directory include: _local.properties, _logging.properties, _de-identify.xml, hotdeploy, library for customizations, projects with assets, and examples folders. On initial start-up of DevTest Workstation, Projects and Examples will be copied or refreshed from the install directory to the data directory.

   When installing, the default data directory is a subfolder named *DevTest* within the user's home folder. Example: ~/DevTest
   Using the %userprofile% (or '~') user home variables might be useful for system administrators who are installing this product on behalf of other users. These variables resolve to the user's home folder of whatever user is using the product.The specified data directory must exist and the user must have write permissions to it. If it does not exist, the installer will create it and will test to be sure it is writable
   If the data directory is different than the installation directory, DevTest creates a lisa.user.properties file in both the installation directory and in the user's home folder. The **lisa.data.dir** property in this property file tells the product where the data directory is located.

7. Click **Next**.

   The Demo Server step opens.

8. If you want the installer to unzip the demo server, select the **Install demo server** option and specify the fully qualified path of the demo server zip file.

9. Click **Next**.

   The Select Additional Tasks step opens.

10. (Optional) If you do not want to create a desktop icon for DevTest Workstation, clear the check box.

11. Click **Next**.

    The Select File Associations step opens.

12. Select all the extensions to work with the Examples Project tutorials provided in *Using CA Application Test*. The file extensions that you can associate with the DevTest Solutions include:

    - *.tst -- Select this extension to create test cases with CA Application Test.
    - *.vsm and *.vsi -- Select this extension to create virtual services with CA Service Virtualization
    - *.ste -- Select this extension to run a suite with Test Runner in CA Application Test.
    - *.stg -- Select this extension to run a test case as staging document in CA Application Test

13. Click **Install** to start the installation.

14. The Installing step opens. When the installation finishes, the Information step opens.

15. Read the information and click **Next**.

16. The Completing the DevTest Solutions Setup Wizard step opens.

17. Click **Finish**

More Information:

- Using an HTTP/S Proxy Server - DevTest Workstation
- Environment Settings
- Using DevTest Workstation with Your Java Environment.

Using an HTTP/S Proxy Server - DevTest Workstation

If you are using a plain HTTP proxy server or an SSL-secured HTTP proxy server, define that proxy server and any hosts to exclude in the **local.properties** file in LISA_HOME.

Follow these steps:

1. Log on to the host where DevTest Workstation is installed.
2. Navigate to LISA_HOME.
3. If local.properties does not exist, copy **_local.properties** and save the copy as local.properties (without the underscore).

4. Open local.properties for edit and locate the section header for either HTTP Proxy Server or HTTPS Proxy Server, depending on whether your proxy server uses plain HTTP or is SSL-secured.

5. Identify your proxy server by FQDN or IP address and port:

   - For an HTTP server, use the lisa.http.webProxy.host and lisa.http.webProxy.port properties

   - For an HTTPS server, use the lisa.http.webProxy.ssl.host and lisa.http.webProxy.ssl. port properties

6. Identify any hosts to exclude from going through your proxy server:

   - For an HTTP server, use the lisa.http.webProxy.nonProxyHosts property

   - For an HTTPS server, use the lisa.http.webProxy.ssl.nonProxyHosts property

7. Verify that you removed the comment symbols in front of the properties.

8. Save the file and exit.

Example:

The first two lines of the following example specify that the URL for the HTTP proxy server is **http://192.168.24.242:49185**.

The third line specifies that the hosts that will not go through this proxy include the loopback address for the localhost (127.0.0.1) and IP addresses in the range 192.168.32.0 through 192.168.32.255. Notice that the pipe symbol (|) is used as the delimiter between the IP addresses to exclude. Notice also that the wildcard (*) represents any valid value, where valid values for an IP address node range from zero to 255. The wildcard character can also be used with FQDNs and host names, if hosts to exclude share a standard naming convention.

```
lisa.http.webProxy.host=192.168.24.242
lisa.http.webProxy.port=49185
lisa.http.webProxy.nonProxyHosts=127.0.0.1|192.168.32.*
```

HTTP/S Proxy Server settings in local.properties

```
## ==============================================
## HTTP Proxy Server
## ==============================================
#lisa.http.webProxy.host=<machine name or ip>
##list of excluded machine names or ip addresses delimited by pipes, * wildcard ac
cepted <machine name or ip>[|<machine name or ip>]*
lisa.http.webProxy.nonProxyHosts=127.0.0.1
#lisa.http.webProxy.port=

## ==============================================
## HTTPS Proxy Server
## ==============================================
#lisa.http.webProxy.ssl.host=<machine name or ip>
##list of excluded machine names or ip addresses delimited by pipes, * wildcard ac
cepted <machine name or ip>[|<machine name or ip>]*
lisa.http.webProxy.ssl.nonProxyHosts=127.0.0.1
#lisa.http.webProxy.ssl.port=

## === Leave blank to use integrated NTLM authentication
#lisa.http.webProxy.host.domain= used for NTLM authentication
#lisa.http.webProxy.host.account=
#lisa.http.webProxy.host.credential=

## === Exclude simple host names from proxy use - default value is true
```

```
#lisa.http.webProxy.nonProxyHosts.excludeSimple=false

## === Preemptively send authorization information rather than waiting for a chall
enge
## ===== valid values are basic or ntlm
#lisa.http.webProxy.preemptiveAuthenticationType=ntlm
```

Environment Settings

DevTest documentation mentions a token that is named **%LISA_HOME%** (for Windows) or **$LISA_HOME** (for OS X or UNIX). This token indicates the location where the DevTest Solution was installed.

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

For example, if you installed DevTest Server into **C:\DevTest_***release_number*, that is the value of **%LISA_HOME%**. DevTest Workstation also has access to the value of this variable in a property named LISA_HOME.

To put more JARs, zips, or directories in the DevTest classpath, you have two options:

- Define the environment variable **LISA_POST_CLASSPATH** and set the resources that you want there.
- Put them in the **%LISA_HOME%/hotDeploy** directory or in the hotDeploy folder in your data directory.

> **Note:** For more information about environment settings, see Common Properties and Environment Variables.

Installing the Demo Server

The optional demo server is a JBoss 4.2.3 application server that has several applications for demonstrating DevTest features.

- The **examples** project contains test cases that use the demo server.
- Some of the tutorials in Getting Started use the demo server.

> **Note:** The demo server uses port 1529, which cannot be in use by any other application. If this port is not available the demo server does not start successfully.

Java 8 is required to run the demo server.

Follow these steps:

1. Download the demo server as described in Download DevTest Installers.

2. Select one of the following approaches to installing:

   - (Preferred) Install DevTest Solutions (Server or Workstation) and have the setup wizard unzip the file into the same installation directory (**LISA_HOME**). The setup wizard also creates a desktop icon for the demo server.

   - Unzip the **DevTestDemoServer.zip** file on your computer. (In Windows, this file is downloaded to your Downloads folder.) Go to the **lisa-demo-server** folder and follow the instructions in the README file. The README file contains platform-specific instructions for starting the demo server.

3. If you select the second approach, be aware of the following information:

   - You must have Java 8 installed separately on your system.

   - Set the environment variable **JAVA_HOME**. This variable is required for JBoss to compile and run JSP files.

   - Do not put the JBoss Server directory on your desktop or any path that contains spaces. JBoss cannot compile the JSPs if there is a space in the path to its directory.

Notes:

- To start the demo server from the command line, go to the **LISA_HOME\DemoServer\lisa-demo-server** directory and start the script for your operating system:

  - (Windows) **start-windows.bat**

  - (UNIX or Linux) **start-unix-linux.sh**

  - (OS/X) **start-osx.command**

- To run the demo server on UNIX or Linux, use the **/bin/bash** shell.

- The demo server runs the DevTest Java Agent by default, reporting as much information as possible back to the Application Insight feature. To turn off this reporting, use the **-noagent** flag. To turn off heap / stack information only, use the **-noheapss** flag.

- If native agent binaries are present on the widely used UNIX and Linux distributions, the demo server launches the native agent instead. To make the demo server use the pure Java agent in this case, pass the **--javaagent** parameter to the startup command. The native agent binaries are only intended for use with Java 1.4 and earlier.

- The demo server database is created when the demo server is started for the first time. This database is located in the **LISA_HOME\DemoServer\lisa-demo-server\jboss\server\default\data\lisa-demo-server.db** directory.

- After the demo server is started, you can access the server with a browser on port 8080.

Running Forward Cars

Forward Cars is a web-based demo application that simulates popular car-buying applications that are available online.

Forward Cars contains the following key areas of functionality:

- **Inventory Search**
  Lets you search the database to find your next car.

- **VIN Lookup**
  Lets you look up a car's unique serial number. The vehicle identification number (VIN) identifies not only the vehicle type and specifications, but also helps authorities and organizations track a vehicle's past records.

- **Loan Application**
  Completes a credit check of a customer who applies for a loan.

- **Loan Status**
  Responds to the loan application with up to three banks that provide their interest rate to service the loan.

The following diagram shows the architecture of Forward Cars. The web container provides the front end of the application. The web container interacts with an ESB layer. The ESB layer uses JSON and REST to communicate with an inventory application. The ESB layer uses the Java Message Service (JMS) to communicate with a loan application.

For more information about using Forward Cars, see Tutorial 11: Run the Forward Cars Demo Application.

Follow these steps:

1. Download and unzip the DevTestDemoServer.zip file on your computer. Forward Cars is included in the Demo Server download file and is located in the **LISA_HOME\DemoServer** file directory.

2. Start Forward Cars from the command line. Go to the **LISA_HOME\DemoServer\carsdemo** directory and start the script for your operating system:

- (Windows) **start.bat**
- (UNIX or Linux) **./start.sh**
- (OS X) **./start.command**

To start Forward Cars without agents, add the **--noagent** option. If you do not need the agents, this option can improve the system performance.

The application can take several minutes to start.

3. Enter the following URL in a web browser:

   `http://localhost:3434/cars-app`

   We recommend using Chrome or Firefox browser when accessing the Forward Cars demo in order to avoid compatibility issues.

   > **Note**: Because Forward Cars uses port 3434, that port cannot be in use by any other application. If this port is not available, Forward Cars does not start successfully.

   The home page opens:

Verifying the DevTest Installation

This section describes how to verify that the DevTest installation was done correctly.

Follow these steps:

1. Start Identity and Access Manager.
2. Prepare to log in to the user interfaces. You can:

- Log in as the standard DevTest Super User.
- Create a DevTest user with the Super User role for yourself and log in with your own credentials.

3. Access each user interface and log in.
4. (Optional) Examine the directories.

Start the DevTest Processes or Services

This section explains the process of starting the DevTest Server with all components available. Use the sequence that is shown to ensure that all components start. Some of the ways you can start the DevTest processes or services are described following the start process.

Note: For more information about the processes, see DevTest Server Components.

If starting Enterprise Dashboard or any of the DevTest web services takes a long time, check your hard disk for fragmentation. Defragging the disk can improve performance.

Start Order Sequence

Start the DevTest Server processes (or services) in the following sequence:

1. Start Identity and Access Manager.
2. Start the Enterprise Dashboard Server.
3. Start each registry.
4. Start DevTest Portal.
5. Start the following components in any order:
   - Broker
   - Coordinator server
   - Simulator server that is associated with each coordinator
   - Virtual Service Environment
6. If used on the DevTest Server, start DevTest Workstation.

Note: To shut down the server components, use the reverse order.

Ways You Can Start DevTest Server

Access the DevTest Server processes in one of the following ways:

- (Windows) Click the **Start** menu and expand **DevTest Solutions**. Start the processes in start order sequence.
- Go to the **LISA_HOME\bin** folder. Start the executables in start order sequence.
- (Windows) Go to **Services** and start the services in start order sequence.
- Open a command prompt as an Administrator or open a terminal window. Go to **LISA_HOME\bin**, and enter the command to start each process in start order sequence. You can also start the respective services or start the associated vmoptions file. Get help by entering the command name followed by **--help**.
- Execute the scripts in the **LISA_HOME\bin** folder. The **startdefservers.bat** script starts all components except the Enterprise Dashboard in the correct order. The **stopdefservers.bat** script stops all components.

Log in as the Standard Super User

Setting up security involves creating role-based user accounts and specifying the credentials in Identity and Access Manager or configuring your LDAP user federation. If you plan to use LDAP for authentication, do not add your own user name and password as described in Create a User with the Super User Role. To access the UIs before you configure LDAP, use the standard user that is defined with the Super User role.

When you access DevTest Workstation, a login dialog opens.

To log in, type *admin* in the **Username** field, type *admin* in the **Password** field, and click **Login**.

When you browse to DevTest Portal, the login area opens.

To log in, type *admin* in the **User Name** field, type *admin* in the **Password** field, and click **Log in**.

Create a User with the Super User Role

Grant yourself full access to DevTest Solutions.

Follow these steps:

1. Start the Identify and Access Manager UI (https://localhost:51111).
2. Log in as the standard administrator user.
    a. Enter *admin* for the user name.
    b. Enter *admin* for the password.
3. Select **Users** under the **Manage** menu.
   The Lookup tab opens on the Users page.

4. Create a user account for yourself with a password that you will not forget. Assign a Super User role for full access to DevTest Solutions.

    a. Enter the user details.

        - Required fields are noted with a red asterisk.
        - Click the question mark beside each field for more information about that field.

    b. Click **Save**.

    c. Click the **Credentials** tab.

    d. Enter a password that you will not forget and click **Reset Password**.

    e. Click **Change Password** on the confirmation window.

    f. Click the **Role Mappings** tab.

    g. Select **Super User** in the **Available Roles** column and click **Add Selected**.

Now, you can access any DevTest user interface or command-line interface and log in with the user ID and password that you defined.

Access the User Interfaces

The Enterprise Dashboard, the registry, and the Portal that you are using must be running for all UIs. See Start the Server Components.

Access the UIs as described here. Then, log in as the standard Super User or log in with your own credentials, where you defined yourself with the Super User Role.

Enterprise Dashboard

To view the list of registries that are connected to the Enterprise Dashboard:

- Browse to the Enterprise Dashboard UI and log in.

  `http://hostname:1506`

- (Windows) Log in to the server where the Enterprise Dashboard is installed. From the Start menu, expand **DevTest Solutions** and select **DevTest Enterprise Dashboard UI**.

Portal

The UI requires the Broker to be running.

CA Application Test and CA Service Virtualization use the Portal for some functionality. See Using CA Application Test and Using CA Service Virtualization.

Before you use the Application Insight feature, install the DevTest Java agent. Browse to the following URL and log in to the DevTest Portal to access Application Insight. Specify the host name of a computer with a running registry. See Using Application Insight.

- Browse to the Portal UI and log in.

  `http://hostname:1507/devtest`

- (Windows) From the Start menu, expand **DevTest Solutions** and select **DevTest Portal UI**.

Workstation

Start DevTest Workstation on your local host. DevTest Workstation is where you create and edit tests and also stage tests. Alternative methods include TestRunner and junitlisa ant task, for example. See Using CA Application Test. The UI requires a coordinator server and a simulator server to be running.

- (Windows) From the Start menu, expand **DevTest Solutions** and select **Workstation**.
- Go to **LISA_HOME\bin** and run the **Workstation.exe**.

If DevTest Workstation displays a “Request Timed Out” error while connecting to the registry, you might have to start the Workstation executable as “Run as Administrator.”

Installing Virtual Service Catalog

Virtual Service Catalog provides a central repository for viewing and sharing virtual services. From a single dashboard, you can quickly view the available services that are deployed to a VSE and accessible in any Enterprise Dashboard that is connected to Virtual Service Catalog.

> **Note:** Before you can use Virtual Service Catalog, you must create users with the appropriate permissions in Identity and Access Manager. Identity and Access Manager must installed as part of the DevTest installation prior to installing Virtual Service Catalog. Identity and Access Manager must also be running before uses can access Virtual Service Catalog. For more information, see Installing and Configuring DevTest Server and Identity and Access Manager.

This section describes how to install Virtual Service Catalog.

- Install Virtual Service Catalog on Windows
- Install Virtual Service Catalog on UNIX
- Install Virtual Service Catalog on OS X

> **Note:** Virtual Service Catalog only supports Enterprise Dashboards and Registries that are on DevTest 10.3 or later.

Install Virtual Service Catalog on Windows

Before you start the installation procedure, download the Virtual Service Catalog (vscatalog) installer files from the Download Center:

Follow these steps:

1. Run the installer file; for example, **vscatalog_win_x64.exe**.
   The Welcome to the Virtual Service Catalog Setup Wizard step opens.

2. Click **Next**.
   The CA End User License Agreement step opens.

3. Read the license agreement, scrolling to the end. Select the **I accept the terms of the License Agreement** option, and click **Next**.
   The Select Destination Directory step opens.

4. Enter the path and folder name for the installation directory and click **Next**.

   - Consider a name that includes the release identifier, if you want to keep the current release separate from older releases.

   - If you browse to a path and enter the name of a new folder, the installation wizard creates the folder.

   > **Important!** The default installation directory for Virtual Service Catalog is **C:\Program Files\CA\vscatalog**. If you are using a version of Windows that prevents you from writing directly to C:\Program Files, you must select a different directory.

   The Identity and Access Manager step opens. Identity and Access Manager is required to install and use Virtual Service Catalog.

5. Enter the URL for your existing IAM service and click **Next**.

   - This URL is the host and port where Identity and Access Manager is installed.

   - Identity and Access Manager must installed as part of the DevTest installation prior to installing Virtual Service Catalog. For more information, see Installing and Configuring DevTest Server

   - Clicking **Validate** confirms that you have entered a URL with a valid format. This validation does not attempt to connect to IAM.

   The Desktop Icons step opens.

6. If you do not want to create a desktop icon for Virtual Service Catalog, clear the check box.

7. Click **Next**.
   The Select Start Menu Folder step opens.

8. Specify whether to add Virtual Service Catalog to your Start menu and if so, whether to add Virtual Service Catalog to the Start menu of other users.

   - To create a Start menu folder with shortcuts for all users, accept all defaults. Optionally, enter a new folder name.

   - To have no Start menu folder, clear the **Create a Start Menu folder** check box.

- To create a Start menu folder and restrict the shortcut display to your Start menu:
  - Accept the selection of **Create a Start Menu folder**.
  - Accept the default name or enter another name.
  - Clear the **Create shortcuts for all users** check box.

9. Click **Next**.
   The Windows Services step opens.

10. (Optional) Select **Install Services** to create the following Windows service:

    - Virtual Service Catalog service.

    This selection adds the service to Administrative Tools, Component Services, Services. The advantage to starting services as opposed to the associated executables is that you reduce the number of icons that are displayed on the system tray. If you want the Startup type to be defined as Automatic so that the services start when the host computer is restarted, select the **Start on bootup** check box. You can configure Automatic startup within Services, if you do not select this check box here.

    To have the services on Windows run under a specific user account, click **Optional Service Account**. To have the services run under the system account, select **System Account**. To have the services run under another account, select **User Account** and enter the account name and password.

11. Click **Next**.
    The Installing step opens. When the installation finishes, the Information step opens.

12. Read the information and click **Next**.
    The Completing the Virtual Service Catalog Setup Wizard step opens.

13. Click **Finish**.

Install Virtual Service Catalog on UNIX

Before you start the installation procedure, download the Virtual Service Catalog (vscatalog) installer files from the Download Center:

The following procedure is based on the graphical version of the installer. To use the command-line version of the installer, add the **-c** option. For example:

```
./vscatalog_linux_x64.sh -c
```

If a graphical front end is not available on your computer, you must use the command-line version.

Note: If you downloaded the generic UNIX installer, ensure that a Java virtual machine (JVM) is on the same computer. If the installer cannot find a JVM, the installer displays a message and exits. For more information, see the section on supplying your own JVM in System Requirements.

Follow these steps:

1. In a terminal window, navigate to the directory where the installer file is located.

2. Ensure that the installer file has the execute permission. For example:

   ```
   chmod 777 vscatalog_linux_x64.sh
   ```

   This command gives the read, write, and execute permissions to each class of user.

3. Run the installer file. For example:

   ```
   ./vscatalog_linux_x64.sh
   ```

   > On Ubuntu systems, if you install as root user, you can not display DevTest icons on the desktop.

   The Welcome to the Virtual Service Catalog Setup Wizard opens.

4. Click **Next**.
   The CA End User License Agreement step opens.

5. Read the license agreement, scrolling to the end. Select the **I accept the terms of the License Agreement** option, and click **Next**.
   The Select Destination Directory step opens.

6. Enter the path and folder name for the installation directory and click **Next**.

   - Do not use a directory that contains spaces.
   - Consider a name that includes the release identifier, if you want to keep the current release separate from older releases.
   - If you browse to a path and enter the name of a new folder, the installation wizard creates the folder.

   The Identity and Access Manager step opens. Identity and Access Manager is required to install and use Virtual Service Catalog.

7. Enter the URL for your existing IAM service and click **Next**.

   - This URL is the host and port where Identity and Access Manager is installed.
   - Identity and Access Manager must installed as part of the DevTest installation prior to installing Virtual Service Catalog. For more information, see Installing and Configuring DevTest Server
   - Clicking **Validate** confirms that you have entered a URL with a valid format. This validation does not attempt to connect to IAM.

   The Desktop Icons step opens.

8. If you do not want to create a desktop icon for Virtual Service Catalog, clear the check box.

9. Click **Next**.
   The Installing step opens. When the installation finishes, the Information step opens.

10. Read the information and click **Next**.
    The Completing the Virtual Service Catalog Setup Wizard step opens.

11. Click **Finish**.

Install Virtual Service Catalog on OS X

Before you start the installation procedure, download the Virtual Service Catalog (vscatalog) installer files from the Download Center:

Follow these steps:

1. Run the installer file, for example, **vscatalog_osx_x64.dmg**.
   The Welcome to Virtual Service Catalog Setup Wizard step opens.

2. Click **Next**.
   The CA End User License Agreement step opens.

3. Read the license agreement, scrolling to the end. Select the **I accept the terms of the License Agreement** option, and click **Next**.
   The Select Destination Directory step opens.

4. Enter the path and folder name for the installation directory and click **Next**.

   - Do not use a directory that contains spaces.

   - Consider a name that includes the release identifier, if you want to keep the current release separate from older releases.

   - If you browse to a path and enter the name of a new folder, the installation wizard creates the folder.

   The Identity and Access Manager step opens. Identity and Access Manager is required to install and use Virtual Service Catalog.

5. Enter the URL for your existing IAM service and click **Next**.

   - This URL is the host and port where Identity and Access Manager is installed.

   - Identity and Access Manager must installed as part of the DevTest installation prior to installing Virtual Service Catalog. For more information, see Installing and Configuring DevTest Server

   - Clicking **Validate** confirms that you have entered a URL with a valid format. This validation does not attempt to connect to IAM.

   The Desktop Icons step opens.

6. If you do not want to create a desktop icon for Virtual Service Catalog, clear the check box.

7. Click **Install**.
   The Installing step opens. When the installation finishes, the Information step opens.

8. Read the information and click **Next**.
   The Completing the Virtual Service Catalog Setup Wizard step opens.

9. Click **Finish**.

More Information:

- Configure Virtual Service Catalog
- Identity and Access Manager
- DevTest Solutions Architecture
- Installing and Configuring DevTest Server

Installing Integration Tools

This section describes how to install and configure third-party tools that can be used with DevTest.

- Install Performance Monitor (Perfmon)
- Install and Configure SNMP
- Run TCPMon
- Install DevTest ALM Plug-in
- Set Up the SAP System Landscape Directory
- Configure CA Agile Central

Install Performance Monitor (Perfmon)

Performance Monitor (Perfmon) is a utility that demonstrates monitoring the performance of the local or remote system. Perfmon demonstrates how to monitor system performance using performance counters.

To use Perfmon to monitor the performance of a Windows system:

- You must have .NET Framework 2.0 compatibility.
- From a command prompt, run the **setup-wperfmon.bat** file that is located in the **LISA_HOME\bin** directory.
- On Windows, the command prompt must be "Run as Administrator".

Note: If you are running Windows 2012, you can achieve .NET Framework 2.0 compatibility by installing .NET Framework 3.5.

In addition, ensure:

- The user ID is the same on both computers.
- The user ID has administrator privileges on both computers.
- File and Printer sharing is turned ON.
- Simple File sharing is turned OFF.
- The default C$ share, ADMIN$ share, or both are enabled.

Sometimes the firewall on the computer to be monitored must be stopped.

To verify that remote monitoring is working:

1. Select **Start, Control Panel, Administrative Tools, Performance**.
2. Add a monitor to the computer that you want to observe.

DevTest and Windows use the same technology to do remote monitoring. If the Windows monitoring works, then DevTest monitoring typically works.

To use Perfmon to gather metrics in DevTest, see Windows Perfmon Metrics.

Install and Configure SNMP

This section describes how to install and configure SNMP to work with DevTest.

- SNMP support on UNIX
- SNMP support on Windows
  - Install Microsoft SNMP Agent on Windows 7 and Windows 10
  - Install Microsoft SNMP Agent on Earlier Windows Versions
  - Configure Microsoft SNMP Agent

The Microsoft Windows implementation of the Simple Network Management Protocol (SNMP) is used for the following tasks:

- Configure remote devices
- Monitor the network performance
- Audit network usage
- Detect network faults or inappropriate access.

SNMP support on UNIX

SNMP support 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.

SNMP support on Windows

Windows 7 and Windows 10 provide an agent that is able to answer SNMP requests and send traps.

Install Microsoft SNMP Agent on Windows 7 and Windows 10

Follow these steps:

1. Open the Windows **Control Panel**.
2. Click **Programs**.
3. Under Programs and Features, select **Turn Windows features on or off**.
   The Windows Features window opens.
4. Select the **Simple Network Management Protocol (SNMP)** check box and click **OK**.
5. Wait for SNMP to be installed.

Install Microsoft SNMP Agent on Earlier Windows Versions

Follow these steps:

1. Open the Windows **Control Panel**.
2. Double-click **Add or Remove Programs**.
   The **Add or Remove Programs** window opens.
3. Click **Add/Remove Windows Components** on the left side of the window.
   The **Windows Component** wizard appears.
4. Select **Management and Monitoring Tools** in the **Components** list and click **Details**.
   The **Management and Monitoring Tools** window opens.
5. Select **Simple Network Management Protocol** from the **Subcomponents of Management and Monitoring Tools** list and click **OK**.
6. Click **Next**.
   The **Windows Components Wizard** installs the Microsoft SNMP agent.
7. When complete, click **Finish**.

Configure Microsoft SNMP Agent

This procedure describes how to configure the SNMP Agent.

Follow these steps:

1. Open the **Windows Control Panel**.
2. Double-click **Administrative Tools**.
   The **Administrative Tools** window opens.
3. Double-click **Services**.
   The **Services** window opens.

4. Double-click the SNMP service.
   The **SNMP Service Properties** window opens.

5. Change **Startup Type** to **Automatic** on the **General** tab.
   This action configures the SNMP service to start the Microsoft SNMP agent on system startup.

6. Click the **Traps** tab.

7. Type the community name that your computer sends trap messages to in the **Community Name** field.

8. Click **Add to list**.

9. Click **Apply** and **OK**.

10. Click **OK**.

To use Windows SNMP to gather metrics, see Using CA Application Test.

Run TCPMon

TCPMon is a utility that lets you monitor the messages that are passed in a TCP-based conversation.

TCPMon consists of the following elements:

- For Windows: A .jar file, a .bat file
- For UNIX: A shell script

To run TCPMon:

Double-click the .bat file on Windows or execute the shell script on UNIX.

You can find a **tcpmon.bat** file in the **LISA_HOME\bin** directory. You can get the latest version of TCPMon from: http://ws.apache.org/commons/tcpmon/.

> **Note:** This section documents the TCPMon version from Apache. This TCPMon version contains a **Sender** tab that is not available in the TCPMon version that is distributed with DevTest.

Using TCPMon as an Explicit Intermediary

The most common usage pattern for the TCPMon is as an intermediary. This usage is specified as explicit because the client has to point to the intermediary, rather than the original endpoint, to monitor the messages.

To start the TCPMon in this configuration:

1. Provide the listen port, the host name, and the port for the listener in the **Admin** tab for TCPMonitor.
2. To open up a new tab (Port 8000) that allows the messages to be seen, click **Add**.

Requests now point to the listener port of the TCPMon instead of the original endpoint.

All messages that are passed to and from localhost:8080 are monitored.

- We set the listener to port 8000 which can be any unused port in the local computer.
- We added a listener with host as **localhost** and port as 8080.
- Point the browser to **localhost:8000** instead of localhost:8080.

Using TCPMon as a Request Sender

TCPMon can also be used as a request sender for web services.

- The request SOAP message can be pasted into the **Sender** window and then sent directly to the server.
- The web service endpoint is entered in the **connection Endpoint** text box.

Install DevTest ALM Plug-in

The plug-in lets you load and run a DevTest test case as an ALM test from the HP ALM suite. The plug-in only supports a 64-bit installation of DevTest.

See Using the DevTest ALM Plug-in for information about using the plug-in.

This section contains the following information:

- Prerequisites
- Installation
- Post Installation Files
- InstallationUtility Command-Line
  - Recreate DevTestConnection.properties File
- Using the HP ALM Client to Run Test Cases

Prerequisites

The plug-in has the following prerequisites:

- LISA_HOME environment variable must be defined and set to the root directory of the DevTest installation.

- The DevTest ALM plug-in runs on Microsoft Windows only.
- DevTest ALM plug-in must be installed where DevTest Workstation and the test files are located.

The following software must be installed:

- DevTest 10.2 64-bit installation (including the corresponding version of Java and browser)
- HP ALM 12.20, 12.21, 12.50, or 12.53
  - HP ALM Connectivity, HP ALM Client Registration
  - HP ALM Client Side components: HP ALM Connectivity, HP ALM Client Registration
- .NET 4.5 Runtime or later

Installation

Follow these steps:

1. Ensure that the .NET runtime is installed on the client computer.
2. Install the HP ALM Client Side components: HP ALM Connectivity and HP ALM Client Registration.
3. Install DevTest.
4. Define the LISA_HOME environment variable and set it to the root directory of the DevTest installation.
5. Go to the LISA_HOME\addons\qc directory and run the executable file for your system configuration:
   - DevTestALMPlugin_x64.exe
6. Complete the wizard steps.
   The DevTestConnection.properties files are created.
7. Ensure all the DevTest services are running.

Post Installation Files

The DevTestConnection.properties file is created in the LISA_HOME\alm-plugin directory.

The file contains the following connection information entered during the installation process:

- dt_hostname
- dt_scheme
- dt_port
- dt_coordinator
- dt_username

- dt_password_encrypted

The following files are installed in the LISA_HOME\bin directory:

- devtestALMPlugin.dll
- InstallationUtility.exe

The following files are installed in the LISA_HOME\doc directory:

- script_template.js
- script_template.vbs

When running ALM tests, the plug-in generates the alm-devtest.log file in the LISA_HOME\alm-plugin folder.

InstallationUtility Command-Line

The plug-in installer uses the InstallationUtility executable to register the plug-in and to create the DevTestConnection.properties file. Use the InstallationUtility executable to change the connection information and generate a new encrypted password in DevTestConnection. properties file.

Run the following command to get the usage information:

```
C:\Program Files\CA\DevTest|bin>InstallationUtility Installation Utility H
```

```
InstallationUtilty command <argument>
```

```
InstallationUtility  -h | --help
```

The following commands are available:

- **register <ddl_name>**
  Registers the dll file.
- **deregister <dll_name>**
  Deregisters the dll file.
- **check <dll_name>**
  Checks the registration of the dll file.
- **config <options>**
  Creates or replaces the configuration file.

The following options are available:

- **-h, --help**
  Displays the help screen.
- **--user <user>**
  The DevTest username. Required for the configuration.
- **--pass <password>**
  The DevTest password. Required for the configuration.
- **--host <hostname>**
  The DevTest host name. Required for the configuration.

- **--port <port>**
  The DevTest port 1505. Required for the configuration.

- **--protocol <HTTP | HTTPS>**
  The DevTest protocol; HTTP or HTTPS. Required for the configuration.

- **--coord <coordinator>**

The DevTest coordinator. Required for the configuration. Set to **Coordinator** for the local coordinator.

Recreate DevTestConnection.properties File

You can use the InstallationUtility executable to edit the connection information and recreate the connection settings to the registry. This process creates a new encrypted password. If a new encrypted password is not necessary, edit the connection information from the DevTestConnection.properties file.

> **Note:** Create a backup of the DevTestConnection.properties file before you generate a new file using the InstallationUtility executable.

The executable has the following recreate format:

```
InstallationUtility config --host localhost --user admin --pass admin --pc
```

> **Note:** You must enter the parameters for all the connection information to generate a new DevTestConnection.properties file.

Using the HP ALM Client to Run Test Cases

The following diagram shows a test case being initiated from an ALM client to run on CA Application Test. Tests are retrieved from the URLs attached in the ALM Client. The ALM test definitions are mapped to the actual DevTest tests using MAR, MAR info, or test, configuration, and staging files. Tests are executed from the HP ALM client, which triggers the plug-in to execute the DevTest tests. The test results are sent back to the HP ALM client.

> DevTest test suites are not supported in HP ALM.

See Run DevTest Tests in HP Quality Center for more information about how to run tests.

Set Up the SAP System Landscape Directory

This page describes how to set up the SAP System Landscape Directory (SLD) for CA Service Virtualization.

Follow these steps:

1. Obtain the following information for the SLD server:
   - Host
   - Port
   - User
   - Password
2. Log on to the System Landscape Directory.

> **Note:** you must have a LcrInstanceWriterLD role to perform this task.

3. Perform one of the following actions:
   - Manually Register DevTest with SLD
   - Import the DevTest SLD XML File

Manually Register DevTest with SLD

This page describes how to manually register DevTest with the SAP System Landscape Directory (SLD). This process creates a product name/version and software component name/version in the SLD software catalog.

Follow these steps:

1. In the SLD software catalog, select **Products**, **New Product Version**.
2. Enter **CA LISA SV** in the **Product Name** field.
3. Enter **CA Technologies** in the **Vendor Name** field.
4. Enter the version number; for example, **V9.5**, in the **Product Version** field.
5. Click **Create**.
6. Enter **CA LISA SV** in the **Technical Name** field.
7. Enter **CA LISA** in the **Software Component Name** field.
8. Enter the version number; for example, **V9.5**, in the **Software Component Version** field.
9. Enter **released** in the **Production State** field.
10. Click **Create**.

Import the DevTest SLD XML File

This page describes how to set up the SAP System Landscape Directory (SLD) by importing the DevTest SLD XML file.

Follow these steps:

1. In the SLD software catalog, select **Administration**, **Content**, **Import**.
2. Enter the path of the provided SLD data zip file in the **Selected File** field.
   This file is located in **LISA_HOME\addons\sap\CALISA.xml**
3. Click **Import Selected File**.

After you install DevTest, refer to the following paths:

- Default installation path: **C:\lisa\bin**
- Default log file path: **C:\Users\<userID>\lisatmp_<version number>**
  For detailed information about each log file and its meaning, refer to Logging.
  All log files are human readable.
- Default configuration file path: **C:\LISA_HOME**

- Default configuration files: **lisa.properties** and **local.properties**
  For detailed information about the order of properties files and setting values in the log files, refer to Properties.
  All property files are human readable java name=value entries.

Configure CA Agile Central

Integrate CA Agile Central with DevTest to view and manage DevTest tests and test cases results in CA Agile Central.

An API key must be created for DevTest to interact with CA Agile Central. The API key is stored as a property in the DevTest configuration file that is known as the **site.properties** file.

Before creating the API key, confirm that you have Editor permissions for the CA Agile Central workspace that you want to integrate with DevTest Solutions. Also confirm that this workspace is set as your default workspace. Select **My Settings** to view the Profile page and review your permissions and default workspace.

Follow these steps:

1. Go to https://rally1.rallydev.com/login/accounts/index.html#/apps.

   Note: You must be logged into CA Agile Central to access this page.

2. Click the **API Keys** tab.
3. Select **Create New API Key**.
4. Enter a name or description of the key.
5. Select **Full Access**.
6. Click **Create**.
   The key is generated.
7. Enter the key as the value for the **rally.api.key** property in the **site.properties** file.
8. (Optional) To configure a connection to CA Agile Central through a proxy, configure the following properties in the **site.properties** file:
   - **rally.proxy.host**
   - **rally.proxy.port**
   - **rally.proxy.username**
   - **rally.proxy.password**

   If your username contains a backslash, use a double backslash \\ in the username.
   For more information about updating the **site.properties** file, see Custom Property Files.

When creating tests in DevTest , the name of the test case must match the name of the test case in CA Agile Central. However, you must remove the colon after the test case prefix and replace it with a dash, with no space. For example, if the name of the test case in CA Agile Central is **TC1255: Inventory_lookupVIN**, the test case name in DevTest must be **TC1255-Inventory_lookupVIN.tst**.

You must also add the CA Agile Central Report Generator to your staging document. The staging document must contain at least one other report generator type, such as Default Report Generator. Do not replace the Default Report Generator with the CA Agile Central Report Generator.

Installing DevTest Solutions with a Silent Install

A silent installation allows for unattended installation and does not prompt you for any input. Use a silent installation when there are similar installations to be performed on more than one computer.

DevTest Solutions uses install4j 6.0.3.

> The **enableTelemetry=** property in your response.varfile determines whether you want to send diagnostic and usage data to CA Technologies to help us improve our products and services. Valid values are true and false. For more information about this option and the data that is collected, see Telemetry and Telemetry Data.

To perform a silent install on Windows:

1. Consult the install4j documentation for detailed information about unattended installations.

2. Consult the DevTest installation instructions for installation options to use in the **response.varfile**.

3. Perform one of the following actions:

   - Install DevTest Solutions and locate the response file in **LISA_HOME\.install4j\response.varfile**. Store this file in a location where you can retrieve it later. The **response.varfile** is used during the silent install process when DevTest is installed on another system.

   - Create a **response.varfile**, using the examples below, changing the values as necessary.

     > If you do not specify the lisaDataDir in the response.varfile, we assume the data directory will be the same as the install directory.

     >

> If your license file is not in the default location, ensure that the **licenseXmlFile=** property in your response.varfile reflects the correct directory.

4. To initiate a silent installation, execute the following command (the example assumes a Windows system):

```
devtest_win_x86.exe –q –varfile response.varfile
```

To perform a silent install on Linux:

1. Consult the install4j documentation for detailed information about unattended installations.

2. Consult the DevTest installation instructions for installation options to use in the **response.varfile**.

3. Install DevTest manually on a Linux system. The following steps assume that the installer, demo server, and license file are located in **/home/user123/DevTest** and that DevTest is installed to **/opt/CA/DevTest**.

> If your license file is in a different directory, ensure that the **licenseXmlFile=** property in your response.varfile reflects the correct directory.

4. Navigate to the **/opt/CA/DevTest/.install4j** folder and copy the response.varfile to a new virtual machine.

5. On the new Linux virtual machine, navigate to **/home/user123/DevTest**. The response.varfile, devtest_linux_x64.sh, devtestlic.xml, and DevTestDemoServer.zip files will be in this directory. Also, the devtest_linux_x64.sh file must have executable permissions (chmod +x devtest_linux_x64.sh).

6. Run the following command in the **/home/user123/DevTest** directory to install DevTest silently:

```
./devtest_linux_x64.sh -q -varfile response.varfile
```

To perform a silent install on OS x:

1. Consult the install4j documentation for detailed information about unattended installations.

2. Consult the DevTest installation instructions for installation options to use in the **response.varfile**.

3. Install DevTest manually on an OS X system. The following steps assume that the installer, demo server, and license file are located in ***/user.home*/Downloads** and that DevTest is installed to **/Applications/CA/DevTest**.

> If your license file is in a different directory, ensure that the **licenseXmlFile=** property in your response.varfile reflects the correct directory.

4. Navigate to the **/Applications/CA/DevTest/.install4j** folder and copy the response.varfile to a new OS X machine.

5. On the new OS X machine, navigate to **/Users/user123/DevTest**. The response.varfile, devtest_osx_x64.dmg, devtestlic.xml, and DevTestDemoServer.zip files will be in this directory.

6. Run the following commands in the **/Users/user123/DevTest** directory to install DevTest silently:

```
hdiutil attach devtest_osx_x64.dmg
/Volumes/DevTest/DevTest\ Solutions\ Installer.app/Contents/MacOS
/JavaApplicationStub -q -varfile response.varfile
hdiutil detach /Volumes/DevTest
```

Sample Response File (Windows)

```
# install4j response file for DevTest Solutions 10.4.1 (build 0.0.0.0)
autostartServices$Boolean=false
caAgreementChoice=2
createDesktopLinkAction$Boolean=true
installDemoServer=No
installEnterpriseDashboard=Yes
installServer=Yes
installServices$Boolean=true
installWorkstation=Yes
licenseXmlFile=C\:\\license\\devtestlic.xml
# If installing SERVER COMPONENTS, we strongly recommend that the lisaDataDir is
BLANK or is the same as the install directory.
# If installing WORKSTATION ONLY, the lisaDataDir is used to define the data
directory location.
# When an administrator is installing WORKSTATION ONLY on behalf of another user,
it is recommended to use %userprofile%\\DevTest or ~/DevTest. Note, the end user
needs to have write permission for the data directory.
# Workstation data directory components: _local.properties, logging.properties, de-
identify.xml, hotdeploy, library for customizations, projects with assets, example
folders
lisaDataDir=
serviceAccount=system
serviceAccountName=S-1-5-18
serviceAccountPassword=
steAssociation$Boolean=true
stgAssociation$Boolean=true
sys.adminRights$Boolean=true
sys.installationDir=C\:\\Program Files\\CA\\DevTest
sys.languageId=en
sys.programGroupAllUsers$Boolean=true
sys.programGroupDisabled$Boolean=false
sys.programGroupName=DevTest Solutions
tstAssociation$Boolean=true
vsiAssociation$Boolean=true
vsmAssociation$Boolean=true
enableTelemetry=true
## ============================================
## IAM variables:
##  'sys.component.38273$Boolean'
##   Possibles values:
##   - true: indicates install Identity Access Manager
##   - false: indicates does not install Identity Access Manager
## The following variables are only valid if 'sys.component.38273$Boolean' is
'true'
##  'dbType$Integer'
##   Possibles values:
##   - 0: EMBEDDED
##   - 1: DB2
##   - 2: Microsoft SQL Server
##   - 3: MySQL
##   - 4: Oracle
##  'dbHost': The database host name
##  'dbPort$Long': The database port number
```

```
##  'dbName': The database name
##  'dbUsername': The username of the user with rights on the database
##  'dbPassword': The password of the user with rights on the database
##  'driverPath': The full path of the Jar with the DB Driver
## The following variables is only valid if 'sys.component.38273$Boolean' is
'false'
##  'iamServerUrl': it is the full URL of the exiting IAM as: https:[HOSTNAME]:
51111/auth
## ==============================================
sys.component.38273$Boolean=true
dbType$Integer=2
dbHost=norto02N230940
dbPort$Long=1433
dbName=keycloak
dbUsername=sa
dbPassword=interOP@123
driverPath=C\:\\drivers\\MSSQL\\sqljdbc4-4.0.jar
```

Sample Response File (Linux)

```
# DevTest Local Install
 sys.adminRights$Boolean=true
 sys.programGroupDisabled$Boolean=false
 installServer=Yes
 installWorkstation=Yes
 createDesktopLinkAction$Boolean=true
 installEnterpriseDashboard=Yes
 sys.languageId=en
 caAgreementChoice=2
 sys.installationDir=/opt/CA/DevTest
 licenseXmlFile=/home/user123/DevTest/devtestlic.xml
 sys.symlinkDir=/usr/local/bin
 installationType$Integer=0
 installDemoServer=Yes
 demoServerZipFile=/home/user123/DevTest/DevTestDemoServer.zip
enableTelemetry=true
## ==============================================
## IAM variables:
##  'sys.component.38273$Boolean'
##   Possibles values:
##   - true: indicates install Identity Access Manager
##   - false: indicates does not install Identity Access Manager
## The following variables are only valid if 'sys.component.38273$Boolean' is
'true'
##  'dbType$Integer'
##   Possibles values:
##   - 0: EMBEDDED
##   - 1: DB2
##   - 2: Microsoft SQL Server
##   - 3: MySQL
##   - 4: Oracle
##  'dbHost': The database host name
##  'dbPort$Long': The database port number
##  'dbName': The database name
##  'dbUsername': The username of the user with rights on the database
##  'dbPassword': The password of the user with rights on the database
##  'driverPath': The full path of the Jar with the DB Driver
## The following variables is only valid if 'sys.component.38273$Boolean' is
'false'
##  'iamServerUrl': it is the full URL of the exiting IAM as: https:[HOSTNAME]:
51111/auth
## ==============================================
sys.component.38273$Boolean=true
dbType$Integer=2
dbHost=norto02N230940
dbPort$Long=1433
dbName=keycloak
dbUsername=sa
dbPassword=interOP@123
driverPath=C\:\\drivers\\MSSQL\\sqljdbc4-4.0.jar
```

Docker Containers

DevTest Solutions supports using Docker to deploy DevTest instances in multiple environments as part of a continuous delivery process.

Docker Prerequisites

> Note: Using Docker with DevTest Solutions requires advanced expertise with Docker images and containers. Do not use DevTest with Docker unless you are thoroughly familiar with Docker fundamentals and processes.

Using DevTest with Docker requires:

- You must have a full version of Linux that supports Docker features.
- You must have Docker installed in your environment (https://docs.docker.com/installation/#installation).
- You must have the Linux version of DevTest Solutions installed.
- DevTest Docker support was developed and tested on Docker version 1.11.2.
- To build our Docker containers with our **gradlew** script, your machine requires unrestricted internet access to download the current libraries to support the Gradle/Docker build process. We do not support running our **gradlew** script on a machine that has restricted access to the internet.

  If you need to deploy your containers on a server that has restricted internet access, you must build your images on another server that does have unrestricted internet access, and then you can move your images from the build server to your Docker deployment server using the following Docker commands.

  ```
  [root@localhost docker]# docker save --output=portal-base.tar devtest/portal-base:latest
  [root@localhost docker]# docker load --input=portal-base.tar
  ```

  > **Note:** Docker support was added in DevTest Solutions 8.3. You must have installed the Linux version of 8.3 or later.

Connecting to Oracle Databases in Docker Containers

The Oracle Client DB Jars require that a *user.timezone* property is set. This value is not passed through Docker containers by default. There are several ways to set this environment variable; one effective method is to add the following line: *-Duser.timezone="-01:00"* (an offset of one hour behind GMT) in the .vmoptions files provided in the Docker images by default. Apply this change to the .vmoptions file for each DevTest component to affect all Docker instances started from the modified containers.

For more on time zones: *http://docs.oracle.com/database/121/NLSPG/ch4datetime.htm#NLSPG259*

Build Baseline Docker Images

DevTest Solutions includes some baseline Docker images. You can use these images as they are or use them as a starting point for creating your own customized images. Before you can view the packaged images, you must perform a Gradle build.

Follow these steps:

1. Ensure that you have completed the prerequisites that are listed previously.
2. Ensure that Docker and the Linux version of DevTest 8.3 or later are both successfully installed.
3. Change to the **docker** subfolder of the DevTest installation.
4. Run the following build:

   > ./gradlew build

   This command uses the DevTest installation from the parent folder to run a build and create the Docker images. On Ubuntu systems, you may need to add a user with permissions to run the command.
5. After the build has completed, use Docker to view the images that were created.

   Note: This build can take 15 minutes or more to complete.

For detailed information about the contents of each image and instructions for using it, see the README file included with each image.

Docker Images

Docker images included in the docker subfolder include:

- **devtest/dradis-base**
  Approximate size: 656.7 MB
  This image can be used to run a standalone Enterprise Dashboard container. See the **dradis-licensed** example for more information.

  Note: Before Enterprise Dashboard will run successfully in a container, you must add your Enterprise Dashboard license file to the images.

- **devtest/portal-base**

  Approximate size: 1.093 GB

This image can be used to run a standalone DevTest Portal container.

- **devtest/demoserver**
  Approximate size: 508.6 MB
  This image can be used to run a standalone DevTest Demo Server container.

- **devtest/servers-base**
  Approximate size: 735.1 MB
  This image can be used to run standalone coordinator, simulator, or VSE containers.

- **devtest/registry-broker-base**
  Approximate size: 1.438 GB
  This image can be used to run standalone registry containers or broker containers. This image is built on top of the **devtest/servers-base** image, so it contains everything the **devtest /servers-base** image contains, plus the necessary additions for the registry and broker processes.

- **devtest/devtest-base**
  Approximate size: 2.083 GB
  This image contains a complete DevTest installation and can be used to run any of the DevTest services or tools.

Note: See the **LISA_HOME/docker\examples\devtest-x11vnc** example for more.

Example

Use the **LISA_MORE_VM_PROPS** property on the command line to point to a running Enterprise Dashboard:

```
  > docker run -P -e LISA_MORE_VM_PROPS="-Dlisa.enterprisedashboard.service.
url=tcp://dradishost.domain.com:2003/EnterpriseDashboard" devtest/registry-base
/opt/bin/Registry
```

Docker Commands to Start DevTest Components in Their Own Containers

In the following examples, Docker is installed on ***10.141.66.40***. This is the IP address that is used for all the components being started in their own containers. You must modify this IP address for your environment.

Starting Enterprise Dashboard Container

Use the dradis-licensed image from the examples folder.

IAM_SERVER_URL is the URL of your Identity and Access Manager instance. You can also enter the host machine IP address and port 51111 if you are running as a service on a host machine.

```
o docker run -d -p 1506:1506 -e LISA_MORE_VM_PROPS="-Diam.server.url=IAM_SERVER_UR
L" --name=dradisEDB dradis-licensed:latest /opt/devtest/bin/EnterpriseDashboard
```

Starting Registry Container

IAM_SERVER_URL is the URL of your Identity and Access Manager instance. You can also enter the host machine IP address and port 51111 if you are running as a service on a host machine.

```
o docker run -d --name registry -p 2010:2010 -p 1505:1505 -p 1528:1528 -e
LISA_MORE_VM_PROPS="-Ddevtest.enterprisedashboard.host=10.141.66.40 -Diam.server.
url=IAM_SERVER_URL" devtest/registry-broker-base /opt/devtest/bin/Registry -n
tcp://10.141.66.40:2010/Registry
```

Starting Broker Container

```
docker run -d --name broker -p 1508:1508 -p 2009:2009 devtest/registry-broker-
base /opt/devtest/bin/Broker -n tcp://10.141.66.40:2009/Broker -m tcp://10.
141.66.40:2010/Registry
```

Starting Coordinator Container

```
docker run -d --name coordinator -p 2011:2011 devtest/servers-base /opt/devtest
/bin/CoordinatorServer -n tcp://10.141.66.40:2011/Coordinator -m tcp://10.
141.66.40:2010/Registry
```

Starting Simulator Container

```
docker run -d --name simulator -p 2014:2014 devtest/servers-base /opt/devtest/bin
/Simulator -n tcp://10.141.66.40:2014/Simulator -m tcp://10.141.66.40:2010
/Registry
```

Starting VSE Container

```
docker run -d --name VSE -p 2013:2013 devtest/servers-base /opt/devtest/bin
/VirtualServiceEnvironment -n tcp://10.141.66.40:2013/VSE -m tcp://10.141.66.40:
2010/Registry
```

Starting Portal Container

IAM_SERVER_URL is the URL of your Identity and Access Manager instance. You can also enter the host machine IP address and port 51111 if you are running as a service on a host machine.

```
 o docker run -d --name DevtestPortal -p 1507:1507 -e LISA_MORE_VM_PROPS="-Diam.
server.url=IAM_SERVER_URL" devtest/portal-base /opt/devtest/bin/Portal -J-
Dregistry.host="registryhost" -J-DlisaAutoConnect="tcp://registryhost:2010
/Registry"
```

Preparing Your DevTest Upgrade

▪ Review authentication component changes ▪ Review new features and changes to DevTest ▪ Ensure you have a DevTest license ▪ Download DevTest Installer	▪ Host name, IP addresses, SSL Certificates, LDAP configuration, database configuration ▪ Review DevTest System Requirements

▪ Assets (Example: your test cases and virtual services are in a source control repository) ▪ Installation folder and data directory (if applicable) ▪ Registry database ▪ Enterprise Dashboard database to remain in compliance with your license agreement.	▪ Ensure configuration files for registry and Enterprise Dashboard have full DBA privileges ▪ Shut down services and applications. This includes anything that connects to the registry. ▪ For 7.5X you must uninstall the Enterprise Dashboard

Once you have finished preparing for the upgrade, check out the following articles:

- Upgrade DevTest
- Additional Upgrade Details
- After You Upgrade

Upgrading the Authentication Component

Starting with DevTest Solutions 10.4, Identity and Access Manager is the component that provides user authentication for all DevTest components. You now use Identity and Access Manager to do the following:

- Manage users and roles
- Manage sessions and events
- Define realm settings
- Federate external user databases (LDAP)

Identity and Access Manager must be running before you start a registry. If Identity and Access Manager is not running, you will not be able to start a registry.

Previously, your LDAP configuration, role definitions, users, and role mapping for users were all stored at the registry level.

Note: Use Identity and Access Manager to create and manage users and roles. Use Access Control (ACL) to assign permissions to individual roles in DevTest Portal.

Important! Identity and Access Manager automatically migrates all of your existing authentication information from earlier versions of DevTest. However, it is important to understand the details of this migration process and the order in which specific steps must be performed to ensure the successful migration of this data. Read through the following sections carefully before performing your upgrade.

A single Identity and Access Manager instance for your entire enterprise allows component access to be managed in a single location. However, if you have multiple registries and need to manage access to each registry separately, you need to install a separate IAM for each registry. This is especially true if users from one registry cannot have rights to another. For more information, see DevTest Process Relationships.

Migrating Authentication Information

The migration of your authentication information is controlled by the **export2iam** file that is located in your **lisatmp** directory. By default, lisatmp is located in the user's home directory or in the directory specified in the **lisa.tmpdir** property in system properties. If you are running Windows services, **lisatmp** is located in the DevTest installation directory.

Initially, the export2iam file does not exist, but when you start a registry for the first time after the upgrade, the migration of your existing data is triggered, and this file is created. The export2iam file then serves as a flag to indicate whether the migration has been run. If the **export2iam** file is present and contains valid content, Identity and Access Manager understands that the migration has already been executed and should not be executed again. If this file is not present, Identity and Access Manager attempts to perform the migration the next time the registry is started.

The migration process migrates your authentication information according to the configurations in the following files:

- local.properties
  This file stores initialization and configuration information. The properties in this file take precedence over the lisa.properties and site.properties files.

- site.properties
  This file stores initialization and configuration information. The properties in this file take precedence over the properties that are defined in lisa.properties. However, they do not take precedence over local.properties.

- **authentication-providers.xml**
  This file stores the details for your LDAP authentication provider.

- **ldap-mappings.xml**
  This file assigns roles to groups of LDAP users.

These files must be configured in your 10.4 environment or copied from your existing environment BEFORE starting any DevTest components. The first time you start a registry after the upgrade, the **export2iam** file is created and the migration is not triggered when the registry is started the next time.

If you ever need to repeat the migration:

1. Shut down the registry
2. Delete the **export2iam** file in the **lisatmp** directory.
3. Restart the registry.

What Gets Migrated?

The migration process migrates the following authentication information:

- **Role Definitions**
  Role definitions are always migrated during this process. However, if the role is already present in Identity and Access Manager, it is not overwritten.
- **LDAP Information**
  LDAP information is only migrated if you configured authentication providers for ACL by modifying the **authentication-providers.xml** file in the installation directory. LDAP mappings in the **ldap-mappings.xml** file are also migrated.
- **User Information**
  User definitions are migrated during this process unless you have a configured LDAP authentication provider. If you have an LDAP provider configuration in the **authentication-providers.xml** file, then user information is migrated from LDAP, and user definitions within DevTest are not migrated. If you are not using the **authentication-providers.xml** file, then your existing DevTest user definitions are migrated to Identity and Access Manager.

Reports

The migration process generates the following report files in your **lisatemp** directory:

- **ExportLDAPReport*.csv**
  This report shows the migration status of each LDAP authentication provider. Valid values in this report are IMPORTED_AS_LDAP, ERROR_WHILE_IMPORTING, and NOT_SUPPORTED.
- **ExportRoleReport*.csv**
  This report shows the migration status of each role that is being migrated. Valid values in this report are ROLE_ADDED, ROLE_ALREADY_EXIST, ROLE_ADDED_FAILURE, UNKNOWN_FAILURE
- **ExportUserReport*.csv**
  This report shows the migration status of each user that is being migrated. Valid values in this report are USER_ADDED, USER_ALREADY_EXIST, USER_ADDED_FAILURE, ROLE_MAPPING_FAILURE, UNKNOWN_FAILURE.

Review these reports after you upgrade to confirm the successful migration of your authentication information.

Video

The following video demonstrates using Identity and Access Manager with LDAP.

The following video demonstrates using Identity and Access Manager with Database Users.

Upgrade DevTest

You can perform a DevTest Solutions in two ways:

- In-place Upgrade: This approach overwrites your existing instance with the new 10.4 instance.
- New Install Upgrade: This approach leaves your existing DevTest instance in place and installs DevTest 10.4 into a new directory.

Both approaches follow the same basic process. Any differences are called out in the procedure below.

Important! Starting with 10.4, authentication for DevTest is managed by Identity and Access Manager. This upgrade process includes migrating your existing authentication information from the registry level of your previous instance. Before proceeding with this upgrade, ensure that you have completed all prerequisite tasks and have reviewed Upgrading the Authentication Component.

Follow these steps:

1. Download the installer.
2. Have your license key available.

   Note: To upgrade from DevTest 10.3 or earlier, you must request a new license file. For more information, see License Administration.

3. Run the installer.
   For assistance with this step, see Installing and Configuring DevTest Server.
4. If you are performing a New Install Upgrade, configure the following property files or copy them from your previous instance:
   - **local.properties**
   - **site.properties**
   - **authentication-providers.xml**
   - **ldap-mappings.xml**
   - **vmoptions**
   - **dradis.properties**
     If you elected to to send diagnostic and usage data to CA Technologies, ensure that the **com.ca.prism.enabled=** property is set to **true**.

   If you are performing an In-place Upgrade, these files are already present and configured.

> **Important!** These files contain your database and LDAP configurations. These properties must be configured BEFORE starting any DevTest component for the first time after the upgrade.

5. Clear your browser cache.
6. Upgrade any related machine to ensure that all Server components and the workstation are on the same release.
7. Start Identity and Access Manager.
8. Start server components: Enterprise Dashboard, registry, Portal, VSE, coordinator, simulator.

> **Note:** The first time that you start a registry after the upgrade, the authentication information that was stored at the registry level is automatically migrated to Identity and Access Manager.

9. Verify the activation for each registry by ensuring that each registry appears correctly in the Enterprise Dashboard.
10. Verify that your users and roles are appearing correctly in Identity and Access Manager.

> **Note:** If the same user name exists in two registries with different roles in each registry, the roles from both registries are added to that user in Identity and Access Manager.

When the upgrade is complete, verify that the upgrade is successful by viewing the After You Upgrade documentation.

After You Upgrade

For information about the order in which to start the services and applications, see Verifying the DevTest Installation. After you perform the upgrade, review the following information.

- Verify the Authentication Migration
- Patches Folder
- Verify Services and Applications
- Open DevTest Workstation

Verify the Authentication Migration

Identity and Access Manager automatically migrates all of your existing authentication information from earlier versions of DevTest. After you have completed the upgrade and started all DevTest components, it is important to verify the migration.

Start the Identify and Access Manager UI (https://localhost:51111) and confirm that your users and roles were successfully migrated.

You can also use the reports that are generated by the migration process for a more detailed analysis of the data that was migrated. These reports located in your **lisatmp** directory. For more information about these reports, see Upgrading the Authentication Component.

For more information about using Identity and Access Manager, see Identity and Access Manager .

Patches Folder

If the existing installation contained a **LISA_HOME\lib\patches** folder, the name of the folder is changed to **patches.backup**. You should not need to apply any of the patches to the new installation.

Verify Services and Applications

- Verify database connectivity.
- Verify Enterprise Dashboard connectivity.
- Verify Portal connectivity.
- Verify DevTest Workstation connectivity.
- Verify component connectivity.
- Deploy a virtual service.
- Verify the deployed virtual service.
- Start a test case.
- Verify the test case result

Open DevTest Workstation

If you plan to use DevTest Workstation, refer to Open DevTest Workstation, Step 4.

Click here for additional details around the upgrade process.

Additional Upgrade Details

This section provides details on how to perform an *in-place upgrade* of DevTest Solutions. For an *in-place upgrade*, run the installer for DevTest 10.4 and specify the same directory as the existing installation.

This section assumes that the earlier version is 8.0 or later.

Important! If you have any questions, contact CA Support *before* you attempt the upgrade.

Product Changes

For information about how the functionality has changed since the version of your existing installation, see the Release Notes.

Licensing

The **devtestlic.xml** file contains the license details.

> **Note:** To upgrade from DevTest 10.3 or earlier, you must request a new license file. For more information, see License Administration and How License Activation Works.

Enterprise Dashboard and Registry

When you do an in-place upgrade from DevTest 8.0 or later, the schema of the Enterprise Dashboard database is updated and the existing data is maintained.

For both registry and Enterprise Dashboard, you do not need to create a new database. You cannot change the type of database during the upgrade process. For example, you cannot switch from Oracle to SQL Server.

The database user must have DBA privileges. Otherwise, the Enterprise Dashboard and registry will fail to start after the upgrade. A permissions check is performed at the beginning of the database upgrade. This checks the appropriate permission level to avoid making modifications to the database.

> To remain in compliance with your license agreement, you must retain all usage data that is in your Enterprise Dashboard database.

The installer does not upgrade the registry database. After finishing the upgrading processes by the installer, the first invocation of the registry will upgrade the registry database.

> **Note:** If the database is terminated abnormally by the user or by the system, the registry will check for the last installed version on the next database start. This information might not be available due to the abnormal termination. The way to resolve this is to either restart the database and the initialization or disable the migration check in the LISA properties file.

- In DevTest 9.1 and later, Enterprise Dashboard properties are kept in the **dradis.properties** file. To avoid confusion in the future, consider removing the "dradis" properties from your **local.properties** and **site.properties** files, as they no longer impact Enterprise Dashboard processing.

- With DevTest 9.1 and later, configuration of the Enterprise Dashboard URL has changed. Therefore, the parameters that are passed when starting a registry must change also, to something similar to this example:

```
'docker run -d -p 1505:1505 –p 1528:1528 –p 2010:2010 --name=Registry -e
LISA_MORE_VM_PROPS="-Ddevtest.enterprisedashboard.host=10.141.66.40 -Ddevtest.
enterprisedashboard.port=1506 -Ddevtest.enterprisedashboard.https.enabled=false"
devtest/registry-broker-base /opt/devtest/bin/Registry -n tcp://10.141.66.40:2010
/Registry'.
```

For details, see Reactivate a Registry or Enterprise Dashboard.

- To support pre-9.1 DevTest registries, the EnterpriseDashboardCIC service provides backwards compatibility and must be started (on an image where dradis-CIC is available). Use a command similar to the following command:

  ```
  docker run –d –p 2003:2003 --name=dradisCIC devtest-licensed:latest /opt
  /devtest/bin/EnterpriseDashboardCIC
  ```

- If the EnterpriseDashboardCIC service is running under Docker, see Docker Containers.

- For more details on backward compatibility with older registries, see Maintain Registries.

Using DDL Scripts to Upgrade Enterprise Dashboard

After you upgrade from a previous DevTest version, you will have DevTest and Enterprise Dashboard schema upgrade scripts in the **LISA_HOME\database\upgrade** folder with all supported databases for previous releases. For example, if you upgrade from 9.0.0 to 10.2.0, you see ten DDL files.

- db2_enterprisedashboard_upgrade_9.0.0.ddl
- db2_upgrade_9.0.0.ddl
- derby_enterprisedashboard_upgrade_9.0.0.ddl
- derby_upgrade_9.0.0.ddl
- mysql_enterprisedashboard_upgrade_9.0.0.ddl
- mysql_upgrade_9.0.0.ddl
- oracle_enterprisedashboard_upgrade_9.0.0.ddl
- oracle_upgrade_9.0.0.ddl
- sqlserver_enterprisedashboard_upgrade_9.0.0.ddl
- sqlserver_upgrade_9.0.0.ddl

To generate upgrade DDLs for other DevTest versions, you must execute with a command-line utility.

- For DevTest schema upgrade scripts, execute:

  ```
  Java.exe -jar C:\DevTest\lib\core\lisa-core-10.2.0.jar -ud 9.0.0
  ```

  This code creates registry upgrade scripts from release 9.0.0 under the current folder.

To specify a destination folder, add more arguments as follows.

```
Java.exe -jar C:\DevTest\lib\core\lisa-core-10.2.0.jar -ud 9.0.0 C:\ddl
```

- For Enterprise Dashboard schema upgrade scripts, execute:

  ```
  Java.exe -jar C:\DevTest\lib\dradis\dradis-10.2.0.jar -ud 9.0.0
  ```

  This code creates Enterprise Dashboard upgrade scripts from release 9.0.0 under the current folder.

  To specify a destination folder, add more arguments as follows.

  ```
  Java.exe -jar C:\DevTest\lib\dradis\dradis-10.2.0.jar   -ud   9.0.0  C:\ddl
  ```

Mixing Component Versions

The registry and all the underlying components, including DevTest Workstation, must belong to the same release. Mixing component versions under a registry is not supported.

The Enterprise Dashboard is backward compatible through version 8.0 and supports connections from registries of multiple versions. However, the Enterprise Dashboard itself must be the most recent version. For example, Enterprise Dashboard 10.2 can accept connections from 8.0 and 8.4 registries. Enterprise Dashboard 8.4 *cannot* accept connections from a 10.2 registry.

Property Files

The following list describes what happens to the main property files in an in-place upgrade:

- The **lisa.properties** file of the existing installation is replaced with the 10.4 version.
- The **_local.properties** file of the existing installation is replaced with the 10.4 version.
- The **_site.properties** file of the existing installation is replaced with the 10.4 version.
- The **_dradis.properties** file of the existing installation is replaced with the 10.4 version.
- If you created a **dradis.properties** file, the file is preserved as is.
  If you are upgrading from a version that predates the dradis.properties file, a **_dradis.properties** file is created. Any customized properties for Enterprise Dashboard in your **local.properties** or **site.properties** file are migrated to a **dradis.properties** file.
- If you created a **local.properties** file, the file is preserved as is.
- If you created a **site.properties** file, the file is preserved as is.

You are not supposed to modify the **lisa.properties** file. If you did make changes to the **lisa.properties** file, the upgrade process overwrites the changes.

Custom Folders

If the existing installation contains any custom folders, the folders are preserved as is.

Custom JAR Files

If the existing installation contains any custom JAR files in the **LISA_HOME\lib** directory, the JAR files are preserved as is.

Memory Allocation Settings

If the existing installation contains any modified **.vmoptions** files in the **LISA_HOME\bin** directory, the files are preserved as is.

Database Safeguards

A permissions check is performed at the beginning of the database upgrade. This checks the appropriate permission level to avoid making modifications to the database.

> **Note**: If the database is terminated abnormally by the user or by the system, the registry will check for the last installed version on the next database start. This information might not be available due to the abnormal termination. The way to resolve this is to either restart the database and the initialization or disable the migration check in the LISA properties file.

Migrating the SDK

Upgrading SDK Components

Using the SDK covers the most common API changes in the most common extensions. This guide is not intended to be comprehensive for everything that can possibly be in an SDK extension. Rather, this guide highlights the SDK API history, including mandatory changes that are required for forward compatibility.

This document is intended for an audience who has studied the DevTest product documentation *and* obtained DevTest training from CA. Mastery of previous versions expedites the DevTest learning curve. However, only proper DevTest education can prepare you to enjoy a smooth transition and realize maximum value.

We highly recommend that you engage professional services to assist with the DevTest upgrade. Contact your account manager for more details.

> **Note:** All SDK components must be compiled with Java 8 to run in DevTest 10.2.

This guide is organized by component. It provides the relevant history of each component, starting with LISA 4.0 or the version in which the component was introduced.

The most effective way to use this guide is as follows:

1. Identify the section for your component type.
2. Identify the subsection for your current version.

3. Follow the notes on the API changes through the latest version and update your code as required.

SDK Documentation

The following SDK documentation is available:

- Using the SDK
- JavaDocs for DevTest Solutions are available in the **doc** folder of your installation directory.

Java Version Notes

DevTest 9.0 or later is supported in Java 8. All SDK components must be compiled with Java 8. Most relevant changes to the core Java language are backwards compatible.

- DevTest 8.0 was supported in Java 7.
- LISA 7.5 was supported in Java 7.
- LISA 4.0 was supported on Java 5.
- LISA 4.5 was supported on Java 5.
- LISA 4.6 was supported on Java 5.
- LISA 5.0 was supported on Java 5.
- LISA 6.0 was supported on Java 6.
- LISA 6.1 was supported on Java 7.
- LISA 7.0 was supported on Java 7.
- LISA 7.1 was supported on Java 7.

For more information about core language changes see the following web pages:

- Java 6 Features and Enhancements: http://www.oracle.com/technetwork/java/javase/features-141434.html
- Java 6 Release Notes: http://www.oracle.com/technetwork/java/javase/releasenotes-136954.html
- Java 7 Features and Enhancements: http://www.oracle.com/technetwork/java/javase/jdk7-relnotes-418459.html
- Java 7 Release Notes: http://www.oracle.com/technetwork/java/javase/7u-relnotes-515228.html

Assertion Updates

SDK Assertion components extend **com.itko.lisa.test.Assertion**.

- 8.X to 9.X
- 9.X to 10.X

8.X to 9.X

The **com.itko.util.ParameterList** class was deprecated in release 9.5. For more information, see Parameters and Parameter Lists.

9.X to 10.X

No changes that affect SDK components.

Companion Updates

SDK Companion components extend **com.itko.lisa.test.SimpleCompanion**.

- 8.X to 9.X
- 9.X to 10.X

8.X to 9.X

The **com.itko.util.ParameterList** class was deprecated in release 9.5. For more information, see Parameters and Parameter Lists.

9.X to 10.X

No changes that affect SDK components.

Filter Updates

SDK Filter components extend **com.itko.lisa.test.FilterBaseImpl**.

- 8.X to 9.X
- 9.X to 10.X

8.X to 9.X

The **com.itko.util.ParameterList** class was deprecated in release 9.5. For more information, see Parameters and Parameter Lists.

9.X to 10.X

No changes that affect SDK components.

Step Updates

Simple SDK Steps or Nodes implement the **com.itko.lisa.test.CustJavaNodeInterface**.

- 8.X to 9.X
- 9.X to 10.X

8.X to 9.X

The **com.itko.util.ParameterList** class was deprecated in release 9.5. For more information, see Parameters and Parameter Lists.

9.X to 10.X

No changes that affect SDK components.

Data Protocol Updates

Data Protocols became extensible in LISA 5.0 and extend **com.itko.lisa.vse.stateful.protocol.DataProtocol**.

- 8.X to 9.X
- 9.X to 10.X

8.X to 9.X

No changes that affect SDK components.

9.X to 10.X

On **com.itko.lisa.vse.stateful.DelayedResponder**:

• **invokeLater(TestExec testExec, Runnable action, long delay)** was deprecated in 8.0.0, replaced by **invokeLater(final TestExec testExec, final Runnable action, final long delay, final long memoryUsed)**. To do the equivalent of the deprecated method, pass 0 as the last parameter.

Transport Protocol Updates

Extensible Transport Protocols were introduced in LISA 5.0 and extend **com.itko.lisa.vse.stateful.protocol.TransportProtocol**.

- 8.X to 9.X
- 9.X to 10.X

8.X to 9.X

No changes that affect SDK components.

9.X to 10.X

No changes that affect SDK components.

Body Carrier Updates

com.itko.lisa.vse.stateful.model.BodyCarrier was introduced in LISA 6.0. This is a base class for objects that need to have both meta data and a body, such as com.itko.lisa.vse.stateful.model.Request, com.itko.lisa.vse.stateful.model.Response, or com.itko.lisa.vse.stateful.model.TransientResponse.

- 8.X to 9.X
- 9.X to 10.X

8.X to 9.X

No changes that affect SDK components.

9.X to 10.X

No changes that affect SDK components.

Request Updates

com.itko.lisa.vse.stateful.model.Request represents the request side of a VSEtransaction.

- 8.X to 9.X
- 9.X to 10.X

8.X to 9.X

No changes that affect SDK components.

9.X to 10.X

No changes that affect SDK components.

Response Updates

com.itko.lisa.vse.stateful.model.Response represents the response side of a VSE transaction during recording.

- 8.X to 9.X
- 9.X to 10.X

8.X to 9.X

No changes that affect SDK components.

9.X to 10.X

No changes that affect SDK components.

Transient Response Updates

com.itko.lisa.vse.stateful.model.TransientResponse represents the response side of a VSEtransaction during playback.

- 8.X to 9.X
- 9.X to 10.X

8.X to 9.X

No changes that affect SDK components.

9.X to 10.X

No changes that affect SDK components.

TCP Delimiter Updates

TCP Delimiters were introduced with the TCP Protocol in LISA 6.0. They implement the **com.itko.lisa.vse.stateful.protocol.tcp.delimiters.TCPDelimiter** interface.

- 8.X to 9.X
- 9.X to 10.X

8.X to 9.X

No changes that affect SDK components.

9.X to 10.X

No changes that affect SDK components.

API Notes

- 8.X to 9.X
- 9.X to 10.X

8.X to 9.X

No changes that affect SDK components.

9.X to 10.X

No changes that affect SDK components.

Getting Started

This section contains an overview of DevTest Portal, tutorials for each functional area of DevTest Solutions, and a glossary.

- Role in the Continuous Delivery Ecosystem
- DevTest Portal
- CA Application Test Tutorials
- CA Service Virtualization Tutorials
- Application Insight Tutorial
- Glossary

DevTest Solutions consists of the following products:

CA Application Test

- Provides an automated testing solution for distributed application architectures.
- Allows product teams to design and execute automated unit, functional, regression, integration, load, and performance tests for multiple layers of a distributed architecture.
- Allows automated testing at the UI layer and for the headless services behind the UI that provide business logic and data to the application.

CA Service Virtualization

- Removes constraints throughout the SDLC by modeling and simulating unavailable or dependent systems.
- Enables parallel development and testing to reduce cycle times, detect defects early and increase IT productivity.
- Reduces demand for lab infrastructure and software to avoid costs and reduce configuration effort.
- “Shifts quality left” for higher performance and less risk.

Role in the Continuous Delivery Ecosystem

CA offers a collection of integrated software planning, development, testing, and delivery tools that create a complete Continuous Delivery Ecosystem. This ecosystem helps you overcome the complexities and obstacles of these new demands and puts innovation in the hands of your customers faster, at lower cost, and with reduced risk. The strategic integration points between these tools help you manage your entire software development lifecycle, end-to-end.

DevTest Solutions is one product in the ecosystem. DevTest Solutions integrates with other products as follows:

- **CA Application Test with CA Agile Central**
  Lets you view and manage the test cases that you create in CA Application Test in CA Agile Central.

- **CA Application Test with CA Agile Requirements Designer**
  Lets you import an existing test from CA Application Test into CA Agile Requirements Designer.

- **CA Service Virtualization with CA Agile Requirements Designer**
  Helps you generate rich data for service virtualization.

- **CA Service Virtualization with CA Test Data Manager**
  Lets you generate realistic virtual data that covers the full range of possible scenarios for effective service virtualization.

- **CA Service Virtualization with CA API Management**
  Lets you capture the request and response data for APIs that CA API Gateway manages.

- **CA Service Virtualization with CA Automic Release Automation**
  Lets you deploy, start, and stop virtual services from CA Automic Release Automation as part of your deployment process.

The following videos demonstrate how the products in the Continuous Delivery Ecosystem work together.

For more information, see the DocOps site for the Continuous Delivery Ecosystem:
https://docops.ca.com/ca-continuous-delivery/4-0/en

DevTest Portal

DevTest Portal is a web-based application that provides simpler access to the most commonly used workflows for DevTest products. DevTest Workstation is the main application for advanced users and contains the full range of functionality available within the DevTest products.

This page describes the available functionality in DevTest Portal. For a quick summary, see DevTest Portal Functionality.

This page also describes the **All Resources** window and how to manage projects and tabs.

- Open DevTest Portal
- Navigating DevTest Portal
- All Resources Window
- Manage Projects
- Manage Tabs

Open DevTest Portal

You open DevTest Portal from a web browser.

Note: For information about the server components that must be running, see Verifying the DevTest Installation.

One possible error message indicates that the user cannot be authenticated and that you should make sure the registry service is running. If you receive this message and the registry service is running, the cause might be a slow network. Analyze your network for impaired performance.

Follow these steps:

1. Complete one of the following actions:
   - Open a supported browser and enter **http://localhost:1507/devtest**.
     If the registry is on a remote computer, replace **localhost** with the name or IP address of the remote computer.
     If the port number was changed from the default value **1507**, use the new port number.
   - Select **View**, **DevTest Portal** from DevTest Workstation.
2. Enter your user name and password.
3. Click **Log in**.

Navigating DevTest Portal

When you log in, DevTest Portal displays a header area, a navigation menu, and a home page.

The following graphic shows these components.

Screen capture of DevTest Portal

Navigation Menu

The navigation menu appears on the left.

The icon in the left portion of the header area lets you collapse, hide, and restore the navigation menu.

The navigation menu contains the following sections:

- **Home**
  Display the home page.

- **Create**
  Create API tests and virtual services.

  - **API Test**

  - **Virtual Service**

  - **Copybook Bundle**

  > **Tip:** When you create artifacts, they are saved into the project that is shown in the **Current Project** drop down. You can select another project before you create.

- **Manage**
  Manage API tests, test cases, test suites, and virtual services.

  > **Tip:** When you manage artifacts, they are chosen from the project that is shown in the **Current Project** drop down. You can select another project.

- - **All Resources**

  - **API Tests**

  - **Tests**

  - **Test Suites**

  - **Virtual Services**

  - **Copybook Bundles**

  - **Data Sets**

- **Monitor**
  Monitor tests and virtual services.

  - **Tests**

  - **Virtual Service Environments**

  - **Server Health**

  - **CVS**

- **Application Insight**
  Analyze and document transactions with the Application Insight feature. You can also manage tickets.

    - **Analyze Transactions**
    - **Document Transactions**
    - **Manage Tickets**
- **Reporting**
  View and filter reports.
    - **CAI Top N and Metrics**
    - **Testing**
    - **Virtual Service Metrics**
    - **Administration**
- **Settings**
  Configure agents and access control.
    - **Agents**
    - **Access Control**
        - **Roles**
        - **Users**
        - **Resource Groups**
        - **Resources**

Home Page

The home page contains a dashboard of portlets that let you monitor the activity of your application. Each portlet displays information of a DevTest component that can be manually refreshed.

> **Note:** The first time that you click a Getting Started link, a login dialog opens. Your login to view the documentation persists throughout your session on DevTest Portal.

The dashboard contains the following portlets:

- **Getting Started**
  Displays links to documentation pages that describe key functions of the portal.
- **Quick Links**
  Displays links to key functions of the portal.
- **Help**
  Displays a link to the DevTest documentation, to ask a question to the DevTest Community, and to report an issue.

- **Virtual Services: Current Status**
  Displays the current total count of transactions, the number of virtual services that are deployed, and the number of virtual services that are recorded.

- **Test: Current Status**
  Displays the available test results. Clicking **Show More...** navigates you to the **Monitoring Test** window.

- **Agents: Current Status**
  Displays the status for an agent. Selecting an agent name or **Show More...** navigates you to the agent management operations in the **Agents** window. To view the current agent information, manually refresh the portlet.

- **New Ticket Alerts**
  Displays a list of new tickets and a count of new, identified, or closed tickets. The name, date, and time of the ticket displays in the list. Selecting the ticket, **New**, **Identified,** or **Closed** navigates you to **Manage Tickets** window. The **Manage Tickets** window lets you view tickets, update existing tickets, view a list of tickets, and search for tickets in the Application Insight database.

- **Path Alerts**
  Displays a list of paths with exception flags. Selecting a path navigates you to the **Analyze Transactions** window.

  Create a ticket by clicking **Create Ticket** , entering or selecting the field options, and click **Submit**.The ticket displays in the **Manage Tickets** window and in the **New Tickets Alerts** pane of the **Home** page.

- **Points of Interest**
  Displays a list of pinned transactions. Selecting the name in the list of **Points of Interest** navigates you to the **Analyze Transactions** window.

All Resources Window

The **Manage, All Resources** window lets you manage multiple projects and their resources. The following resources are available:

- API tests
- Test cases
- Test suites
- Virtual services
- Copybook bundles
- Data sets

The resources appear in a table.

You can show and hide columns. By default, only the **Last Updated** column is hidden.

You can change the number of resources per page. The default setting is ten resources.

To display only one type of resource, click the resource type following **All Resources** in the left navigation menu. Each resource type opens in a new tab.

Manage Projects

You can manage projects from DevTest Portal.

To add a project:

1. Click **Manage Projects** .
   The **Manage Projects** window appears.
2. Click **Add Project**.
3. Enter the project name.
4. (Optional) Enter a description.
5. Click **OK**.

To delete a project:

1. Click Manage Projects .
   The **Manage Projects** window appears.
2. Click **Delete** next to the project you want to delete.
3. Click **Yes**.

> **Tip:** To delete multiple projects simultaneously, select more than one project and click **Delete Projects**.

To copy a project:

1. Click Manage Projects .
   The **Manage Projects** window appears.
2. Click **Copy** next to the project you want to copy.
3. Update the project name.
4. Click **OK**.

To rename a project or update a description:

1. Click Manage Projects .
   The **Manage Projects** window appears.

2. Click **Rename** next to the project you want to update.

3. Update the project name or description.

4. Click **OK**.

To upload the contents of a MAR file into a project:

1. Click Manage Projects .
   The **Manage Projects** window appears.

2. Click **Upload archive to the project** next to the project you want to update.

3. Drag and drop a MAR file or click **Upload MAR** to select a MAR file.

4. (Optional) Select the **Force to merge conflicted files** check box to indicate that duplicate files from the imported MAR should override files of the same name that are already in the project.

5. Click **OK**.

6. A window displays a list of all resources for which the upload failed.

> **Note:** You can select and upload one MAR file at a time. If you upload the wrong MAR file, remove it and select the correct one.

When uploading a MAR to a project, if the MAR contains any entries that do not belong to any DevTest project folders in the MAR, they are considered as unusable files, and they are not uploaded.

Manage Tabs

You can manage multiple tabs by changing the order that they appear or by closing them. The **Home** tab cannot be closed when it is the only open tab. If you close all tabs, the **Home** tab opens.

The following options for closing tabs are available:

- **Close tab**
  This option closes the selected tab.

- **Close all other tabs**
  This option closes all the other tabs.

- **Close tabs to the right**
  This option closes the tabs to the right of the selected tab.

Follow these steps:

1. To move a tab, drag-and-drop the tab to the new location.
2. To close tabs, right-click a tab and select an option.

Customize the Home Page Dashboard

Customize the home page dashboard by changing the layout, resizing, removing, and adding portlets. The title of the portlets and the Test filter options can be edited.

Follow these steps:

1. Go to the **Home** tab.
2. To change the layout, click **Enable edit mode** , click **Change location** and hold, then drag-and-drop the portlet into the desired position.
3. To change the title, click **Configure** , enter the title, and click **Save**.
4. (Optional) To change filter options for the **Test: Current Status** portlet, select a userid from the **Executed by** drop-down or select a timeframe from the timeframe drop-down, click **Save**, and click **Close**.
5. To remove a portlet, point to **Add or remove dashboard content** and clear the desired check box or click **X** in the portlet.
6. To add a portlet, point to **Add or remove dashboard content** and select the check box.
7. Point outside of the popup to close the popup.
8. To reset to the default layout, click **Reset to default configuration** .
9. To undo the changes, click **Undo changes** .
10. To save the changes, click **Save changes** .

Update Preferences

You can set preferences to view help tips and select a default language from the **Preferences** dialog. You access this dialog by clicking the down arrow next to the user name in DevTest Portal.

- Change Your Password
- Enable and Disable Help Tips
- Select Default Language

Change Your Password

Follow these steps:

1. Select **Set Password** from the drop-down list of the down arrow.
2. Enter your old and new passwords.
3. Click **Save**.

Enable and Disable Help Tips

DevTest Portal provides tips on how to perform key tasks. You can enable and disable the help tips.

Follow these steps:

1. Select **Preferences** from the drop-down list of the down arrow.
2. To enable the help tips, select the **Enable help for first-time user** check box and click **OK**.
3. To disable the help tips, do one of the following steps:

- - From the **Preferences** dialog, clear the **Enable help for first-time user** check box and click **OK**.
  - From the help tip, click **X** and select **Yes** to remove all help tips.
  - From the help tip, click **X** and select **No** to remove the individual help tip.

Select Default Language

DevTest Portal lets you select the default language of the portal and the default project.

> **Note:** When using a browser with an automatic translation capability, you must set the language in the **Preferences** dialog.

Follow these steps:

1. Select **Preferences** from the drop-down list of the down arrow.
2. S elect a language from the **Language** drop-down list.
3. Click **OK**.

CA Application Test Tutorials

This section contains a series of tutorials that illustrate various aspects of CA Application Test. The tutorials are sequential. Complete them in the order presented.

The first few tutorials walk you through using DevTest Workstation to build simple test cases. You become familiar with basic concepts such as projects, properties, data sets, filters, and assertions.

The subsequent tutorials help you acquire deeper knowledge about how to set up test steps to interact with and test several common technologies. These technologies include Java objects, web pages, Enterprise JavaBeans (EJBs), web services, and databases. You also learn how to stage a quick test.

To perform the tutorials, you must have DevTest Workstation installed and you must have access to a registry.

Some tutorials use the Demo Server as the system under test. For information about installing the Demo Server, see Installing.

- Tutorial 1 - Projects, Test Cases, and Properties
- Tutorial 2 - Data Sets
- Tutorial 3 - Filters and Assertions
- Tutorial 4 - Manipulate Java Objects (POJOs)
- Tutorial 5 - Run a Demo Server Web Application
- Tutorial 6 - Test a Website
- Tutorial 7 - Test an Enterprise JavaBean (EJB)
- Tutorial 8 - Test a Web Service
- Tutorial 9 - Examine and Test a Database
- Tutorial 10 - Stage a Quick Test
- Tutorial 11: Run the Forward Cars Demo Application

Tutorial 1 - Projects, Test Cases, and Properties

Contents

- Step 1 - Start DevTest Workstation
- Step 2 - Create a Project
- Step 3 - Create a Test Case
- Step 4 - Add a Property to the Project Configuration
- Step 5 - Add a Test Step
- Step 6 - Add a Log Message
- Step 7 - Add a Second Log Message
- Step 8 - Run the My Output Log Message Step
- Step 9 - Observe Property Values
- Step 10 - Run the Output Log Message Step
- Tutorial 1 - Review

Tutorial Tasks

In this tutorial, you:

- Create a project
- Create a test case
- Add properties
- Add simple test steps

- Use the Interactive Test Run Utility

Prerequisites

- DevTest Workstation is installed and DevTest license credentials are entered.
- You have reviewed the Glossary.

Step 1 - Start DevTest Workstation

Follow these steps:

1. Start the registry.
   - If your computer has DevTest Server installed:
     a. Start the Enterprise Dashboard by selecting **DevTest Solutions**, **Enterprise Dashboard Server** from the list of programs in your Windows Start menu. Wait until the "Grails application running" message appears.
     b. Start the registry by selecting **DevTest Solutions**, **Registry** from the list of programs in your Windows Start menu.
   - If your computer has DevTest Workstation installed, use a registry that is running on another computer.
2. Select **DevTest Solutions**, **Workstation** from the list of programs in your Windows Start menu.
3. When the **Set DevTest Registry** dialog opens, select a registry and click **OK**.
4. The **Login** dialog opens. Enter a valid username and password and click **Login**.

Step 2 - Create a Project

The project that you create holds all the test case example files that are required for the tutorials.

Follow these steps:

1. From the DevTest Workstation main menu, select **File**, **New**, **Project**.
   The **Create New Project** dialog opens.
2. In the **Project Name** field, remove the default value and type **My Tutorials**.
3. Click **Create**.
   The My Tutorials project is created.

Step 3 - Create a Test Case

A test case is a specification of how to test a business component in the system under test.

Follow these steps:

1. In the **Project** panel, right-click the **Tests** folder and select **Create New Test Case**.
2. Set the file name to **tutorial1**.

3. Click **Save**.
   DevTest Workstation opens a new tab labeled tutorial1. The green arrow in the model editor represents the start of the test case.

Step 4 - Add a Property to the Project Configuration

In this step, you set a global property in the project configuration. You access this property later in the tutorial.

The default configuration has the name **project.config**, and is created automatically for a new project. The project.config file is located in the **Configs** folder in the **Project** panel. The file extension is not shown. You can add the properties to the project.config file and, if necessary can also create a configuration file.

Follow these steps:

1. In the **Project** panel, double-click **project** in the My Tutorials > **Configs** folder.
   The properties editor opens.
2. To add a row, click **Add** at the bottom of the properties editor.
3. In the **Key** field, type **config_prop**.
4. In the **Value** field, type **42**.
5. From the main toolbar, click **Save**.

Step 5 - Add a Test Step

A test case includes one or more test steps. In this procedure, you add an Output Log Message test step to write text to the log file.

Follow these steps:

1. Click the tutorial1 tab.
2. Click **Add Step**, select **Utilities**, and select **Output Log Message**.
   A step named Output Log Message is added to the model editor.
3. Right-click the Output Log Message step and select **Rename**.
4. Change the name to My Output Log Message.
5. Make sure that My Output Log Message is still selected. In the right pane, click the arrow next to Output Log Message.
   The Output Log Message tray opens.

Step 6 - Add a Log Message

With the log editor open, you add a log message that includes various properties.

The properties in the log message originate from several sources:

- The LISA_HOME property is automatically set.
- The java.version property is a system property.

- You added the config_prop property to the project configuration in Step 4.
- You create a property with the name **MyOutputLogMessage_step_prop** in the log message itself.

The syntax for a property is **{{property_name}}**.

Follow these steps:

1. In the log editor, delete the placeholder text.
2. Copy and paste the following text into the log editor:

```
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 co
nfiguration.
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 va
lue of config_prop.
Create a new property named MyOutputLogMessage_step_prop: {{MyOutputLogMessa
ge_step_prop=100}}.
The MyOutputLogMessage_step_prop property has been assigned the value 100.
```

The log editor looks like the following graphic.

Output Log Message example for Tutorial 1

Step 7 - Add a Second Log Message

The second test step in the test case writes a different message to the log file.

Follow these steps:

1. Close the first log message step by clicking the arrow at the upper left corner of the window.
2. Click **Add Step**, select **Utilities**, and select **Output Log Message**.
   A step with the name **Output Log Message** is added to the model editor and the **Output Log Message** tray opens.
3. In the log editor, delete the placeholder text.
4. Copy and paste the following text into the log editor:

```
The current value of config_prop is: {{config_prop}}.
The current value of MyOutputLogMessage_step_prop: {{MyOutputLogMessage_step
_prop}}.
```

> **Note:** The log message does not change the values of **config_prop** or **MyOutputLogMessage_step_prop**.

5. Close the second log message step by clicking the arrow at the upper left corner of the window.

6. From the main application toolbar, click **Save**, or select **File**, **Save**, tutorial1.

Step 8 - Run the My Output Log Message Step

The Interactive Test Run (ITR) utility enables you to walk through and verify a test case.

Follow these steps:

1. From the toolbar, click **Start ITR** .
   The ITR opens. The ITR contains an **Execution History** pane on the left and a set of tabs on the right.

2. In the **Execution History** pane, click **Execute Next Step**.
   The My Output Log Message step is run. The **Response** tab displays the response from the My Output Log Message step. The actual values replace the properties.

Step 9 - Observe Property Values

The ITR also lets you observe how the properties are created and modified.

Follow these steps:

1. Click the **Properties** tab in the ITR.
   The **Properties** tab displays the value of each property before and after the execution of the My Output LogMessage step. A value that the step created is highlighted in green. A value that was modified in the step is highlighted in yellow. Notice that the value of config_prop was changed from 42 to 21.

2. Compare these values with the response in Step 8.

Step 10 - Run the Output Log Message Step

In this procedure, you use the ITR to run the second step in the test case.

Follow these steps:

1. In the ITR, click **Execute Next Step** to run the Output Log Message step.

2. To view the response, click the **Response** tab. Although you set config_prop to 42 in the project.config file, you changed the value to 21 in the My Output Log Message step, and the value did not change in the Output Log Message step. The value of the **MyOutputLogMessage_step_prop** property also carried over from the My Output Log Message step to the Output Log Message step.

3. To view the current and previous property values, click the **Properties** tab.

4. When you are done, close the tutorial1 and project tabs.

Tutorial 1 - Review

In this tutorial, you took a first look at properties. You saw that properties are denoted by using a special syntax, {{property_name}}. You can set properties by 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:

- Learned how to create and save a 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

Tutorial 2 - Data Sets

Contents

- Step 1 - Create a Data Set
- Step 2 - Create a Test Case
- Step 3 - Add a Property to the Project Configuration
- Step 4 - Add a Test Step for Output Log Message
- Step 5 - Create Another Output Log Message Step
- Step 6 - Execute the Test
- Step 7 - Add the Data Set
- Step 8 - Change the Data Set Behavior
- Tutorial 2 - Review

In this tutorial, you learn how to create and use a simple data set. You also learn how to provide the data in a data set to a test case.

Tutorial Tasks

In this tutorial, you will:

- Create a simple data set
- Use the data set in various ways
- Iterate through a series of test steps using a data set

Prerequisites

- You have completed Tutorial 1 - Properties.
- DevTest Workstation is open.

Step 1 - Create a Data Set

In this tutorial, you use a comma-delimited text file as the data set. This option is only one of several options available to create a data set. After you create the text file, you import it into the My Tutorials project.

Follow these steps:

1. In a text editor such as Notepad, create a text file.

2. Copy and paste the following properties and values into the text editor. Do not use spaces in the text file.

   ```
   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
   ```

   The first row specifies the names of the properties to which this data is assigned (month, day, year). The remaining rows specify the data that is read and used in the test case. Two of the rows include a property with the name **yearglobal**.

3. Save the file as dates.txt.

4. In the **Project** panel, right-click the **Data** folder in the My Tutorials project and select **Import Files**.

5. Navigate to the folder where you saved the dates.txt file and select the file name.

6. Click **Open**. The dates.txt file now appears in the **Data** folder.

Step 2 - Create a Test Case

You add a test case to the My Tutorials project.

Follow these steps:

1. In the **Project** panel, right-click the **Tests** folder and select **Create New Test Case**.

2. Make the file name **tutorial2**.

3. Click **Save**.

Step 3 - Add a Property to the Project Configuration

The dates.txt file includes a property with the name **yearglobal**. In this procedure, you add the yearglobal property to the project configuration.

Follow these steps:

1. In the **Project** panel, double-click project.config.

2. Click **Add** to add a row.

3. In the **Key** field, enter yearglobal.

4. In the **Value** field, enter 1999.

5. Click **Save**.

Step 4 - Add a Test Step for Output Log Message

To write text out to the log, use a test step, the Output Log Message step.

Follow these steps:

1. Click the tutorial2 tab.

2. Click **Add**.
   The **Add step** menu is displayed.

3. Select **Utilities** and select Output Log Message.
   A step with the name **Output Log Message** is added to the model editor.

4. Right-click **Output Log Message** and select **Rename**. Change the name to DSstep1.

5. In the right pane, click the arrow next to Output Log Message.
   The **Output Log Message** tray opens.

6. Delete the placeholder text.

7. Enter the following log message:

   ```
   Date is: {{month}}/{{day}}/{{year}}
   ```

   Note: The curly brackets are important. The test case runs correctly only if they are included.

8. To close the Output Log Message tray, click anywhere in the model editor.

9. Click **Save**.

Step 5 - Create Another Output Log Message Step

Create another test step similar to the DSstep1 test step.

Follow these steps:

1. Click **Add**.
   The **Add step** menu is displayed.

2. Select **Utilities** and select **Output Log Message**.
   A step with the name **Output Log Message** is added to the model editor.

3. Right-click Output Log Message and select **Rename**. Change the name to DSstep2.

4. In the right pane, click the arrow next to Output Log Message.
   The **Output Log Message** tray opens.

5. Delete the placeholder text.

6. Enter the following log message:

   ```
   Date is: {{month}}/{{day}}/{{year}}
   ```

7. To close the **Output Log Message** tray, click anywhere in the model editor.

8. Click **Save**.

Step 6 - Execute the Test

To execute the test and see what happens, use the Interactive Test Run (ITR).

Follow these steps:

1. From the toolbar, click **Start ITR**.
   The ITR opens.

2. In the **Execution History** pane, click **Automatically execute test** .

3. When the test is complete, click **OK**.

4. In the **Execution History** pane, click DSstep1 and DSstep2.
   Notice that the month, day, and year properties have not been replaced with actual values. This result is expected, because you have not added the data set to the test case.

Step 7 - Add the Data Set

You now add the dates.txt data set to the DSstep1 test step.

Follow these steps:

1. In the model editor, select DSstep1.

2. In the right pane, double-click the **Data Sets step** tab.

3. Click **Add** below the Data Sets element.

4. From the Common DataSets list, select **Read Rows from a Delimited Data File**.
   The data set is added to the test step.
   The data set editor opens in the right pane.

5. In the data set editor, set the name to DatesDS.

6. Click **File Location**, then navigate to and select the dates.txt file in the LISA_HOME\Projects\My Tutorials\Data directory.

7. Click **Test and Keep**.
   If the test is successful, the **Data Set Editor** window returns a "Test successful" message.

8. Click **OK**.

9. From the toolbar, click **Start ITR**, then select **Start new ITR**.

10. In the **Execution History** pane, click **Automatically execute test** .

11. When the test is complete, click **OK**.

12. In the **Execution History** pane, click DSstep1 and DSstep2.
    The first row of data in the data set is displayed in the **Response** tab. Both step responses display the same date because we read only from the data set in DSstep1.

Step 8 - Change the Data Set Behavior

You now modify the data set so that it loops through the test step until all the rows in the data set are read.

Follow these steps:

1. In the model editor, select the **DSstep1** test step.

2. In the step elements panel of DSstep1, click the arrow next to **DatesDS** under the Data Sets element.
   The data set editor opens.

3. In the **At end of data** field, select the **Execute** option.

4. Click the drop-down arrow on the **Execute** field and select **End the Test** from the list of choices that appear.
   This setting causes the test to end when all the data rows have been read.

5. Click **Test and Keep**.

6. Click **OK** to close the test successful message.

7. In the model editor, select the DSstep2 test step.

8. In the **Step Information** tab, set the **Next** drop-down list to DSstep1.
   This setting causes the two test steps to loop. The arrows in the model editor show the order of execution: DSstep1, followed by DSstep2, followed by DatesDS.

9. From the toolbar, click **Start ITR**, then select **Start new ITR**.

10. In the ITR, click **Automatically execute test** .
    The test case runs in a loop until there are no more data rows in the data set.

11. When the test is complete, click **OK**.

12. Click **Save**.

Tutorial 2 - Review

In this tutorial, you:

- Created a comma-delimited data set
- Used the data set for running a simple test case
- Learned how a test step accesses the data in the data set

Tutorial 3 - Filters and Assertions

Contents

- Step 1 - Create a Test Case from an Existing Test Case
- Step 2 - Change Action of Test Step
- Step 3 - Add an Assertion
- Step 4 - Test the Assertion
- Step 5 - Add a Filter
- Step 6 - Test the Filter
- Tutorial 3 - Review

In this tutorial, you modify the test case that was created in Tutorial 2 to include a filter and an assertion.

For an introduction to filters and assertions, see Filters and Assertions.

Tutorial Tasks

In this tutorial, you:

- Save an existing test case with a new name
- Add an assertion to a test step
- Add a filter to a test step

Prerequisites

- You have completed Tutorial 2 - Data Sets.
- DevTest Workstation is open.

Step 1 - Create a Test Case from an Existing Test Case

In this step, you open tutorial2.tst and save it as tutorial3.tst.

Follow these steps:

1. Open the tutorial2.tst test case in the My Tutorials project.
2. From the menu bar, select **File**, **Save As**.

3. In the **File name** field, enter **tutorial3**.

4. Click **Save**.
   The tutorial3 test case is created and saved under the My Tutorials project.

Step 2 - Change Action of Test Step

Change the **Next Steps** action of both test steps so that DSstep1 is the next step. Only the first step reads from the data set.

Follow these steps:

1. In the model editor, select DSstep1.

2. In the **Step Information** tab, change the Next step to **DSstep1**.
   With this action, the output goes back to the same step DSStep1. For the time being, alert icons appear next to DSStep2.

3. In the model editor, double-click DSstep2 and change the Output Log Message as follows:

   ```
   Date contains 1999. It is: {{month}}/{{day}}/{{year}}.
   ```

   Note: The curly brackets are important. The test case runs correctly only if they are included.

4. Click **Save**.

Step 3 - Add an Assertion

You can add various types of assertions to a test case. In this procedure, you add an XML assertion with the name **Ensure Result Contains String**.

The assertion logic is:

- If the response contains the string 1999, then the DSstep2 step is run next.
- If the response does not contain the string 1999, then the DSstep1 step is run next.

Follow these steps:

1. In the model editor, select DSstep1.

2. Open the **Assertions** tab.

3. Click + **Add**.

4. From the XML submenu, select **Ensure Result Contains String**.

Screenshot of XML submenu, Ensure Result Contains String.

5. The new assertion that is applied to DSstep1 is added to the **Assertions** tab. The assertion editor opens.

6. In the assertion editor, do the following:

    a. In the **If** list, select True.

    b. In the **then** list, select Go To: DSstep2.

    c. In the **Log** field, enter The string 1999 was found.

    d. In the **Contains String** field, enter 1999.

Screenshot of Result as String Contains Given String assertion dialog

7. Click **Save**.

Step 4 - Test the Assertion

To determine whether the assertion works as expected, use the Interactive Test Run (ITR) utility.

Follow these steps:

1. Start a new ITR session.
2. In the **Execution History** pane, click **Automatically execute test** .
3. When the test is complete, click **OK**.
4. Review the **Response** tab.

> **Note:** The DSstep2 step is executed next when DSstep1 encounters a date in which the year is 1999.

ITR Response tab for Tutorial 3

5. Click the **Properties** tab and review the behavior of the properties.

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_DOC_PATH	C:\Lisa\Projects\My Tut...	C:\Lisa\Projects\My Tut...
LISA_LAST_STEP	end	DSStep2
lisa.DSStep2.rsp	Date contains 1999. It is...	Date contains 1999. It i...
lisa.end.rsp	The test has ended	
year	1999	1999
day	8	8
robot	0	0
lisa.DSStep2.rsp.time	0	0
LISA_TC_PATH	C:\Lisa\Projects\My Tut...	C:\Lisa\Projects\My Tut...
LISA_HOST	Diana-PC	Diana-PC
LISA_PROJ_NAME	My Tutorials	My Tutorials
lisa.end.rsp.time	0	
LISA_USER	arhoades@Diana-PC	arhoades@Diana-PC
instance	0	0
testCaseId	66366438313934302D6...	66366438313934302D6...
LISA_TC_URL	file:/C:/Lisa/Projects/My...	file:/C:/Lisa/Projects/My...
lisa.DatesDS.returnedPr...	[DatesDS_RowNum, mo...	[DatesDS_RowNum, mo...
LASTRESPONSE	The test has ended	Date contains 1999. It i...
config_prop	42	42
testCase	tutorial3	tutorial3
lisa.DSStep1.rsp.time	0	0
yearglobal	1999	1999
lisa.DSStep1.rsp	Date is: 5/8/1999	Date is: 5/8/1999
LISA_PROJ_ROOT	C:/Lisa/Projects/My Tut...	C:/Lisa/Projects/My Tut...
month	5	5
LISA_DOC_URL	file:/C:/Lisa/Projects/My...	file:/C:/Lisa/Projects/My...
DatesDS_RowNum	4	4

Screenshot of properties tab for Tutorial 3

6. Click the **Test Events** tab and review the events that were generated.

Interactive Test Run - 6

Execution History
- DSStep1
- DSStep1
- DSStep1
- DSStep1
- DSStep2
- End the Test

Response | Properties | Test Events

EventID	Timestamp	Short	Long
Cycle ended ...	Mon Jun 27 14:08:1...	66366438313934302D626...	N/A
Cycle ending	Mon Jun 27 14:08:1...	66366438313934302D626...	Signaled...
Cycle history	Mon Jun 27 14:08:1...	66366438313934302D626...	

Screenshot of ITR Test Events tab for Tutorial 3

Step 5 - Add a Filter

You can add various types of filters to a test case. In this procedure, you add a utility filter with the name **Store Step Response**. This type of filter lets you save the step response as a property.

Follow these steps:

1. In the model editor, select DSstep1.

2. Open the **Filters** tab.

3. Click **Add**.

4. From the **Utility Filters** submenu, select **Store Step Response**.
   The filter editor opens.

5. In the filter editor, set the property name to **DSstep1_response_prop**.
   This property is where the step response is stored.

Screenshot of Store Step Response filter for Tutorial 3

6. In the model editor, double-click DSstep2 and add the following text to the end of the output log message:

   ```
   The value of DSstep1_response_prop is: {{DSstep1_response_prop}}.
   ```

7. Click **Save**.

Step 6 - Test the Filter

To determine whether the filter works as expected, use the Interactive Test Run (ITR) utility.

Follow these steps:

1. Start a new ITR session.

2. In the **Execution History** pane, click **Automatically execute test** .

3. When the test is complete, click **OK**.

4. Review the **Response** tab.
   The DSstep2 test step now displays the additional text that you added to the output log message.

Screenshot of ITR Response tab for Tutorial 3

5. Click the **Properties** tab and observe where the DSstep1_response_prop property is created and modified.

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_DOC_PATH	C:\Lisa\Projects\My Tutorials\Tests	C:\Lisa\Projects\My Tutorials\Tests
LISA_LAST_STEP	DSStep2	DSStep1
lisa.DSStep2.rsp	Date contains 1999. It is: 5/8/1999....	
year	1999	1999
day	8	8
robot	0	0
lisa.DSStep2.rsp.time	0	
LISA_TC_PATH	C:\Lisa\Projects\My Tutorials\Tests	C:\Lisa\Projects\My Tutorials\Tests
LISA_HOST	Diana-PC	Diana-PC
LISA_PROJ_NAME	My Tutorials	My Tutorials
LISA_USER	arhoades@Diana-PC	arhoades@Diana-PC
DSstep1_response_prop	Date is: 5/8/1999	Date is: 5/8/1999
instance	0	0
testCaseId	30636437383534612D646530382D3...	30636437383534612D646530382D3...
LISA_TC_URL	file:/C:/Lisa/Projects/My%20Tutorial...	file:/C:/Lisa/Projects/My%20Tutorial...
lisa.DatesDS.returnedProps	[DatesDS_RowNum, month, lisa.Dat...	[DatesDS_RowNum, month, lisa.Dat...
LASTRESPONSE	Date contains 1999. It is: 5/8/1999....	Date is: 5/8/1999
config_prop	42	42
testCase	tutorial3	tutorial3
lisa.DSStep1.rsp.time	0	0
yearglobal	1999	1999
lisa.DSStep1.rsp	Date is: 5/8/1999	Date is: 5/8/1999
LISA_PROJ_ROOT	C:/Lisa/Projects/My Tutorials	C:/Lisa/Projects/My Tutorials
month	5	5
LISA_DOC_URL	file:/C:/Lisa/Projects/My%20Tutorial...	file:/C:/Lisa/Projects/My%20Tutorial...
DatesDS_RowNum	4	4

Screenshot of properties tab for Tutorial 3

6. Click the **Test Events** tab and review the events that were generated.

Response | Properties | Test Events

EventID	Timestamp	Short	Long
Info message	Mon Jun 27 14:14:34 CDT 2011	DSStep2	Date contains 19...
Log message	Mon Jun 27 14:14:34 CDT 2011	Will execute the default next step	

Screenshot of ITR Test Events tab for Tutorial 3

Tutorial 3 - Review

In this tutorial, you:

- Took a first look at filters and assertions.
- Opened and modified an existing test case.
- Learned how to add a simple assertion.
- Learned how to add a simple filter.
- Used the Interactive Test Run utility to validate that the assertion and filter worked as expected.

More Information

DevTest provides filters and assertions to cover most of the situations that you encounter in your test case development. If no appropriate filter exists, DevTest provides a mechanism for developing custom filters and assertions through the Software Development Kit (SDK). See Using the SDK for more information.

Tutorial 4 - Manipulate Java Objects (POJOs)

Contents

- Step 1 - Create a Test Case
- Step 2 - Create a Dynamic Java Execution Test Step
- Step 3 - Make a Call on the Java Object
- Step 4 - Add an Inline Filter
- Step 5 - Verify the Property Created
- Tutorial 4 - Review

In this tutorial, you create and manipulate a simple Java object and use the **java.util.Date** class to create a date object.

First, you construct the object and review how to call methods on the object. Then you incorporate the object into a simple DevTest model editor.

Tutorial Tasks

In this tutorial, you:

- Use the Dynamic Java Execution test step
- Use the Complex Object Editor for simple objects
- Use inline filters and save the results into a property

Prerequisites

- You have completed Tutorial 3 - Filters and Assertions.
- DevTest Workstation is open.

Step 1 - Create a Test Case

To create a test case:

- In the My Tutorials project, create a test case with the name **tutorial4**.

Step 2 - Create a Dynamic Java Execution Test Step

The Dynamic Java Execution test step lets you create a Java object from a class in the DevTest classpath. In the following procedure, you use the **java.util.Date** class.

Follow these steps:

1. Click **Add Step**.
   The **Add step** menu is displayed.

2. Select **Java/J2EE** and then select **Dynamic Java Execution**.
   The Dynamic Java Execution editor opens.

Screenshot of Dynamic Java Execution editor for Tutorial 4

3. In the Local JVM Settings area, ensure that **Make New Object of Class** is selected.

4. In the field to the right of **Make New Object of Class**, type **java.util.Date**.

5. Click **Construct/Load Object**.
   The Complex Object Constructor wizard appears. The first step shows the available constructors.

6. Select the Date(java.lang.Long) constructor.

Screenshot of Complex Object Constructor

7. Click **Next**.

8. Click **Finish**.
   The Complex Object Editor opens.

Screenshot of COE for Tutorial 4

Now you have a Java object to manipulate in the Complex Object Editor.

Step 3 - Make a Call on the Java Object

The Complex Object Editor is divided into two panels. The left panel contains the **Object Call Tree** , which tracks method invocations and their input parameters and return values. The following icons are used to identify the branches in the object call tree:

The type (class) of the currently loaded object, followed by the response from calling the 'toString' method of the object.

The Constructor that was called. This icon 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 the call has been executed) for the enclosing method.

The contents of the right panel vary depending on what is selected in the left panel.

To call the Java object:

1. In the right panel of the Complex Object Editor, click the **Call Sheet** tab.
   The **Call Sheet** tab shows the available methods that you can call.

2. Double-click the **setYear()** method. Or, you can select the **setYear()** method and click **Add selected method to Object Call Tree**.

Screenshot of the Call Sheet tab for the Dynamic Java Execution step.

The **setYear()** method is added to the **Object Call Tree**. The right panel now displays the **Call** tab and **Docs** tab.
The **Call** tab lists the argument information.

3. In the **Value** field for arg1, enter **104**.

4. Click **Execute**.

Screenshot of Dynamic Java Execution step COE Call Tree tab for Tutorial 4

5. In the **Object Call Tree**, select the **java.util.Date** object.

6. In the **Data Sheet** tab, verify that the year field is now set to 104.

Screenshot of Dynamic Java Execution step COE Data Sheet tab for Tutorial 4

Step 4 - Add an Inline Filter

You can add an inline filter from the Complex Object Editor. Inline filters (and assertions) do not result in a filter being added to the test step in the elements panel. Inline filter management is always done in the Complex Object Editor.

Follow these steps:

1. With the **java.util.Date** object selected, click the **Call Sheet** tab.

2. To retrieve the date to be placed in a property, invoke the **toString()** method.
   The right panel now displays the **Call** tab and **Docs** tab.

3. To add an inline filter, enter **Date_prop** as the property name in the **Save Result in Property** field in the **Status/Result** area on the **Call** tab.

Screenshot of COE Call tab for Tutorial 4

4. Click **Execute**.

5. Click **Save**.

Step 5 - Verify the Property Created

You can display the **Property Window** to verify that the **Date_prop** property was created.

Follow these steps:

1. Click **Show model properties** on the test case toolbar. Or, you can select **Help**, **View Properties** from the main menu.

2. Click **Refresh** .

3. Locate the **Date_prop** property.

Key	Value
yearglobal	1999
config_prop	42
LISA_LAST_STEP	
LISA_DOC_PATH	C:\Lisa\Projects\My Tutorials\Tests
lisa.designtime.testcaseinfo	com.itko.lisa.editor.TestCaseInfo@ebacb7
Date_prop	Wed Dec 31 18:00:00 CST 1969
lisa.Step1.rsp	Wed Dec 31 18:00:00 CST 1969
LISA_TC_PATH	C:\Lisa\Projects\My Tutorials\Tests
robot	0
LISA_HOST	Diana-PC
LISA_PROJ_NAME	My Tutorials
instance	0
LISA_USER	arhoades@Diana-PC
testCaseId	33626165353535642D643838352D3461
LISA_TC_URL	file:/C:/Lisa/Projects/My%20Tutorials/Tests
testCase	Test Case
LISA_PROJ_ROOT	C:/Lisa/Projects/My Tutorials
LISA_DOC_URL	file:/C:/Lisa/Projects/My%20Tutorials/Tests

Screenshot of Property window for Tutorial 4

4. Click **Close**.

Tutorial 4 - Review

In this tutorial, you:

- Created a test step to manipulate a Java object of **java.util.Date** type.
- Used the Complex Object Editor to manipulate the Java object.
- Learned how to add inline filters to objects and save results into a property.

Tutorial 5 - Run a Demo Server Web Application

Contents

- Step 1 - Start the Web Application
- Step 2 - Log in to the Web Application
- Step 3 - Create an Account
- Step 4 - Close an Account
- Step 5 - Log Out from the Web Application
- Tutorial 5 - Review

In this tutorial, you step through a simple web application that accompanies DevTest.

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 application, you can view the profile of the user, create an account, add addresses, and so on.

The goal of this tutorial is for you to become familiar with the application. This application is used in subsequent tutorials as the system under test.

Prerequisites

- The Demo Server is running.

Step 1 - Start the Web Application

Follow these steps:

1. Open a new browser window.
2. Enter the following URL, where localhost is the path with the IP address for your computer.

   `http://localhost:8080/lisabank/`

   The login page appears.

LISA Bank login page

Step 2 - Log in to the Web Application

You use the predefined user name **lisa_simpson**.

Follow these steps:

1. In the **Name** field, enter **lisa_simpson**.

2. In the **Password** field, enter **golisa**.

3. Click **Login**.
   The welcome page appears. The left side contains buttons for various actions that you can perform: **View Profile**, **New Account**, **Close Account**, **Add Address**, **Delete Address**, and **Log Out**.

LISA Bank welcome page

Step 3 - Create an Account

Notice that the current user does not have any accounts.

Follow these steps:

1. Click **New Account**.

2. From the **Account Type** list, select SAVINGS.

3. In the **Account Name** field, enter **My Savings**.

4. In the **Initial Balance** field, enter **100.00**.

5. Click **Add Account**.

LISA Bank new account page

The new savings account is added to the **Accounts** section.

LISA Bank with new savings account added

6. Repeat the preceding steps to create two more accounts: CHECKING and AUTO_LOAN. Set the initial balance of the checking account to $600.00. Set the initial balance of the auto loan to $10000.00.
   Do not use commas in the **Initial Balance** field.

LISA Bank My Money Home screen

Step 4 - Close an Account

The application lets you close accounts.

Follow these steps:

1. Click **Close Account**.
2. Select **My Savings** from the list and click **Select Account**.

LISA Bank close account page

3. Click **Confirm Delete**.
   The **My Savings** account is removed from the **Accounts** section.

Step 5 - Log Out from the Web Application

To log out from the web application, click **Log Out**.

Tutorial 5 - Review

In this tutorial, you:

- Logged in to the LISA Bank application.
- Created new accounts.
- Closed an account.

Tutorial 6 - Test a Website

In this tutorial, you use the web recorder to record the path through a website. You then create test steps of HTTP/HTML Request for each HTTP request/response pair.

The HTTP/HTML Request test step enables you to make requests of a web server and receive results in a test case. To verify that the pages work as expected, test a simple website.

Tutorial Tasks

In this tutorial, you:

- Create a test case that contains HTTP/HTML Request steps with the web recorder

- Edit and run the test case that the recorder produces
- Add a data set to the test case

Prerequisites

- You have completed Tutorial 5 - Run a Demo Server Web Application.
- DevTest Workstation is open.
- You have access to the demo server.

Tutorial Parts

- Tutorial 6 - Part A - Record the Test Case
- Tutorial 6 - Part B - Run the Test Case
- Tutorial 6 - Part C - Modify Request Test Steps

Tutorial 6 - Part A - Record the Test Case

Contents

- Step 1 - Create a Test Case
- Step 2 - Start the Web Recorder
- Step 3 - Record the LISA Bank Application
- Step 4 - Stop the Web Recorder

With the web recorder, create a test case that contains HTTP/HTML Request steps.

Step 1 - Create a Test Case

Follow these steps:

1. In the My Tutorials project, create a test case with the name **tutorial6a** in the **Tests** subfolder.
   The model editor opens.

Step 2 - Start the Web Recorder

In this tutorial, you record a website through HTTP proxy.

Follow these steps:

1. Select **Actions**, **Record Test Case for User Interface**, **Web Recorder (HTTP proxy)** from the main menu.
   The **Test Recorder** dialog opens.

2. In the **Opening URL** field, enter the following URL. Replace the IP address in the path with the IP address for your computer.

Screenshot ot Test Recorder dialog for Tutorial 6

3. Click **Start Recording**.
   The **Test Recorder** window opens. The **Test Recorder** window contains two tabs: **Browser** and **Recorded Elements**. The login page of the LISA Bank application appears in the Browser tab.

Step 3 - Record the LISA Bank Application

While you perform actions in the LISA Bank application, the request and response information for each page visited are recorded.

Follow these steps:

1. In the **Name** field, enter **lisa_simpson**. In the **Password** field, enter **golisa**.

LISA Bank login screen for Tutorial 6

2. Click **Login**.
   The welcome page appears.

3. In the **Accounts** section, click the account number of the checking account.
   The **Account Activity** window appears.

4. Click **Deposit**.

LISA Bank deposit screen for Tutorial 6

5. In the **Deposit Money** area, enter the password **golisa**, a description of the transaction, and the amount for this deposit.

6. Click **Deposit**.

LISA Bank deposit screen for Tutorial 6 for second deposit

The **Account Activity** window shows the updated balance and a record of the deposit.

LISA Bank account activity screen for Tutorial 6

7. From the left navigation pane, click **Log Out**.

Step 4 - Stop the Web Recorder

After you stop recording, you can view details about the transactions.

Follow these steps:

1. At the bottom of the **Test Recorder** window, click **Stop Recording**.
   Filters and properties are automatically created for the web page references. The form fields for the deposit are also displayed. The left pane shows a list of transactions (steps). The right pane shows the step detail and response for the selected transaction.

Screenshot of Test Recorder, Step Detail tab for Tutorial 6

2. Click **Commit Edits**.
   The parameters page appears.

3. Click **Add to Test** and **Close**.

Screenshot of Test Recorder, Parameters page for Tutorial 6

The model editor is populated with a new test case, having all the transaction information from the saved recording. Each test step in the test case represents an HTTP request.

Screenshot of test case for Tutorial 6

4. Save the test case.

Tutorial 6 - Part B - Run the Test Case

In this section, continue from Part A to run the saved test case in the Interactive Test Run (ITR) utility and view the results.

Follow these steps:

1. Start a new ITR session.

2. In the **Execution History** pane, click **Automatically execute test**.

3. When the test is complete, click **OK**.

Screenshot of ITR run for Tutorial 6

The **View** tab shows the rendered pages while the ITR replays the deposit into the checking account.
The **Source** tab shows the HTML code for the page that is captured in the step.
DevTest acts as the browser and sends the same HTTP requests to the web server.

4. Close the ITR utility.

Tutorial 6 - Part C - Modify Request Test Steps

Contents

- Step 1 - Copy a Test Case
- Step 2 - Add a Data Set
- Step 3 - Modify the POST Parameters for the Recorded Deposit
- Step 4 - Stage the Test Case
- Step 5 - View the New Deposits in LISA Bank
- Tutorial 6 - Review

The web recorder produces an HTTP/HTML Request test step for each request.

You can edit and modify these test steps like the other test steps in DevTest. 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 LISA Bank test, replace these hard-coded description and deposit amount values (for example, "cash" and "1000.00") with properties from a data set. You previously worked with data sets in Tutorial 2 - Data Sets.

In the LISA Bank5 test step from the recording results, the Host Name and Port parameters are parameterized and added to the configuration. The values for description and amount are hard-coded.

In this part of the tutorial, we use a numeric counting data set to parameterize the test case so it deposits different amounts of money. When you then run the test case, it uses deposit values different from the ones recorded.

Step 1 - Copy a Test Case

Follow these steps:

1. Ensure that the tutorial6a test case is open in the model editor.
2. Select **File**, **Save As** from the menu bar.
3. In the **File name** field, enter tutorial6c.
4. Click **Save**.

Step 2 - Add a Data Set

In the following procedure, you add a numeric counting data set. This type of data set enables you to assign a number to a property. You can change the number by a fixed value each time the data set is used.

Follow these steps:

1. In the model editor, select the first test step.
2. In the right pane, double-click the **Data Sets** step tab.
3. Click **Add** below the **Data Sets** element.
4. From the **Common Datasets** list, select **Create a Numeric Counting Data Set**.
   The data set editor opens in the right pane.
5. Enter the following values:
   a. In the **Name** field, enter DepositsDS.
   b. In the **At end** field, select the **Execute** option and select **End the Test** from the list.
   c. In the **Property Key** field, enter ds_counter.
   d. In the **From** field, enter 100.
   e. In the **To** field, enter 105.

f. In the **Increment** field, enter 1.

6. Click **Test and Keep** to test the data set.
   You see a success message that shows the first row of data in the data set:

Screenshot of Data Set Editor success message for Tutorial 6

7. Click **OK**.

Step 3 - Modify the POST Parameters for the Recorded Deposit

You now use the ds_counter property (which you created in the data set) to specify varying amounts of money for the deposit.

Follow these steps:

1. In the model editor, double-click the LISA Bank5 step.

2. In the **POST Parameters** area, change the value of the **description** key to **deposit {{ds_counter}}**.

3. Change the value of the **amount** key to **{{ds_counter}}**.

Screenshot of modifying Post parameters for Tutorial 6

4. Save the test case.

Step 4 - Stage the Test Case

To stage (or run) a Quick Test:

1. From the toolbar, click **Stage a quick test**.
2. In the **Stage a Quick Test** window, ensure that **If test ends, restart it** is selected.

Stage a Quick Test window

3. Click **OK**.
   The **Test Monitor** opens, but the test has not been started yet.

Stage Test Case Current Interval Metrics window

4. Click **OK**.

5. From the toolbar, click **Play**.
   The line graphs show the progress of the test.

Stage Test Case Current Interval Metrics window while test is running

Step 5 - View the New Deposits in LISA Bank

Follow these steps:

1. Log in to the LISA Bank application again with the user **lisa_simpson** and password **golisa**.

2. To view the deposits, click the account number link for the checking account.
   Notice how the deposits start with 100 and increase by 1 until the amount 105 is reached.

Screenshot of LISA Bank with incrementing account balance for Tutorial 6

Tutorial 6 - Review

In this tutorial, you used a numeric counting data set to provide input to the recorded test.

You:

- Copied a test case and added a numeric counting data set.
- Modified the POST Parameters for the recorded deposit.
- Staged a quick test.

Tutorial 7 - Test an Enterprise JavaBean (EJB)

Contents

- Step 1 - Create a Test Case
- Step 2 - Create a Configuration
- Step 3 - Add an EJB Test Step
- Step 4 - Connect to the Server
- Step 5 - Locate the EJB Interface
- Step 6 - Configure the EJB
- Step 7 - Add an Assertion
- Step 8 - Verify the Method Execution
- Step 9 - Add Another EJB Test Step
- Tutorial 7 - Review

The LISA Bank application provides a full set of EJBs to interact with an account, get the user and account information from the Java interface.

This tutorial uses the Enterprise JavaBean Execution test step to call EJB methods in a test case and test the response with an assertion. You test a simple EJB to verify that the addUser and deleteUser methods work as expected.

Tutorial Tasks

In this tutorial, you:

- Use the Enterprise Java Bean Execution step.
- Use the Complex Object Editor with EJB objects.

Prerequisites

- You have completed Tutorial 6 - Test a Website.
- DevTest Workstation is open.
- You have access to the demo server.

Step 1 - Create a Test Case

Follow these steps:

1. In the **Project** pane, right-click the **Tests** folder and select **Create New Test Case**.
2. Set the file name to **tutorial7**.
3. Click **Save**.

Step 2 - Create a Configuration

You previously worked with configurations in Tutorial 2 - Data Sets.

Follow these steps:

1. Open the project.config file.
2. If the configuration does not contain the User and Password properties, add these properties. You do not need to set the values.
3. Create a configuration with the name **config7**.
4. Add the User property to the config7 configuration and set the value to **Lisa7**.
5. Add the Password property to the config7 configuration and set the value to **Pass7**.

6. Click **Save**.
7. In the **Project** pane, right-click the config7 configuration and select **Make Active**.
   The configuration now appears in green.

Step 3 - Add an EJB Test Step

The Enterprise JavaBean Execution test step enables you to make calls on a running EJB.

Follow these steps:

1. Click the tutorial7 tab.
2. Click **Add Step**.
3. Select **Java/J2EE** and select **Enterprise JavaBean Execution**.
   The **New EJB Setup** wizard appears.

Step 4 - Connect to the Server

The **New EJB Setup** wizard prompts you to specify the connection information for the EJB server. Enter information for the computer where the demo server is running.

Follow these steps:

1. From the **Select Server From List** drop-down list, select **JBoss 3.2/4.0**.
2. In the **Host Name or IP Address** field, enter **localhost** or the name or IP address of the remote computer.
3. Click **Next**.
   The list of JNDI names is retrieved from the EJB server.

Step 5 - Locate the EJB Interface

The **New EJB Setup** wizard prompts you to specify the name of the EJB interface.

Follow these steps:

1. In the **Remote** tab, select EJB3UserControlBean/remote.

2. Click **Next**.
   The Complex Object Editor opens.

Step 6 - Configure the EJB

Follow these steps:

1. If you use the same EJB object repeatedly, and the **Keep EJB Object Reference** check box is not already selected, select the check box.

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.

3. In the **Object Editor** area, select the **Call Sheet** tab and select the addUser method.

4. Click **Add selected method to Object Call Tree**.

The **Object Call Tree** now displays the addUser method.

Important! You can only add User and Password once. To execute this tutorial more than once, change the values that are associated with User and Password.

Step 7 - Add an Assertion

To enter the method parameters and add an inline assertion:

1. In the **Object Call Tree** pane, select **Expert Mode** if it is not already enabled.
2. Select the **Use Property** check boxes for each argument.
   Use Property is a column heading in the **Parameters** area.
3. In the **Value** column for arg1, select **User** from the **Defaults** list of properties.
4. In the **Value** column for arg2, select **Password** from the **Defaults** list of properties.
5. In the **Status/Result** area, add the inline assertion by selecting **Exact** and clearing the **True** check box.
6. In the **Comparison on Result NOT Exactly** field, enter **True**.
7. From the **Exact** drop-down, select **Fail the Test**.
8. Click **Execute**.

The parameters to the method are displayed in the **Object Call Tree**, next to the **Input Parameter** icon. The return value of this method is the value of the User field, "Lisa7," in the **Object Call Tree**.

9. Test the addUser method again by clicking **Execute**.

10. The return changes to null and an error is produced because the user has already been added.

Step 8 - Verify the Method Execution

From the LISA Bank application, you can verify that the user was added.

Follow these steps:

1. Go to the LISA Bank application.
2. Log in as user **admin** with the password **admin**.
3. To confirm that Lisa7 was added, view the list of users.

Step 9 - Add Another EJB Test Step

Now try the preceding steps again to invoke the deleteUser method.

Follow these steps:

1. Repeat the tutorial beginning with Step 3 to add an EJB step with the name **DeleteUser**.
2. Use the method parameter property User.

3. Click **Execute** to execute this method and get results.

4. The return is true, indicating the user has been deleted.

5. Click **Save**.

Tutorial 7 - Review

In this tutorial, you:

- Created a test case consisting of two EJB test steps.
  The EJB object was loaded from the example application on the demo server.

- Created an EJB test step and loaded an EJB.
- Used the Complex Object Editor to manipulate EJB objects.

Tutorial 8 - Test a Web Service

Contents

- Step 1 - Create a Test Case
- Step 2 - Add a Web Service Execution (XML) Test Step
- Step 3 - Create a Web Service Client
- Step 4 - Execute the Web Service Request
- Step 5 - View the Request and Response
- Tutorial 8 - Review

In this tutorial, you use the Web Service Execution (XML) test step to call web service operations in a test case. You then test the request and response. These web service operations provide the same functionality as the equivalent method calls in the EJB used in Tutorial 7.

Tutorial Tasks

In this tutorial, you:

- Add the Web Service Execution (XML) test step.
- Execute a web service operation.

Prerequisites

- You have completed Tutorial 5.
- DevTest Workstation is open.
- You have access to the demo server.

Step 1 - Create a Test Case

Follow these steps:

1. Right-click on the **Tests** folder in the **Project** panel, and select **Create New Test Case**.
2. Set the file name to **tutorial8**.
3. Click **Save**.
4. In the **Project** panel, right-click on project.config and select **Make active**.

Step 2 - Add a Web Service Execution (XML) Test Step

The Web Service Execution (XML) test step enables you to execute an operation on a SOAP-based web service.

Follow these steps:

1. Click the tutorial8 tab.

2. Click **Add Step**.

3. Select **Web/Web Services** and select **Web Service Execution (XML)**.
   A Web Service step is added to the model editor.

4. To open the Web Service Execution (XML) editor, double-click the Web Service step.

Screenshot of Web Service Execution (XML) editor for Tutorial 7

5. Click **New Document**.

Step 3 - Create a Web Service Client

Now specify the operation to be called, and create a SOAP message to send to the operation.

Follow these steps:

1. In the **WSDL URL** field, enter the following location.

> **Note:** The WSSERVER and WSPORT properties represent the server and port.

```
http://localhost:8080/itko-examples/services/UserControlService?wsdl
```

2. In the **Service** field, select UserControlServiceService.
3. In the **Port** field, select UserControlService.
4. In the **Operation** field, select the addUser operation.
5. In the **On Error** field, select Abort the Test.
   DevTest uses this criteria to build the web service client. The Visual XML editor shows a graphical view of the SOAP message.

Screenshot of Visual XML editor for Tutorial 8

6. Save the test case.

Step 4 - Execute the Web Service Request

Follow these steps:

1. Click **Execute WS Request**.
   The test is executed.

Step 5 - View the Request and Response

The **Request** tab shows the resulting request data that was sent after any post processing (for example, substituting DevTest properties). The **Response** tab shows the resulting response data that was received.

Follow these steps:

1. To view the request upon execution, click the **Request** tab.

Screenshot of Web Service Execution (XML) Request tab for Tutorial 8

2. To view the response upon execution, select the **Response** tab.

Screenshot of Web Service Execution (XML) Response tab for Tutorial 8

Tutorial 8 - Review

In this tutorial, you:

- Created a test case with the Web Service Execution (XML) test step.
- Executed the addUser operation.
- Viewed the request and response for this operation.

Tutorial 9 - Examine and Test a Database

Contents

- Step 1 - Create a Test Case
- Step 2 - Add Database Properties to the Configuration
- Step 3 - Add a SQL Database Execution (JDBC) Test Step

- Step 4 - Connect to the Database
- Step 5 - Execute a SQL Query
- Step 6 - Add an Assertion
- Step 7 - Run the Test Case
- Step 8 - Change the Assertion
- Step 9 - Add a Filter
- Step 10 - Test the Filter and Assertion
- Tutorial 9 - Review

In this tutorial, you examine and test a database table that is part of the web application in Tutorial 5.

You use the SQL Database Execution (JDBC) step to interact with a database in a test case and test the response with an assertion. You examine the Users table from a Derby database that is part of the application.

Tutorial Tasks

In this tutorial, you:

- Use the SQL Database Execution (JDBC) step.
- Store application properties in a configuration.
- Add and modify an assertion.
- Add a filter.

Prerequisites

- You have completed Tutorial 5.
- DevTest Workstation is open.
- You have access to the demo server.

Step 1 - Create a Test Case

Follow these steps:

1. In the **Project** pane, right-click on the **Tests** folder and select **Create New Test Case**.
2. Set the file name to **tutorial9**.
3. Click **Save**.

Step 2 - Add Database Properties to the Configuration

Store the properties that are necessary to connect to the database in the configuration. This practice is a standard DevTest practice that increases the portability of test cases.

Follow these steps:

1. If project.config is not the active configuration, then right-click project.config in the **Project** pane and select **Make Active**.

2. Open the project.config configuration.

3. Add the following properties:

   - **DBDriver**
     org.apache.derby.jdbc.ClientDriver

   - **DBConnect**
     jdbc:derby://localhost:1529/lisa-demo-server.db

   - **DBUserID**
     sa

   - **DBPwd**
     sa

Properties Editor

Key	Value	Encrypt
DBDriver	org.apache.derby.jdbc.ClientDriver	☐
DBConnect	jdbc:derby://localhost:1529/lisa-demo-server.db	☐
DBUserID	sa	☐
DBPwd	sa	☐

4. Click **Save**.

Step 3 - Add a SQL Database Execution (JDBC) Test Step

The SQL Database Execution (JDBC) test step enables you to connect to a database using JDBC and make SQL queries on the database.

Follow these steps:

1. Click the tutorial9 tab.

2. Click **Add Step**.

3. Select **Other Transactions** and select **SQL Database Execution (JDBC)**.
   JDBC is added to the model editor.

To open the step editor, double-click the JDBC step.

Screenshot of JDBC step editor for Tutorial 9

Step 4 - Connect to the Database

To provide the connection information, use the properties that you added to the project.config configuration.

Follow these steps:

1. Enter the following values in the **Connection Info** and **Execution Info** areas of the step editor. Notice that when you enter the password, the value is masked.

   - **JDBC Driver**
     {{DBDriver}}

   - **Connect String**
     {{DBConnect}}

   - **User ID**
     {{DBUserID}}

   - **Password**
     {{DBPwd}}

2. Click **Test Connection** at the bottom of the step editor.
   A message indicates that the connection is valid.

Connection Attempt Successful Message

3. Click **OK**.

Step 5 - Execute a SQL Query

Now specify and run a SQL statement that retrieves data from the Users table.

Follow these steps:

1. In the **SQL Statement** pane, enter the following statement:

```
SELECT LNAME, LOGIN FROM Users
```

SQL Statement pane for Tutorial 9

2. Click **Test/Execute SQL** at the bottom of the step editor.
   A message confirms a valid query and displays the number of rows returned.

Screenshot of JDBC Step Editor message about Result Set results

3. Click **OK**.
   The **Result Set** tab is displayed.

Screenshot of JDBC Result Set tab for Tutorial 9

Step 6 - Add an Assertion

Add an assertion that tests for the presence of a specific last name in the result set.

Follow these steps:

1. In the **Result Set** tab, select a cell in the LNAME column.

2. Click **Generate Assertion for the Value of a Cell**.
   The **Generate JDBC Result Set Value Assertion** dialog opens.

Generate JDBC Result Set Value Assertion dialog for Tutorial 9

3. From the drop-down list, select the **Fail the Test** option.
   If the last name that you selected is not found, then the test fails.

Screenshot of Generate JDBC Result Set Value Assertion dialog for Tutorial 9

4. Click **OK**.

5. Click **Save**.

Step 7 - Run the Test Case

Follow these steps:

1. From the toolbar, click **Start ITR**.

2. Click **Execute Next Step**.
   The test runs successfully. The result set is shown in the **Response** tab.

Screenshot of ITR Results tab for Tutorial 9

3. Retract the ITR tray.

Step 8 - Change the Assertion

You now modify the assertion to cause the test to fail.

Follow these steps:

1. In the model editor, click the JDBC SELECT Users test step.

2. Open the **Assertions** tab in the **Element Tree**.

Screenshotof Assertions tab for JDBC Select Users step for Tutorial 9

3. Double-click the assertion that you created earlier.
   The assertion editor is opened. The lower portion indicates that the assertion checks the first column of the result set for the specified value.

4. Change the value of the **Regular Expression** field to Johns.

Screenshot of ResultSet Contents pane for Tutorial 9

5. Start a new ITR and run the test case again.
   The test fails.

6. Retract the ITR tray.

Step 9 - Add a Filter

Add a database filter that captures the value in the first column and fourth row of the result set. The value is stored in a property.

Follow these steps:

1. In the model editor, select the JDBC SELECT Users test step.

2. Open the **Filters** tab in the **Element Tree**.

3. Click **Add**.

4. From the **Database Filters** submenu, select **Extract Value from JDBC Result Set**.
   The filter editor opens.

5. In the **Column** field, enter *1.* Or, you can enter the actual column name, which is *LNAME.*

6. In the **Row** field, enter *3.*
   This field is zero-based. Therefore, the value 3 refers to the fourth row.

7. In the **Property** field, enter *DBProperty.*

Screenshot for Parse JDBC Result Sset for Value dialog for Tutorial 9

8. Click **Save**.

Step 10 - Test the Filter and Assertion

Follow these steps:

1. Start a new ITR and run the test case again.
   The test fails because Johns was not found in the result set.

2. Click the **Test Events** tab.

3. Click the **Property set** event.
   Notice that DBProperty was set to the value specified by the filter.

4. Click the **Assertion fired** event.
   The **Long Info Field** area indicates that the assertion fired because the first column of the result set did not contain the value Johns.

Screenshot of ITR Long Info Field for Tutorial 9

5. Click the **Properties** tab.

6. Locate and review the **DBProperty** row.

Screenshot for ITR results for Tutorial 9

Tutorial 9 - Review

In this tutorial, you created a test case to query a database. You used the Users table from the Apache Derby database that accompanies the applications on the demo server. You learned how to:

- Connect to the database.
- Execute a SQL query against the database.
- Add assertions and filters.

Tutorial 10 - Stage a Quick Test

Contents

- Step 1 - Open the Test Case
- Step 2 - Review the Test Case
- Step 2 - Part A - Run a Quick Test
- Step 2 - Part B - View Generated Reports
- Tutorial 10 - Review

In this tutorial, you use the quick staging option (quick test) to learn how to stage tests and read subsequent reports. You run the quick test on the multi-tier-combo example that accompanies DevTest. Running a quick test is the simplest way to stage a test.

Tutorial Tasks

In this tutorial, you:

- Use the multi-tier-combo test case.
- Use the quick test feature.
- Select and format reports.

Prerequisites

- You have completed Tutorials 5 through 9.
- DevTest Workstation is open.
- You have access to the demo server.

Step 1 - Open the Test Case

Open a test case from the examples project.

Follow these steps:

1. Select **File**, **Open**, **Test Case**, **File System** from the main menu.
2. Navigate to the LISA_HOME\examples\Tests folder.
3. Select multi-tier-combo and click **Open**.
   The multi-tier-combo test case opens in the model editor.

Step 2 - Review the Test Case

Review the various types of test steps in this test case. For example:

- Add User is a Web Service Execution (Legacy) step.
- Get User is an Enterprise JavaBean Execution step.
- Verify User Added is a SQL Database Execution (JDBC) step.

- Deposit Money is a JMS Messaging (JNDI) step.

Screenshot of multi-tier-combo test case for Tutorial 9

You used many of these steps in tutorials 6 through 9. In this tutorial, you use all the test steps to build a more realistic test case involving several layers of the application.

Step 2 - Part A - Run a Quick Test

Follow these steps:

1. From the menu bar, click **Stage a Quick Test** on the test case toolbar.
   To stage a quick test, the example test case can be open in the model editor. Or, you can right-click on the test in the **Project** panel and can enter the parameters to stage a quick test from there.

Screenshot of Stage Quick Test dialog for multi-tier-combo for Tutorial 10

2. In the **Stage a Quick Test** dialog, complete the following required information:

- **Run Name**: Enter a unique name (Tutorial10QuickTest).
- **Number of Instances**: Enter a number of users to run the test concurrently (4).
- **Stage Instances To**: Select the name of the coordinator server or stage it locally.
- To restart the test, select **If test ends, restart it**.

3. Click **OK**.
   The **Test Run** window opens, but the test has not started yet.

4. To start the test running, click from the main toolbar.
   The test begins, and the graph immediately plots results.

Screenshot of staged test running for Tutorial 10

You can roll over the graph lines to view descriptions.

Screenshot of Quick Test graph with popup information box

5. To select which events to display, select the **Events** tab.

Screenshot of Events Tab after running Quick Test

Step 2 - Part B - View Generated Reports

View execution and performance data for test cases and suites that were run in DevTest Portal.

Follow these steps:.

1. From DevTest Portal, select **Reporting**, **Testing** from the left navigation menu.
   The **Testing window** opens.

Screen capture of Testing window

2. Click the report name to display the report.
   The report opens in a new tab.

3. To filter the report, select or enter the filter options in the **Refine by** pane and click **Apply.**

4. Analyze the data.

For more information about creating a PDF or CVS file and other options, see Testing Reporting.

Tutorial 10 - Review

In this tutorial, you tested the multi-tier-combo example using a quick test. You learned how to:

- Review a test case containing several types of test steps.
- Configure and run a test in the quick test feature.
- Examine a report that is generated from the test run.

Tutorial 11: Run the Forward Cars Demo Application

In this tutorial, you step through the key features of Forward Cars demo application.

- Step 1 - Start Forward Cars
- Step 2 - Perform an Inventory Search
- Step 3 - Change the Enterprise Service Bus Location
- Step 4 - Add a Car
- Step 5 - Edit a Car's Information
- Step 6 - Delete a Car
- Step 7 - Look up a Car's History Report Through the VIN
- Step 8 - Log in to Forward Cars
- Step 9 - Apply for an Auto Loan
- Step 10 - Check on Your Loan Status

- Step 11 - Log Out from Forward Cars
- Tutorial 11 - Review

Step 1 - Start Forward Cars

Forward Cars is included in the Demo Server download file and is located in the **LISA_HOME\DemoServer** file directory. Make sure that you have downloaded and unzipped **DevTestDemoServer.zip** file on your computer, as described in Running Forward Cars.

Follow these steps:

1. Start Forward Cars from the command line. Go to the **LISA_HOME\DemoServer\carsdemo** directory and start the script for your operating system:

   - (Windows) **start.bat**
     If the Windows machine has UAC enabled (the default for Windows 7), the start script must run with Administrative privileges.

   - (UNIX or Linux) **./start.sh**

   - (OS X) **./start.command**

   The application can take several minutes to start.

2. Enter the following URL in a web browser:

   `http://localhost:3434/cars-app`

   > **Note**: Because Forward Cars uses port 3434, that port cannot be in use by any other application. If this port is not available, Forward Cars does not start successfully.

   The home page opens:

Step 2 - Perform an Inventory Search

Use the search functionality to locate a specific vehicle through detailed search criteria.

Follow these steps:

1. From the home screen, click **Inventory Search**.
   All the available vehicles display.

2. To search for a particular make of car, click the name under the **Make** column. In this instance, click **Audi**.
   All Audi models display.

3. To clear the search, click the **Make** name (Audi) in the left column.

   All the vehicles display again.

4. You can also search for a specific vehicle details using text. Enter a key term or terms (separated by commas), such as **Black**, in the **Search** text box. Click **Clear Search** to display all the vehicles again.

   If a search does not return any results, no cars matched your search criteria. Click **Clear Search** and try again.

> **Note**: To identify key terms that you can search for, click **View Details** for any car. Any of the unique identifiers that display can be used in a text search.

Step 3 - Change the Enterprise Service Bus Location

The default installation routes most transactions via an Enterprise Service Bus (ESB) that is running on port 7000. If you are using Forward Cars in a virtual server environment, for example, you can change the ESB location.

Follow these steps:

1. From the home page, click **Inventory Search**.
   All the available vehicles display.

2. Click the **Lock** icon in the upper left corner.

   The Admin functionality displays.

3. In the URL text field, enter the new host address. In this instance, enter **7001**.

4. Click **Set ESB**.

5. Refresh the web browser.
   7001 is included in the host address. In a virtual environment, if that was a legitimate address, Forward Cars would now route requests from the new database location.

6. Set the host address to **7000** again, then click **Set ESB**.

7. Refresh the web browser.

Step 4 - Add a Car

Add a car to the inventory database.

Follow these steps:

1. From the home page, click **Inventory Search**.
   All the available vehicles display.

2. Click the **Lock** icon.
   The Admin functionality displays.

3. Click **Add a Car**.

   The **Create Car** page opens.

4. Click **Populate Form**.

   The fields are populated.

5. Click **Create** to add the new car to the database.

   The message **car created** displays at the top of the page. The new car is visible at the bottom of the page.

Step 5 - Edit a Car's Information

Edit details about an existing car in the database.

Follow these steps:

1. From the home page, click **Inventory Search**.
   All the available vehicles display.

2. Click the **Lock** icon.
   The Admin functionality displays.

3. Click **Edit** on any car that you want to edit.
   The **Edit** window opens.

4. Edit one or more fields, then click **Save**.
   The vehicle is updated with the new information.
   The message **car updated** displays at the top of the page.

Step 6 - Delete a Car

Remove a car from the inventory.

Follow these steps:

1. From the home page, click **Inventory Search**.
   All the available vehicles display.

2. Click the **Lock** icon.
   The Admin functionality displays.

3. Click **Delete** on any car that you want to delete.

4. Click **OK**.
   The vehicle is removed from the inventory.
   The message **car deleted** displays at the top of the page.

Step 7 - Look up a Car's History Report Through the VIN

Use the VIN to look up the history report for the vehicle.

Follow these steps:

1. From the home page, click **Inventory Search**.
   All the available vehicles display.

2. Locate a car that you want to view the history report for. Click **View Details**.
   The **Car Detail** window opens.

3. Click the **VIN** at the bottom of the window:

   The VIN is then pre-populated on the Vehicle History Report page.

4. Click **Lookup**.

   The Vehicle History Report refreshes to display the history report for the selected car.

5. You can also look up the VIN directly through the home screen. First, on the Vehicle History Report page, copy the VIN.

6. Click **Home** to return to the home page.

7. Click **VIN Lookup**.

   The Vehicle History Report page displays.

8. Enter the **VIN** and click **Lookup**.

   The Vehicle History Report refreshes to display the history report for the selected car.

Step 8 - Log in to Forward Cars

Use the predefined user name **lisa.simpson**.

Follow these steps:

1. Click **Guest** in the upper-right corner, then select **Log In**.

2. In the **Name** field, enter **lisa.simpson**.

3. In the **Password** field, enter **golisa**.

4. Click **Sign In.**
   The home page opens

The home page contains buttons for various actions that you can perform: **Inventory Search**, **VIN Lookup**, **Loan Application**, **Loan Status**, and **Log Out**.

Note: Any time that you refresh the browser window, you must log in again.

Step 9 - Apply for an Auto Loan

Forward Cars completes a credit check for a user who applies for a loan. You must be logged in first.

Follow these steps:

1. From the home page, click **Loan Application**.
   The Finance Application displays for Lisa Simpson.

2. Click **Apply**.
   The **Application Submitted** message opens.

3. Click **OK**.

You can also apply through a loan when you view details for a specific car.

Follow these steps:

1. Make sure that you are logged in.

2. From the home page, click **Inventory Search**.

3. On the **Inventory** page, click **View Details** for a car.
   The **Car Detail** page opens.

4. Click **Auto Finance**.
   The finance application displays for Lisa Simpson.

5. Click **Apply**.
   The **Application Submitted** message opens.

6. Click **OK**.
   The home page opens.

Step 10 - Check on Your Loan Status

You can check the status of the loans that you submitted to see if they were approved.

Follow these steps:

1. From the home page, click **Loan Status**.
   The **Application Results** page opens.

2. Click a record for Lisa Simpson.
   The page refreshes to display the lender proposals.

Step 11 - Log Out from Forward Cars

To log out from the web application, click **Lisa Simpson** in the upper right corner, then select **Logout**.

Tutorial 11 - Review

In this tutorial, you learned how to:

- Start Forward Cars
- Perform an inventory search
- Change the ESB location
- Add a car
- Edit a car's information
- Delete a car
- Look up a car's history report through the VIN
- Log in to Forward Cars
- Apply for an auto loan
- Check on your loan status
- Log out from Forward Cars

CA Service Virtualization Tutorials

This section contains the following CA Service Virtualization tutorials:

- Tutorial 1: REST-based Virtual Service
- Tutorial 2: JMS Virtual Service Using DevTest Portal
- Tutorial 3: JMS Virtual Service Using DevTest Workstation
- Tutorial 4: HTTP/S Virtual Service
- Tutorial 5: Virtual Service from WSDL
- Tutorial 6: Virtual Service from Swagger (OpenAPI)

Tutorial 1: REST-based Virtual Service

The DevTest Home\Projects\Cars v9 folder contains artifacts to demonstrate a REST-based virtual service with its associated auto-generated baseline test that you can use to see how CA Service Virtualization can work with the Forward Cars demo application.

- Step 1 - Start DevTest Portal
- Step 2 - Start VSE
- Step 3 - Start Demo Server
- Step 4 - Deploy a REST-based Virtual Service
- Step 5 - Monitor the REST-based Virtual Service
- Step 6 - Run a Baseline Test Against the REST-based Virtual Service
- Step 7 - Run a Baseline REST-based Test Against the Live Forward Cars Application

Step 1 - Start DevTest Portal

Follow these steps:

1. Ensure that the Enterprise Dashboard, registry, a coordinator server, and a simulator are running.

   - If your computer has DevTest Server installed:

   - Start the Enterprise Dashboard by selecting **DevTest Solutions**, **Enterprise Dashboard Server** from the list of programs in your Windows Start menu. Wait until the "Grails application running" message appears.

   - Start the registry by selecting **DevTest Solutions**, **Registry** from the list of programs in your Windows Start menu.
     If your computer has DevTest Workstation installed, use a registry that is running on another computer.
     Start a coordinator server by selecting **DevTest Solutions**, **Coordinator Server** from the list of programs in your Windows Start menu.

   - Start a simulator by selecting **DevTest Solutions**, **Simulator Server** from the list of programs in your Windows Start menu.

2. Select **DevTest Solutions**, **Portal UI** from the list of programs in your Windows Start menu.

3. Open a supported browser and enter http://localhost:1507/devtest.
   If the registry is on a remote computer, replace **localhost** with the name or IP address of the remote computer.
   If the port number was changed from the default value **1507**, use the new port number.

4. Click **Log in**.
   The **Login** dialog opens. Enter a valid username and password and click **Login**.

Step 2 - Start VSE

Follow this step:

- Select **DevTest Solutions**, **Virtual Service Environment** from the list of programs in your Windows Start menu.
  When the "Virtual service environment is now ready" line appears, VSE has initialized.

Step 3 - Start Demo Server

When DevTest and VSE are both running, you can start the demo server that you use to complete this tutorial.

Follow these steps:

1. Select **DevTest Solutions**, **Cars Example Servers** from the list of programs in your Windows Start menu.
   The application can take several minutes to start.

2. Enter the following URL in a web browser:

   ```
   /localhost:3434/cars-app
   ```

Note: Because Forward Cars uses port 3434, that port cannot be in use by any other application. If this port is not available, Forward Cars does not start successfully.

Step 4 - Deploy a REST-based Virtual Service

Follow these steps:

1. Navigate to the DevTest Portal.
2. In the **Current Project** drop-down on the upper right corner of the page, select **Cars v9**.
3. From the left nav bar, select **Manage, Virtual Service**s.
4. In the **Actions** column beside the **cars-rest** virtual service, select the **Options** icon, then select **Deploy**.
   The **Deploy Virtual Service** window appears.
5. Click **Deploy**.

Step 5 - Monitor the REST-based Virtual Service

Follow these steps:

1. From the left nav bar, select **Monitor, Virtual Service Environments, VSE**.
   The Monitor Virtual Services window appears.
2. You can see that the **cars-rest** virtual service is running on port 7001, which is the port specified in the **project.config** file being used as the configuration file for this project.

Step 6 - Run a Baseline Test Against the REST-based Virtual Service

Follow these steps:

1. From the left nav bar, select **Manage, Tests**.
   The Manage Tests tab appears.
2. In the **Actions** column beside the **cars-baseline-rest-test** test, select the **Options** icon, then select **Run with Options**.
   The **Run Test Case** window appears.
3. In the **Configuration** field, verify that the **project.config** file is selected.
   Remember, the **project.config** file directs that the test run against port 7001, or the port for the virtual service.
4. Click **Run**.
   The **Monitoring Tests** tab opens.
5. When the test has completed, select the **VSE** tab to monitor the virtual service.
6. Click the **Refresh** icon at the upper right corner of this screen to refresh the status of the virtual service.
   Notice that the Transaction Count for this virtual service shows 7, indicating that the test performed 7 transactions against the virtual service.

Step 7 - Run a Baseline REST-based Test Against the Live Forward Cars Application

Follow these steps:

1. From the left nav bar, select **Manage, Tests**.
   The Manage Tests tab appears.

2. In the **Actions** column beside the **cars-baseline-rest-test** test, select the **Options** icon, then select **Run with Options**.
   The **Run Test Case** window appears.

3. In the **Configuration** field, verify that the **live-esb.config** file is selected.
   The **live-esb.config** file directs that the test run against port 7000, or the port for the live Forward Cars application.

4. Click **Run**.
   The **Monitoring Tests** tab opens.

5. When the test has completed, select the **VSE** tab to monitor the virtual service.

6. Click the **Refresh** icon at the upper right corner of this screen to refresh the status of the virtual service.
   Notice that the Transaction Count for this virtual service still shows 7, indicating that the test that just completed did not perform any transactions against the virtual service.

Tutorial 2: JMS Virtual Service Using DevTest Portal

This tutorial describes how to use a loan application virtual service via JMS using DevTest Portal. The DevTest Home\Projects\Cars v9 folder contains artifacts to demonstrate a virtual service to apply for a loan application with the Forward Cars application using JMS.

- Step 1 - Start DevTest Portal
- Step 2 - Start VSE
- Step 3 - Copy JAR Files
- Step 4 – Configure Forward Cars to Use Proxy JMS Queues
- Step 5 - Deploy a Loan Application Virtual Service
- Step 6 - Monitor the Virtual Service
- Step 7 – Apply for a Loan Application in Forward Cars
- Step 8 - Check on Your Loan Status

Step 1 - Start DevTest Portal

Follow these steps:

1. Ensure that the Enterprise Dashboard, registry, a coordinator server, and a simulator are running.

   - If your computer has DevTest Server installed:

   - Select **DevTest Solutions**, **Enterprise Dashboard Server** from the list of programs in your Windows Start menu. Wait until the "Grails application running" message appears.

- Select **DevTest Solutions**, **Registry** from the list of programs in your Windows Start menu.
  If your computer has DevTest Workstation installed, use a registry that is running on another computer.

- Select **DevTest Solutions**, **Coordinator Server** from the list of programs in your Windows Start menu.

- Select **DevTest Solutions**, **Simulator Server** from the list of programs in your Windows Start menu.

2. Select **DevTest Solutions**, **Portal UI** from the list of programs in your Windows Start menu.

3. Open a supported browser and enter http://localhost:1507/devtest.
   If the registry is on a remote computer, replace **localhost** with the name or IP address of the remote computer.
   If the port number was changed from the default value **1507**, use the new port number.

4. Click **Log in**.
   The **Login** dialog opens. Enter a valid username and password and click **Login**.

Step 2 - Start VSE

Follow this step:

- Select **DevTest Solutions**, **Virtual Service Environment** from the list of programs in your Windows Start menu.
  When the "Virtual service environment is now ready" line appears, VSE has initialized.

Step 3 - Copy JAR Files

Two JAR files are required for deploying the JMS-based virtual service and must be copied into the DevTest directory.

Follow these steps:

1. Download the following files from the Internet.

   - activemq-client-5.12.1.jar

   - hawtbuf-1.11.jar

2. Copy them to the following directory:
   DEVTEST_HOME\hotDeploy

Step 4 – Configure Forward Cars to Use Proxy JMS Queues

Follow these steps:

1. If Forward Cars is running, stop the application.

2. Open *DEVTEST_HOME* \DemoServer\carsdemo \cars-loan.properties.

3. Edit the following properties to point to the PROXY JMS destinations to enable recording and playback for a JMS service, as follows:

cars.partner.responseQueue=PROXY.CARS.PARTNER.RESPONSE

cars.partner.bofaRequestQueue=PROXY.CARS.PARTNER.BOFA

cars.partner.chaseRequestQueue=PROXY.CARS.PARTNER.CHASE

cars.partner.citiRequestQueue=PROXY.CARS.PARTNER.CITI

4. Start Forward Cars from the command line. Go to the *DEVTEST_HOME\DemoServer\carsdemo* directory and start the script for your operating system:

   - (Windows) **start.bat**
     If the Windows computer has UAC enabled (the default for Windows 7), the start script must run with Administrative privileges.

   - (UNIX or Linux) ./**start.sh**

   - (OS/X) **./start.command**

   The application can take several minutes to start.

5. Enter the following URL in a web browser:

   /localhost:3434/cars-app

   The home page opens. Because Forward Cars uses port 3434, that port cannot be in use by any other application. If this port is not available, Forward Cars does not start successfully.

Step 5 - Deploy a Loan Application Virtual Service

Follow these steps:

1. Navigate to DevTest Portal.

2. In the **Current Project** drop-down on the upper right corner of the page, select **Cars v9**.

3. From the left nav bar, select **Manage, Virtual Service**s.

4. In the **Actions** column beside the **loan-offers** virtual service, select the **Options** icon, then select **Deploy**.
   The **Deploy Virtual Service** window appears.

5. Click **Deploy**.

Step 6 - Monitor the Virtual Service

Follow these steps:

1. From the left nav bar, select **Monitor, Virtual Service Environments, VSE**.
   The Monitor Virtual Services window appears.

2. You can see that the loan-offers virtual service is running on the PROXY JMS destinations that you specified in the cars-loan.properties file (Step 4).

Step 7 – Apply for a Loan Application in Forward Cars

Follow these steps:

1. Navigate to Forward Cars.
2. From the home page, click **Guest** in the upper-right corner, then select **Log In**.
3. In the **Name** field, enter **lisa.simpson**.
4. In the **Password** field, enter **golisa**.
5. Click **Sign In**.
   The home page opens.
6. From the home page, click **Loan Application**.
   The Finance Application displays for Lisa Simpson.
7. Click **Apply**.
   An Application Submitted message opens.
8. Click **OK**.

Now, if you repeat step 6, you can see that the transaction count is going up, indicating that the virtual service is returning those loan responses.

Step 8 - Check on Your Loan Status

You can check the status of the loans you have submitted to see if they were approved.

Follow these steps:

1. From the home page, click **Loan Status**.
   The **Application Results** page opens.
2. Click “Lisa” in the First Name field.
   The **Application Results** page refreshes to display the lender proposals.

Tutorial 3: JMS Virtual Service Using DevTest Workstation

This tutorial describes how to create a JMS recording of a loan application using the Forward Cars application. If you do not want to use the prerecorded loan-offers service that DevTest Solutions provides (in Tutorial 2), you can create your own virtual service using DevTest Workstation.

- Step 1 – Copy JAR Files
- Step 2 – Configure Forward Cars to Use Proxy JMS Queues
- Step 3 - Open DevTest Workstation
- Step 4 - Record a JMS Session to Apply for a Loan in Forward Cars
- Step 5 – Apply for a Loan Application in Forward Cars
- Step 6 - Complete the JMS Recording in DevTest Workstation
- Step 7 - Change the Forward Cars Application ID to a Magic String

Step 1 – Copy JAR Files

Two JAR files are required for deploying the JMS-based virtual service and must be copied into the DevTest directory.

Follow these steps:

1. Extract the following files from the cars .war file or download them from the Internet.
   - activemq-client-5.12.1.jar
   - hawtbuf-1.11.jar
2. Copy them to the following directory:
   DEVTEST_HOME\hotDeploy

Step 2 – Configure Forward Cars to Use Proxy JMS Queues

Follow these steps:

1. If Forward Cars is running, stop the application.
2. Open *DEVTEST_HOME* \DemoServer\carsdemo \cars-loan.properties.
3. Edit the following properties to point to the PROXY JMS destinations to enable recording and playback for a JMS service, as follows:

   cars.partner.responseQueue=PROXY.CARS.PARTNER.RESPONSE

   cars.partner.bofaRequestQueue=PROXY.CARS.PARTNER.BOFA

   cars.partner.chaseRequestQueue=PROXY.CARS.PARTNER.CHASE

   cars.partner.citiRequestQueue=PROXY.CARS.PARTNER.CITI
4. Start Forward Cars. Go to the *DEVTEST_HOME**DemoServer**carsdemo* directory and start the script for your operating system:
   - (Windows) **start.bat**
     If the Windows computer has UAC enabled (the default for Windows 7), the start script must run with Administrative privileges.
   - (UNIX or Linux) ./**start.sh**
   - (OS/X) **./start.command**

   The application can take several minutes to start.
5. Enter the following URL in a web browser:

   /localhost:3434/cars-app

   The home page opens. Since Forward Cars uses port 3434, that port cannot be in use by any other application. If this port is not available, Forward Cars does not start successfully

Step 3 - Open DevTest Workstation

Follow these steps:

1. Start the Enterprise Dashboard by clicking **Start Menu**, **All Programs**, **DevTest Solutions**, **Enterprise Dashboard Server**. Wait until the "Grails application running" message appears.

2. Start the registry by clicking **Start Menu**, **All Programs**, **DevTest Solutions**, **Registry**.

3. Click **Start**, **All Programs**, **DevTest Solutions**, **Workstation**.
   The Set DevTest Registry dialog opens.

4. Select a registry and click **OK**.

5. The Login dialog opens. Enter a valid username and password and click **Login**.
   The DevTest Workstation opens.

6. Start CA Service Virtualization by clicking **Start**, **All Programs**, **DevTest Solutions**, **Virtual Service Environment**.

Step 4 - Record a JMS Session to Apply for a Loan in Forward Cars

Follow these steps:

1. In DevTest Workstation, open the Cars v9 project. The project is located in the *DEVTEST_HOME*\Projects\ folder.

2. Double-click the project.config file. The JMS assets display in the Assets panel:

3. You can verify the assets to ensure that the recording and playback is set up properly. Double-click an asset, then in the asset editor, click the green Verify button.

4. View the log window for a success or failure message.

5. Click **VSE Recorder** in the main toolbar.

6. On the **Basics** tab, click the blue folder icon at the bottom left of the window.

7. Navigate to Projects\Cars v9\Data, and select the **loan-offers-jms.vrs** file. This file contains the required configuration for the recording.

8. Click **Open**.

9. On the **Basics** tab, in the **Write image to** field, rename the file from newimage to a unique name.

10. In the **Model file** field, enter the same unique name with a .vsm extension.

11. Click **Next**.

    The proxy recorder setup page opens. The JMS-related configuration is already established.

    Three request channels for the loan are submitted to three different banks, as specified in the cars-loan.properties file (Step 2). The recorder forwards those requests to the live request queues, where they are processed by the Forward Cars application. The banks then process and return the response to the proxy response queue.

12. Click **Next** to start the recording session.

    The recording feedback page opens. The green status icons indicate that the setup was successful.

13. Keep DevTest Workstation open, with the recording feedback page displayed.

Step 5 – Apply for a Loan Application in Forward Cars

Follow these steps:

1. Navigate to Forward Cars.

2. From the home page, click **Guest** in the upper-right corner, then select **Log In**.

3. In the **Name** field, enter **lisa.simpson**.

4. In the **Password** field, enter **golisa**.

5. Click **Sign In**.
   The home page opens.

6. From the home page, click **Loan Application**.
   The Finance Application displays for Lisa Simpson.

7. Click **Apply**.
   An Application Submitted message opens.

8. Click **OK**.

Step 6 - Complete the JMS Recording in DevTest Workstation

Follow these steps:

1. Back in DevTest Workstation, the table in the Summary tab lists each request channel and response channel.
   Notice that the Bank Partners Response Channel has increased to three, for each loan application response that was generated in Forward Cars.

2. Click **Next** to finish the recording.

3. Keep the Request and Response data protocols as-is and click **Next**.

4. Under **Remaining Transactions**, the three requests for each bank display. Under **Token Identification**, the three responses for the loan application display.

5. Click **Next**, then **Finish**.

6. On the Project Panel in DevTest Workstation, the new JMS service image and virtual service model display under the VirtualServices folder.

Step 7 - Change the Forward Cars Application ID to a Magic String

Once the recording is complete, you need to change the magic string of the application ID so it can play back properly.

Follow these steps:

1. In DevTest Workstation Project panel, double-click the new virtual service image that you created.
   The Transactions page opens.

2. The Response panel on the right shows the response to the requests. When you deploy the service and apply for a loan application, the <applicationId> that displays in the Response panel must be changed to the default (or Meta) response. In this example, the application ID is 2.

   Replace 2 with the following magic string:
   {{=request_applicationId;}}

3. Click the **Save** icon.

Now that you have recorded the JMS service image, you can deploy it from DevTest Workstation or DevTest Portal.

Tutorial 4: HTTP/S Virtual Service

In this CA Service Virtualization tutorial, you:

- Create and activate a configuration file

- Configure the VSE Recorder
- Record a test case
- Deploy a virtual service model
- Test against a virtual service model

This section contains the following pages:

- Prerequisites
- Process for Creating and Testing a VSI

Prerequisites

The following steps are prerequisites to completing this tutorial:

- DevTest Workstation and VSE are installed.
- You have reviewed the following pages:
  - Glossary
  - Using CA Service Virtualization
  - CA Service Virtualization Concepts

Process for Creating and Testing a VSI

After you complete the prerequisites, complete the following steps:

- Step 1 - Start DevTest Workstation and Portal
- Step 2 - Start VSE
- Step 3 - Start Demo Server
- Step 4 - Run a Test Case
- Step 5 - Create a Configuration File
- Step 6 - Activate the Configuration File
- Step 7 - Configure the VSE Recorder
- Step 8 - Record the Test Case
- Step 9 - Deploy the Virtual Service Model
- Step 10 - Test Against the Virtual Service Model
- Review the Tutorial

Step 1 - Start DevTest Workstation and Portal

Follow these steps:

1. Ensure that the registry is running.
   - If your computer has DevTest Server installed:
   - Start the Enterprise Dashboard by selecting **DevTest Solutions**, **Enterprise Dashboard Server** from the list of programs in your Windows Start menu. Wait until the "Grails application running" message appears.
   - Start the registry by selecting **DevTest Solutions, Registry** from the list of programs in your Windows Start menu.
   - If your computer has DevTest Workstation installed, use a registry that is running on another computer.
2. Select **DevTest Solutions, DevTest Portal UI** from the list of programs in your Windows Start menu.
3. Select **DevTest Solutions**, **Workstation** from the list of programs in your Windows Start menu.
   The **Set DevTest Registry** dialog opens.
4. Select a registry and click **OK**.
5. The **Login** dialog opens. Enter a valid username and password and click **Login**.
6. Continue with "Step 2 - Start VSE".

Step 2 - Start VSE

Follow these steps:

1. Click **Start**, **All Programs**, **DevTest Solutions**, **Virtual Service Environment**.
   When the "Virtual service environment is now ready" line appears, VSE has initialized.
2. Continue with "Step 3 - Start Demo Server".

Step 3 - Start Demo Server

When DevTest Workstation and VSE are both running, you can start the demo server that you use to complete this tutorial.

Follow these steps:

1. Select **DevTest Solutions**, **DevTest Demo Server** from the list of programs in your Windows Start menu.
   When the "Started in XXs:YYms" line appears, the demo server has started.
2. Continue with "Step 4 - Run a Test Case".

Step 4 - Run a Test Case

To verify that LISA Bank is available, run a test case in the **Interactive Test Run (ITR)**.

Follow these steps:

1. Double-click **webservices.tst** in the **Project** panel.
   The webservices test case opens in a tab.

2. Click **ITR** to open the ITR.

3. To run the test case on the demo server, click **Automatically Execute Test** .
   When the test completes successfully, you know that the demo server is available and the test case runs correctly.

4. Close the **ITR** window.

5. Close the test case by clicking **X** on the editor tab.

6. Continue with "Step 5 - Create a Config File".

Step 5 - Create a Config File

The **webservices** test case is the source application driver. You use the VSE Recorder in DevTest Workstation to listen and record the transactions of this test case. Insert the VSE Recorder between the source client application and the live system. The live LISA Bank system web service is on localhost port 8080. Configure your endpoint so that it uses the listen port for the VSE Recorder, 8001. The recorder intercepts the request and response on the way to and from the live web service application on port 8080.

Each of the steps in your test case contains a field where you can specify an endpoint for the step. By default, DevTest uses the {{ENDPOINT1}} property to define this endpoint. By using a property and not hard-coding the endpoint, you can quickly change the end point for each step by changing the project configuration for the {{ENDPOINT1}} value.

The default configuration is named project.config, and is created automatically for a new project. The project.config file is located in the **Configs** folder in the **Project** panel. You can also create a configuration file.

Follow these steps:

1. Right-click the **Configs** folder in the **Project** panel and select **Create New Config**.

2. Enter the name of the new configuration: **VSRecorder**.

3. Click **OK**.
   The properties editor opens the VSRecorder config file.

To add a property to the configuration file:

1. To add a row, click **Add** at the bottom of the properties editor.

2. Select ENDPOINT1 from the **Key** field.

3. Enter the following value in the **Value** field:

   ```
   http://localhost:8001/itkoExamples/EJB3UserControlBean?wsdl
   ```

> **Note:** The port from the test case (8080) is changed to the listen port for the VSE Recorder (8001). The recorder can intercept the request and response on the way to and from the live web service application on port 8080.

4. Click **Save** on the main toolbar.

5. To close the config file, click **X** on the editor tab.

6. Continue with "Step 6 - Activate Config File".

Next Step.

Step 6 - Activate Config File

For DevTest to use the newly created config file as the primary or active config file, activate it.

To activate the config file:

1. In the **Project** panel, right-click the **VSRecorder** config file.

2. Select **Make Active**.

To confirm that the ENDPOINT1 value from the VSRecorder config file is being used:

1. Double-click the **webservices** test case in the **Project** panel.
   The editor opens.

2. Double-click the **Add User** test step and verify that the **Endpoint** is on port 8001.

Screen shot highlighting the Endpoint field on the Add User Object XML test step

3. Continue with "Step 7 - Configure the VSE Recorder".

Step 7 - Configure the VSE Recorder

With the endpoint set, you are now ready to configure the VSE Recorder.

Follow these steps:

1. Click **VSE Recorder** in the main toolbar.
   On the **Basics** tab, configure the settings for the virtual service image, which stores the data. Configure the setting for the virtual service model, which creates the transaction workflow model.

2. Replace the default service image name, *newimage.vsi*, in the **Write image to** field with *VSETutorial.vsi*.

3. Leave the **Import traffic** field blank.
   For this tutorial, you will not import an existing traffic file.

4. Select the transport protocol **HTTP/S**.
   For this tutorial, you do not de-identify the data because you do not pass sensitive data (such as Social Security numbers or bank account numbers). Continue to the **Export to** field, which is used to export a raw traffic file of the recording. The exported file serves as raw backup of the entire recording session.

5. Click **Browse**, name the file **rt_VSETutorial**, and save it in the **VServices** folder.
   The "rt_" prefix indicates this file is a raw traffic file.

6. Click **Browse** to create the virtual service model file.

7. Enter the name **VSETutorial** and click **Save**.

8. Leave the **VS Model style** selections as they are, and click **Next**.
   On the next window, you designate the ports for the recorder.

9. By default, the recorder listens and records on port 8001, which you set as the endpoint in the VSRecorder config file. Add **localhost** to the **Target host** field.

10. Update the **Target port**, or the port the live LISA Bank system transactions are on, to **8080**
    .

Screenshot of VSI Recorder port selection screen, listening on port 8001 and Target port of 8080

11. Select the **Gateway** option for the **Recorder passthru style**.
    With these settings completed, you can start recording.

12. Click **Next**.

13. Continue with "Step 8 - Record the Test Case".

Step 8 - Record the Test Case

Follow these steps:

1. Click **Next** to start recording.
   As the window shows, the recorder is listening on port 8001 to the target port 8080. The window also shows a session and transaction count. Because the test case has not been run, the count is at 0. When the test case finishes running, this window has three transactions that are counted.

2. In DevTest Workstation, click the test case tab, open the **ITR**, and execute the steps automatically.

3. Navigate to the recorder and verify that the three transactions were successfully recorded. With the test case recorded, you can end the recording session by clicking **Next.** DevTest processes the recording and presents a list of transactions. For more advanced virtual service transactions, modify the post-recording model and data on the next window. For our test case, we are ready to continue, so click **Next**.
   On the next window, the base path, binding, and logic settings are correct.

4. Click **Next**.
   The next window indicates that a Web Services (SOAP) data protocol handler has been selected for the request side.

5. Leave this selection unchanged, and click **Next**.

6. Make no changes on the next window and click **Enter**.
   DevTest processes the recording a final time and creates the virtual service model and the virtual service image files.

Screenshot of VSI Recorder processing complete screen

You can open and edit each of these files in DevTest Workstation.

7. To see the virtual service model and virtual service image files, select the **Open the service image** and **Open the generated virtual service model** check boxes.

8. Click **Finish**.
   DevTest closes the recorder and opens the selected files in DevTest Workstation.

9. Continue with "Step 9 - Deploy the VSM".

Step 9 - Deploy the VSM

To deploy the Virtual Service Model (VSM):

1. To close the **Demo Server** window, click **X** in the upper right corner.
   Closing this window shuts down the demo server, which hosts LISA Bank. This shutdown leaves the test case without a system under test and allows you to use the virtual service you created.

2. Open DevTest Workstation and open the **DevTest Portal**, which provides the UI for managing virtual services.

3. Click **Monitor**, **Virtual Service**s, **VSE**.

4. In DevTest Workstation, right-click **VSETutorial** virtual service model in the **Project** panel and select **Deploy/Redeploy to VSE**.
   On the **Deploy Virtual Service** dialog, the virtual service model and the VSRecorder config file are selected. The **Group Tag** is blank. The **Concurrent capacity** is set to 1, **Think time scale** is at 100%, and **If service ends, automatically restart it** is selected. Because the **Start the service on deployment** check box is selected, the virtual service starts when you deploy it.
   To share this model and the configurations, or to deploy it remotely, click **Save as MAR** to create a MAR file and distribute the model as necessary. All the prepopulated configurations are correct, so click **Deploy**, and the virtual service is deployed and displays in the DevTest Portal.

VSETutorial virtual service model appears in the VSE tab of the DevTest Portal

5. To return to the DevTest Portal, click the DevTest Portal tab in your browser.
   The VSETutorial virtual service model appears in the VSE tab.
   The virtual service is started and ready to process transactions, which are tracked in the **Txn Count** column. The service initially shows that it has processed 0 transactions.

6. Run the test case and return to this window.

7. Continue with "Step 10 - Test Against the VSM".

Step 10 - Test Against the VSM

To test against the Virtual Service Model (VSM):

1. Open the **webservices** test case and run it in the **ITR**.

2. Return to the virtual service when the ITR completes.
   The transaction count is now at 3, confirming you are testing against the virtual service.

   > **Note:** You can also stage and execute a test case or suite against the virtual service.

VSETutorial with 3 transactions

3. In DevTest Workstation, right-click the **webservices** test case in the **Project** pane and select **Stage a Quick Test**.

4. Enter **10** for the **Number of Instances**.

5. Clear the **If test ends, restart it** check box, and click **OK**.

6. Click **Play** with the **Test Monitor** window open and let the test run.
   When you return to the DevTest Portal after running the test case with 10 instances, you see the transaction count is 33.

VSETutorial with 33 transactions

7. Continue with "Review the Tutorial".

Review the Tutorial

In this tutorial, you created and tested a virtual service to review the basic functionality of VSE.

In the tutorial, you:

- Created and activated a configuration file
- Configured the VSE Recorder
- Recorded a test case
- Deployed a virtual service model
- Used the ITR and staged a test to test against a virtual service model

Tutorial 5: Virtual Service from WSDL

This tutorial describes how to create a virtual service from a WSDL document, using DevTest Portal. You will use the creditreport.wsdl file included in the examples folder of your installation directory to create this virtual service.

- Step 1 - Start DevTest Portal
- Step 2 - Start VSE
- Step 3 - Create a Credit Inquiry Virtual Service
- Step 4 - View the Virtual Service
- Step 5 - Configure Forward Cars to Point to Virtual Service
- Step 6 - Run the Baseline Test
- Step 7 – Apply for a Loan Application in Forward Cars
- Step 8 - Reset the Configuration File

Step 1 - Start DevTest Portal

1. Ensure that the Enterprise Dashboard, registry, a coordinator server, and a simulator are running.
   If your computer has DevTest Server installed:

1.
   - Select **DevTest Solutions**, **Enterprise Dashboard Server** from the list of programs in your Windows Start menu. Wait until the "Grails application running" message appears.
   - Select **DevTest Solutions**, **Registry** from the list of programs in your Windows Start menu.

   If your computer has DevTest Workstation installed, use a registry that is running on another computer.

   - Select **DevTest Solutions**, **Coordinator Server** from the list of programs in your Windows Start menu.
   - Select **DevTest Solutions**, **Simulator Server** from the list of programs in your Windows Start menu.

2. Select **DevTest Solutions**, **Portal** from the list of programs in your Windows Start menu.

3. Open a supported browser and enter **http://localhost:1507/devtest**.
   If the registry is on a remote computer, replace **localhost** with the name or IP address of the remote computer.
   If the port number was changed from the default value **1507**, use the new port number.

4. Click **Log in**.
   The **Login** dialog opens. Enter a valid username and password and click **Login**.

Step 2 - Start VSE

1. Select **DevTest Solutions**, **Virtual Service Environment** from the list of programs in your Windows Start menu.
   When the "Virtual service environment is now ready" line appears, VSE has initialized.

Step 3 - Create a Credit Inquiry Virtual Service

1. Navigate to DevTest Portal.

2. In the **Current Project** drop-down on the upper right corner of the page, select **Cars v9**.

3. From the left nav bar, select **Create**, **Virtual Service** from the left navigation menu in DevTest Portal.

4. Click **Specification**.

5. Provide basic information about the virtual service:

   a. Select **VSE** as the VSE server.

   b. Enter **credit-inquiry** in the Service Name field.

   c. Enter **Virtual Credit Inquiry Service for Forward Cars** in the Description field.

6. Click **Select File** and select the **creditreport.wsdl** file in the *LISA_HOME*/examples/Data directory.

7. Click **Continue**.

8. Click **Finish** in the Configure Connection & Resources dialog.

   The Collect Transactions pane opens. This pane displays a list of transactions and their requests and responses. For more information, see View and Edit VSE Transactions.

9. Accept the default selections and click **Next**.

   The Configure pane opens. This pane lets you select which data protocols to use. For more information, see Select Data Protocols.

10. Accept the default selections and click **Next**.

    The Configure DPH pane opens. This pane lets you configure any protocols that you selected in the Configure pane. For more information, see Configure Data Protocols.

11. Accept the default selections and click **Next**.

The Save pane opens.

12. Select **Save & Deploy Virtual Service**.

13. Select the **Create baseline tests** check box.

14. Click **Save**.

15. Click **OK** in the dialog that confirms your virtual service was successfully created and saved.

Step 4 - View the Virtual Service

1. From the left nav bar, select **Monitor, Virtual Service Environments, VSE**.

   The Monitor Virtual Services window appears.

2. You can see that the **credit-inquiry** virtual service is running.

   > **Note:** If the service is not running, Click in the **Actions** column for the **credit-inquiry** service and select **Start**.

3. Note the port where the virtual service is running. You will need this port number in the next step.
   Also note that the transaction count (Txn Count) is zero.

Step 5 - Configure Forward Cars to Point to Virtual Service

1. If Forward Cars is running, stop the application.

2. Open *LISA_HOME* \DemoServer\carsdemo**cars-loan.properties**.

3. Edit the following property to point to the port for the virtual service that you created (instead of the default port 3600):

   - **cars.partner.creditInquiryEndPoint**=http://localhost:3600

4. Save your changes and close the property file.

5. Start Forward Cars. Go to the *DEVTEST_HOME\DemoServer\carsdemo* directory and start the script for your operating system:

   - (Windows) **start.bat**

     If the Windows computer has UAC enabled (the default for Windows 7), the start script must run with Administrative privileges.

   - (UNIX or Linux) ./**start.sh**

   - (OS/X) **./start.command**

   The application can take several minutes to start.

6. Enter the following URL in a Web browser:

   /localhost:3434/cars-app

   The home page opens. Because Forward Cars uses port 3434, that port cannot be in use by any other application. This port must be available, for Forward Cars to start successfully.

Step 6 - Run the Baseline Test

1. From the left nav bar, select **Manage, Tests**.

2. Click in the **Actions** column for the credit-inquiry baseline test and select **Run with Options**.

3. Select the configuration file created for the credit-inquiry baseline test from the **Configuration** list.

4. Click **Run**.
   The Monitoring Tests tab opens. When the test is complete, a green checkmark displays in the Status column.

5. Open the VSE tab again.
   If you closed the tab, you can select **Monitor, Virtual Service Environments, VSE**. If the tab is still open, you can select it at the top of the main DevTest Portal pane.

6. Notice that the transaction count increased.

Step 7 – Apply for a Loan Application in Forward Cars

1. Navigate to Forward Cars (/localhost:3434/cars-app).

2. From the home page, click **Guest** in the upper-right corner, then select **Log In**.

3. In the **Name** field, enter **lisa.simpson**.

4. In the **Password** field, enter **golisa**.

5. Click **Sign In**.
   The home page opens.

6. From the home page, click **Loan Application**.
   The Finance Application displays for Lisa Simpson.

7. Click **Apply**.
   An Application Submitted message opens.

8. Click **OK**.

9. Open the VSE tab again.
   Notice that the transaction count increased. If you repeat step 7, you can see that the transaction count is going up. This indicates that the virtual service is returning those loan responses.

Step 8 - Reset the Configuration File

When you are done with this tutorial, it is a good idea to stop your virtual service and reset cars-loan.properties to the default port.

1. Open the VSE tab again.
2. Click in the **Actions** column for the **credit-inquiry** service and select **Stop** .
3. Click **OK** in the confirmation dialog.
4. If Forward Cars is running, stop the application.
5. Open *LISA_HOME* \DemoServer\carsdemo**cars-loan.properties**.
6. Edit the following property to point to the default port:
   - **cars.partner.creditInquiryEndPoint**=http://localhost:3600
7. Save your changes and close the property file.

Tutorial 6: Virtual Service from Swagger (OpenAPI)

This tutorial shows you how to create a virtual service from a Swagger (OpenAPI) 2.0 or 3.0 document.

You will use a Swagger document that is included in the Forward Cars demo application. The Swagger document describes the inventory service.

This tutorial assumes that you have a Windows installation of DevTest Solutions.

For information about the supported browsers, see System Requirements.

Step 1 - Start DevTest Workstation

You will use DevTest Workstation later in the tutorial to create and deploy the virtual service.

Follow these steps:

1. Ensure that the Enterprise Dashboard and the registry are running.
   If your computer has DevTest Server installed:
   - Select **DevTest Solutions**, **Enterprise Dashboard Server** from the list of programs in your Windows Start menu. Wait until the "Grails application running" message appears.
   - Select **DevTest Solutions**, **Registry** from the list of programs in your Windows Start menu.

   If your computer has DevTest Workstation installed, you will need access to a registry that is running on another computer.

2. Select **DevTest Solutions**, **Workstation** from the list of programs in your Windows Start menu.
   The Set DevTest Registry dialog opens.

3. Select a registry and click **OK**.
   The Login dialog opens.

4. Enter a valid username and password and click **Login**.

Step 2 - Start DevTest Portal

You will use DevTest Portal later in the tutorial to monitor the virtual service.

Follow these steps:

1. Select **DevTest Solutions**, **Portal** from the list of programs in your Windows Start menu.

2. Open a supported browser and enter **http://localhost:1507/devtest**.
   If the registry is on a remote computer, replace **localhost** with the name or IP address of the remote computer.
   If the port number was changed from the default value **1507**, use the new port number.

3. Enter a valid username and password and click **Log in**.

Step 3 - Start the Virtual Service Environment

The virtual service that you deploy requires a virtual service environment to be running.

To start the virtual service environment, select **DevTest Solutions**, **Virtual Service Environment** from the list of programs in your Windows Start menu. Wait until the "Virtual service environment is now ready" message appears.

Step 4 - Create a Virtual Service from Swagger

DevTest Workstation includes a wizard that lets you create a virtual service from a Swagger document.

The following graphic shows the wizard step in which you specify the document.

Screen capture of virtual service wizard.

In this procedure, you also deploy the virtual service.

Follow these steps:

1. Go to DevTest Workstation.
2. Create a new project.
3. From the **File** menu, select **New**, **VS Image**, **From Swagger Specification**.
4. In the **Write image to** field, change the default file name to **cars.vsi**.
5. In the **Model file** field, type **cars.vsm**.
6. Click **Next**.
7. In the **URL** field, browse to the **LISA_HOME\examples\Data** directory and select the **cars-inventory-swagger.json** file.
   The **Endpoint** field and **Methods** pane are populated with information from the Swagger document. You cannot modify the **Endpoint** field.
8. In the **Listen on port** field at the bottom of the window, change the default value to **3550**.
9. Click **Next**.
10. Add the REST data protocol to the list of request-side data protocols.
11. Click **Next**.
    The step for configuring the REST data protocol opens.

12. Click **Next**.

13. Click **Finish**.
    The **cars.vsi** and **cars.vsm** files are added to the project.

14. Right-click **cars.vsm** in the **Project** panel and select **Deploy/Redeploy to VSE@Default**.

15. Click **Deploy**.
    The virtual service is deployed.

16. Click **OK**.

Step 5 - Access the Monitor in DevTest Portal

DevTest Portal lets you view the runtime behavior of the virtual service. For example, you can view the number of transactions that the virtual service has processed.

The following graphic shows the location of the port number and transaction count.

Screen capture of port number and transaction count.

Follow these steps:

1. Go to DevTest Portal.

2. From the left navigation menu, select **Monitor, Virtual Service Environments, VSE**.
   The VSE window appears.

3. Observe that the **cars** virtual service is running on port 3550 and has a transaction count of 0.

Step 6 - Configure Forward Cars to Point to Virtual Service

You update a port number in one of the Forward Cars properties files to match the value from Step 4.

> **Note:** After you finish the tutorial, we recommend that you stop Forward Cars and reset the port number to the default value of 3500.

Follow these steps:

1. If Forward Cars is running, stop it.

2. Open the **cars-esb.properties** file in the **LISA_HOME\DemoServer\carsdemo** directory.

3. Set the value of the **inventory.port** property to **3550**.

4. Save the **cars-esb.properties** file.

5. Start Forward Cars by selecting **DevTest Solutions**, **Cars Example Servers** from the list of programs in your Windows Start menu. The application can take several minutes to start.

Step 7 - Perform Actions in Forward Cars

Now that the virtual service is running, you can interact with it.

To access Forward Cars, you use port 3434. The port that you changed in the properties file is a different, back-end port.

As shown in the following graphic, the virtual service will return broken images and garbled text. This behavior is expected when you create virtual services from specification documents.

Screen capture of broken image.

Follow these steps:

1. Open a supported browser and enter **http://localhost:3434/cars-app.**
2. Click **Search**.
3. Click **View Details**.
4. Click **OK**.
5. Return to DevTest Portal.
6. Observe that the transaction count is now 5.

Review

In this tutorial, you learned how to:

- Create a virtual service from a Swagger 2.0 document
- Deploy the virtual service
- Monitor the runtime behavior of the virtual service

Application Insight Tutorial

In this tutorial, you generate transactions by using the LISA Bank demo application. You then view the transactions in the **Analyze Transactions** window of DevTest Portal.

- Prerequisites
- Step 1 - Configure the Capture Levels
- Step 2 - Generate Transactions from the Demo Application
- Step 3 - View the Transactions in the Analyze Transactions Window

Prerequisites

Ensure that the following components are running:

- Enterprise Dashboard Server
- Registry
- Demo server
- Broker
- Portal

Step 1 - Configure the Capture Levels

Each protocol that the DevTest Java Agent can capture has a capture level.

The default capture level is **Counts**. To perform the steps later in this tutorial, you must set the capture level for the relevant protocols to **Full Data**.

The following graphic shows the **Adjust Captures for Protocol** pane in the **Agents** window.

Screen capture of Adjust Captures for Protocol pane.

Follow these steps:

1. Open DevTest Portal. If the portal is running on a local computer, the URL is **http://localhost:1507/devtest**.

2. Select **Settings, Agents** from the left navigation menu.
   The **Agents** window opens.

3. In the **Agents** pane, select the **JBoss_LISABank** agent.

4. If the **Adjust Captures for Protocol** pane is collapsed, expand the pane.

5. Ensure that the capture levels for the following protocols are set to **Full Data**:

   - HTTP Server
   - JDBC
   - EJB
   - Logging

Step 2 - Generate Transactions from the Demo Application

DevTest Solutions includes a demo application named LISA Bank.

In this procedure, you log in to the demo application, create an account, and log out.

Follow these steps:

1. In a web browser, enter **http://localhost:8080/lisabank**. If the demo server is running on another computer, replace **localhost** with the host name or IP address of the remote computer.
   The login page opens.

2. In the **Name** field, type **lisa_simpson**.

3. In the **Password** field, type **golisa**.

4. Click **Login**.
   The welcome page opens.

5. Create an account by performing these steps:

   a. Click **New Account**.

   b. In the **Account Name** field, type **My Checking**.

   c. In the **Initial Balance** field, replace the default value with **500**.

   d. Click **Add Account**.

6. Click **Log Out**.

Step 3 - View the Transactions in the Analyze Transactions Window

The **Analyze Transactions** window displays the transactions that have been generated. These transactions are also referred to as *paths*.

The following graphic shows a list of transactions in the **Analyze Transactions** window. By default, the transactions appear in reverse chronological order.

	Name	Start Time	Wall Time	CPU Time	Agent
	/lisabank/logout.do	10/06/2017 5:17:03 PM 785	83 ms	78 ms	JBoss_LISABank
	/lisabank/createaccount.do	10/06/2017 5:17:00 PM 530	647 ms	202 ms	JBoss_LISABank
	/lisabank/buttonclick.do	10/06/2017 5:16:35 PM 035	382 ms	140 ms	JBoss_LISABank
	/lisabank/login.do	10/06/2017 5:16:21 PM 087	2721 ms	2215 ms	JBoss_LISABank
	/lisabank/home.do	10/06/2017 5:16:00 PM 869	455 ms	327 ms	JBoss_LISABank
	/lisabank/	10/06/2017 5:16:00 PM 677	179 ms	156 ms	JBoss_LISABank
	/lisabank/	10/06/2017 5:16:00 PM 622	0 ms	0 ms	JBoss_LISABank

Screen capture of transactions in Analyze Transactions window.

The following graphic shows a path graph for one of the transactions.

Screen capture of path graph for one transaction.

Follow these steps:

1. Return to DevTest Portal.
2. Select **Application Insight**, **Analyze Transactions** from the left navigation menu.
3. Locate the transaction that has the name **/lisabank/createaccount.do**.
4. Click the link in the **Name** column.
   The transaction detail dialog opens. The upper area contains the path graph. The lower area contains information about the selected node.
5. Click an EJB node in the path graph.
6. Review the information that appears in each tab.
7. Click a JDBC node in the path graph.
8. Review the information that appears in each tab.
9. Close the dialog.

Glossary

This glossary defines common terms that are used in DevTest Solutions.

assertion

An *assertion* is an element that runs after a step and all its filters have run. An assertion verifies that the results from running the step match the expectations. An assertion is typically used to change the flow of a test case or virtual service model. Global assertions apply to each step in a test case or virtual service model. For more information, see Assertions.

asset

An *asset* is a set of configuration properties that are grouped into a logical unit. For more information, see Assets.

audit document

An *audit document* lets you set success criteria for a test, or for a set of tests in a suite. For more information, see Building Audit Documents.

baseline

A *baseline* is a test case that helps you determine whether a system is still functioning the same way later. For more information, see Creating Baselines.

companion

A *companion* is an element that runs before and after every test case execution. Companions can be understood as filters that apply to the entire test case instead of to single test steps. Companions are used to configure global (to the test case) behavior in the test case. For more information, see Companions.

configuration

A *configuration* is a named collection of properties that usually specify environment-specific values for the system under test. Removing hard-coded environment data enables you to run a test case or virtual service in different environments simply by changing configurations. The default configuration in a project is named project.config. A project can have many configurations, but only one configuration is active at a time. For more information, see Configurations.

conversation tree

A *conversation tree* is a set of linked nodes that represent conversation paths for the stateful transactions in a virtual service. Each node is labeled with an operation name, such as **withdrawMoney**. An example of a conversation path for a banking system is **getNewToken**, **getAccount**, **withdrawMoney**, **deleteToken**. For more information, see Using CA Service Virtualization.

coordinator

A *coordinator* receives the test run information as documents, and coordinates the tests that are run on one or more simulator servers. For more information, see Coordinator Server.

data protocol

A *data protocol* is responsible for parsing requests in CA Service Virtualization. For more information, see Using Data Protocols.

data set

A *data set* is a collection of values that can be used to set properties in a test case or virtual service at run time. Data sets provide a mechanism to introduce external test data into a test case or virtual service. Data sets can be created internal to DevTest, or externally (for example, in a file or a database table). For more information, see Data Sets.

de-identify

De-identifying (formerly called *desensitizing)* is used to convert sensitive data to user-defined substitutes. Credit card numbers and Social Security numbers are examples of sensitive data. For more information, see De-identifying Data.

event

An *event* is a message that is broadcast, informing any interested parties that an action has occurred. For more information, see Understanding Events.

filter

A *filter* is an element that runs before and after a test step, enabling you to change the data in the result, or to store values in properties. For more information, see Filters.

group

A *group*, or a *virtual service group,* is a collection of virtual services that have been tagged with the same group tag so they can be monitored together in DevTest Portal.

Interactive Test Run (ITR)

The *Interactive Test Run (ITR)* utility lets you run a test case or virtual service model step by step. You can change the test case or virtual service model at run time and rerun to verify the results. For more information, see Interactive Test Run (ITR) Utility.

magic date

During a recording, a date parser scans requests and responses. A value matching a wide definition of date formats is translated to a *magic date*. Magic dates are used to verify that the virtual service model provides meaningful date values in responses. An example of a magic date is {{=doDateDeltaFromCurrent("yyyy-MM-dd","10");/*2012-08-14*/}. For more information, see Magic Strings and Dates.

magic string

A *magic string* is a string that is generated during the creation of a service image. A magic string is used to verify that the virtual service model provides meaningful string values in the responses. An example of a magic string is {{=request_fname;/chris/}}. For more information, see Magic Strings and Dates.

match tolerance

Match tolerance is a setting that controls how CA Service Virtualization compares an incoming request with the requests in a service image. The options are Exact, Signature, and Operation. For more information, see Match Tolerance.

metrics

Metrics let you apply quantitative methods and measurements to the performance and functional aspects of your tests, and the system under test. For more information, see Generating Metrics.

Model Archive (MAR)

A *Model Archive (MAR)* is the main deployment artifact in DevTest Solutions. MAR files contain a primary asset, all secondary files that are required to run the primary asset, an info file, and an audit file. For more information, see Working with Model Archives (MARs).

Model Archive (MAR) Info

A *Model Archive (MAR) Info* file is a file that contains information that is required to create a MAR. For more information, see Working with Model Archives (MARs).

navigation tolerance

Navigation tolerance is a setting that controls how CA Service Virtualization searches a conversation tree for the next transaction. The options are Close, Wide, and Loose. For more information, see Navigation Tolerance.

node

Internal to DevTest, a test step can also be referred to as a *node*, explaining why some events have node in the EventID.

path

A *path* contains information about a transaction that the DevTest Java Agent captured. For more information, see Using Application Insight.

path graph

A *path graph* contains a graphical representation of a path and its frames. For more information, see Path Graph.

project

A *project* is a collection of related DevTest files. The files can include test cases, suites, virtual service models, service images, configurations, audit documents, staging documents, data sets, monitors, and MAR info files. For more information, see Project Panel.

property

A *property* is a key/value pair that can be used as a run-time variable. Properties can store many different types of data. Some common properties include LISA_HOME, LISA_PROJ_ROOT, and LISA_PROJ_NAME. A configuration is a named collection of properties. For more information, see Properties.

quick test

The *quick test* feature lets you run a test case with minimal setup. For more information, see Stage a Quick Test.

registry

The *registry* provides a central location for the registration of all DevTest Server and DevTest Workstation components. For more information, see The Registry.

service image

A *service image* is a normalized version of transactions that have been recorded in CA Service Virtualization. Each transaction can be stateful (conversational) or stateless. For more information, see Service Images.

simulator

A *simulator* runs the tests under the supervision of the coordinator server. For more information, see Simulator Server.

staging document

A *staging document* contains information about how to run a test case. For more information, see Building Staging Documents.

stitching

The process by which the DevTest Java Agent assembles partial transactions into complete transactions.

subprocess

A *subprocess* is a test case that another test case calls. For more information, see Building Subprocesses.

test case

A *test case* is a specification of how to test a business component in the system under test. Each test case contains one or more test steps. For more information, see Building Test Cases.

test step

A *test step* is an element in the test case workflow that represents a single test action to be performed. Examples of test steps include Web Services, JavaBeans, JDBC, and JMS Messaging. A test step can have DevTest elements, such as filters, assertions, and data sets, attached to it. For more information, see Building Test Steps.

test suite

A *test suite* is a group of test cases, other test suites, or both that are scheduled to execute one after other. A suite document specifies the contents of the suite, the reports to generate, and the metrics to collect. For more information, see Building Test Suites.

think time

Think time is a setting that controls how long a test case waits before executing a test step. For more information, see Add a Test Step (example) and Staging Document Editor - Base Tab.

transaction frame

A *transaction frame* encapsulates data about a method call that the DevTest Java Agent intercepted. For more information, see Business Transactions and Transaction Frames.

virtual service model (VSM)

A *virtual service model* receives service requests and responds to them in the absence of the actual service provider. For more information, see Virtual Service Models.

virtual service environment (VSE)

The *virtual service environment (VSE)* is a DevTest Server component that you use to deploy and run virtual services. For more information, see Using CA Service Virtualization.

Using

Using CA Application Test

- Using CA Application Test
- The Registry
- Coordinator Server
- Simulator Server
- Command-Line Utilities
- Using DevTest Portal with CA Application Test
- Using DevTest Workstation with CA Application Test

Using CA Service Virtualization

- Using CA Service Virtualization
- SV Installation and Configuration
- CA Service Virtualization Concepts
- Using DevTest Portal with CA Service Virtualization
- Using DevTest Workstation with CA Service Virtualization
- VSE Manager
- VSE Commands
- Java Agent VSE Properties

Using Application Insight

- Getting Started with Application Insight
- Agent Configuration
- Analyzing Business Transactions
- Using the Shelf

- Create and Manage Tickets
- Creating Virtual Services
- Creating Baselines
- Create Data Sets
- Create Request/Response Pairs from Transaction Frames
- Document Transactions
- Native MQ Application Insight
- Application Insight Command-Line Tool
- VS Traffic to CA Application Insight Companion
- DevTest Extension for Google Chrome
- CAI Top N and Metrics Reports

Using the SDK

- Using the SDK
- Integrating Components
- Testing Integrated Components
- Extending DevTest Solutions
- Extending Test Steps
- Extending Assertions
- Extending Filters
- Custom Report Generators
- Custom Report Metrics
- Custom Companions
- Using Hooks
- Custom Data Sets

Using CA Application Test

CA Application Test lets you build and run test cases and suites.

Note: DevTest Portal is a web-based application that provides simpler access to the most commonly used workflows for DevTest products. DevTest Workstation is the main application for advanced users and contains the full range of functionality available within the DevTest products. For a quick summary of the functionality available in the portal, see DevTest Portal Functionality.

The following video uses the multi-tier-combo test case to highlight key features in CA Application Test.

More Information:

- Using DevTest Portal with CA Application Test
- Using DevTest Workstation with CA Application Test

The Registry

The registry provides a central location for the registration of all DevTest Server and DevTest Workstation components.

- Start the Registry
- Create a Named Registry
- Change the Registry

The registry tracks the locations of any DevTest run-time components and provides lookup to their locations for each registered component. The registry also provides the web consoles for server administration, reporting, CVS, and Application Insight. The common JMS provider that is used for the component-to-component communication is started and run in the registry process.

The fully qualified name of the registry is **tcp://hostname-or-IP-address:2010/registry-name**. For example:

- tcp://localhost:2010/Registry
- tcp://myserver:2010/Registry
- tcp://myserver.example.com:2010/Registry
- tcp://172.24.255.255:2010/Registry

DevTest Workstation includes the Registry Monitor, which lets you monitor the test cases, simulators, coordinators, and virtual environments for a test suite.

Start the Registry

> **Note:** Identity and Access Manager must be running before you start a registry. If Identity and Access Manager is not running, you will not be able to start a registry.

If the registry is not installed locally, log in to the server where it is installed. To start the registry, follow one of these steps.

Valid on Windows

- Open a command prompt, navigate to the **LISA_HOME\bin** directory, and enter the following command:

  ```
  Registry
  ```

- Select **DevTest**, **Registry** from the list of programs in your Windows Start menu.

- Double-click the **Registry.exe** file in the **LISA_HOME\bin** directory.

Valid on UNIX

- Open a terminal window, navigate to the **LISA_HOME/bin** directory, and enter the following command:

  ```
  ./Registry
  ```

  Wait until the following message appears:

  ```
  Registry Ready.
  ```

Create a Named Registry

To create a named registry, run the registry executable with the **-n** option:

```
LISA_HOME\bin\Registry -n RegistryName
```

The following examples create a registry with the name **registry1**.

Valid on Windows

```
cd C:\Lisa\bin
Registry -n registry1
```

Valid on UNIX

```
cd Lisa/bin
./Registry -n registry1
```

Change the Registry

While you are working in DevTest Workstation, you can switch to another registry.

Follow these steps:

1. Select **System**, **Registry**, **Change DevTest Registry** from the main menu.
   The **Set DevTest Registry** dialog appears.

2. Enter the registry name, or open the drop-down list and select a previously used registry.

3. Select or clear the check box.
   Selected: The Set DevTest Registry dialog opens when you start DevTest Workstation.
   Cleared: DevTest Workstation connects to the last connected registry on startup.

4. Click **OK**.

> **Tip:** If you see a "Too many open files" exception in **registry.log**, especially on Linux, ask the administrator to increase per-user and system-wide open file limits in "/etc/security/limits.conf" and "/etc/sysctl.conf".
>
> Do not forget to restart DevTest components afterward.

Coordinator Server

Coordinator servers receive the test run information as documents, and coordinate the tests that run on one or more simulator servers.

- Create a Coordinator Server
- Monitor Coordinator Servers

When you stage a test, you choose which coordinator server to coordinate the test from. You also specify a staging document that specifies which simulator servers to use when running instances of the test. Any coordinator server can launch instances on any simulator server (based on the staging document).

The coordinator server manages metric collection and reporting, and communicates the test data to DevTest Workstation for monitoring purposes. A DevTest Server environment can have, and commonly does have, multiple coordinator servers.

The coordinator server runs tests when presented with a staging document, test case, and configuration.

The **lisa.coordName** property sets the default name of a coordinator server.

```
lisa.coordName=Coordinator
```

The fully qualified name of a coordinator server is **tcp://hostname-or-IP-address:2011/coordinator-name**.

Create a Coordinator Server

To create a coordinator server, run this command:

```
LISA_HOME\bin\CoordinatorServer -n CoordinatorServerName -m RegistryName
```

The following examples create a coordinator server with the name **coordinator1**. The associated registry has the name **registry1**.

Valid on Windows

```
cd C:\DevTest\bin
CoordinatorServer -n coordinator1 -m tcp://localhost:2010/registry1
```

Valid on UNIX

```
cd DevTest/bin
./CoordinatorServer -n coordinator1 -m tcp://localhost:2010/registry1
```

You can display the version number by using the **--version** option.

For information, see Running Server Components as Services.

Monitor Coordinator Servers

If DevTest Workstation is attached to the associated registry, then a running coordinator server appears in the Registry Monitor.

To monitor coordinator servers from the Registry Monitor:

1. Click the **Toggle Registry** icon in DevTest Workstation.
2. Click the **Coord Servers** tab.

The coordinator servers also appear in the DevTest Portal .

To monitor coordinator servers from the DevTest Portal:

1. Select **Monitor**, **Tests**, from the DevTestPortal.
2. The Coordinator Server column shows the coordinators.
   Click the arrow at the column heading to sort by coordinator.

Simulator Server

Simulator servers run the tests under the supervision of the coordinator server.

- Create a Simulator Server
- Monitor Simulator Servers

Virtual users or test instances are created and run on the simulator servers. The number of virtual users, and hence the number of simulator servers that are deployed, depend on the nature of the tests being performed. Each virtual user communicates with the client system.

For large tests with many virtual users, virtual users can be distributed among several simulator servers.

The **lisa.simulatorName** property sets the default name of a simulator server.

```
lisa.simulatorName=Simulator
```

The fully qualified name of a simulator server is **tcp://hostname-or-IP-address:2014/simulator-name**.

Create a Simulator Server

To create a simulator, run the following command:

```
LISA_HOME\bin\Simulator -n SimulatorName -m RegistryName
```

The following examples create a simulator with the name **simulator1**. The associated registry has the name **registry1**.

Valid on Windows

```
cd C:\DevTest\bin
Simulator -n simulator1 -m tcp://localhost:2010/registry1
```

Valid on UNIX

```
cd DevTest/bin
./Simulator -n simulator1 -m tcp://localhost:2010/registry1
```

You can specify the number of virtual users for this simulator by using the -i option. For example:

```
Simulator -n simulator1 -m tcp://localhost:2010/registry1 -i 100
```

You can display the version number by using the --version option.

When creating a simulator, you can override the default port number by adding a colon and the nondefault port number to the -n option. For example:

```
Simulator -n testSim1:35001
```

To run multiple simulators, use the command prompt to create the simulators as shown previously.

If the port number is not specified in the name, the simulator tries port 2014 first. If 2014 is taken, the simulator tries 2015, 2016, and so on, up to port 2024 before stopping.

For more information about port usage, see Default Port Numbers.

For information about running the simulator as a service, see Running Server Components as Services.

For information about configuring multiple simulators, see the following video.

Monitor Simulator Servers

If DevTest Workstation is attached to the associated registry, then a running coordinator server appears in the Registry Monitor.

To monitor simulator servers from the Registry Monitor:

1. Click the **Toggle Registry** icon in DevTest Workstation.
2. Click the **Simulators** tab.

Simulator servers also appear in the DevTest Portal.

To monitor simulator servers from the DevTest Portal:

1. Select **Monitor**, **Tests** from the DevTest Portal.
2. The Status tab displays the simulators used for each test.

Command-Line Utilities

The **LISA_HOME\bin** directory contains the following command-line utilities:

- **Application Insight Command-Line Tool**
  The **PFCmdLineTool** command lets you perform various Application Insight tasks from the command line. For more information, see Application Insight Command-Line Tool.

- **CVS Manager**
  CVS Manager lets you add or remove monitors to the CVS Dashboard through a command-line option. For more information, see CVS Manager.

- **Make Mar**
  Make Mar lets you show the contents of MAR info files (stand-alone or in an archive), or create model archive files from MAR info files. For more information, see Make Mar.

- **Service Image Manager**
  Service Image Manager is used to import transactions into a service image (new or existing), and to combine two or more service images. For more information, see ServiceImageManager .

- **Service Manager**
  Service Manager is used to verify the status of, reset, or stop a running server process. For more information, see Use the ServiceManager.

- **Test Runner**
  Test Runner is a "headless" version of DevTest Workstation with the same functionality but no user interface. For more information, see Test Runner.

- **VSE Manager**
  VSE Manager is used for managing virtual service environments. For more information, see VSE Manager Commands.

Using DevTest Portal with CA Application Test

DevTest Portal is a web-based application that provides simpler access to the most commonly used workflows for DevTest products. DevTest Workstation is the main application for advanced users and contains the full range of functionality available within the DevTest products.

This section provides detailed information about using the Portal to build and run test cases and suites for supported features.

More Information:

- DevTest Portal
- Create an API Test
- Monitor Tests
- Manage Test Artifacts

Create an API Test

The **API Test** window lets you build a test manually or with request/response pairs from an existing API test or from the file system.

Follow these steps:

1. Select **Create**, **API Test** in DevTest Portal.
2. Enter the name of your test in the **Name** field. After the test runs, the application appends a datestamp and timestamp to the name you enter to ensure that it is unique.
3. Select a project from the **Project** drop-down list, or enter a project name. If the project exists, this API test is added to it. If it does not exist, it is created.
4. Enter the base portion of the URL for the request/response pairs in the **Base URL** field. The base URL is the URL to which a request connects when it is executed. For example, enter:

   ```
   http://localhost:8080
   ```

5. (Optional) Enter a short description of the test in the **Description** field.

More Information:

- Create a Test by Importing R/R Pairs from a Test
- Create a Test by Importing R/R Pairs from a File
- Create a Test Manually
- Edit an API Test

Create a Test by Importing R/R Pairs from a Test

You can create a test by loading an existing API test into the Test Editor.

Follow these steps:

1. From the **R/R Pair Editor** toolbar, click **Import R/R pairs from existing tests** .
   The **Load Existing API Tests** window opens.
2. The list shows all the projects available on the resource server and the API tests available under each project.
   If you have not created an API test in DevTest Portal, you will not see any items in the drop-down list.
3. To filter the project names, enter a string in the **Filter** field.
4. To import an API test, select it and click **Load**.

Create a Test by Importing R/R Pairs from a File

You can create a test by loading request/response pairs from a zip file into the Test Editor.

Follow these steps:

1. From the **R/R Pair Editor** toolbar, click **Import R/R pairs from local zip files** .
   The **Import R/R Pairs** window opens.

2. To designate a zip file containing r/r pair files, browse the file system, or drag and drop a zip file into the window.

Import R/R Pairs window

Note the requirements for the r/r pair files in the **Resources** section of the window. If you upload a zip file, the request/response pairs must be in the root of the zip file.

> **Note:** If you created your zip file using the native OSX compress utility, this utility can create an additional _MACOSX folder in the file. This folder prevents DevTest from finding your request/response pairs. You must delete this folder or use a different utility to create the zip file before you can import your file successfully.

3. Select the r/r pair files to import. To select all r/r pair files, select the **File Name** check box.

4. Click **Import**.

Ensure that the installation directory is enabled for Read/Write access for all users. Otherwise, the import of r/r pairs will fail because the import will store data in the projects folder.

Create a Test Manually

To create a test manually, use the Test Editor.

Follow these steps:

1. Click **Add a R/R Pair** .
   The first R/R pair, numbered 0, appears. Its default name is **Baseline-0**. To rename it, double-click the name. When you are done editing, press **Enter** to exit the edit mode.

2. On the right pane, under the R/R pair name, select the **Web Service** from the drop-down list. Valid options are REST and SOAP.

3. Perform one of the following actions:

   - For REST, enter a **Type** (GET, POST, PUT, or DELETE) and a **Resource Path** (the path to the web service).

   - For SOAP, enter a **Version** (SOAP 1.2 or SOAP 1.1), an **Endpoint** (the specific endpoint for this request) and a **SOAP Action** (optional for SOAP 1.2).

4. Enter the text for the request in the **Request** field.

5. Enter the text for the response in the **Assertion on Response** field.

Edit an API Test

To edit a test, use the **Test Editor**.

DevTest Portal test editor

To change the name of a test, double-click the name, enter a new name, and click **Enter**.

To create an API test, click **Create**. If the test is created successfully, the new test is automatically opened in a **Test Edit** page.

To create the API test and run it, click **Create and Run**. If the test is created successfully, the monitoring portlet appears.

> **Important!** The most likely and common cause for DevTest to append a test file name with ".invalid" is that a Java class needed for the test (for example, in a Dynamic Java Execution step) is not available in DevTest. When such a test file is saved or closed, DevTest makes a copy of the test and renames the copy of the test to ".invalid" extension.

Monitor Tests

The **Monitoring Tests** window lets you see API tests, test cases, and test suites that have been run on CA Application Test. You can also stop or kill a running test from this window.

When no tests or suites meet the selection criteria, you can click **Create a Test** or **Execute a Test**.

Click the filter button to toggle the search options.

- Filter Tests and Suites
- Monitor Tests and Suites
- Stop or Kill Tests

Filter Tests and Suites

To filter the test cases and test suites that are returned from the database, click **Toggle filter options** .

Complete the following fields:

- **From**
  Defines the beginning date and time for test cases and test suites.

- **To**
  Defines the ending date and time for test cases and test suites.

- **Executed By**
  Designates the user who submitted the test case or test suite.

To display test cases and test suites that were run in the past week, leave the default value of **Last 7 Days**. The drop-down list allows you to select various date and time ranges.

When the results are displayed, you can further filter using the filter options at the top of the pane.

- **Filter by Name**
  To show only the results from a specific test case or suite, enter a name or partial name of the test or suite.

- **Filter by Executed By**
  To show only those test cases and suites that a specific user submitted, enter a userid or partial userid.

You can also filter by type, by selecting either **Tests** or **Suites**. To filter by result, select **Passed**, **Failed**, **Aborted**, or **Running**.

To select how many rows to display, click **Page Size** .

Monitor Tests and Suites

The **Suites/Tests** pane displays the test cases and suites that match the filter criterion entered.

The Summary icons display the number of cycles that passed, failed, aborted, or are running.

The following fields are displayed:

- **Name**
  Displays the name of the test case or suite. A plus sign or minus sign precedes a suite name, which you click to expand or collapse the test cases in the suite. Test suites that were created with the DevTest Portal have a date stamp and timestamp that is appended to the suite name. Mouse over the name and an Info icon appears to the right of the name. Click that icon for more information about the test or suite.

- **Status**
  Indicates the status of the test case or test suite: Passed, Failed, Failed to Stage, Aborted, or Running.

- **Errors/Warnings**
  Shows the number of errors (shown on a gray circle) and warnings (shown on a yellow circle) for each test case and test suite.

- **Start Time**
  Displays the start date and time for each test case and test suite.

- **Duration**
  Shows, in seconds, the duration of the execution of the test case or test suite.

- **Executed By**
  Displays the userid of the submitter of the test case or test suite.

- **Coordinator Server**
  Displays the coordinator server that is used by this test case or test suite.

- **Description**
  Displays the description of the test case or test suite.

Stop or Kill Tests

Follow these steps:

1. Put your cursor on the name of a running test and right-click.

2. From the menu, select **Stop Test** or **Kill Test**.
   Stop Test stops the test after it reaches the next logical end/fail. The test does not start a new cycle.
   Kill Test stops the test immediately after the current step completes.

Display Details of Tests and Suites

You can display details of tests and suites from the Suites/Tests pane. Click a test case name to show details about the execution of that test case. The **Status** tab displays.

Test Cases with More Than One Cycle

For a test case that has more than one cycle, the Cycle Details are displayed, showing the status of each cycle.

Monitor Test Results with Cycles

The following fields are displayed for each cycle:

- **Start Time**
  Displays the start date and time for each cycle in the test case.

- **Errors/Warnings**
  Shows the number of errors (shown on a gray circle) and warnings (shown on a yellow circle) for each test case and test suite.

- **Status**
  Indicates the status of the test case or test suite:

  - Passed
  - Failed
  - Aborted
  - Running

- **VUser**
  Shows the number of virtual users for each cycle.

- **Cycle Number**
  Displays an index of cycle runs. If there is only one cycle in the test run, **Cycle Number** is 0.

- **Duration**
  Shows, in seconds, the duration of the execution of the test case or test suite.

- **Coordinator Server**
  Displays the coordinator server that is used by the cycle.

- **Simulator Server**
  Displays the simulator server that is used by the cycle.

Cycle Details

To display cycle details, perform one of the following actions:

- For test cases that have more than one cycle, double-click the cycle timestamp

- For test cases that have only one cycle, double-click the test case name

Monitor Multi-Tier Combo in Portal with RR

Each test step in the test case displays. All data sets, filters (global and nonglobal), assertions (global and nonglobal), companions, and event actions that are associated with each step also display.

To expand all the steps in the rest case, click **Expand All**. To collapse all the steps, click **Collapse All**.

To the left of each element name, an arrow displays. To display detailed information about the element, including request/response, events, errors or warnings, and properties, click the arrow.

If a step produced errors or warnings, the number of errors and warnings displays in a gray circle (for errors) or orange circle (for warnings) to the right of the step name. Mouse over the number to show the error or warning text.

If a step has a request and response, click **Request** or **Response** to display the request or response details.

To search the text of a request or a response, click **Search** . Enter a query string and click **Enter**.

Note: If you do not see events or properties for a step that had events or properties, inspect the staging document for that test to be sure that it is set to record events, requests, responses, screenshots, and properties that are set or referenced. You can use the **Run1User1CycleShowAll** staging document to ensure all is captured.

To expand the view of a request or a response, click **Expand** .

To view the staging document that a test uses, click the **Staging Doc** tab.

To view documentation that is associated with a test, click the **Documentation** tab.

Display Details of Assertions, Filters, Data Sets, and Companions

To display the details of the results of assertions, filters, data sets, and companions, click the arrow to the left of the element name.

Details of Assertions

You can select to see details for fired assertions or evaluated assertions by selecting the appropriate check box on the filter bar.

Details for fired assertions display the message that the assertion produced and the result of the assertion.

Result of a fired assertion in the DevTest Portal, Monitor Tests

Details for evaluated assertions display the message that the assertion produced and the result of the assertion.

Result of an evaluated assertion in the DevTest Portal, Monitor Tests

Details of Filters

Details for filters display the properties that the filter set.

Result of a filter in the DevTest Portal, Monitor Tests

Details of Data Sets

Details for data sets display the data set contents and the properties that the data set changed.

Details of Companions

Details for companions display what action the companion performed.

Companions Panel

Monitor with CVS (Continuous Validation Service)

Continuous Validation Service (CVS) lets you schedule tests and test suites to run regularly over an extended time period.

CVS has a 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 can monitor each test run.

A monitor contains a single test or an entire test suite. A service contains one or many monitors.

A coordinator manages the tests, which run on a simulator. State is maintained in a database on the DevTest registry.

You can select to either run a service or individual monitors in it from the **CVS Dashboard**.

Prerequisite

CVS runs in a DevTest Server environment.

There must be a coordinator server and a simulator server running, registered with a DevTest registry.

For details on setting up the DevTest Server environment, see Installing.

You can open the **CVS Dashboard** from DevTest Portal or DevTest Workstation.

Open the CVS Dashboard

To open the CVS Dashboard from DevTest Portal:

Select **Monitor**, **CVS** from the left navigation panel.

To open the CVS Dashboard from DevTest Workstation:

Select **View**, **CVS Dashboard** from the left navigation panel.

> **Note:** Closing the **CVS Dashboard** or closing DevTest Workstation does not interfere with the CVS scheduled tasks. When you reconnect to the registry, you see the dashboard with the current data and status information.

CVS Dashboard Overview

The **CVS Dashboard** has the following tabs. To refresh the list that is displayed on the **CVS Dashboard**, click **Refresh** at the bottom of the window.

Overview Tab or Monitor Tab

In DevTest Portal, the first tab on the **CVS Dashboard** is named "Overview." In DevTest Workstation, the first tab is named "Monitor." Their functionality is the same.

The Overview tab displays a list of all the monitors (tests and test suites) that are added to the **CVS Dashboard**. Your **CVS Dashboard** lists all monitors running on an attached DevTest registry, not only the ones that you initiate.

The **CVS Dashboard** opens with the **Overview** tab active.

You can add many monitors to run in one or many services. A service contains one or more monitors.

You can schedule to run a service. All monitors in that service are run at the scheduled time. The services are run in the background at scheduled intervals.

You can either add the monitors in one service or add the monitors in different services. All the monitors added in one service are shown added after that service name.

Top Panel

All the tests in a specific service that have run are shown with a colored ball:

- **Green:** Completed tests.
- **Red:** Failed tests.
- **Yellow:** Tests that had an error in their staging documents.

The icons show the Test Run Completion, Failure, Error, Staging Error, or Unknown Error. These icons show the state of the monitor after it has run.

At the right side, the top panel shows a graphical representation of the jobs that are currently running.

Bottom Panel

The bottom panel displays the status of each monitor that is run in a service.

The bottom panel shows all the monitors that are running or have finished their last run. The monitors that have finished running completely (are not scheduled to run again) do not display in this list.

The table shows the following information:

- **Monitor Name:** The name that is given to the monitor
- **Running:** Shows whether this monitor is running (true/false)
- **Active:** Shows whether this monitor is Active (true/false)
- **Documents:** The names of the documents, such as test case, staging, and suite documents that are 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

You can customize the columns by arranging them in sorted order or by showing or hiding the columns.

To see the monitor-related information, double-click a monitor.

Toolbar

The **Toolbar** tab includes a toolbar with the following buttons:

- **Re/Deploy Monitor:** Deploy or redeploy a monitor to the dashboard. See Deploy a Monitor to CVS.
- **View Attributes:** View monitor-related information.
- **Delete:** Delete a monitor from the dashboard.
- **Run Immediately:** Run the monitor immediately.
- **Activate:** Click to deactivate the monitor. Deactivate means it remains in the list, but does not run. You can also click this button to reactivate a previously deactivated monitor.
- **View Results:** Click to view the results of the run in the Report Viewer.

When you run the **CVS Dashboard** directly from DevTest Workstation (by selecting to view the dashboard from DevTest), you also see the following buttons:

- Refreshes the list on the dashboard
- Auto refreshes the list on the dashboard after a specific period.

When you run the CVS Dashboard from DevTest Portal, you see the following buttons on the upper right side of the window:

- Refreshes the list on the dashboard.
- Auto refreshes the list on the dashboard after a specific period.

When you run the **CVS Dashboard** directly from your browser, these buttons do not appear.

Graph Tab

The **Graph** tab displays the tests in the **CVS Dashboard** in a graphical format. This tab provides a graphical summary of the tests that have passed and failed.

The **Graph** tab consists of three graphical displays, which show and track the last 60 minutes of activity.

If you run the **CVS Dashboard** from DevTest Portal, the **Select Service** field lets you enter specific monitors to view.

If you run the **CVS Dashboard** from DevTest Workstation or a web browser, the **Filter Monitor** lets you select the specific monitors to view.

Follow these steps:

1. Click **Filter Monitors** at the top of the window.
   A list of available monitors opens.

2. Select the monitor that you want to review.
   The details display in the following sections:

   - **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.

   - **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. Hover over a monitor to show the average response time in a tooltip.

   You can select to display the test/suite to be seen in the graphical format in the **Pass/Fail By Monitor**.

Events Tab

The **Events** tab displays various events corresponding to the stages of the monitor run, such as starting of a test run, ending, and failing.

The following information is displayed for the last 60 minutes of activity:

- **Time Stamp:** The time the event occurred.

- **Service Name:** The name of the service in which the monitor resides.

- **Monitor Name:** The name of the monitor in which the event occurred.

- **Documents:** A list of the documents that are associated with the monitor for which the event occurred.

- **Action:** The action reported by the event. Start events are blue, staging error events are yellow, completion events are green, and failed events are 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. If you double-click a **Message** field, you can see the full text of the message.

If you run the **CVS Dashboard** from DevTest Portal, refreshes the list on the dashboard. Selecting the **Auto Refresh** check box automatically refreshes the display.

If you run the **CVS Dashboard** from DevTest Workstation or a web browser, you can display the events in real time by selecting **Auto Refresh** at the bottom of the panel, or you can refresh the list manually by clicking **Refresh**, with the **Auto Refresh** box that is cleared. You can also adjust the column sizes so that all columns can display on the window.

Test Cycle with Errors

At times, the test cycle fails. The cycle that has errors can be seen in the **Actions** list in different color. You can double-click any event to see the associated messages in an expanded window.

Deploy a Monitor to CVS

To deploy a monitor to CVS, use one of the following techniques.

- Deploy a Monitor through DevTest Portal
- Deploy a Monitor through DevTest Workstation
- Deploy a Monitor through the CVS Dashboard
- Deploy a Monitor through the cvsMonitors Directory
- Deploy a Monitor through CVS Manager

Deploy a Monitor through DevTest Portal

Before you can perform this procedure, a Model Archive for the monitor must be created.

Follow these steps:

1. In the **Overview** tab of the **CVS Dashboard**, click **Re/Deploy**.
   The **Re/Deploy Monitor** window for the new monitor opens.

2. Enter the name of the MAR file in the **Model Archive** field.

3. If the MAR has previously been deployed, select or clear the **Replace the monitor if it exists** check box.

4. Click **Re/Deploy**.
   The monitor is added to the **CVS Dashboard**.

Deploy a Monitor through DevTest Workstation

The only prerequisite for this approach is the existence of a test case or suite.

Follow these steps:

1. From the **Project** panel at the left of DevTest Workstation, right-click a test case or suite and select **Deploy as monitor to CVS**.
   You see a collection of tabs to provide monitor information, the same as needed for Creating Monitor MAR Info Files.

2. Click **Deploy** to deploy the monitor.

Deploy a Monitor through the CVS Dashboard

Before you can perform this procedure, a Model Archive for the monitor must be created.

Follow these steps:

1. In the **Monitor** tab of the **CVS Dashboard**, click **Re/Deploy Monitor**.
   The **Re/Deploy Monitor** window for the new monitor opens.

2. Enter the name of the MAR file in the **Model Archive** field.

3. If the MAR has previously been deployed, select or clear the **Replace the monitor if it exists** check box.

4. Click **Re/Deploy**.
   The monitor is added to the **CVS Dashboard**.

Deploy a Monitor through the cvsMonitors Directory

Before you can perform this procedure, a Model Archive for the monitor must have been created
.

Follow these steps:

1. Go to the **LISA_HOME\cvsMonitors** directory.

2. Add the MAR file to the directory.
   Later, you can update the monitor by placing a new version of the MAR file in the same directory.

Deploy a Monitor through CVS Manager

For more details about the command-line option for deploying a CVS Monitor, see CVS Manager.

Email Notification Settings

Every time that you schedule a new test in the CVS utility, you must deploy a monitor in the **CVS Dashboard**.

In the monitor window, you can also set an email address. If you set an email address, you receive an email notification every time the monitor is run in the specified time period.

> To send an email to multiple email addresses, enter an email address for a distribution list in this field.

To set the email notification, update the **lisa.properties** file.

You also must change the mail server configuration and must enter the mail host as shown in the following example.

```
# If you use performance monitoring alerts, this is the "from" email address of th
ose 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=localhost
```

To uncomment this information in the **lisa.properties** file, remove the **#** from the first column of the file.

Restart DevTest to set the mail server configuration.

When you deploy the test to run in CVS, complete the following field:

- **Notification email:** Enter the email address for the email notification of the test run result.

Every time the test is run, you receive an email.

Run a Monitor Immediately

When you add the services in the top panel, they are run automatically in the background at the scheduled time intervals.

To run a specific monitor immediately, use the toolbar.

Follow these steps:

1. Deploy the monitor.
2. When it is listed in the bottom panel of the **Overview** tab, select it.
3. Click **Run Immediately** to run the monitor immediately.
   You do not see the monitor running in the bottom panel.
4. To see the monitor that has run, go to the **Events** tab of the **CVS Dashboard,** with a suffix **-now** (for example, **Test 3-now**).

View Test Details

You can view the details of a test run in the **CVS Dashboard**.

Follow these steps:

Do one of the following:

- Double-click the service (only available when running the **CVS Dashboard** through DevTest Workstation).
- Double-click the monitor.

The test-related detail window opens.

You can also display the **Events** tab in the CVS Dashboard. You can adjust the column sizes so that all columns can display. To see the full text of the message, double-click the **Message** field.

Any reports that are generated during scheduled test runs are available and can be seen in the Report Viewer.

CVS Manager

The CVS Manager command-line utility lets you manage the set of monitors that are deployed to the Continuous Validation Service. The utility is located in the **LISA_HOME\bin** directory.

This utility has the following format:

```
CVSManager [-h] [-m registry-spec] [-d archive-file] [-r archive-file] [-l] [-D] [-A] [-e] [x] [-X] [-s name] [-n name] [-u username] [-p password] [--version]
```

- **-h, --help**
  Displays help text.

- **-m *registry-spec*, --registry=*registry-spec***
  Defines the registry to which to connect.

- **-d *archive-file*, --deploy=*archive-file***
  Deploys the specified model archive to CVS as a monitor. The monitor that is defined in the archive must refer to a monitor and service name combination that does not exist.

- **-r *archive-file*, --redeploy=*archive-file***
  Redeploys the specified model archive to CVS as a monitor. The monitor that is defined in the archive must refer to a monitor and service name combination that exists.

- **-l, --list**
  Lists the currently deployed monitors with information about each.

- **-D, --pause**
  Pauses the scheduled execution of the indicated monitor.

- **-A, --resume**
  Resumes the scheduled execution of the indicated monitor.

- **-e, --execute-now**
  Causes the indicated monitor to be executed immediately, regardless of its schedule. This action does not affect any scheduled executions of the monitor.

- **-x, --remove**
  Removes a monitor from CVS. Use the service name and monitor name arguments to indicate which monitor to remove.

- **-X, --remove-all**
  Removes all monitors from CVS. If a service name is specified, then only monitors defined with that service name are removed.

- **-s *name*, --service-name=*name***
  Specifies the service name for the monitors to affect.

- **-n *name*, --monitor-name=*name***
  Specifies the monitor name to affect.

- **-u *username*, --username=*username***
  Specifies the DevTest security user name. This option is required.

- **-p *password*, --password=*password***
  Specifies the DevTest security password. This option is required.

- **--version**
  Print the version number and exit.

Example: Deploy Monitor

The following example deploys a monitor to CVS.

```
CVSManager -d monitor.mar -u user -p password
```

Example: Delete Monitor

The following example deletes that same monitor (assuming the service and monitor names).

```
CVSManager -x -s OrderManager -n CheckOrders -u user -p password
```

The following example deletes all monitors in the OrderManager service.

```
CVSManager -X -s OrderManager -u user -p password
```

The following example deletes all monitors.

```
CVSManager -X -u user -p password
```

Manage Test Artifacts

To manage test artifacts, on the left navigation panel, select **Manage** and select from the following options:

- **API Tests**: API tests that were created with DevTest Portal
- **Tests**: Tests that were created through DevTest Workstation
- **Test Suites**: Test suites that were created through DevTest Workstation

To select a project, use the **Project** drop-down. Projects available in the drop-down are projects that are in the **Projects** directory of the **LISA_HOME** directory.

- Manage API Tests
- Manage Tests
- Manage Test Suites

Manage API Tests

The **Manage API Tests** option lets you edit tests that were created in DevTest Portal.

Follow these steps:

1. Select Manage, API Tests.
   The **Manage API Tests** window opens.
2. Select a test from the list of tests, and click it.
   The **Test Editor** opens.
3. Use the **Test Editor** as described in Edit an API Test.

Manage Tests

You manage tests from the **Tests** window.

Follow these steps:

1. Select **Manage**, **Tests**
   The **Tests** window opens.

2. To view information and documentation about a test, click **Edit** .

3. To delete a test, click **Delete** .

4. To run a test, click **Options** and select **Run**.

You can run a test with the options to change the configuration file, staging document, or coordinator server.

If you are running a test that was created as a baseline from a virtual service, be sure to select **Run with Options**. Then ensure that the values of WSSERVER and WSPORT in the configuration file are set correctly. In some cases, the default value of WSSERVER is CHANGE_ME.

Follow these steps:

1. From the Tests window, click **Options** and select **Run with Options**.
   The **Run Test Case** dialog opens.

2. To change the staging document for the test, select a staging document from the drop-down list.

3. To change the coordinator server for the test, select a coordinator server from the drop-down list.

4. To change the configuration file for the test, select a file name from the drop-down list in the **Configuration** field.

5. To display the keys and values in the configuration file, click **Configuration details** .
   The blue keys and values are the system properties and cannot be edited or deleted. The keys and values in black are user-defined properties and can be edited and deleted from the project.config file.

6. To edit the project configuration and alternate configuration properties, double-click the key or value and edit the field.

7. To add a new property to the project configuration, perform the following steps:

   a. Click **Options** and select **Add New Property**.

   b. Double-click the **Key** and **Value** field and enter a name.

8. To revert an alternate configuration value to the default setting, click **Revert** .

9. To delete a project configuration property, click **Delete** .

10. To add temporary properties for one runtime, perform the following steps:

a. Click **Additional Properties** and click **Add.**
A table displays with a new row.

b. Double-click the **Name** and **Value** field and enter a name.

c. To delete the additional property, click **X**.

11. Click **Run**.

To download the MAR file with the test, click **Options** and select **Download MAR**.

Add Local Properties for One Project

To locally display properties for a project (config file), add custom properties to the project.config file. These properties are then available in the scope of that project.

Follow these steps:

1. Open a DevTest Solutions project in the DevTest Workstation.

2. Open project.config file that is located in ***PROJECT_ROOT*/Configs**.
The Properties Editor opens.

3. Click the plus icon at the bottom to add a property.

4. Enter **lisa.report.properties** in the **Key** field.

5. Enter ***BuildNumber, VersionNumber, AppName*** in the **Value** field.
Or
Edit and save the project.config file in the file system by adding the following line to the file:

   lisa.report.properties

   For example:

   ```
   lisa.report.properties=BuildNumber,VersionNumber,AppName
   ```

6. In DevTest Portal, click the test case, then select **Run with options**.

7. Click the **Additional Properties tab** (at the bottom).
The locally defined properties for the config file now display under the drop-down list, after the delimiter at the bottom of the list.

Add Global Properties for All Projects

If you want properties to consistently display for all projects within the scope of the DevTest Solutions/instance, you can add custom global properties to the DevTest System Properties File (lisa.properties). These properties affect all projects created within that DevTest Solutions instance.

Follow these steps:

1. Open the DevTest System Properties File (select **System, Edit DevTest Properties** from the DevTest Workstation main menu).

2. Include the properties as a new line:
   lisa.global.report.properties
   For example:

   ```
   lisa.global.report.properties=BuildNumber,VersionNumber,AppName
   ```

3. Close the file.

4. In DevTest Portal, click the test case, then select **Run with options**.

5. Click the **Additional Properties** tab (at the bottom).
   The defined global properties now display under the drop-down list, after the delimiter at the bottom of the list.

Manage Test Suites

You can manage test suites from the **Test Suites** window.

Follow these steps:

1. Select **Manage**, **Test Suites**
   The **Test Suites** window opens.

2. To view information and documentation about a test suite, click **Edit** .

3. To delete a test suite, click **Delete** .

4. To run a test suite, click **Options** and select **Run**.

You can run a test suite with the options to change the configuration file.

Follow these steps:

1. Click **Options** and select **Run with Options**.
   The **Run Test Suite** dialog opens.

2. To change the configuration file for the test suite, select a file name from the drop-down list in the **Configuration** field.

3. To display the values in the configuration file, click **Configuration details** .
   Keys and values in blue are the system properties and cannot be edited or deleted. The keys and values in black are user-defined properties and can be edited and deleted from the project configuration.

4. To edit the project configuration and alternate configuration property, double-click the key or value and edit the field.

5. To add a new property to the project configuration, perform the following steps:

   a. Click **Options** and select **Add New Property**.

   b. Double-click the Key and Value field to enter the name.

6. To revert an alternate configuration value to the default setting, click **Revert** .

7. To delete a project configuration property, click **Delete** .

8. To add temporary properties for one runtime, perform the following steps:

   a. Click **Additional Properties** and click **Add.**
   A table displays with a new row.

   b. Double-click the **Name** and **Value** field and enter a name.

   c. To delete the additional property, click **X**.

9. Click **Run**.

To download the MAR file for the test suite, click **Options** and select **Download MAR**.

Testing Reports

You can view reports with execution and performance data for test cases or test suites that were run in DevTest Portal. Reports can be viewed, managed, shared, and exported. These reports are divided into the following categories:

- Testing Execution Reports
- Testing Performance Reports

For more information about viewing and manipulating reports see View and Navigate Reports.

View and Navigate Reports

DevTest reports let you view and analyze detailed data about tests, test suites, virtual services, virtual service environments, and the data that is captured during business transactions. This page contains the following sections to help you view and navigate these reports:

- Open the Testing Window
- Search for Reports
- View Reports
- Filter Reports
- Legend Filters
- Export Report Data
- Send a Report Link
- Share a Report URL

For detailed information about specific reports, refer to the following pages:

- Testing Execution Reports
- Testing Performance Reports
- Virtual Service Metrics Reports
- CAI Inventory Reports

- CAI Top N Reports
- User Activity Report

Open the Testing Window

To access a particular report:

1. Click **Reporting** on the left navigation pane of the DevTest Portal.
2. Click one of the following options:
   - **CAI Top N and Metrics**
   - **Testing**
   - **Virtual Services Metrics**
   - **Administration**

   The reporting windows display a list of available reports for the selected category of reports.

Each reporting window provides two views to make it easier to find the report you want.

- Click **Category** to list the available reports by category when multiple categories are available (default view).
- Click **List** to view all available reports in a single, alphabetical list.

Search for Reports

Find reports using the **Search** bar at the top of the reporting window. To search for reports, enter all or part of the name in the **Search** bar and click . For example, enter **history** to display all the reports that contain “history” in the report title.

View Reports

Click **Launch Report** (or the name of a report) in a reporting window, to open the selected report in a new tab. The report data displays in the left pane, and the report filter options display in the Refine By pane on the right. The filter options that display can vary for each report.

Some reports have a legend that you can use to filter the data. The legend is color-coded to represent the values or properties in the report. For more information, see **Filter Reports** below.

In most reports, you can hover your mouse over graphical elements, such as line, bar, or pie charts, to view more detail about that specific element. You can also click some of the graphical elements to drill down into sub-level reports that provides even more detailed data.

When a report contains several pages, use the navigation arrows at the bottom to move between the pages. When a top-level report calls a sub-level report, use the breadcrumbs in the top left corner to navigate back to the original report.

To zoom into a portion of the report, click and drag the pointer over the area. Click **Reset zoom** to restore the report to the original display.

Filter Reports

Use the **Refine By** pane to the right of the report data and the legend at the bottom of the report to filter the information that displays in the report. Click **Get Report** to refresh the report with your selected filters.

Testing Reports Filters

The following filter options are available in the Refine By pane for Testing reports. The filter options that are available vary by report.

- **Time Range, From, To**

  Click **Set search duration** to change the default time range, From date and time, and To date and time. Select **Make Default** to save your changes as the default values for this report. Click **OK** to apply your changes to the report.

  Once you set a custom time range, you can click **Return to previous** to return to the previously set time range.

- **Auto Refresh**
  Select this check box to automatically refresh the report data on a specified interval. When you select this check box, you can also specify the interval, in seconds. The default value is 60 seconds.

- **Summarize By**
  Select a name to group the report by a specific runtime attribute. The Test Execution Summary report and the Test Execution Run Comparison report contains the Summarize By option.

- **Test Attributes**
  This option is available when tests or suites are run with attributes defined. Filter the report by selecting a name and value from drop-down list. Click **+** to add more attributes. You can add a maximum of five attributes to filter the report.

- **Suite/Test List**
  Lists the available suites or tests that are based on the time range of the report. You can search the list for a specific suite or test by entering the whole or partial name in the **Search** bar. Select the checkbox of the suite or test name you want to generate a report.

- **Test Type**
  Displays the test type that is used to filter the report. Test type options are test or suite.

- **Executed By**
  Select from the list of users.

- **Summary Period**
  Groups the results by the selected summary period. Selecting **Week** groups all the results by a weekly period starting on the first day of the week.

- **Last Run Status**
  Filtered by the status of the most recent test execution.

- **Execution Count**
  Limits the results to the tests that have executed repeatedly by the execution count or more.

- **Report Test Type**
  Filters the results by the type of test, which can be either part of a suite or a stand-alone test.

- **Interval**
  Groups the results by the selected interval period. Selecting **5 seconds** groups all the results into five second blocks. The interval is not dependent on the start of a defined calendar period.

- **Pass**
  Shows the test cases/suites that have passed. The Pass state included tests that completed with errors but did not fail according to defined assertions in the test.

- **Fail**
  Shows the test cases/suites that have failed. A failed test refers to a test with a defined assertion. When the test was run, it did not meet the criteria defined in the assertion.

- **Aborted**
  Shows the test cases/suites that have aborted. An aborted test indicates a “false failure” due to an inability to connect to the system under test.

- **Suite/Test Started Time**
  Displays a list of start times for a selected test or suite.

Virtual Service Metrics Reports Filters

The following filter options are available in the Refine By pane for Virtual Service Metrics reports. The filter options that are available vary by report.

- **Time Range, From, To**

  Click **Set search duration** to change the default time range, From date and time, and To date and time. Select **Make Default** to save your changes as the default values for this report. Click **OK** to apply your changes to the report.

  Once you set a custom time range, you can click **Return to previous** to return to the previously set time range.

- **Auto Refresh**
  Select this check box to automatically refresh the report data on a specified interval. When you select this check box, you can also specify the interval, in seconds. The default value is 60 seconds.

- **Virtual Service Environment List**
  Displays the available virtual service environments in alphabetical order.

- **Virtual Service List**
  Displays the available virtual services in alphabetical order. You can search for a specific virtual service by entering the whole or partial name. As you type, the corresponding name displays in the list. You can only select one virtual service from the list to generate a report.

- **Summary Period**
  You can select a day, week, or month period.

CAI Top N and Metrics Reports Filters

The following filter options are available in the Refine By pane for CAI Top N and Metrics reports. The filter options that are available vary by report.

- **Time Range, From, To**

  Click **Set search duration** to change the default time range, From date and time, and To date and time. Select **Make Default** to save your changes as the default values for this report. Click **OK** to apply your changes to the report.

  Once you set a custom time range, you can click **Return to previous** to return to the previously set time range.

- **Auto Refresh**
  Select this check box to automatically refresh the report data on a specified interval. When you select this check box, you can also specify the interval, in seconds. The default value is 60 seconds.

- **Category**

- Select a category or multiple categories from the list.

- **Agents**
  Select an agent or multiple agents from the list.

- **Top N Results**
  The default percentage is 10.

Administration Reports Filters

The following filter options are available in the Refine By pane for Administration reports. The filter options that are available vary by report.

- **Time Range, From, To**

  Click **Set search duration** to change the default time range, From date and time, and To date and time. Select **Make Default** to save your changes as the default values for this report. Click **OK** to apply your changes to the report.

  Once you set a custom time range, you can click **Return to previous** to return to the previously set time range.

- **Auto Refresh**
  Select this check box to automatically refresh the report data on a specified interval. When you select this check box, you can also specify the interval, in seconds. The default value is 60 seconds.

- **Authorization**
  Select to display the user activity that was allowed or denied.

- **Roles**
  Select the user role types to filter the report.

- **Users**
  Select the users to filter the report.

- **Actions**
  Select the actions to filter the report.

- **Targeted Resource**
  Enter a value to filter the report based on the content of the targeted resource column.

- **Details**
  Enter a value to filter the report with other details.

Legend Filters

Many of the reports contain a color-coded legend at the bottom of the report page.

In many reports you can click an item in the legend to hide or show the corresponding data in the report. This is particularly useful in reports that allow you to drill down into sub-level reports. This feature lets you filter your report data before drilling down. When data is hidden, the legend item turns gray.

Export Report Data

You can export the report data into a PDF or a CSV file to save or print.

Follow these steps:

1. Open a report.
2. Click to generate a PDF or click to generate a CSV file.
   The report displays in another browser window.
3. To save, click **Save**.
4. To print, click **Print**.

Send a Report Link

Click to send an email with the link to the report.

Share a Report URL

Click to view and copy the URL for a report.

Reports in DevTest Portal are not exact replicas of reports that were available in the LISA Console (1505).

More Information:

- Testing Execution Reports
- Testing Performance Reports
- Virtual Service Metrics Reports
- CAI Inventory Reports

- CAI Top N Reports
- User Activity Report

Testing Execution Reports

Execution reports contain information about how tests and suites were run. The reports contain top-level and sub-level reports. All reports are available from the Reporting option on the left navigation pane of the DevTest Portal. However, when you access a sub-level report directly from a top-level report, your filter options carry through to the sub-level report.

For example, you can access the Test Execution Detail report directly from the Reporting menu and then select the appropriate filters in this report. However, you can also select the Test Execution History report first. From this report, you can click a specific bar in the chart to view the Test Execution Detail report for the data in the selected bar.

The top-level reports are the root reports that contain information that can be filtered. The sub-level reports are determined by the information from the root report and cannot be filtered. The reports follow a sequence when you select information from charts or tables within a top-level and sub-level report.

The following diagram illustrates the reporting flow.

The following reports are available:

- Test Execution History Summary
- Test Execution Detail
- Test Execution Detail Drill Down

- Test Execution Run Comparison
- Test Execution Summary
- Test/Suite Execution History
- Suite Execution Detail

For more information about viewing and manipulating reports see View and Navigate Reports.

Test Execution History Summary

The Test Execution History Summary lets you review the number of tests that were executed during a specified time period. You can use this report to see the tests that are passing, failing, and aborting and drill down into the details of tests. This analysis provides a better understanding of why tests are passing and failing.

This test displays a separate bar in the graph for tests in the following categories:

- **Pass Count**
  The number of tests that completed successfully.

  > **Important!** Tests that completed with errors are included in the pass count rather than the fail count. To view any errors that are associated with tests in a Pass status, click the green Pass Count bar in the report. The Test Execution Detail report opens and lists all the test steps for the passing tests. Errors are shown in the Reason for Failure column.

- **Failed Count**
  The number of tests that failed to complete successfully. A Failed status indicates a test that contains an assertion. When the test was run, the criteria of the assertion were not met, which caused the test to fail.

- **Abort Count**
  The number of tests that aborted. Aborted tests are “false failures” that aborted due to an inability to connect to the system under test.

The Test Execution History Summary displays the number of tests that were executed daily, weekly, or monthly by execution state. The report displays information in a bar graph.

To view the details behind any bar in the graph, click the bar in the report. The Test Execution Detail report opens and lists more detail for all the associated test steps.

Click an item in the legend at the bottom of the report to hide or show the corresponding data. For example, click **Abort Count** to hide that data in the graph. When data is hidden, the legend item turns gray.

Test Execution Detail

The Test Execution Detail lets you view more details for each test step that was executed for a particular test execution instance. This report provides valuable information about the state and performance of individual test steps as you analyze and debug problem tests.

The report data is in a table format and includes the execution date and time, status, and duration of each step. Click the step number to view the request and response information for a specific step. The Test Request/Response Pair report opens.

Note: Step duration indicates the amount of time, in milliseconds, for the step to complete. This duration does not include any think time that is assigned to the step.

Important! Test steps that completed with errors still appear in a Pass status. The details of any error or failure messages display in the Reason for Failure column.

The final page of this report displays a summary of the staging information for this test.

You can access the Test Execution Detail report directly from the Reporting menu and then select the appropriate filters for your report. However, you can also access this report as a sub-level report from the Test Execution History Summary.

Test Execution Detail - webservices

Show details on each step executed for a particular test execution instance.

Execution Time: 07/01/2016 09:43:33 AM

Step Number	Step Name	Last Execution Date/Time	Step Status	Reason for failure	Step Duration(ms)
9001	Add User	07/01/2016 09:43:34 AM	PASS	-	1409
9002	Add User	07/01/2016 09:43:34 AM	PASS	-	1515
9003	Add User	07/01/2016 09:43:34 AM	PASS	-	1698
9004	Add User	07/01/2016 09:43:34 AM	PASS	-	1753
9005	Add User	07/01/2016 09:43:34 AM	PASS	-	1758
9006	Add User	07/01/2016 09:43:34 AM	PASS	-	1803
9007	Add User	07/01/2016 09:43:34 AM	PASS	-	1985
9008	Add User	07/01/2016 09:43:34 AM	PASS	-	2088
9009	Add User	07/01/2016 09:43:34 AM	PASS	-	2178
9010	Add User	07/01/2016 09:43:34 AM	PASS	-	2286
9011	Add User	07/01/2016 09:43:34 AM	PASS	-	2338
9012	Add User	07/01/2016 09:43:34 AM	PASS	-	2485
9013	Add User	07/01/2016 09:43:34 AM	PASS	-	2521

07/06/2016 10:20:16 AM **Executed By:** admin Page: 1 / 60

Test Execution Detail Drill Down

To export a PDF of a detailed view on each step executed for a particular test execution instance, click the **Export to pdf with Drilled Down details** icon on the right.

Home | Testing | Test execution detail

Export report to pdf with Drilled Down details

Test Execution Detail - ClearTest

Show details on each step executed for a particular test execution instance.

Start Time: 07/11/2018 03:49:35 PM End Time: 07/11/2018 03:49:39 PM Duration: 4.005 s

Step Number	Step Name	Last Execution Date/Time	Step Status	Event Description	Step Duration(ms)
603	http GET	07/11/2018 03:49:35 PM	PASS		469
603	http GET	07/11/2018 03:49:35 PM	PASS	[STEP] The assertion of type "HTTP Response Code" had outcome:	469
603	http GET	07/11/2018 03:49:35 PM	PASS	Requested URL:http://www.google.com/ Response:	469
603	http GET	07/11/2018 03:49:35 PM	PASS	THINK for 637	469
604	http GET	07/11/2018 03:49:36 PM	PASS		1314
604	http GET	07/11/2018 03:49:36 PM	PASS	Requested URL:http://www.google.com/ Response:	1314
604	http GET	07/11/2018 03:49:36 PM	PASS	[STEP] The assertion of type "HTTP Response Code" had outcome:	1314
604	http GET	07/11/2018 03:49:36 PM	PASS	THINK for 701	1314

Test Request/Response Pair

The Test Request/Response Pair report provides the request, response, and error details for a selected step within a test.

You can access this report directly from the Reporting menu and then select the appropriate filters. However, you can also access this report as a sub-level report from the Test Execution Detail report.

Note: If the Request and Response sections of this report are empty, you have defined the staging document for this test to not record this information for your reports. Typically, you chose not to record this information to save space in your report files. For more information, see Test Suite Editor - Reports Tab.

The final page of this report displays a summary of the staging information for this test.

Test Request/Response Pair

Show request/response on the selected step.

Based On: Step name = Add User

Execution Time: 07/01/2016 09:43:34 AM

Test Name: webservices Step Name: Add User Step Number: 9,001

REQUEST


```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <soapenv:Body>
  <addUser xmlns="http://ejb3.examples.itko.com/">
   <!--username is optional-->
   <username xmlns="">webapp-1470572205</username>
   <!--password is optional-->
   <password xmlns="">example-pwd</password>
  </addUser>
 </soapenv:Body>
</soapenv:Envelope>
```


RESPONSE


```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"><env:Header></env:Header><env:Body><ns2:addUserResponse
xmlns:ns2="http://ejb3.examples.itko.com/"><return><email></email><fname></fname><lname></lname><login>webapp-1470572205</login>
<newFlag>true</newFlag><phone></phone><pwd>AL2slstNsa1Saz09CwL0/ThKVnU=</pwd><roleKey>REGULAR</roleKey><ssn></ssn></return>
</ns2:addUserResponse></env:Body></env:Envelope>
```


07/06/2016 12:35:08 PM Page 1 of 2

Test Execution Run Comparison

The Test Execution Run Comparison report lets you compare different runs of the same test. This comparison allows you to pinpoint when a particular test passed and when it failed. As with most reports, you can summarize your report by any runtime attribute. This report is particularly useful when you summarize by build number to identify the build where the test passed and the build where it failed.

The report shows the execution state for tests that were run during a specified time range. Each test run displays as a separate bar in the graph.

The following execution states are highlighted in each bar in the graph:

- Passed
  The test completed successfully.

Important! Tests that completed with errors are a Passed status, not a Failed status. To view any errors that are associated with tests in a Passed status, click the green Passed bar in the report. The Test Execution Detail report opens and lists all the test steps for the test run. Errors are shown in the Reason for Failure column.

- Failed
  The test failed to complete successfully. A Failed status indicates a test that contains an assertion. When the test was run, the criteria of the assertion were not met, which caused the test to fail.

- Aborted
  The test aborted. Aborted tests are “false failures” that aborted due to an inability to connect to the system under test.

To view the details behind any bar in the graph, click the bar in the report. The Test/Suite Execution History report opens and lists more detail for each execution of the selected test or suite.

Click an item in the legend at the bottom of the report to hide or show the corresponding data. For example, click **Aborted** to hide that data in the graph. When data is hidden, the legend item turns gray.

Test Execution Summary

The Test Execution Summary report lists all the test and suites that were executed during a specified time period. This report contains the following fields:

- **Test/Suite Name**
  Name of the test or suite. Click the name to view addition detail in the Test/Suite Execution History report.

- **Type**
  Indicates whether the line item is a test or a suite.

- **Last Execution Date/Time**
  The date and time the test or suite was last executed.

- **Duration(s)**
  The total time taken, in seconds, for the test or suite to run from beginning to end, including think time.

> **Note**: When the duration of an individual test step is reported, the think time is excluded. Because the reported duration of the test or suite includes the think time, the duration of the test or suite is typically greater than the sum of the duration of the test steps.

- **Times Executed**
  Number of times the test cycled (ran completely from beginning to end) during the selected time frame. Click to view more detail in the Test/Suite Execution History report.

- **Tests Passed**
  The test or suite completed successfully. Click the value in this column to view more details for the passing executions of the test or suite in the Test/Suite Execution History report.

> **Important!** Tests that completed with errors are a Passed status, not a Failed status. To view any errors that are associated with tests in a Passed status, click count in the Passed column.

- **Tests Failed**
  The test failed to complete successfully. A Failed status indicates a test that contains an assertion. When the test was run, the criteria of the assertion were not met, which caused the test to fail. Click the value in this column to view more details for the failed executions of the test or suite in the Test/Suite Execution History report.

- **Tests Aborted**
  The test aborted. Aborted tests are “false failures” that aborted due to an inability to connect to the system under test. Click the value in this column to view more details for the aborted executions of the test or suite in the Test/Suite Execution History report.

- **Last Run Status**
  The status of the test or suite the last time it was run.

Test/Test Suite Execution Reports
List of tests and test suites executed in a given period.

From: 01/01/1970 12:00:00 AM To: 10/30/2017 01:13:40 PM

Test/Suite Name	Type	Last Execution Date/Time	Duration(s)	Times Executed	Tests Passed	Tests Failed	Tests Aborted	Last Run Status
DevTest_config_info	Test	10/17/2017 04:45:13 PM	86716	3	47	0	39	ABORT
REST Example	Test	10/17/2017 04:34:31 PM	89256	1	622	0	0	PASS
AllTestsSuite	Suite	10/17/2017 04:45:13 PM	86728	2	5	11	5	ABORT

Test/Suite Execution History

The Test/Suite Execution History report lets you view more details for each test step that was executed for a selected time period. This report provides valuable information about the state and performance of individual tests and suites as you analyze and debug problems.

You can access the Test/Suite Execution History report directly from the Reporting menu and then select the appropriate filters for your report. However, you can also access this report as a sub-level report from the Test Execution Summary report.

This report contains the following fields:

- **Test/Suite Name**
  Name of the test or suite. Click the name to view more detail for each test step in the Test Execution Detail report.

- **Type**
  Indicates whether the line item is a test or a suite.

- **Last Execution Date/Time**
  The date and time the test or suite was last executed.

- **Duration(s)**
  The total time taken, in seconds, for the test or suite to run from beginning to end, including think time.

  > **Note:** When the duration of an individual test step is reported, the think time is excluded. Because the reported duration of the test or suite includes the think time, the duration of the test or suite is typically greater than the sum of the duration of the test steps.

- **Times Executed**
  Number of times the test cycled (ran completely from beginning to end) during the selected time frame.

Passed
The test or suite completed successfully.

> **Important!** Tests that completed with errors are a Passed status, not a Failed status. To view any errors that are associated with tests in a Passed status, click the test/suite name.

Failed
The test failed to complete successfully. A Failed status indicates a test that contains an assertion. When the test was run, the criteria of the assertion were not met, which caused the test to fail.
Aborted
The test aborted. Aborted tests are “false failures” that aborted due to an inability to connect to the system under test
Status
The status of the test or suite the last time it was run in the specified time period.

> Entries with an **UNKNOWN** status show up in the Test/Suite Execution History Report if ***all*** of the following conditions are true:
>
> 1. The test or suite of tests did not have any steps that passed.
> 2. The test or suite of tests did not have any steps that failed.
> 3. The test or suite of tests did not have any steps that aborted.
> This scenario can happen if a test or suite of tests was staged but never executed.

Test/Suite Execution History - TC1255-new

Summary of test/suite execution within a given time frame or last run.

From: 06/07/2016 01:36:46 PM **To:** 07/07/2016 01:35:05 PM

Test/Suite Name	Type	Last Execution Date/Time	Duration(s)	Times Executed	Passed	Failed	Aborted	Status
TC1255-new	Test	06/29/2016 11:32:46 AM	1	1	1	0	0	PASS
TC1255-new	Test	06/29/2016 02:07:01 PM	2	1	1	0	0	PASS
TC1255-new	Test	06/29/2016 02:09:11 PM	2	1	1	0	0	PASS
TC1255-new	Test	06/29/2016 02:10:05 PM	2	1	1	0	0	PASS
TC1255-new	Test	06/29/2016 02:14:41 PM	2	1	1	0	0	PASS
TC1255-new	Test	06/30/2016 08:55:32 AM	2	1	1	0	0	PASS
TC1255-new	Test	06/30/2016 08:57:38 AM	1	1	1	0	0	PASS
TC1255-new	Test	06/30/2016 09:15:43 AM	2	1	1	0	0	PASS
TC1255-new	Test	06/30/2016 09:16:46 AM	2	1	1	0	0	PASS
TC1255-new	Test	06/30/2016 09:19:28 AM	1	1	1	0	0	PASS
TC1255-new	Test	06/30/2016 09:21:30 AM	2	1	1	0	0	PASS
TC1255-new	Test	06/30/2016 09:22:20 AM	1	1	1	0	0	PASS
TC1255-new	Test	06/30/2016 09:23:30 AM	1	1	1	0	0	PASS

07/07/2016 01:37:08 PM **Executed By:** admin Page: 1 / 2

Suite Execution Detail

The Suite Execution Detail lets you view more details for each test that was executed for a particular suite execution instance. This report provides valuable information about the state and performance of individual tests within a suite.

The percentages of passed, failed, and aborted tests show in a pie chart at the top of the report. Click an item in the legend below the pie chart to hide or show the corresponding data. For example, click **Abort** to hide that data in the graph. When data is hidden, the legend item turns gray.

You can access the Suite Execution Detail report directly from the Reporting menu and then select the appropriate filters for your report. However, you can also access this report as a sub-level report from the Test/Suite Execution History report.

This report contains the following fields:

- **Test Name**
  Name of the test. Click the name to view more details in the Test Execution Detail report.
- **Last Execution Date/Time**
  The date and time the test was last executed.
- **Duration(s)**
  The total time taken, in seconds, for the test to run from beginning to end, including think time.

> **Note:** When the duration of an individual test step is reported, the think time is excluded. Since the reported duration of the entire test includes the think time, the duration of the test or suite is typically greater than the sum of the duration of the test steps.

- **Times Executed**
  Number of times the test was run during the selected time frame.

- **Reason for Failure**
  Error or failure messages associated with the test display in this column.

- **Status**
  The status of the test the last time it was run in the specified time period.

More Information:

- View and Navigate Reports
- Testing Performance Reports
- Virtual Service Metrics Reports
- CAI Top N and Metrics Reports
- User Activity Report

Testing Performance Reports

Performance reports contain information about how your tests and suites performed at run time.

You can use the Cycle Performance Summary and the Request/Second reports together to gain better insight into the performance of your system under test during load and performance testing. The other reports in this section help you understand how the user is affected during high system loads. These reports focus primarily on response time.

The following reports are available:

- Cycle Performance Summary
- Request per Interval
- Average Transaction Response
- HTTP Summary
- HTTP Traffic Details
- Performance Summary
- Test Metrics
- Top N Steps of Test and Suites

For more information about viewing and manipulate reports see View and Navigate Reports.

Cycle Performance Summary

The Cycle Performance Summary provides insight into the performance of a test or suite during load and performance testing. This report shows the minimum, maximum, and average cycle execution time, in milliseconds. It also shows the standard deviation for a test or suite within a specified time frame. The cycle execution time is the time taken for one iteration of a test or suite to run completely from beginning to end. This includes the time taken to execute each step as well as the think time.

When reviewing the standard deviation, average cycle execution time, and minimum cycle execution time (as shown in the bar graphs), use the legend on the left side of the report to determine the number of milliseconds. When reviewing the maximum cycle execution time (as shown in the line graph), use the legend on the right side of the report to determine the number of milliseconds.

Click an item in the legend at the bottom of the report to hide or show the corresponding data. For example, click **Standard Deviation** to hide that data in the graph. When data is hidden, the legend item turns gray.

The second page of this report shows the response time for each cycle in a cluster chart to help you more easily identify outliers. The final page of this report provides additional information about the number of cycles that were running simultaneously during a given time slot.

Screen capture of the Cycle Performance Summary report

Screen capture of the last page of the Cycle Performance Summary report

Request per Interval

The Request per Interval report displays the number of requests per a specified interval for tests and suites. The default show the number of requests per second. This report is useful for analyzing load testing that is based on the number of requests per interval.

The Request per Interval report shows the number of requests that were active, not just requests that started or stopped. For example, if a request was running for three seconds, it is counted in all three seconds on the chart. The number of requests display as a line chart.

Screen capture of the Request per Interval report

Average Transaction Response

The Average Transaction Response report displays the response time for each step in a selected test. The response times are plotted in a line chart. This report helps you to identify individual steps with a longer or shorter response time than expected.

The blue points represent the response times in milliseconds for a transaction over time. Hover your mouse over the blue points to display test/suite name, response time, day, month, and time of the transaction.

The final page of this report displays a summary of the staging information for this test.

Screen capture of the Average Transaction Response report

HTTP Summary

The HTTP Summary report helps you to determine the page weight in a web site by showing you the number of bytes being streamed. This report displays a cumulative HTTP traffic summary in two line charts.

The HTTP Summary report is limited to the HTTP/HTML test step type. It does not display data for REST, Web Service Execution, or any other test step type. To see the Cumulative HTTP Traffic Summary report, the following property must be set in either local.properties or site.properties:

lisa.commtrans.ctstats=true.

HTTP Summary - multi-tier-combo [Run1User1Cycle0Think]

Cumulative HTTP traffic summary

From: 07/07/2016 02:56:31 PM To: 07/07/2016 03:56:31 PM

HTTP Traffic	MBytes
Bytes Read	0.07
Bytes Sent	0.00
Total	0.07

HTTP Transaction type	Count
200	6
400	0
500	0
Others	0

07/07/2016 03:57:10 PM Page: 1 / 1

HTTP Traffic Details

The HTTP Traffic Details report provides additional information about the performance of a web site. It shows the average response in milliseconds for each test step as well as the percentage of the response time for the network and server.

This report also shows the HTTP response code for each step, which can be useful for identifying the type of traffic and the specific steps where errors are occurring.

Http Traffic Details - multi-tier-combo [run25for3m]

Displays http traffic metrics for the selected test/test suite.

From: 07/04/2016 01:25:52 PM To: 07/11/2016 01:25:52 PM

Step Name/URL	Response Code	Total Request	Average Response	Average Network(%)	Average Server(%)
Step Name - Account Activity					
http://localhost:8080/lisabank/buttonclick.do	200	106	48.0	56.1	43.9
Step Name - Add User Object XML					
http://localhost:8080/itkoExamples/EJB3UserControlBean	200	106	15.0	30.1	69.9
Step Name - Delete User XML					
http://localhost:8080/itkoExamples/EJB3UserControlBean	200	106	21.0	12.8	87.2
Step Name - Login					
http://localhost:8080/lisabank/login.do	200	106	53.0	79.8	20.2
Step Name - Logout					
http://localhost:8080/lisabank/logout.do	200	106	23.0	84.1	15.9
Step Name - Pay Bill Online					
http://localhost:8080/lisabank/depositmoney.do	200	106	360.0	7.4	92.6

07/11/2016 01:25:53 PM Page: 1 / 1

Screen capture of the HTTP Traffic Details report

Performance Summary

The Performance Summary report displays the response time and standard deviation for each step in a selected test. This report helps you analyze the performance of specific steps in a test. The standard deviation lets you identify steps with unpredictable or unstable response times.

When determining the minimum, maximum and average response times (as shown in the bar graphs), use the legend on the left side of the report to determine the number of milliseconds. When determining the standard deviation (as shown in the line graph), use the legend on the right side of the report to determine the number of milliseconds.

Click an item in the legend at the bottom of the report to hide or show the corresponding data. For example, click **Min Response Time (ms)** to hide that data in the graph. When data is hidden, the legend item turns gray.

The final page of this report displays detailed information about the performance of each step in the test, and a summary of the staging information for this test.

Step Name	Total Time (ms)	Sample Size	Min Response Time (ms)	Max Response Time (ms)	Average Time	Median (ms)	Standard Deviation	90th Percentil (ms)
ALL	4612	92	10	109	50	54.0	29.6	87.0
Create User - XML	699	23	24	50	30	29.0	6.66	39.5
Get User - XML	349	23	10	33	15	13.0	5.6899999999999995	20.0
List Users - JSON	1626	23	58	109	70	66.0	10.82	80.0
List Users - XML	1938	23	71	107	84	82.0	9.22	95.0

Name: REST Example
C:/Program Files/CA/DevTest/examples/Tests/REST Example.tst
Config: REST Example.config
C:/Program Files/CA/DevTest/examples/Configs/project.config
Staging: QuickStageRun
C:/Program Files/CA/DevTest/examples/StagingDocs/_quick_stage_document_.stg

Test Metrics

The Test Metrics report displays the event metrics for tests or test suites in a line chart by date. To obtain meaningful data from this report, you must first define the events you want to capture in a staging document. As you define events in the staging document, you select a color code for those events. The legend of events and their associated colors display in the second page of the report.

The second page of the report contains the following fields:

- **Long Description**
  The event description that you defined in the Long Name field of the staging document.

- **Sample Rate**
  The frequency with which sampling is performed and the values of your defined metrics are recorded. The sample rate can be set in seconds, minutes, hours, or daily. This value is set at the bottom of the metrics tab of the staging document.

- **Counter**
  When set to **True**, the event metric value is defined as a cumulative value, that is a running total.
  When set to **False**, the event metric value is a single value at the time of sampling.

- **Event**
  The name of the event that you defined in the Short Name field of the staging document.

- **Reg Expression**
  If you defined an event to search for a regular expression, that expression displays in this column. For example, you can create an event that only records specific types of errors that contain a defined regular expression.

For more information about defining events in a staging document, see Staging Document Editor - Metrics Tab.

Screen capture of the Test Metrics report

Top N Steps of Test and Suites

The Top N Steps of Test and Suite report displays the longest running test steps for a defined time period. This report helps you to identify long-running test steps and determine which steps are most resource intensive. Hover your mouse over an individual test step in the pie chart to see the total execution time for the step in milliseconds.

By default, this report shows the top ten longest running steps. However, you can define the number of steps that display in the **Refine By** pane to the right of the report data.

Click a step in the legend at the bottom of the report to hide or show that step in the pie chart.

Screen capture of the Top N Steps of Test and Suites report

More Information:

- View and Navigate Reports
- Testing Execution Reports
- Virtual Service Metrics Reports
- CAI Top N and Metrics Reports
- User Activity Report

Using DevTest Workstation with CA Application Test

DevTest Portal is a web-based application that provides simpler access to the most commonly used workflows for DevTest products. DevTest Workstation is the main application for advanced users and contains the full range of functionality available within the DevTest products.

DevTest Workstation is an integrated environment for developing, staging, and monitoring tests.

You can work in a DevTest Workstation version or in a DevTest Server environment.

In DevTest Workstation, the tests are managed and run in the Workstation environment. DevTest Workstation is a test client that QA/QE, development, and business analysis teams use to test the following components:

- Rich browser and web user interfaces

- The building blocks below the user interface

DevTest Workstation is used to build and stage unit, functional, integration, regression, and business process testing. DevTest Workstation requires a registry to run. Tests are managed and run in the DevTest Workstation environment (test authoring IDE), which runs embedded coordinator and simulator servers.

In DevTest Server, the tests are also managed and run in the Workstation environment. The workstation then connects to the server to deploy and monitor tests that were developed in DevTest Workstation.

The server-side engine for test cases and virtual services (test and virtual service model authoring IDE) manages, schedules, and orchestrates test cases continually for unit, functional, load, and performance tests.

Open DevTest Workstation

When you open DevTest Workstation, you are prompted to specify a registry.

If your computer has an installation of DevTest Server, then you can use:

- A registry that is running on your local computer
- A registry that is running on a remote computer

Use a registry that is running on a remote computer if your computer has an installation of DevTest Workstation.

For more information about how to specify the registry with SSL enabled, see Use SSL to Secure Communication Between Components.

To open DevTest Workstation:

1. Do one:
   - Open a command prompt, go to the **LISA_HOME\bin** directory, and run the DevTest Workstation executable.
   - If you have a DevTest application icon

on your desktop, double-click the application icon.
   - Select **DevTest**, **DevTest Workstation** from the list of programs in your Windows Start menu.

   The **Set DevTest Registry** dialog opens.

Set DevTest Registry dialog

2. Accept the default registry, or specify a different registry. Because DevTest Workstation is a GUI, there is no way to set a default remote registry on startup. Specify a registry at this prompt, or to change registries, use the Toggle Registry command.

3. Click **OK**.

4. (Optional) If this is the first time you have logged on to Release 10.2 of DevTest Workstation, and if DevTest has been installed with a data directory separate from the installation directory, the **Migrating Data with Projects** dialog opens. If you select **Skip**, you must manually copy examples and projects to the data directory, or you can invoke this dialog the next time you start DevTest Workstation by deleting the .devtest file in your data directory. **Skip** or **Next** will also copy the default projects, examples, and template files (_local.properties, _logging.properties, and _de-identify.xml) from your installation folder to your data directory.

   a. To have the installation copy files to the specified data directory, select an existing data directory and click **Next**.
      Either a confirmation window or an error window opens.

   b. For the error window, click **Previous** and select another data directory.
      For the confirmation window, click **Finish**.

5. The **Login** dialog opens.

Login dialog box

Enter your user name and password, and click **Login**.

Main Menu

The main menu in DevTest Workstation includes options for all the major functions available. The options available are dynamic to the selections at times. Some menus and also drop-down lists and toolbars are available throughout DevTest Workstation.

> **Note:** The items on this menu are dynamic. Your DevTest administrator controls the items that are available as part of your security profile.

Project Panel

A project is a collection of related DevTest files. The files can include test cases, suites, virtual service models, service images, configurations, audit documents, staging documents, data sets, monitors, and MAR info files.

In DevTest Workstation, only one project can be open at a time.

The **LISA_HOME\Projects** directory is created when you create a project for the first time.

All the folders in the **Project** panel of DevTest Workstation are the same as the folders that are created in the file system. You can examine the **LISA_HOME** directory to see the folder structure and files.

You can import files into a project.

Project Panel Layout

The **Project** panel contains a project tree.

When you create a project from scratch, the project tree has the following structure:

- **VirtualServices**
  - **Images**
  - **VRScenarios**
- **Tests**

  - **Subprocesses**
  - **AuditDocs**
  - **Suites**
  - **StagingDocs**
- **Configs**
- **Data**

To open or close the **Project** panel, click **Project**.

The toolbar in the **Project** panel has icons for the following actions:

- Refresh the project
- Dock or undock the **Project** panel
- Close the **Project** panel

The project includes a **.settings** directory, which does not appear in the **Project** panel. The **.settings** directory is used for saving some settings internally and can be seen in the file system in the **Projects** directory.

Project Panel Right-Click Menu

You can create various documents in a project from the **Project** panel. When you right-click a choice in the **Project** panel, the menu that appears is dynamic to the selection.

The order of these choices varies depending on what choice is selected when you right-click.

The right-click menu choices described here are also available from the main menu by selecting **File**, **New**.

- **Create New Test Case**
  For detailed information, see Create a Test Case.
- **Create New VS Model**
  For detailed information, see Work with Virtual Service Models. This feature requires an extra license.
- **Create New VS Image**
  For detailed information, see Create a Service Image. This feature requires an extra license.
- **Create New Staging Document**
  For detailed information, see Building Staging Documents.
- **Create New Suite**
  For detailed information, see Building Test Suites.
- **Create New Test Audit**
  For detailed information, see Building Audit Documents.

One or more of the following choices could also be available:

- **Add New Folder**
- **Import Files**
- **Rename Project**
- **Delete**
- **Paste**
- **View/Edit Project Metadata**

If you want the documents you create to default to the matching folder name, right-click that choice when adding. If you add a document without right-clicking the folder name, DevTest adds it to the bottom of the **Project** panel list. To correct, you then must go to the root directory and manually move the file into the correct folder and then reopen the project.

You are not limited to keeping a specific type of file (such as .tst) under the recommended folder (such as **Tests**). The file can stay anywhere in the project. The only exceptions are .config files; they must be located in the **Configs** folder.

Examples Project

When you open DevTest Workstation for the first time, the **Project** panel displays a project with the name **examples**.

Open any of the sample files by double-clicking them. The appropriate editor opens in the right panel.

The actual project files are located in the **LISA_HOME\examples** directory.

MARInfos

- **AllTestsSuite**
  Suite MAR that includes the test suite AllTestsSuite, with all the .tst files and accompanying data files to run the AllTestsSuite. The suite also includes the 1User1Cycle0Think staging document, the DefaultAudit audit document, and the project.config configuration file.

- **creditCheckValidate**
  Test-based Monitor MAR that includes the creditCheckValidate test case and the monitorRunBase staging document.

- **DatabaseModel**
  Virtual Service MAR that includes the DatabaseModel virtual service model and virtual service image.

- **OnlineBankingExternalCreditCheck-local**
  Test-based Monitor MAR that includes the test case webservices-xml-fail, staging document Run1User1Cycle, and project.config.

- **OnlineBankingJMStest-local**
  Test-based Monitor MAR that includes the async-consumer-jms test case, the Run1User1Cycle staging document, and the project.config configuration file.

- **OnlineBankingTransactionMonitor-local**
  Test Based Monitor MAR that includes the following items:

  - The multi-tier-combo test case
  - The Run1User1Cycle staging document
  - All the data files to support the test case
  - The project.config configuration file

- **OnlineBankingWebServices-local**
  Test-based Monitor MAR that includes the webservices-xml test case, Run1User1Cycle staging document, and the project.config configuration file.

Audit Docs

- **DefaultAudit**
- **main_all_should_fail**
- **ws_security-xml**

Configs

- **project**
  The project.config file contains intelligent defaults for many properties.

Data

The Data directory contains files that are necessary to run some of the examples for the demo server.

Monitors

- **creditCheckValidate.tst**
  Test case that is used for CVS monitor demos. Fails randomly on a specific cid.
- **monitorRunBase.stg**
  Staging document with one user, one cycle, and 100 percent think time, with Application Insight disabled and no maximum run time.
- **monitorRunMultiple.stg**
  Staging document with one user, run continuously, 136 percent think time, with Application Insight enabled and a maximum run time of 15 seconds.
- **monitorSLARun.stg**
  Staging document with one user, one cycle, and 100 percent think time, with Application Insight disabled and no maximum run time. JMX and JBOSS metrics are selected to record.
- **selenium.capabilities.conf**
  Sample skeleton parameter file with Selenium web driver parameters that you can set to customize options for Selenium test cases.
- **serviceValidator.tst**
  Used for CVS monitor demos. Fails randomly on a specific cid.
- **userAddDelete.tst**
  This test is used in the monitor setup for CVS demos.

Setup

The Setup directory contains batch files to start all DevTest components, stop all DevTest components, and load CVS monitors.

To use the scripts with access control (ACL) enabled, add the user name and password options to the Service Manager commands in the script. The password is not automatically encrypted, so be sure to protect the file by using the appropriate method for your operating system.

Staging Documents

- **1User0Think_RunContinuously**
  This staging doc runs a single virtual user with zero think time. This staging document also runs the test or tests "continuously," which does not necessarily mean "forever".
  When a test that this staging doc runs meets ALL of the following conditions and runs out of data, the run finishes:
  - A data set is on the first step of the test.
  - The data set "End of data" step is "End the test."
  - The data set is "global," not "local."
- If there are multiple data sets matching these conditions, then the first data set to expire finishes the run. For a good example, review the multi-tier-combo.tst test case.
- **1user1cycle0think**
  This staging document runs a test with a single user one time with a 0 percent think time scale.
- **ip-spoofing**
  To test IP spoofing support with your DevTest installation, use this staging document. IP spoofing is enabled in this staging document in the IP Spoofing tab. If you are running the examples server, an IP spoofing test web page is available at http://localhost:8080/ip-spoof-test.
- **jboss-jmx-metrics-localhost**
  This staging document runs a test with three users, run continuously, with a maximum run time of 440 seconds and 100 percent think time. The document has all four report generator parameter check boxes selected, and specifies all JBOSS JMX metrics to be collected.
- **Run1User1Cycle**
  This staging document runs a test with a single user one time with 100 percent think time scale.
- **Run1User1CyclePF**
  This staging document runs a test with a single user one time with 100 percent think time scale, with Application Insight enabled.
- **Run1User1CycleShowAll**
  This staging document runs a test with a single user one time with 100 percent think time scale. The staging document also selects all four check boxes in the default report generator so more things show in the web base model execution page.

Subprocesses

- **ws-sec-sub**
  This one-step test case is marked as a subprocess and can be called from any Execute Subprocess step.

Test Suites

- **AllTestsSuite**
  The AllTestsSuite runs all the tests in the DevTest Tests directory, using the 1user1cycle staging document and a default audit document of AuditDocs\DefaultAudit.aud. For report metrics, it only records requests and responses, and produces the default metrics. The Run Mode is serial.

- **FastAllTestsSuite**
  The AllTestsSuite runs all the tests in the DevTest Tests directory, using the 1user1cycle0think staging document and a default audit document of AuditDocs\DefaultAudit.aud. For report metrics, it only records requests and responses, and produces the default metrics. The Run Mode is parallel.

Test Cases

- **AccountControlMDB**
  A simple JMS test that adds a user with an account to LISA Bank. We expect and assert on patterns in the responses from the two steps.

- **async-consumer-jms**
  An example of an "async consumer" queue where the test case continually accepts messages from a response queue/topic. The queue also makes the messages available to the test case in the order in which they arrived.
  The first step creates the queue (internal to the test).
  The second step fires three messages to a JMS queue on the demo server. The async queue receives the messages.
  The third step validates that the async queue received three messages.

- **DevTest_config_info**
  This test case fetches diagnostic information about the computer running DevTest. The results can help Support solve configuration issues.

- **ejb3EJBTest**
  A pure EJB test of the LISA Bank functionality. Usually you would test applications by recording a web browser or other UI. Those tests are "end to end" integration tests. These sorts of tests are "lower down the food chain" and require more technical authors (though you still do not need to write any code).
  These tests enable the development team to use to test constantly and validate the code without having a user interface, which may not exist or may change so frequently that the tests cannot keep up.

- **ejb3WSTest**
  This model thoroughly tests the LISA Bank web services. The model is almost identical in functionality to the ejb3EJB test and useful for the same reasons (see that test case documentation).

- **ip-spoofing**
  This example test case demonstrates IP spoofing support in DevTest.
  This test requests the URL "http://localhost:8080/ip-spoof-test" using a REST step, a web page that contains the IP address of the requesting client. The test then makes a SOAP request to the following URL, which identifies a web service containing an operation that returns the IP address of the requesting client:

  ```
  http://localhost:8080/itko-examples/ip-spoof-test/webservice
  ```

  The rest case executes both requests in a loop for ten times. You can stage this test with the IP Spoofing Test staging document "ip-spoofing.stg". With the correct network interface configuration, you see different IP addresses being used among virtual users while they make the HTTP and SOAP requests.

- **jms**
  This test case is a simple JMS example showing how to send XML/text messages and objects in native Java format.

- **load-csv-dataset-web**
  This test model uses a comma-separated values (CSV) file as a data set to test a web application. The demo web app that is shipped with DevTest lets us add and remove users from the database.

- **log-watcher**
  This example shows how to fail a test by watching a log file for ERROR or WARNING messages. The example uses a data set to feed the example AntiPattern bean two numbers to divide. Approximately halfway through the data set we give 0 as the operand. This operand causes a divide-by-zero exception to occur on the server. The AntiPattern bean logs the exception and returns -1 as a result.
  This example is a common anti-pattern: internal errors occurring but external parties have no idea that the result is incorrect. The result looks believable but it is wrong. The expected result is that the EJB propagates the exception back to the caller.
  This test case fails with Application Insight because the agent records the fact that an exception was logged. Thus, DevTest determines that something is wrong.
  An alternative to using Application Insight is to set up a global assertion to watch the server log file. We define what comprises a regular expression: in this case simply the test "ERROR". The regular expressions can be as simple or as complicated as you want.
  Usually you would set the assertion to fail the test immediately. In this case, we advance to the "Error detected in log file" step and end the test normally.
  DevTest expects applications under test that log errors or warnings never to pass. Consider using an equivalent companion in your test cases by default.

- **main_all_should_fail**
  This test is an example of negative testing. Because we expect test steps to fail, we feed a service data that we expect will cause an error.
  The test has a companion, the NegativeTestingCompanion, which fails the test if any steps succeed.
  In this case, we try to create users in the demo server that exist. We get this data from the database itself (the username data set queries the table directly). If any step passes, the overall test fails.
  The only step that does pass is the "quietly succeed" step. This step lets you mark steps that you expect to pass in this sort of scenario as "quiet" in the editor so the NegativeTestingCompanion excludes them.

- **main_all_should_pass**
  This test calls a subprocess to insert a unique user name into the demo server USERS table. The data set is interesting because it relies on a data sheet to draw values from a unique code generator. The same thing could have been done with a unique code generator with a "counter" data set. This example demonstrates how one data set can influence another.
  Data sets are evaluated in the order they are specified. Each time the step is executed, the UniqueUser property is assigned a new value. The data sheet refers to {{UniqueUser}} four times, so we get five unique values.
  If any of the steps fail, the test fails immediately.
  Compare this test to "main_all_should_fail," which is a similar test where we expect each step to fail and we fail the test if anything passes. This process is known as negative testing.

- **multi-tier-combo**
  The multi-tier-combo test uses a variety of service endpoints to validate the LisaBank example. We test SOAP, EJB, JMS, Selenium, and web transactions and validate these transactions in a variety of ways including directly validating the demo server database. This test requires a Firefox browser for the Selenium steps to run.

The multi-tier-combo test uses various service endpoints to validate the LISA Bank example. The test case tests SOAP, EJB, JMS, and web transactions and validates these transactions in several ways including directly validating the demo server database.
The test also demonstrates how you can build complex SOAP objects from spreadsheets. A spreadsheet with the name **multi-tier-users.xls** in the Data folder of the project backs the **User** data set on the first step.
If you run this test in the Interactive Test Run (ITR) window, it creates a single user from the first row of the spreadsheet, then the test finishes.
The test restarts until it reaches the end of the data set if you stage it with the example **1User0Think_RunContinuously** staging document. This process is the preferred way to iterate repeatedly over a large data set. You could introduce a loop in the test case that is not as flexible.
If you let the staging document control the data set ending the test, then you can spread the test over many virtual users. Or, you can control the pacing of the test with think times and other parameters.
Only global data sets that are set on the FIRST step in the test affect the staging document "end the continuous test run" behavior. If the data set is local to the test or is declared elsewhere in the test, the "run continuously" behavior really does mean "run forever".

- **multi-tier-combo-se**
  The multi-tier-combo-se test uses various service endpoints to validate the LISA Bank example. We test SOAP, EJB, JMS, and web transactions and validate these transactions in various ways including directly validating the demo server database.
  The test also demonstrates how you can build complex SOAP objects from spreadsheets. The **User** data set on the first step is backed by a spreadsheet with the name **multi-tier-users.xls** in the **Data** folder of the project.
  If you run this test in the Interactive Test Run window (ITR), it creates a single user from the first row of the spreadsheet and then the test finishes.
  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 is nowhere near as flexible.
  If you let the staging document control the data set ending the test then you can spread the test over many virtual users or control the pacing of the test with think times.
  Note that the staging document "end the continuous test run" behavior 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" behavior really does mean "run forever".

- **rest-example**
  The rest-example test demonstrates how to execute RESTful services. The Demo Server contains a JAX-RS example. Each step in this test shows how to interact with that service using both XML and JSON.

- **scripting**
  CA Application Test can take advantage of JSR-223 scripting engines, allowing you to use a large variety of scripting engines in script steps, assertions, data protocols, match scripts and almost anywhere using {{=%language%}} syntax.

- **sendJMSrr**
  This test case lets you test a virtual service that has been created and deployed from JMS request/response pairs.

- **service-validation**
  This test is a simple example of service validation. The test calls one web service and one EJB service and inspects the underlying database with SQL to validate that the services function appropriately.

- **web-application**
  This test is a simple web test that was generated using the web recorder. The test contains some basic assertions such as "assert nonempty response," which is automatically generated. The test also contains some "assert title" assertions that were created by parsing the HTML responses for the <title> tag. These assertions help to ensure that the recorded page is the same when we play back the test.

- **webservices**
  This test case adds, gets, and deletes a user from the EJB3 web services. The test uses a unique code generator to create a number that has a prefix that is a value {{user}} from the config file. The password is hard-coded in the config file.

- **ws_attachments**
  This test case tests our ability to send and receive inline base64 encoded data blobs and XOP /MTOM attachments. Filters and assertions on steps verify that the requests and responses look correct.

- **ws_security**
  The ws_security test case shows how to use signed and encrypted SOAP messages. The first two steps succeed and the last two steps fail. The calls are the same, but the web service does not accept messages that are unencrypted or unsigned.

- **ws_security-xml**
  This test model shows how to use signed and encrypted SOAP messages. The first two steps should succeed and the last two steps should fail. (The calls are the same but the web service does not accept messages that are unencrypted or signed.)
  This test plan requires that your Java environment supports unlimited strength encryption. If either of the first two steps fail, a likely suspect is that your JCE jars must be updated to enable unlimited strength encryption. The JCE jars can be downloaded from www.oracle.com. Search on the keywords "JCE unlimited strength" and pick the JCE library matching your Java version, such as JCE 7 for Java 7. After you install the JCE jars, your DevTest services must be restarted.
  In the audit document, two occurrences of Step Error are expected. We audit for two Step Errors because we expect both of the last two steps here to raise Step Errors. Thus, the audit document can also audit how many occurrences of an event are received. If we add a third Step Error entry to the audit document, the auditor will fail this test because only two Step Error events are raised.

Tests that Fail

- **webservices-xml-fail**
  This test case adds, gets, and deletes a user from the EJB3 web services. The test uses a unique code generator to create a number that has a prefix that is a value {{user}} from the config file. The password is hard-coded in the config file.

VServices

Forward Cars includes a Swagger document that describes the inventory service. The **CarsInventoryFromSwagger** virtual service was created from the Swagger document.

- **statefullATM.tst**

- **statelessATM.tst**

- **web-app-proxy.tst**

- **CarsInventoryFromSwagger.vsm**

- **DatabaseModel.vsm**
- **kioskV4model.vsm**
- **kioskV5.vsm**
- **kioskV6.vsm**
- **WebServicesModel.vsm**
- **webservices-vs.vsm**

Images

- **CarsInventoryFromSwagger.vsi**
- **DatabaseModel.vsi**
- **kioskV4ServiceImage.vsi**
- **kioskV6.vsi**
- **si-kioskV5-dynamic.vsi**
- **si-kioskV5.vsi**
- **WebServicesModel.vsi**

Create a Project

You can manually create a project or you can create a project from an existing Model Archive (MAR) file.

> **Note:** You can only create a project from a VSE MAR file. You cannot add a MAR to an existing project with this dialog.

Follow these steps:

1. Select **File**, **New**, **Project** from the main menu.
   The **Create New DevTest Project** dialog appears.
2. In the **Project Name** field, enter the name of the project.
3. To create the project in a location other than the **LISA_HOME\Projects** directory, click the icon next to the **Project Name** field and specify the location.
4. To create the project from an existing MAR file, do the following actions:
   a. Select the **Base the new project on one of the following** check box.
   b. Select the **MAR file** option.
   c. Specify the MAR file name and the directory location.

5. Click **Create**.

Open a Project

You can open a project from DevTest Workstation or the command line.

To open a project from DevTest Workstation:

1. Select **File**, **Open**, **Project** from the main menu.
   If you have already opened some projects, you see a list of recently opened projects that you can select from. After it is selected, the project will open.

2. To browse for a project file, select **File System** from the list of choices and the **Open Project** window opens.

3. Select the project and click **Open**.

To open a project from the command line:

1. Navigate to the **LISA_HOME\bin** directory.

2. Run the DevTest Workstation executable and specify the project directory as an argument. The project directory can be an absolute path or a relative path. The following example uses an absolute path:

   ```
   LISAWorkstation "C:\Program Files\CA\Lisa\Projects\Project1"
   ```

Quick Start Window

When you open DevTest Workstation, the **Quick Start** window is the first window that you see. This window has some of the most useful options in DevTest Workstation.

> **Note:** The **Quick Start** window is always available as a tab after more tabs are opened.

The following graphic shows the **Quick Start** window. Depending on your screen resolution, you could see the expanded or the compact wording for each option. To view all of the options, click the down arrow at the bottom of the window.

Quick Start Window

- Open Recent
- Create a Web Services Test
- Create a Swagger Test
- Create a Web Test
- Create a Virtual Service Model
- Record a Web Service Virtual Service Image
- Create a Service Image from WSDL
- Learn More
- DevTest Workstation Memory Meter

Open Recent

The **Open a Recent Document** option is selected by default.

The right pane displays any recently opened projects, test cases, suites, staging documents, virtual service models, and virtual service images.

Create a Web Services Test

The **Create a Web Services Test** option lets you create a Web Service test case. The parameters that are needed are displayed in the right pane.

Quick Start New WS Test option

Follow these steps:

1. Enter the **WSDL URL**, **Service**, and **Port** details.
2. Select the operations that you want to test from **All** or **None** or select each item individually.
3. In the right pane, enter the name of the test and select the path where you would like to save it in the project folder. When you select the path, it is seen in the **Path** field. In this pane, you can right-click on any folder and can create one or can rename or delete an existing one.
4. To create the test case, click the green arrow .

For more information, see Generate a Web Service.

Create a Swagger Test

The **Create a Swagger Test** option lets you create baseline tests from a Swagger 2.0 document.

Quick Start Swagger option

The baseline tests are saved to a subfolder of the **Baselines** folder. The subfolder also includes an audit document, a staging document, and a suite for running the baseline tests.

A configuration file is saved to the **Configs** folder. The configuration file contains any properties that are required to run the baseline tests. Before you run the baseline tests or test suite, mark the configuration as active.

For each method that you select, one baseline is generated for each MIME type. Within each baseline, one step is generated for each possible response.

You can test this feature with the Forward Cars demo application. Forward Cars includes a Swagger document that describes the inventory service. The document is available at the following URL:

```
http://localhost:3500/cars-inventory/swagger.json
```

If Forward Cars is running on another computer, replace **localhost** with the host name or IP address of the other computer.

The Swagger document is also available in the **Data** folder of the **examples** project. The file name is **cars-inventory-swagger.json**.

Follow these steps:

1. In the **URL** field, enter a URL or browse to the file that contains the Swagger 2.0 document.

2. Click **Refresh Swagger Cache** .
   If the Swagger parsing is successful, the **Endpoint** field and **Methods** pane are populated with information from the Swagger document. You cannot modify the **Endpoint** field.

3. In the **Methods** pane, select the methods in the endpoint to virtualize.
   By default, all the methods are selected. You can click **Select All** or **Select None**, as appropriate.

4. Locate the field in the lower right corner and enter a name for the subfolder where the baseline tests are saved.

5. To create the test case, click the green arrow .

   The baseline tests are created. In addition, the methods that you selected are disabled.

6. To create another baseline test with any remaining methods, select the methods and choose a new baseline test name.

Create a Web Test

The **Create a Web Test** option lets you create a web test.

Quick Start New Web Test option

Follow these steps:

1. Enter the URL for the web test.
2. Select **Capture HTTP traffic only**.
3. Select your options in the check boxes:
   - **HTML Responses Only**
     Captures only the HTML responses.
   - **Use External Browser**
     Opens an external browser window.
   - **Configure IE for DevTest**
     Configures Internet Explorer for DevTest.
4. In the right pane, enter the name of the test and select the path where you would like to save it in the project folder. When you select the path, it is seen in the **Path** field. In this pane, you can right-click on any folder and can create, rename, or delete one.
5. To create the test case, click the green arrow .

For more information, see Record a Website.

Create a Virtual Service Model

The **Create a VS Model** option lets you create a virtual service model and a corresponding service image.

Quick Start Create VSM option

Follow these steps:

1. Select the transport protocol from the list of available protocols. Your selection here will determine the next sequence of windows.

2. Enter the name of the service image.

3. Enter the name of the VSM and select the path where you would like to save it in the project folder. When you select the path, it is seen in the **Path** field.

4. To create the test case, click the green arrow .

For more detailed information about the windows and options for VSM and VSI creation, see Create a Service Image.

Record a Web Service Virtual Service Image

The **Record a WS VS Image** option lets you create a service image from a web service. The parameters that are needed are displayed in the right pane.

Quick Start Record WS VSI option

Follow these steps:

1. Enter a port on which to listen and record.
2. Enter the URL of the web service to record.
3. Enter the target host and port to record.
4. To determine whether to use SSL with this service image, use the **SSL** check box.
5. Enter the name of the VSM and SI.
6. To create the test case, click the green arrow .

Create a Service Image from WSDL

The **Create an SI from a WSDL** option lets you create a service image from a WSDL document. The properties that are required are displayed in the right pane.

Quick Start VSI from WSDL option

Follow these steps:

1. Enter the **WSDL URL**, **Service**, and **Port** details. If the **WSDL URL** you enter does not contain properties in its path, **Service** and **Port** are populated automatically.
2. To select the operations to test, click **All** or **None** or select each operation individually.
3. Enter the name of the service image and select the path in the project folder at which to save it.
4. To create the test case, click the green arrow .

Learn More

The **Learn More** option window displays a link to the documentation, online support, and the DevTest global community.

All available user documentation for DevTest is accessible from this menu.

DevTest Workstation Memory Meter

At the bottom right corner of the DevTest Workstation main window, the DevTest Workstation Memory Meter displays the run-time memory usage and availability information.

- To run garbage collection, click the Memory Meter.
- To generate a heap dump file (HProf), right-click the Memory Meter. Select the **Dump Heap** option.

> **Note:** This functionality is a diagnostic tool that can help CA Support determine the causes of OutOfMemory conditions. The heap is automatically dumped if an OutOfMemory condition occurs; this option is for manually triggering a heap dump.

After the dump is created, a message indicates that the heap dump has been taken.

Tray Panels

DevTest Workstation has tray panels for some features, such as the Output Log Message step.

You can control whether the opening and closing of tray panels uses animation. By default, animation is disabled to help improve the performance for users who access DevTest Workstation through a remote desktop.

To enable the animation, add the following line to the **site.properties** or **local.properties** file:

```
lisa.ui.tray.animation=true
```

Building Test Cases

Building test cases requires knowledge about the following concepts:

- Setting properties
- Using configurations and applying them to your project
- Applying filters
- Adding assertions
- Adding data sets
- Adding companions

This section also introduces the Complex Object Editor.

> **More Information:**
>
> - Anatomy of a Test Case
> - Properties
> - Configurations
> - Assets
> - Filters

- Assertions
- Data Sets
- Companions
- Complex Object Editor (COE)
- Building Test Steps
- Creating Test Cases
- Building Subprocesses

Anatomy of a Test Case

A test case is a specification of how to test a business component in the system under test. A test case is stored as an XML document, and contains all the information that is necessary to test the component or system.

A test case is a workflow with the test steps connected by paths that represent successful and unsuccessful step conclusions. Assertions can accompany the step and different paths are provided based on the firing of any of the assertions.

Note: Save your test cases regularly.

More Information:

- Test Case Quick Start
- Multi-tier-combo Test Case
- Elements of a Test Case
- Elements of a Test Step

Test Case Quick Start

To start working with test cases:

1. Start the registry.
   See DevTest Registry.

2. Create or open a project in DevTest Workstation.
   See Project Panel.

3. Create a test case in the project.
   See Create a Test Case.
   You can open a test case from inside a project or from outside a project by selecting **File**, **Open**, **Test Case** from the main menu.

The **multi-tier-combo.tst** test case is shown in the following graphics. For more information about the multi-tier-combo test case, see Multi-tier-combo Test Case.

DevTest Workstation is divided into the following main areas (from left to right):

- Project Panel
- Model editor
- Element Panel

Project Panel

The **Project** panel, which is located in the left portion of the window, is dockable. You can open or close the **Project** panel by using the **Project**

button on the left.

When DevTest Workstation first opens, the last project you had open opens by default.

Project panel

For more information, see Project Panel.

Model Editor

The model editor is where you create and view the test case workflow. The test case workflow consists of all test steps, filters, and assertions that are applied to a specific test case.

The model editor is the place to create and view test cases. The middle portion of the window displays a tab that is the name of the test case.

Screenshot of multi-tier-combo test case

The green arrow marks the start of a test case.

The workflow in the model editor provides a graphical view of a test case. This view is helpful, because it gives a quick visual validation on the test case workflow.

In the model editor, an icon in the workflow represents each step in the test case. The icons change according to the type of test step. For example, if you have a database-related step, a database icon and the associated filters and assertions are attached to the step.

Element Panel

The **Element** panel contains the elements that are required for a test case or a test step.

Screenshot of Element panel for a test case

Screenshot of test step collapsed

To change the step name back to the default step name, click **Revert to Default Name** .

You can add or delete an element by clicking the required test case or test step element.

Some elements can be applied at the global level (to an entire test case). Some elements can be applied at a step level (only to a specific test step).

Enter some required information at the beginning of a test case.

For example:

- **Test Case Information**: Enter the test case name and select whether this case is a subprocess.
- **Documentation**: Enter the documentation for the test case.

After you add steps to this test case, a workflow of steps begins to form. A new set of elements appears at a test step level in the **Element** panel.

Multi-tier-combo Test Case

The examples project contains a test case with the name **multi-tier-combo**.

This test case uses various service endpoints to validate the LISA Bank demo application. The case tests SOAP, EJB, JMS, and web transactions and validates these transactions in various ways, including directly validating the demo server database.

This test case also demonstrates how to build complex SOAP objects from spreadsheets. The **multi-tier-users.xls** spreadsheet in the project Data folder backs the User data set on the first step.

Running this test in the Interactive Test Run (ITR) window, the test creates a single user from the first row of the spreadsheet and finishes.

If you stage the test with the example **1User0Think_RunContinuously** staging document, the test restarts until it reaches the end of the data set. This method is the preferred way to iterate repeatedly over a large data set. You can introduce a loop in the test case, but that is not as flexible.

If you let the staging document control the data set ending the test, then you can spread the test over many virtual users. Or, you can control the pacing of the test with think times, for example.

Only global data sets that are set on the first step in the test affect the staging document "end the continuous test run" behavior. If the data set is local to the test or it is declared elsewhere in the test, the "run continuously" behavior really does mean "run forever."

Notice the **example** project folders being opened in the **Project** panel and a set of test case elements in the **Element** panel.

Here you can see in the model editor section the test case information.

Screenshot of multi-tier-combo test case

Elements of a Test Case

The elements of a test case help in building the whole test case. The following graphic shows how the **Test Case** panel on the right side of DevTest Workstation displays the test case elements.

Screenshot of Element panel for a test case

Test Case Information

The **Test Case Information** tab is where you can change the name of a test case.

This tab is also used as an entry point for creating a subprocess or for converting a test case into a subprocess. A subprocess is a test case that another test case calls instead of running as a stand-alone test.

Screenshot of the Test Case Information tab

For more information about subprocesses, see Building Subprocesses.

Companions

A companion is an 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.

To open the companion editor, double-click the companion in the **Elements** panel.

Screenshot of Companions editor

For more information, see Companions.

Global Assertions

An assertion is an element that runs after a step and all its filters have run. Assertions verify that the results from running the step match your expectations. A *global assertion* applies to the entire test case.

To open the assertion editor, double-click the assertion in the **Global Assertion** list.

Screenshot of Global Assertions tab

For more information, see Assertions.

Global Filters

A filter is an element that runs before and after a test step. Filters give you the opportunity to change the data in the result, or store values in properties. A *global filter* applies to the entire test case.

To open the filter editor, double-click the filter in the **Global Filter** list.

Screenshot of Global Filters list

For more information, see Filters.

Documentation

The **Documentation** area lets you add the documentation for your test case. This text is not used 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 who use this test case.

Screenshot of Documentation tab

Elements of a Test Step

A test step is a workflow test case element 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 are typically chained together to build workflows as test cases in the model editor. From each step, you can create filters to extract data or create assertions to validate response data.

These elements are discussed in detail in Building Test Steps.

Screenshot of test step collapsed

Step Information

The step information section provides a place to document basic information about the test step.

You can enter the step name, think time, **Execute on** details, and **Next step** details. You can also specify to run the step using global filters and to run the step quietly.

Screenshot of step information section

For more information, see Building Test Steps.

Log Message

A log message is a text field in which you can enter a message for a step. This message is seen upon execution of the test step or case.

Screenshot of log message section

For more information, see Test Step Logger.

Assertions

An assertion is an element that runs after a step and all its filters have run. Assertions verify that the results from running the step match your expectations. The result of an assertion is always Boolean - either true or false.

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 change the test case workflow by introducing conditional logic (branching) into the workflow.

Screenshot of assertions section

For more information, see Assertions.

Filters

A filter is an element that runs before and after a test step. Filters give you the opportunity to change the data in the result, or store values in properties.

Screenshot of filters section

For more information, see 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 ability provides a mechanism to introduce external test data to a test case.

Screenshot of data sets section

For more information, see Data Sets.

Properties Referenced

This section contains a list of properties that the test step uses or references.

To open the extended view and get its variable value, select and right-click the property.

Screenshot of properties referenced section

For more information, see Properties.

Properties Set

This section contains a list of properties that the test step sets. The **Properties Referenced** and **Properties Set** are for a specific step and change when another step is selected.

Screenshot of properties set section

For more information, see Properties.

Documentation

The **Documentation** area lets you add the documentation for your test step. This text is not used in any process, but it is a convenient place: and more importantly, a good practice to put a description of your test step, and notes for other users who use this test step.

Screenshot of documentation section

Properties

Test properties are name/value pairs, also known as key/value pairs.

The key to data independence, reusability, and portability in test cases is the ability to replace specific data values abstracted from them with variables. These variables are referred to as *properties*. Some properties are predefined and guide how the application operates. You create other properties while you are building your tests.

A sound understanding of properties is important to the creation of test cases. In the context of a test case, any time there is something that can change, it is appropriate to use a property. This scenario includes values in test steps and values in configurations, for example.

Properties can be defined in several ways. After they are 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 few exceptions, be overridden in a test case.

Whenever a property value is set, a Property set event is recorded. The event contains 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. Many properties that are created during a test run that are available to the subsequent test steps. For example, the **lisa.stepname.rsp** property contains the response for the stepname step.

The main property files are:

- **lisa.properties**
- **local.properties**
- **site.properties**

For more information about these files, see Property Descriptions.

Specify a Property

Specify a property using the following syntax:

```
{{property_name}}
```

When a property is identified and ready for use, the current value of the property replaces **{{property_name}}**. Sometimes a property is expected, and is the only choice. Other times, you are asked for the property name explicitly. In these cases, you enter the property name without the braces. Use the brace notation when properties are embedded in a text string. Other syntax allows property expressions to be used: **{{=expression}}** or **{{property_name=expression}}**.

A property name can contain spaces. However, using spaces is not recommended. The characters that define the property syntax ({,}, and =) cannot be used. If you reference a nonexistent or invalid property, DevTest leaves it in the braces.

When editing properties files, which contain UNIX-style line feeds, use an appropriate editor such as WordPad.

> **Note:** Property names that start with **lisa.** are reserved for internal use. DevTest may hide or delete properties that start with **lisa.**

More Information:

- Property Expressions
- String Patterns
- Property Sources
- Common Properties and Environment Variables
- View Properties

Property Expressions

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 BeanShell can evaluate. BeanShell is a Java interpreter environment. Further, these expressions could be string patterns, which give real-looking fake strings, appropriate for most purposes.

For more information about BeanShell, see Using BeanShell in DevTest or www.beanshell.org.

To use a property expression, use one of the following formats:

- **{{=expression}}**
  BeanShell is used to evaluate the expression and replace {{expression}} with the result of the evaluation. For example, {{=Math.random()}} evaluates the static Java method and replaces the {{}} construct with the random number that was returned.

- **{{key=expression}}**
  Using {{rand=Math.random()}} sets a property **rand** equal to the random number that was returned, and replaces the {{}} construct with the random number.

You can reference properties by name only in a property expression (that is, without the braces) because they are already defined as properties. If the property is not found, the property expression is returned inside braces to indicate that there is a problem in the expression.

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. This step only writes to a log and displays the response in the Interactive Test Run (ITR) **Response** tab. Therefore, you can experiment with using property expressions.

The following example uses the **multi-tier-combo** test case in the examples directory**.**

Screen capture of multi-tier-combo test case.

This graphic shows our example in the ITR utility.

Screen capture of ITR running multi-tier-combo test case.

This graphic shows the **Properties** tab in the ITR. The tab that is shown is for the step **Get User**.

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
EJBSERVER	localhost	localhost
LISA_DOC_PATH	C:\Lisa11\examples\Tests	C:\Lisa11\examples\Tests
JMSCONNECTIONFACT...	ConnectionFactory	ConnectionFactory
LISA_LAST_STEP	end	WriteFile
lisa.BinFolder.rsp.time	375	375
ENDPOINT1	http://localhost:8080/itk...	http://localhost:8080/it...
lisa.end.rsp	The test has ended	
lisa.HotDeployFolder.rs...	31	31
lisa.WriteFile.rsp	==========...	==========...
JNDIPORT	1099	1099
order.step.2.queue	queue/C	queue/C
lisa.JavaConfiguration.r...	156	156
lisa.scriptEngine	NotSerializableStateWra...	NotSerializableStateWra...
DBNAME	itko_examples	itko_examples
DBUSER	sa	sa
LISA_TC_PATH	C:\Lisa11\examples\Tests	C:\Lisa11\examples\Tests
user	webapp	webapp
DBPORT	3306	3306
robot	0	0
LISA_PROJ_NAME	examples	examples
DBCONNURL	jdbc:derby://localhost:1...	jdbc:derby://localhost:1...
lisa.LibFolder.rsp.time	15	15
LISA_HOST	Frasetto	Frasetto
lisa.end.rsp.time	0	
LISA_USER	kfrasetto	kfrasetto
WSPORT	8080	8080
DBPASSWORD	sa	sa
instance	0	0

Screen capture of Properties tab in the ITR.

This graphic shows the **Test Events** tabs in the ITR.

Response | Properties | Test Events

Timestamp	EventID	Short	Long
13:34:13,596	Step history	ABEFD4BED91028...	
13:34:13,596	Step started	Pay Bill Online	Withdraw money...
13:34:13,596	Property set	lisa_last_pftxn	<removed>
13:34:14,045	Property set	lisa.Pay Bill Onlin...	200
13:34:14,045	Property set	lisa.Pay Bill Onlin...	http://localhost:8...
13:34:14,045	Property set	lisa.Pay Bill Onlin...	Server=Apache-...
13:34:14,046	Step request	Pay Bill Online	POST /lisabank/d...
13:34:14,046	Step target	Pay Bill Online	http://localhost:8...
13:34:14,046	Info message	Pay Bill Online	Requested URL:h...
13:34:14,046	Property set	LISA_COOKIE_loc...	[version: 0][nam...
13:34:14,046	Step response time	Pay Bill Online	446
13:34:14,046	Step bandwidth c...	Pay Bill Online	16558
13:34:14,046	Step response	Pay Bill Online	<!-- jspstart: roo...
13:34:14,071	Property set	lisa.Pay Bill Onlin...	Integrator of type...
13:34:14,071	Property set	lisa.Pay Bill Onlin...	<?xml version="...
13:34:14,071	Pathfinder	Pay Bill Online	
13:34:14,071	Assert evaluated	Pay Bill Online [A...	Assert of type [A...
13:34:14,071	Assert evaluated	Pay Bill Online [Si...	Assert of type [Si...
13:34:14,071	Log message	Will execute the ...	Withdraw money...

Screen capture of the Test Events tab in the ITR.

Look for the event **Property set** in the **Test Events** tab.

Java developers can also take advantage of the BeanShell environment in the JavaScript step to test property expressions.

String Patterns

String patterns are special types of property expressions that have a syntax {{ =[:patternname:] }}. For example, to format a first name, the string pattern property could be {{ =[:First Name:] }}. The property would evaluate to a fake first name that looks like a real name.

The following video provides an overview of string patterns.

This behavior is much better than the possibility of dealing with random strings that do not look like a real name. The string pattern functionality supports many patterns in addition to first names: last names, dates, Social Security numbers, credit card numbers, credit card expiration dates, and many more. This fake data comes in the TESTDATA table in the reportdb database.

The recommendation is that if you need a first name in your test case, you use {{ =[:First Name:] }} in a data set. The "{{=[" part is a signal to use the string pattern and it has a list of things "registered" that it recognizes.

For example, using the following information in a log step:

```
{{=[:First Name:]}} {{=[:Middle Initial:]}} {{=[:Last Name:]}}
{{=[:Street Address:]}}
{{=[:City:]}}, {{=[:State Code:]}}
{{=[:ZIP Code:]}} {{=[:Country:]}}
SSN: {{=[:SSN:]}}
Card: {{=[:Credit Card:]}} Expires {{=[:CC Expiry:]}}
Phone: {{=[:Telephone:]}}
Email: {{=[:Email:]}}
```

provides the following response:

```
Marilyn M Mcguire
3071 Bailey Drive
Oelwein, IA
50662 US
SSN: 483-16-8190
Card: 4716-2361-6304-6128 Expires 3/2014
Phone: 319-283-0064
Email: Marilyn.C.Mcguire@spambob.com
```

If you run the step again, you get a different set of data. DevTest tracks the number of rows in the test data database and randomly selects a row between 1 and N.

To open the following window, which shows all existing string patterns and the documentation, click the vertical **Patterns** tab on the far right side of the page.

Screenshot of Property Window, Patterns tab

Implementation Information

The data is stored by default in the reports database in the TESTDATA table.

DevTest Workstation verifies whether there is any data in the TESTDATA table when it starts. 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 is recreated. Reading the database is done only at startup and takes approximately 15 seconds.

Creating Your Own String Pattern

When you add your own data, the only thing to be careful about is to assign the ID correctly. Start with 1 and increase 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.

Property Sources

Properties can originate from several sources that include:

- DevTest
- Environmental variables
- Command-line variables on startup
- Configurations
- Companions
- Test steps
- Filters
- Data sets
- String patterns

Because the properties can be overridden, it is important to understand the property hierarchy (the order in which properties are read in a test case).

The following hierarchy is used:

1. Properties that are loaded during the setup of a test.
2. Operating system environment variables (like java.version or os.user, for example).
3. DevTest property files.
4. Command-line attributes.
5. The default configuration.
6. Any alternative configuration properties (from active configuration or runtime configuration file).
7. Properties that are set during a test run.
8. Properties in companions.

9. Properties that are set during test execution (for example, in data sets, filters and steps). Properties set during test execution override values that were set earlier.

Common Properties and Environment Variables

Common properties and environment variables include:

- **HOT_DEPLOY**
  Points to a project-specific hotDeploy directory.

- **LASTRESPONSE**
  The response to the last executed step.

- **LISA_HOME**
  Points to the install directory, and is automatically set. This value includes a final slash. To reference a directory such as **examples**, specify:

  ```
  {{LISA_HOME}}examples
  ```

  No slash is needed before the directory name.

- **LISA_HOST**
  The name of the system on which the testing environment is running.

- **LISA_JAVA_HOME**
  The Java VM to use. Set this property only if you do not want to use the built-in VM. If Java is not installed, DevTest uses the bundled JRE it comes with. You also must rename the jre directory in the install directory to something like jre_notinuse.

- **LISA_POST_CLASSPATH**
  Used to add information after the DevTest classpath. DevTest does not use the OS environment CLASSPATH variable. To add your own JARs after the DevTest classpath, use LISA_POST_CLASSPATH.

- **LISA_PRE_CLASSPATH**
  Used to add information before the DevTest classpath. DevTest does not use the OS environment CLASSPATH variable. To add your own JARs before the DevTest classpath, use LISA_PRE_CLASSPATH.

- **LISA_PROJ_NAME**
  The name of the project to which the current document belongs. LISA_PROJ_NAME refers to the main calling test project.

- **LISA_RELATIVE_PROJ_NAME**
  The name of the project to which the current document belongs. LISA_RELATIVE_PROJ_NAME refers to the project for the currently executing test or subprocess. If the test does not call any subprocesses, LISA_PROJ_NAME and LISA_RELATIVE_PROJ_NAME are the same.

- **LISA_PROJ_PATH**
  The fully qualified path of the project directory. The value is operating system-dependent. A backslash (\) is used as the separator character on Windows. A forward slash (/) is used as the separator character on other operating systems. The following example is based on a Windows installation:

  ```
  C:\Program Files\CA\DevTest\Projects\MyProject
  ```

The one limitation to using LISA_PROJ_PATH in a Custom Java step is that the syntax {{LISA_PROJ_PATH}} is not supported. The Custom Java step invokes a Java compiler to compile the script and Java treats backslashes as escape characters in strings. Therefore, this specific string raises a compiler error. The workaround is to use LISA_PROJ_PATH as a variable. For example:

```
File f = new File ( LISA_PROJ_PATH );
```

- **LISA_RELATIVE_PROJ_PATH**
  The fully qualified path of the project directory of the currently executing test or subprocess. LISA_PROJ_PATH refers to the main calling test. See LISA_PROJ_PATH for details. If the test does not call any subprocesses, LISA_PROJ_PATH and LISA_RELATIVE_PROJ_PATH are the same.

- **LISA_PROJ_ROOT**
  The fully qualified path of the project directory. The value is operating system-independent. A forward slash (/) is used as the separator character on all operating systems, including Windows. The following example is based on a Windows installation:

  ```
  C:/Program Files/CA/DevTest/Projects/MyProject
  ```

- **LISA_RELATIVE_PROJ_ROOT**
  The fully qualified path of the project directory of the currently executing test or subprocess. LISA_PROJ_ROOT refers to the main calling test. See LISA_PROJ_ROOT for details. If the test does not call any subprocesses, LISA_PROJ_ROOT and LISA_RELATIVE_PROJ_ROOT are the same.

- **LISA_PROJ_URL**
  The URL of the project directory. For example:

  ```
  file:/C:/Program%20Files/CA/DevTest/Projects/MyProject
  ```

- **LISA_RELATIVE_PROJ_URL**
  The URL of the project directory of the currently executing test or subprocess. LISA_PROJ_URL refers to the main calling test. See LISA_PROJ_URL for details. If the test does not call any subprocesses, LISA_PROJ_URL and LISA_RELATIVE_PROJ_URL are the same.

- **LISA_TC_PATH**
  The fully qualified path of the directory where the test case is located.

- **LISA_TC_URL**
  The URL of the directory where the test case is located.

- **LISA_USER**
  The user that loaded the test case.

Property Files

The main property files are:

- lisa.properties
- local.properties
- site.properties

More Information:

- DevTest Property File (lisa.properties)
- Custom Property Files

View Properties

The **Properties** pane lets you view the properties that are associated with a test case. This pane also simplifies the process of adding properties to a specific step.

Follow these steps:

1. Open the test case to modify.
2. Complete one of the following actions:
   - Click the vertical **Properties** tab on the far right side of the page.
   - Click **Help, View Properties** on the main menu.
   - Click **Show model properties** on the test case toolbar.

   The **Properties** pane opens.
3. Navigate to the text field in the element tree where you want to add a property and click inside the text field.
4. Select the property to add in the **Properties** pane.
   - To search for a property by name, use the **Find** field at the bottom of the pane.
   - To restrict the list to a specific category of properties, select a property from the list at top of the pane .
   - To view properties that are global to this instance of DevTest, click **Global Properties** on the far right side of the page.
5. Click **Add** at the bottom of the **Property** pane.
   The property is added to the selected field.

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, you can run the same test on different environments by simply using a different configuration. Configurations are used everywhere in DevTest: for example, in a test case document, test suite document, staging document, test case execution, or test suite execution.

A configuration must be defined at the project level. You can specify the values of these properties at the beginning of a test case.

The default configuration of any project is **project.config**. You can create more configurations in a project and can make one active for a specific test case or suite.

If you create a configuration, you are not able to add any new keys in it. To add keys in the new config, add them in the project.config file. You can then select the newly defined keys added in the project.config file from the drop-down available in the new config file.

Note: A configuration file cannot contain a blank line.

Properties are added to your configuration automatically while you develop your test.

For example, when you enter the WSDL name in a test, the defined server name and port are replaced 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.

As another example, when you work with Enterprise JavaBeans (EJBs) or Java objects, you can switch hot deploy directories, or add extra JAR files to your class path, to use different versions of your Java code. Two standard properties exist for these locations: HOT_DEPLOY and MORE_JARS. You can set these properties in your configuration.

For information about other properties, see Properties.

Configurations are for storing properties that are related to the system under test. Avoid using them for storage of "test-like" parameters and global parameters. These parameters can be stored in a companion.

Note: Backslashes "\" are not preserved in configuration files. If you edit the config file manually and you add something with a backslash, the file is overwritten without the backslashes.

A configuration file is a text file with a .config extension that contains the properties as key/value pairs. Configuration files are integral to any test run.

Configuration files can be created or edited in any text editor and can be saved with a .config extension.

When starting a test run, you can select a configuration file. For more information, see Apply a Configuration While Running a Test Case.

We recommend that you establish a naming convention for configuration files, making it easier to identify alternate configurations.

These configuration files must be imported into DevTest.

More Information:

- Project Configuration
- Add a Configuration
- Mark a Configuration as Active
- Edit a Configuration
- Apply a Configuration While Running a Test Case

Project Configuration

Every project has a configuration file with the name **project.config**. This file is also known as the *project configuration*.

In the **Configs** folder of the **Project** panel, the project configuration appears in orange. The file extension is not shown.

You cannot rename or delete the project configuration.

When you double-click the project configuration, the **Properties Editor** window opens. The following graphic shows the **Properties Editor** window. The editor has three columns: Key, Value, and Encrypt.

Screenshot of Properties Editor

These parameters are standard parameters that are available in all configurations.

The project configuration contains the superset of the keys that are defined in every other configuration. To add parameters in other configurations, the parameters must be included in the project configuration. You can then select from the drop-down list in the other configurations.

By default, the project configuration is the active configuration of a project. You can change which configuration in a project is active.

Add a Configuration

A project can have many alternate configurations, but it can have only one active configuration. Any alternate configuration or the default configuration can be made active. The alternate configurations can only override default properties.

To add keys in the new configuration, add them in the project configuration. You can select the newly defined keys from the drop-down in the new configuration file. In a new configuration file, you are not able to add any new keys.

When you create a configuration file, you can only add keys that are already defined in the project configuration. Some standard keys are provided with the installation.

Follow these steps:

1. Right-click the **Configs** folder in the **Project** panel and select **Create New Config**.
2. Enter the name of the new configuration.
3. Click **OK**.

Mark a Configuration as Active

When you want to apply a different configuration to your project, make it active. In the **Configs** folder of the **Project** panel, you can mark any configuration as active. The active configuration applies to the whole project, not a single test case.

If the project configuration is active, it is marked orange and other configurations are black.

If a secondary configuration is active, it is marked purple and the project configuration remains orange.

Project configs folder

Each test case in a project shares the active configuration that is applied to the project. You cannot assign a separate configuration for each test case in a project.

If an active configuration is selected, **Stage a Quick Test** uses it. If not, it uses the default configuration (project.config).

To mark a configuration as active, right-click the configuration in the **Configs** folder and select **Make Active**.

Edit a Configuration

In any configuration other than the project configuration, you can add properties only if they exist in the project configuration.

A best practice is to use properties in the path names that are stored in the configuration. Properties such as **LISA_HOME** or **LISA_PROJ_ROOT** allow for portability of test cases.

A property value can contain multiple lines.

The extended view consists of a dialog for editing a property value. This view can be useful when the value is long or when the value contains multiple lines. To access the extended view, right-click the property value cell and select **Launch Extended View**.

Follow these steps:

1. Double-click the configuration in the **Configs** folder.
   The Properties Editor appears.

2. Add a property by clicking **Add** at the bottom of the Properties Editor.
   A new line is added to the property list.

3. Click the drop-down to select the chosen key.
   Common property names appear in the drop-down.
   Values:

   - **HOT_DEPLOY**
     Indicates the location of the hot deploy directory.

   - **MORE_JARS**
     Add more JAR files to your class path.

   - **NOTIFY_ON_FAIL**
     Defines an email address to be notified when a test case fails.

   - **RESOURCE_GROUP**
     Filters out the selection of coordinators and VSEs based on the resources that are defined in a resource group.

Apply a Configuration While Running a Test Case

For information about applying a configuration when running a test case, see Stage a Test Case.

Assets

An *asset* is a set of configuration properties that are grouped into a logical unit.

Typically, an asset represents a point of communication with an external application or an intermediate client/server component necessary for communication with an application. Types of assets include JMS connection factory, JMS destination, JNDI context, and SAP JCo destination.

The main benefit of assets is reuse. You do not need to enter the same properties multiple times. Instead, you define the properties once as part of an asset.

The following parameters are common to all assets:

- Name
- Description
- Run-time scope

Assets are associated with a configuration. When you open a configuration, the asset browser is located to the right of the properties editor. The project configuration contains the superset of all the assets in the other configurations. You can configure an asset in one of the other configurations to override the corresponding asset in the project configuration.

You can create assets from scratch or from test steps.

More Information:

- Asset Browser
- Asset Editor
- Asset Inheritance
- Runtime Scope
- Create Assets
- Create Assets from Test Steps
- Modify Assets
- Verify Assets

Asset Browser

When you open a configuration, the asset browser is located to the right of the properties editor.

Note: If the asset browser is empty, then no assets have been created.

The main area contains a tree structure. The top-level nodes are the *asset categories*. The second-level nodes are the *asset types*.

The name and description parameters appear below the main area.

An add button appears in the lower left corner.

The following graphic shows the asset browser. The assets in this example are divided into two categories: JMS and JNDI. The JMS category has four asset types: JMS Connection Factory, JMS Destination, JMS Session, and JMS Connection. The JNDI category has one asset type: JNDI Context.

Screenshot of asset browser

If an asset is associated with a project configuration, you can create a copy of the asset by right-clicking the asset and choosing **Duplicate**.

If an asset is also associated with another configuration, right-click the asset and select **Duplicate in project config** to create a copy of the asset.

Asset Editor

You use the asset editor to perform the following tasks:

- Create assets from scratch
- Modify assets

The asset editor has two modes: basic and advanced.

The following graphic shows the basic mode. The name and description parameters are standard for all assets. The parameters below the first horizontal separator vary depending on the asset type.

Screenshot of asset editor in basic mode

The following graphic shows the advanced mode. You can display the advanced mode by clicking **PRO** in the upper right corner.

Screenshot of asset editor in advanced mode

An asset can have a parameter whose value is another type of asset. For example, the JMS session asset has a parameter in which you specify a JMS connection asset. The JMS connection asset has a parameter in which you specify a JMS connection factory asset.

Some parameters let you change the editor so that you can enter a property as the value.

Some parameters that provide a discrete set of values let you change the editor so that you can enter the value directly.

You can use the green button in the title bar to verify the asset.

Asset Inheritance

Every project has a project configuration. A project can also have one or more other configurations.

The project configuration contains the superset of all the assets in the other configurations.

You can configure an asset in another configuration to override the corresponding asset in the project configuration.

The following graphic shows an example. The project configuration has an asset with the name **myAsset**. This asset includes the following parameters: name, description, and mode. The second configuration has the same asset. However, the value of the mode parameter in the second configuration is different from the value of the mode parameter in the project configuration. Thus, the asset in the second configuration is overriding the asset in the project configuration.

The asset can even be a different class, as long as the type is the same.

For example, assume that the project configuration has an IBM MQ connection factory asset. In the additional configuration, you can change the asset to be a TIBCO connection factory asset. This change is possible because the IBM MQ connection factory asset and the TIBCO connection factory asset have the same type: JMS Connection Factory.

The asset browser uses color coding to show the inheritance setting for the assets in each additional configuration:

- If an asset is gray, then it does not override an asset in the project configuration.
- If an asset is black, then it overrides an asset in the project configuration.

Runtime Scope

The runtime scope specifies the minimum level at which an open asset instance is cached and reused at runtime.

Each asset has a runtime scope. In addition, an operation in a test step can have a runtime scope.

The valid values are:

- **Step:** An open asset instance is cached and reused during the step in which it is accessed. When the step is finished, the asset instance is closed.
- **Model:** An open asset instance is cached and reused during the model execution in which it is accessed. When the model execution is finished, the asset instance is closed.
- **Staged:** An open asset instance is cached for all instances of the model in which it is accessed to reuse. When all instances of the staged model are finished, the asset instance is closed.
- **Global:** An open asset instance is cached for all clients in the current JVM to reuse globally. When an asset instance has been idle for a short time, the asset instance is closed. This scope is the largest possible scope.
- **Default:** An open asset instance is assigned to the smallest scope available.

If an asset is used in an operation, the runtime scopes for the asset and the operation can differ. DevTest evaluates the runtime scopes and determines the proper one to use.

Create Assets

You can create assets in the asset browser or the asset editor.

Follow these steps:

1. Complete one of the following actions:
   - In the asset browser, click **Add** in the lower left corner and select the asset type.
   - In the asset editor, click **Add New Asset** and select the asset type. The asset types that are available depend on the parameter where the icon appears.
2. Complete the necessary information. If you leave the **Name** field blank, a name is automatically generated.
3. Click **OK**.

Create Assets from Test Steps

You can create assets from JMS-related test steps.

Multiple assets can be created from a single test step. For example, using a **JMS Messaging (JNDI)** step can result in the following asset types:

- JMS connection factory
- JMS connection
- JMS session
- JMS destination
- JNDI context

If you reexport from a test step, any changes that you made to an asset from the previous export are overridden.

Follow these steps:

1. Open a test case.
2. Select one or more test steps.
3. In the model editor toolbar, click the **Generate assets from the selected steps** icon.

Modify Assets

This page describes how to modify an asset.

The changes that you can make include the asset class. For example, you can change a JMS session asset to a JMS queue session asset or a JMS topic session asset. The icon for changing the asset class is located in the title bar of the asset editor.

Follow these steps:

1. Do one of the following actions:
    a. In the asset browser, double-click an asset.
    b. In the asset browser, right-click an asset and select **Edit** or **Override**.
    c. In a step editor, click **Edit Selected Asset**.
    d. In the editor of another asset that includes the asset as a parameter, click **Edit Selected Asset**.
    e. Complete the modifications.
2. Click **OK**.

Verify Assets

You can verify an asset from the asset editor during the following procedures:

- Creating assets
- Modifying assets

The verification process tries to access the object that the asset represents. The verification process does not perform the functional operation.

Follow these steps:

1. Ensure that the application, or at least the endpoints that it uses, are running and available to DevTest.
2. In the asset editor, click the green **Verify** button.
3. View the log window for a success or failure message.

Filters

A *filter* is an element that runs before and after a test step, enabling you to change the data in the result, or to store values in properties. Use filters to extract values from web pages, XML and DOM responses, Java objects, text documents, or other test step responses.

Most filters execute after the step runs.

After 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. Filters can be used to save a property value to a file, or convert a property to be the "last response". Filters are used mainly to set properties.

A filter can be applied 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 *global* filter is defined at the test case level and executes before or after test steps that are 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 that is defined at the test step level is a step filter, and executes before or after each execution of that test step.

You can add as many global and step filters as necessary. The filters are executed in the order that they appear in the test case.

More Information:

- Add a Filter
- Drag and Drop a Filter

Add a Filter

You can add a filter in the following ways:

- Add a Filter Manually
- Add a Filter from an HTTP Response
- Add a Filter from a JDBC Result Set
- Add a Filter from a Returned Java Object

Add a Filter Manually

To add a filter manually, select the filter type from a list and enter the parameters for the filter.

You can add two types of filters manually: global filters and step filters.

Global filters apply to and automatically run for every step in the test case, unless a step is instructed otherwise.

A step filter applies only to one step and executes for that step only.

To add a global filter manually:

1. To open the **Test Case Elements** panel, open a test case and click anywhere in the editor.
2. On the Global Filters element, click **Add** to add a global filter.
3. You are prompted to select a filter type of Integration Support for CAI or Integration Support for webMethods Integration Server.
   For more information about adding each of these types of filters, see Integration Support for CAI or Integration Support for webMethods Integration Server.

4. When you have at least one global filter on a test case, for each step, by default, the **Use Global Filters** check box is selected. If you do not want to apply a global filter for a step, clear the box.

Screenshot of test step collapsed

To add a step filter manually:

1. Select the step for which you want to apply the filter and in the right panel, click the **Filter** element.

Screenshot of filters section

2. To list the available filters, click **Add** on the filter element, or right-click the step and select **Add Filter** and select the appropriate filter for this step.
The **Filter** menu opens, listing the available filters. Each filter has its own editor and applicable parameters. For more information about each filter type, see Types of Filters.

Add 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.

This example uses the response of the login step in the multi-tier-combo test case in the examples directory. The point of this example is to capture the text where **MyMoney Home** currently appears on the window. (It is not always the same text).

Follow these steps:

1. Run the multi-tier-combo test case in the ITR, and then select the Login test step.

Screenshot of multi-tier-combo in ITR with Login step selected

2. Select the text MyMoney Home in the **View** tab

3. To see that this text is selected in the tree view, click the **DOM Tree** tab.

Screenshot of multi-tier-combo in ITR, DOM Tree view of Login step

4. To apply an inline filter, double-click the login step in the model editor to open the HTTP /HTML Step Editor.

Screenshot of multi-tier-combo in ITR

5. Move to the **DOM Tree** tab, find MyMoney Home in the DOM Tree view, and select it.

6. At the bottom of the window, in the **Select a Command** box, select **Parse Value Filter** from the drop-down list.

7. In the dialog that is displayed, enter the name for the Property Key "wasAdded":

HTML/XML Filter Info dialog

8. Click **OK**.

Parse Text from XML Filter

You can also add an assertion here. For example, you would probably want to test the value of the property **wasAdded**, to see if it is in fact equal to Added user. More information is available in Adding Assertions.

The generated filter appears as a filter in the login test step.

The same filtering capabilities are available when an HTML response is displayed in the step editor.

Add 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.

This example demonstrates how to add a filter from the JDBC Result Set response using the response from the Verify User Added step in the multi-tier-combo test case.

Follow these steps:

1. To open the step editor, double-click the Verify User Added step.

Screenshot of multi-tier-combo Verify User Added step

2. To get values in the result set, edit the SQL statement to read **select * from users** and click **Test/Execute SQL**.

Add a Filter from a JDBC Result Set SQL statement

3. To get values in the result set, click the **Result Set** tab and click **Test/Execute SQL**.

SQL Database Execution (JDBC) - Verify User Added

Result Set

LOGIN	PWD	NEWFLAG	FNAME	LNAME	EMAIL	PHONE	ROLEKEY	SSN
dmxxx-009	tIDAdNa3...	1	first-9	last-9	test@test...		1	
Testuser	IGyAQTu...	0	Test	User	anne@itk...	817-433-...	1	433-87-3...
TEST1	nU4eI71b...	1					1	
First1	8FePHnF0...	1	Firstname1	Lastname1	first1@itk...	555-555-...	1	555-55-8...
Last1	i+UhJqb9...	1	Firstname2	Lastname2	last1@itko...	333-448-...	1	533-88-9...
test1	nU4eI71b...	1	First1	Last1	test1@itk...	214-111-...	1	111-11-1...
test2	nU4eI71b...	1	First2	Last2	test2@itk...	215-222-...	1	222-22-2...
admin	0DPiKuNIr...	1	iTKO	Admin	lisabank-a...	123-4567	2	434-47-5...
sbellum	26yJsXNp...	1	Sara	Bellum	sbellum@...	232-4345	1	614-40-1...
wpiece	/UuJ0Me...	1	Warren	Piece	wpiece@...	455-3232	1	546-71-4...
areck	AHDRRjD...	1	Amanda	Reckonwith	areck@my...	555-2244	1	350-02-1...
boaty	RQIil0Ldp...	1	Boaty	Rabbit	boaty@ra...	333-4521	1	616-51-0...
itko	qUqP5cyx...	1	itko	test	itko.test@...	650-234-...	1	140-72-2...
lisa_simpson	60fAFoq+...	1	lisa	simpson	lisa.simps...	123-456-...	1	295-20-0...
virtuser	nU4eI71b...	1	Virtual	User	virtuser@i...	123-456-...	1	297-55-9...
demo	89yJPVNn...	0	DEMOFIRST	DEMOLAST	demo@itk...	817-433-...	1	677234567

Base | Result Set

Add Filter from a JDBC Result Set - Verify User Added

4. Click the cell in the result set tab that represents the location of the information to capture (sbellum).

5. Click **Generate Filter for Current Col/Row Value** .

6. In the dialog that opens, enter the property key *theLogin*.

Add Filter from a JDBC Result Set - Verify User Added - Enter property key for value

7. Click **OK**.
   DevTest adds a filter with the name **Parse Result Set for Value** in the list user step.

8. To see the filter, click the filter editor.

Parse Result Set for Value window

In the example, the value in the cell in the first column, and eighth row, **sbellum**, is stored in the property theLogin.

Applying a Second Filter

A second filter 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.

Follow these steps:

1. From the result set, select the two values in two different columns from the same row, using the Ctrl key.

LOGIN	PWD	NEWFLAG	FNAME	LNAME	EMAIL	PHONE	ROLEKEY	SSN
dmxxx-009	tIDAdNa3...	1	first-9	last-9	test@test...		1	
Testuser	IGyAQTu...	0	Test	User	anne@itk...	817-433-...	1	433-87-3...
TEST1	nU4eI71b...	1					1	
First1	8FePHnF0...	1	Firstname1	Lastname1	first1@itk...	555-555-...	1	555-55-8...
Last1	i+UhJqb9...	1	Firstname2	Lastname2	last1@itko...	333-448-...	1	533-88-9...
test1	nU4eI71b...	1	First1	Last1	test1@itk...	214-111-...	1	111-11-1...
test2	nU4eI71b...	1	First2	Last2	test2@itk...	215-222-...	1	222-22-2...
admin	0DPiKuNIr...	1	iTKO	Admin	lisabank-a...	123-4567	2	434-47-5...
sbellum	26yJsXNp...	1	Sara	Bellum	sbellum@...	232-4345	1	614-40-1...
wpiece	/UuJ0Me...	1	Warren	Piece	wpiece@...	455-3232	1	546-71-4...
areck	AHDRRjD...	1	Amanda	Reckonwith	areck@my...	555-2244	1	350-02-1...
boaty	RQIil0Ldp...	1	Boaty	Rabbit	boaty@ra...	333-4521	1	616-51-0...
itko	qUqP5cyx...	1	itko	test	itko.test@...	650-234-...	1	140-72-2...
lisa_simpson	60fAFoq+...	1	lisa	simpson	lisa.simps...	123-456-...	1	295-20-0...
virtuser	nU4eI71b...	1	Virtual	User	virtuser@i...	123-456-...	1	297-55-9...
demo	89yJPVNn...	0	DEMOFIRST	DEMOLAST	demo@itk...	817-433-...	1	677234567

Add a Filter from a JDBC Result Set - Result set with two cells selected

2. Select **Filter for a value** and then get another column value filter using the icon. To create this filter, select two cells in the same row. One is the search column and the other is the column whose value you want to extract.

3. In the dialog that opens, select or reassign the columns for the search and the value.

4. Enter the property key *theEmail*.

Generate Value for Value Filter dialog

5. Click **OK**.
   DevTest adds a filter with the name **Get Value For Another Value in a ResultSet Row** in the Verify User Added step.

Get Value For Another Value in a ResultSet Row filter

The Get Value For Another Value in a ResultSet Row filter looks for **sbellum** in the LOGIN column. If the filter finds **sbellum**, it stores the value in the EMAIL column in the same row in a property with the name **theEmail**.

> **Note:** The same filtering capabilities are available when a JDBC result set is displayed in the Step Editor.

Add a Filter from a Returned Java Object

When the test step result is a Java object, you can use the **Inline Filter** panel in the Complex Object Editor to filter the returned value directly from the method call.

This example uses the Get User step (EJB step) in the multi-tier-combo test case in the examples directory.

Follow these steps:

1. To open the step editor for the Get User step, double-click the step.

2. Click **Next**. On the next window, click **Finish**.

Get user step Object Editor

3. To open the Object Call Tree, click **Show Editor**.

Object Call Tree

4. Enter an input parameter "itko" in the value field.

5. Click **Execute** to execute this method.
   The returned value upon executing the getLogin method is stored in the property getUserObject. Notice that in this case the returned value is an object (of type UserState). You also can add an assertion here.

6. Select the **Expert Mode** check box.
   The **Call** tab displays the filter parameters.

Object Call Tree Expert Mode

You could also call a method on the returned object to get the login value for this user, and save the login in another property.

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 Complex Object Editor.

Drag and Drop a Filter

You can drag and drop filters in the model editor from one test step to another.

Follow these steps:

1. Click the filter that is attached to a test step; for example, Step1.

2. Drag and drop that filter to another test step in the model editor; for example, Step2. The dragged filter is applied to Step2.

Assertions

An *assertion* is a DevTest code element that runs after a step and all its filters have run. An assertion verifies that the results from the step match expectations.

The result of an assertion is a Boolean value (true or false).

The outcome can 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 change the test case workflow dynamically by introducing conditional logic (branching) into the workflow. An assertion operates very much like an "if" conditional block in programming.

For example, you can create an assertion for a JDBC step that ensures that only one row in the result set contains a specific name. If the results of the JDBC step contain multiple rows, the assertion changes the next step to execute. In this way, an assertion provides conditional functionality.

The test case flow is often modeled with one of the following two possibilities:

- The next step that is defined for each step is the next logical step in the test case. In this case, the assertions are pointing to a failure.
- The next step is set to fail, and the assertions all point to the next logical step.

The choice depends, usually, on the actual logic being employed.

Note: If an assertion references an unresolved property, a model definition error is raised. The model definition error does not cause the test to terminate, but it cautions the test author that an unresolved property was encountered. The problem that an unresolved property creates is that an assertion cannot give the proper verdict because the assertion does not have enough information. Not having enough information results in false positives or false negatives. (Most of the assertions return "false" if an unresolved property is encountered, but that is not an enforced rule.) By running a test in the ITR and inspecting the test events panel for model definition errors, you can determine if any unresolved properties exist.

You can add as many assertions as are necessary, which gives you the capability to build a workflow of any complexity. Assertions are the only objects that can change the DevTest workflow.

Note: Assertions are executed in the order that they appear, and the workflow logic usually depends on the order that the assertions are applied.

After an assertion fires, the next step can be configured as the assertion determines, and the remaining assertions are ignored. Each time an assertion is evaluated and fired, an event is generated.

Global and Step Assertions

As with filters, you can apply assertions as global. That is, assertions can apply either to the entire test case cycle or as a step assertion, where they apply only to a specific step.

More Information:

- Add an Assertion
- Assertions Toolbar
- Reorder an Assertion
- Rename an Assertion
- Drag and Drop an Assertion
- Configure the Next Step of an Assertion

Add an Assertion

You can add an assertion in the following ways:

- Add an Assertion Manually
- Add an Assertion from an HTTP Response
- Add an Assertion from a JDBC Result Set
- Add an Assertion for Returned Java Object

Add an Assertion Manually

To add an assertion manually, select the assertion type from a list and enter the parameters for the assertion. Manual assertions are of two types:

- Global assertions are defined at the test case level. A global assertion applies to all the steps in the test case and is automatically run for every step, unless a node is instructed otherwise.
- Step assertions are defined at the test step level. This type of assertion applies only to that step and executes for that step only.

Add a Global Assertion

Follow these steps:

1. Open a test case and in the right panel click the **Global Assertions** element.
   You can apply the following types of global assertions:

1. - **HTTP**
     - Simple Web Assertion
     - Check Links on Web Responses
   - **XML**
   -
     - Ensure Step Response Time

- **Other**
  - Ensure Result Contains Expression
  - Ensure Step Response Time
  - Scan a File for Content

The following graphic shows a global assertion that is applied to the multi-tier-combo test case.

Add a Step Assertion

Follow these steps:

1. Complete one of the following actions:
   - Select the step for which you want to apply the assertion and in the right panel click the Assertion element.
   - Right-click the step, select **Add Assertion**, and select the appropriate assertion for the step.

   The following graphic shows step assertions that are applied to the Add User step in the multi-tier-combo test case.

2. To add an assertion, click **Add** on the assertion toolbar.
   Or, you can right-click a step in the model editor to add an assertion. The assertion panel appears and shows a menu of assertions that can be applied to the step.

Select and Edit an Assertion

Follow these steps:

1. Complete one of the following actions:
   - Click the step to which the assertion is applied.
   - Click the assertion that relates to that step in the **Assertion** tab.
2. To open the assertion editor, double-click the assertion. The editor is unique for each type of assertion.

Add 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.

This example of the HTTP/HTML response uses the login step in the multi-tier-combo test case. The point of this example is to test whether the text "MyMoney Home" appears in the response.

Follow these steps:

1. Run the multi-tier-combo test case in the ITR.
2. Double-click the Login step in the model editor.

3. Select the text "MyMoney Home" in the **View** tab.

4. To view and verify that this text is selected in the tree, click the **DOM Tree** tab.

5. From the **Select a Command** pull-down menu at the bottom of the panel, select **Make Assert on Selection**.

6. In the displayed window, enter the expression that you expect the selected text to match, then select the appropriate assertion behavior.

In this example, the assertion fires if the text "MyMoney Home" is not present, and then redirects to the fail step.

7. Click **OK** to save the assertion.
The assertion that was generated appears in the login step.

Running one Filter and one Assertion

Alternatively, if you wanted a filter to capture the value "MyMoney Home," and run it as an assertion, use the Parse Value filter.

The HTML/XML Filter Info window shows that the Property Key value is the filter to apply and Expression is the assertion to fire.

As a result, one filter and one assertion are added to the login step and appear n in the model editor.

Note: The same assertion capabilities are available when an HTML response is displayed in the step editor.

Add 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 example shows how to add an assertion in this way.

Here is an example for a result set response, using the response of Verify User Added step in multi-tier-combo test case in the examples directory (multi-tier-combo.tst).

Follow these steps:

1. Select the Verify User Added step, and double-click it to open its editor window. Edit the SQL statement so it reads **select * from users**.

2. To run the query, click **Test/Execute SQL**.

3. Select the **Result Set** tab and click the cell in the result set that represents the information that you want to test for (for example, **sbellum**).

SQL Database Execution (JDBC) - Verify User Added

Result Set

LOGIN	PWD	NEWFLAG	FNAME	LNAME	EMAIL	PHONE	ROLEKEY	SSN
dmxxx-009	tIDAdNa3...	1	first-9	last-9	test@test...		1	
Testuser	IGyAQTu...	0	Test	User	anne@itk...	817-433-...	1	433-87-3...
TEST1	nU4eI71b...	1					1	
First1	8FePHnF0...	1	Firstname1	Lastname1	first1@itk...	555-555-...	1	555-55-8...
Last1	i+UhJqb9...	1	Firstname2	Lastname2	last1@itko...	333-448-...	1	533-88-9...
test1	nU4eI71b...	1	First1	Last1	test1@itk...	214-111-...	1	111-11-1...
test2	nU4eI71b...	1	First2	Last2	test2@itk...	215-222-...	1	222-22-2...
admin	0DPiKuNIr...	1	iTKO	Admin	lisabank-a...	123-4567	2	434-47-5...
sbellum	26yJsXNp...	1	Sara	Bellum	sbellum@...	232-4345	1	614-40-1...
wpiece	/UuJ0Me...	1	Warren	Piece	wpiece@...	455-3232	1	546-71-4...
areck	AHDRRjD...	1	Amanda	Reckonwith	areck@my...	555-2244	1	350-02-1...
boaty	RQIil0Ldp...	1	Boaty	Rabbit	boaty@ra...	333-4521	1	616-51-0...
itko	qUqP5cyx...	1	itko	test	itko.test@...	650-234-...	1	140-72-2...
lisa_simpson	60fAFoq+...	1	lisa	simpson	lisa.simps...	123-456-...	1	295-20-0...
virtuser	nU4eI71b...	1	Virtual	User	virtuser@i...	123-456-...	1	297-55-9...
demo	89yJPVNn...	0	DEMOFIRST	DEMOLAST	demo@itk...	817-433-...	1	677234567

4. Click **Generate Assertions for the Value of a Cell** in the toolbar below the **Result set** window.
   We want to test that **sbellum** appears in a cell in the LOGIN column.

5. In the **Generate JDBC Result Set Value Assertion** dialog, enter the test step (fail) to redirect if the value is not found:

DevTest creates an assertion with the name Ensure JDBC Result Set Contains Expression in the Verify User Added step.

Note: The same assertion capabilities are available when a JDBC result set is displayed in the step editor.

Add an Assertion for Returned Java Object

When the test step returns a Java object, use the **Complex Object Editor** inline assertion panel to add an assertion on the returned value directly from the method call. The following example shows how to add an assertion this way.

This example uses the Get User (an EJB step) step in multi-tier-combo test case in the examples directory.

Follow these steps:

1. Enter an input parameter **itko**, and execute the method call getUser. Then, execute the getPwd call on the UserState object that was returned from that call.

Screenshot of the Complex Object Editor inline assertion panel to add an assertion on the returned value directly from the method call

2. To open the **Status/Result** pane where you can add the assertion, select the **Expert Mode** check box in the left pane.

Screenshot of the Status/Result pane

The returned value upon executing the getPwd method is stored in the property CurrentPassword.

3. Add an assertion that tests if the returned value is equal to the string "test". If it is not that, then redirect to the Fail step.

4. Click **Execute** to execute this step.

> **Note:** Inline assertions (and filters) do not result in an assertion being added to the test step. Inline assertion management is always done in the **Complex Object Editor**.

For more details on the **Complex Object Editor**, see Complex Object Editor (COE).

Assertions Toolbar

All the elements have a toolbar to add/delete/reorder at the bottom of the element.

Reorder an Assertion

You may need to reorder assertions, because assertions are evaluated in the order in which they appear. Changing the order of the assertions can affect the workflow.

To reorder an assertion:

- Select the assertion in the **Elements** tab and click **Move Up** or **Move Down** on the toolbar.
- Drag-and-drop the assertion in the model editor to the target destination.

Rename an Assertion

You can rename an assertion.

Complete one of these steps:

- 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.
  To change the assertion name back to the default assertion name, select the assertion and click **Revert to Default Name** .

Drag and Drop an Assertion

To drag and drop assertions in the model editor from one test step to another test step:

1. Click the assertion in one test step; for example, Step1.
2. Select and drag the assertion to other test step in the model editor; for example, Step2.
3. The dragged assertion is then applied to Step2.

Configure the Next Step of an Assertion

An assertion added to a step can be seen in the model editor.

Get User step with an assertion highlighted

After the assertion is added to a step, you can select its next step to be executed, if you want the workflow to be altered.

To configure the next step of an assertion:

1. To open the menu, right-click the assertion in the model editor.
2. Select **If triggered**, then and do one of:
   - Select to generate a warning or error.
   - Select to end, fail, or abort the step.
   - Select the next step to be executed.

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 capability provides a mechanism to introduce external test data to a test case.

Data sets are often rows of data that can be inserted into properties as name/value pairs. However, sometimes a data set returns a single property value.

Data sets can be created internal to DevTest, or externally: for example, in a file or a database table.

While a test is running, DevTest assigns properties to the steps specified in the data set editor. When the last data value or values are read from the data set, one of the following actions can occur:

- The data can be reused starting at the top of the data set
- The test can be redirected to any step in the test case

Data sets can be global or local.

More Information:

- Global and Local Data Sets
- Random Data Sets
- Example Scenarios of Data Sets
- Add a Data Set

- Rename a Data Set
- Data Set Next Step Selection
- Data Sets and Properties

Global and Local Data Sets

Data sets can be global or local.

Global Data Sets

By default, a data set is global.

The coordinator server is responsible for providing 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 Sets

You can make the data set local by selecting the **Local** check box while building the data set.

Each instance gets (essentially) its own copy of the data set.

A local data set provides one copy of the data set to each instance being run.

Example

This example has the following components:

- Three concurrent virtual users
- A local data set with 100 rows of data
- A test case that loops over 100 rows of data and stops

Each virtual user sees all the 100 rows of data in the data set.
For a local data set

A single run (one vuser): Test Case A, gets 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 three virtual users or it is staged to run continuously, each test case receives the next row of data. DevTest shares a data set across multiple runs, using the instructions that are specified in the staging document.

Each virtual user (instance) gets its own copy of the data set that is local.

When local is marked and Test Case A has a loop, the test reads every row of data in the data set. Each run gets 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.

Looping can be implemented several ways:

- A test is set to finish when all the data in the data set is exhausted.
- A test is set to reuse the data set when the data is exhausted.
- A test step can call itself, or you can configure a series of steps that loops until the data in the data set is exhausted.

Use a numeric counter data set to cause a specific step to run a fixed number of times. You can set the frequency at the step level or the 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. To achieve this test, set a single step to call itself until the data set (with 100 rows of data) is exhausted. When the data set is exhausted, the test can redirect to the next natural step in the case, or the End step. Alternatively, you can use a counter data set with the user ID/password data set.

If you configure a global data set in the first step of a test case to end the test when the data set is exhausted, all test instances end for a staged run.

This overrides other staging parameters such as steady state time. Local data sets do not end the staged run in this fashion nor will data sets on steps other than the first test.

Random Data Sets

A *random data set* is a special type of data set; you can think of it as a wrapper around another data set.

You can select a data set to be randomized for specific steps, and also select the maximum number of records for randomization.

When a random wraps a data set, *n* copies of the data set are added to the random list, where *n* = Max Number of rows. So when *n*=10, DevTest makes 10 copies of the rows from the data set.

When a step references the random data set, a random row is selected from the data set.

If you have a random data set with the name **RAND1** reads one column with the name **Fruit**, and the **Max Records to Fetch** is set to 10, the random number **RAND1** is generated to pick a row. **RAND1** is in the range 0 through *n*-1. Fruit is the value of the "Fruit" column that is found in that row.

To make a data set random:

1. Select the data set to make random. This example uses the Read Rows from Delimited File data set.

Read Rows from Delimited File data set window, with Random check box selected

2. Select the **Random** check box.

3. Enter the **Max Records to Fetch** number.
   This number is the maximum number of records to use from the data set. If the number is larger than the records in the data set, the smaller number is used to create the random set.

Example Scenarios of Data Sets

To show the behavior of tests using data sets, consider the following scenarios:

A test has 15 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 starts over and reads the first row, and so forth. This cycle continues until all 15 users have run the test. The first row was read eight times; the second 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.

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 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 restarts with the first row until the test run ends.

Scenario 2: At end of data -> Execute End step

The test run ends after 1500 cycles.

A test has 10 virtual users, and the data set has 10,000 rows of data. The staging document specifies that the test will run for 2 minutes.

The 10 users start and read the first 10 rows of data and continue consuming rows of data from the 10,000 rows. How fast they run determines how far down the data set they go. At the two-minute mark, the test ends.

Add a Data Set

To add a data set:

1. In the model editor, select a test case.
2. In the right panel, expand the **Data Sets** tab.
3. Click **Add** to open the data set panel listing the Common Data Sets.
4. Click the required data set to open the appropriate Data Set Editor. The editor is specific to each data set.

Data Set Editor Example

The following graphic shows the editor for the Read Rows From a Delimited File data set.

Read Rows from Delimited File data set window, with Random check box selected

The top panel of the data set editor is common to all data set types. The options are:

- **Name**
  The name of the data set.

- **Local**
  To add a local data set, select the check box. The default is cleared (global).

- **Random**
  To make the data set a random data set, select the **Random** check box and enter the maximum records to fetch.

- **At end of data**
  Instructions for how to proceed after all the data is read. You can:

  - **Start over**
    Continue reading data from the top of the data set.

  - **Execute**
    Select the step to execute after all the data is read. The pull-down menu is prepopulated with all the available steps in the test case.

- **Test and Keep**
  After all the parameters are entered, click this button to test the data set, and to load it into the steps in the test case.

The bottom panel of the data set editor is specific to the data set being created. For the Read Rows From a Delimited File data set, enter:

- **File Location**
  Enter 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 example, LISA_HOME).

- **Delimiter**
  Enter the delimiter being used. Any value can be a delimiter. Some common delimiters are provided in the drop-down list.

This data set requires a delimited text file. In the following example, the first line specifies the property names: UserID and Password. The subsequent lines list the data values to be used for these properties.

Notepad file with five users listed

When you click **Test and Keep**, the first row of data is loaded. A message confirms that the data set can be read and shows the first row of data.

Data Set Editor message that test was successful

Ending a Test Case by a Data Set

The following conditions must be met for a data set to end a test run:

- The data set must be global.
- The data set must be configured to "**At end of data: Execute end**".
- The data set must be increased on the first step of the test case.

Apply a global data set to the first step in a test case carefully because it pulls down the entire staged run before completing the cycles.

Rename a Data Set

To rename a data set:

- Select the data set in the **Elements** panel and click the **Rename** icon on the toolbar.
- Select the data set in the model editor and right-click to open a rename menu.
- Select the data set in the **Elements** panel and click **Revert to Default Name** to reset the data set name to the default.

Move a Data Set

You can move data sets in the model editor from one test step to another by using drag-and-drop.

Follow these steps:

1. Click the data set attached to a test step: for example, Step 1.

2. Select the data set and drag it to the target test step: for example, Step 2, in the model editor and leave it there.

3. The dragged data set is then applied to Step 2.

Data Set Next Step Selection

You can select the next step or the end step to be executed after the data set fires.

Follow these steps:

1. To open the menu, right-click the data set in the model editor.

2. Click the **At end** menu and select the next step to be executed.

Data Sets and Properties

Data sets can use properties.

Important uses of properties include the following:

- 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
  Using a property instead of a hard-coded value increases data set portability. Because properties used in this way must typically be available early in the test run, you must define them in one of the following ways:

  - As a system property

  - In a configuration

  You can define a dummy value in your configuration and can modify it later. This value allows the file to be found at design time. This use of properties is important when you are running your tests on DevTest Server.

- Data set values can contain properties. These values are 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 use properties from the test case, but global data sets cannot because all virtual users share them.

Companions

A *companion* is an element that runs before, after, or both before and after every test case execution. Companions are used to configure behavior that is global to the test case. These behaviors can include simulating browser bandwidth and browser type, setting synchronization points in load tests, and creating a sandbox class loader. In a way, companions set a context for the test case execution.

Companions work as helpers for the test case, before any of the steps are executed.

At the bottom of every element, there is a toolbar that has icons to add, delete, and reorder.

More Information:

- Add a Companion
- DevTest Hooks

Add a Companion

To add a companion:

1. Open a test case and click the **Companion** element in the right panel.
2. Click + **Add**. The **Companion** menu opens.
3. Each companion has a different editor. To open the appropriate editor, select the required companion. Each companion type, with all its parameters, is described in detail in Companion Descriptions.

DevTest Hooks

DevTest *hooks* are an automatic global means to execute logic at start/end of a test case. Hooks apply that logic to all test cases in the environment where the hook is deployed.

Hooks work similarly to companions; they run before a test starts and after a test finishes.

The difference is, hooks are defined at the application level, and every test case in DevTest runs the hooks. If a hook exists, it is used for every test case.

A *hook* is a mechanism that allows for the automatic inclusion of test setup logic or teardown logic, or both, for all the tests running in DevTest. An alternate definition of a hook is a system-wide companion. Anything that a hook can perform can be modeled as a companion.

Hooks are used to configure test environments, prevent tests that are improperly configured or do not follow defined best practices from executing, and provide common operations.

Hooks are Java classes that are on the DevTest class path. Hooks are NOT defined in . lisaextensions file, but in local.properties or lisa.properties as

```
lisa.hooks=com.mypackage1.MyHook1, com.mypackage2.MyHook2
```

A hook is executed or invoked at the start and end of all subprocesses in a test case in addition to the test case itself. If a test case has three subprocesses, the hook logic is executed four times. The logic is executed once for the main test and once for each of three subprocesses.

To prevent a subprocess from executing **startHook, endHook**, or both, include the following action:

```
String marker = (String)testExec.getStateValue(TestExec.
FROM_PARENT_TEST_MARKER_KEY)
"marker" is set to "true" only when it is a subprocess.
if (!"true".equals(marker))
all start/end Hook logic here that should not be executed when in sub process
```

Differences between Hooks and Companions

- Hooks are global in scope. Testers do not specifically include a hook in their test case as is required for companions. Hooks are registered at the system level. If you need every test to include the logic and do not want users to omit accidentally it, a hook is a better mechanism.

- Hooks are deployed at the install level, not at the test case level. If a test runs on two computers where only one has a hook registered, the hook runs only when the test is staged on the computer where the hook is deployed. Companions that are defined in the test case 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 because they are rendered in the model editor.

Complex Object Editor (COE)

The Complex Object Editor (COE) lets you interact with Java objects without having to write Java code.

You can change the current value of an input parameter, make method calls, and examine return values. You can also do simple in-line filtering and add simple assertions on the return value.

Many test steps involve the manipulation of Java objects. You work in the Complex Object Editor whether you are working directly with Java objects (as in a Dynamic Java Execution step or an Enterprise JavaBean (EJB) step) or indirectly (such as input parameters to a web service, or return messages from an Enterprise Service Bus (ESB)).

Wherever it appears, the Complex Object Editor (COE) looks the same.

For example, when you invoke the Dynamic Java Execution step, using the Customer class, you see the following window:

COE - Dynamic Java Execution step

Select to use the Local JVM and enter *com.itko.examples.dto.Customer* in the **Make New Object Of Class** field.

To load the object into the object editor, click **Construct/Load Object**.

After the object is loaded, the Complex Object Editor is invoked.

COE - Dynamic Java Execution step after COE is invoked

The object editor is divided into two panels.

The left panel, the **Object Call Tree**, tracks method invocations, and their input parameters and return values.

The right panel is the **Object State**, with a set of dynamic tabs that show you available options.

> **Note:** If the response is XML and it is larger than 5 MB, it is displayed in plain text with no DOM view. To adjust the default limit, configure the **gui.viewxml.maxResponseSize** property.

The previous window shows a Java object of type **Customer** loaded in the editor. This object was loaded using the Dynamic Java Execution test step, but any number of operations could have loaded it. No calls have been invoked on the object.

More Information:

- Object Call Tree Panel
- Data Sheet and Call Sheet Panels
- Using Data Sets in the COE
- Usage Scenarios for Simple Objects
- Usage Scenarios for Complex Objects

Object Call Tree Panel

As you manipulate your Java object (Customer), the **Object Call Tree** expands to track the calls that are invoked and the associated parameter values.

As an example, an **Object Call Tree** after several methods have been invoked follows:

Object Call Tree after several methods have been invoked

Object Call Tree Icons

The following icons are used to identify the branches in the **Object Call Tree**:

Icon	Description
Icon for the Complex Object Editor type	The type (class) of the object currently loaded, followed by response from calling toString method of object.
Icon for the Complex Object Editor constructor	The constructor that was called. This is shown if multiple constructors exist.

Icon for the Complex Object Editor non-executed method call	A method call that has not been executed.
Icon for the executed method call	A method call that has been executed.
Icon for the Complex Object Editor input parameter	The input parameters (type and current value) for the enclosing method.
Icon for the Complex Object Editor return value	The return value (current value if call has been executed) for the enclosing method.

Clicking an item in the **Object Call Tree** displays the appropriate set of tabs in the **Data Sheet** and **Call Sheet** in the right panel.

Right-clicking an item in the **Object Call Tree** displays a menu. You can execute all calls or you can mark all calls as unexecuted in the **Object Call Tree**.

Data Sheet and Call Sheet Panels

The right panel contains the following tabs:

- Data Sheet
- Call Sheet
- Doc

Data Sheet

The data that is shown in black can be edited in this tab. The data values are edited in the **Value As String** column. These values are always primitives or strings.

Dimmed values cannot be edited in this tab, but can be edited in other windows. For example, the address field is an object of type Address that cannot be edited in this tab.

Call Sheet

In the **Call Sheet** tab, the Complex Object Editor shows the methods/fields calls that are available, and their return types.

COE Call Sheet panel

To add a method, select it in the **Methods/Fields** list and click **Add Method** at the bottom of the tab. The selected method now appears in the **Object Call Tree** (in the left panel).

When you are in the object call tree, you can provide input parameters and can invoke the method.

Doc

The **Doc** tab displays any Java API documentation that has been made available for this class.

Object Interaction Panels

Each action that is displayed in the **Data Sheet** and **Call Sheet** has a different interface.

The tab and its interface change depending on what you have selected in the **Object Call Tree** in the left panel.

COE Data Sheet panel

Object Panels

The **Data Sheet**, **Call Sheet**, and **Doc** tabs are available when an object is selected in the **Object Call Tree**.

COE Call Sheet panel

Method Call Panels

If you select a method call in the **Object Call Tree**, the **Call** and **Doc** tabs are available in the right panel.

COE method call selected

You provide values for the input parameters here.

The information about each parameter is provided, so you only supply the value. This example 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. Entering an object as an input parameter requires more work. We describe several approaches to providing objects in the subsequent sections.

Notice that the **Expert Mode** at the bottom of the left panel is not selected, indicating that we are in the Simple mode.

Simple and Expert Mode

Two editing modes are available: simple and expert.

Simple Mode is useful when your object is a simple object, such as a Java Bean with only a default constructor and several setter/getter methods. This is the classic Data Transfer Object (DTO). These objects are common as inputs to web service calls. With objects of this type, you can switch back and forth between simple and expert mode. A DTO that contains a DTO as a property can be manipulated in simple mode. We see examples of this activity later.

Use Expert Mode for more complex objects, such as objects that have multiple constructors. Some composite objects that contain other complex objects cannot use the simple mode. The simple mode option is disabled if the current object requires expert mode.

All the graphics that were shown previously have used simple mode.

The following graphic shows the example that is shown in the previous graphic, but with expert mode selected.

COE 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) in the object editor.

> **Note:** The in-line filters and assertions that are applied are not seen in the filters or assertions list.

Several other differences become apparent in our later examples.

Input Parameter Panels

If you select an input parameter in the **Object Call Tree**, the tabs available in the right panel vary depending on the input parameter type: Primitives/Strings or Objects.

For input parameters that are primitives and strings, the following panel is displayed.

COE input parameter panel

You can edit the value in this panel in the Simple Value field.

To use a property as the value, click the **Property Value** option button to display the following panel.

COE Property Value selected

To open the available property keys, type the property name, use the pull-down menu, or click **List**.

For input parameters that are objects, the following panel is displayed.

COE Data Sheet, Object State

Return Value Panels

If you select a return value in the **Object Call Tree**, the tabs available in the right panel vary depending on the input parameter type.

For input parameters that are primitives and strings, the following panel is displayed.

COE String Value of Object panel

For input parameters that are objects, the following panel is displayed.

COE String Value of Object panel

Using Data Sets in the COE

A common way to provide data for a Java DTO object is a data set.

A data set, Read DTOs from Excel File, is provided for this purpose.

If the Excel data set exists, the property that contains the data set can be entered as a value for the DTO object.

COE - Using Excel Data Set as input

To initiate the creation of a new DTO data set from the object editor:

1. When a DTO object appears in the call list, right-click the **Name** or **Actual Type** to open a menu.

COE - Right-click menu on DTO object

2. Select the data set Read From Excel Data set to display the following window.

COE - Create XLS Data Set dialog

3. The parameters on this window are required to initiate the creation of this data set.

   Complete the following fields:

- **Property Key**
  The property that stores the current values from the data set.

- **Type of File**
  Select if it is an existing XLS file or to make a new XLS file.

- **File**
  The name of the Excel file that is the template for the DTO data.

- **Open file in Excel**
  Opens the spreadsheet in Excel.

- **End test when no more rows**
  Ends the test after all rows are read.

Usage Scenarios for Simple Objects

The following examples are based on the standard Java classes for simple objects. We have used classes from the demo server included with DevTest that are easy to reproduce 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.

For a simple DTO, parameter values can be entered into the **Data Sheet** panel as fixed values or properties. In the previous example, city could be set equal to the property currentCity.

To enter parameters using the DTO setters in the Call sheet panel:

COE Call Sheet tab with getter selected

1. In the **Call Sheet** tab, select a getter, such as setCity(java.lang.String city) and click **Add Method**.
   The method runs and **COE** opens in the **Call** tab.

COE Call tab opened

2. Enter the parameter value as a fixed value or a property. Use the DevTest property syntax, propname.

3. Click **Execute** to invoke the method.

4. 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, Expert Mode must be used, because the Date type is not a DTO. In fact the COE forces Expert mode.

COE in Expert Mode

But we can still enter parameter values and invoke methods. In the previous example, we execute the parse method, which requires one input parameter. We have entered it as a string value, 9/1/2007.

Note: In Expert Mode, we use the **Null** or **Use Property** check box to denote the parameter type. If neither is selected, fixed value is entered. We would not use the {{ }} property syntax here. Even if we were entering a property rather than a string value, we would enter only the property name.

Because we are in Expert Mode, we can add inline filters and assertions.

Usage Scenarios for Complex Objects

The following examples are based on the standard Java classes for complex objects. We have used classes from the demo server included with DevTest that are easy to reproduce in your environment.

Complex DTO Object Scenario 1

A complex DTO Object, com.itko.examples.dto.Customer, is loaded in the Complex Object Editor 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.

Each of the properties must be given values before the Customer object can be used.

- **locations**
  An array of Address objects

- **poAddr**
  An Address object

- **since**
  A Java Date object

- **types**
  An array of integers

1. Starting with the poAddr object, identify the setPoAddr method in the **Call Sheet**.

2. Select the setPoAddr method and double-click or click **Add Method** to run this method.

3. This DTO enables the use of Simple mode. Do not select the **Expert Mode** check box. The poAddress property is identified as type Address.

4. In a Simple mode, when you clear the **Null** parameter, the Address object is expanded to expose its properties. You know, from the previously illustrated Simple Data Object Scenario, that Address has simple properties. You can enter them as values or properties in the **Value** column.

5. Click **Execute** to invoke the setPoAddr method.

6. Select the setTypes method on the **Call Sheet** and click **Invoke Method**.

7. **Types** is an array of integers. To add as many integers as the array requires, click **Add** at the bottom. In the previous example, we added four elements and entered values for each.

8. Click **Execute** to invoke the setTypes method.

9. Select the setLocations method on the **Call Sheet** and click **Invoke Method**.

10. **Locations** is an array of Address objects. Click **Add** to add as many elements (of type Address) as required.

11. In the previous example, we added three Address objects. Two are complete, and we are ready to expand the third Address object to enter property values. When complete, click **Execute** to invoke the setLocations method. Notice that you can click one of the Location elements in the Object Call Tree to display and edit properties in the Data Sheet tab.

This holds true for all the properties that are listed in the **Object Call tree**.

12. Select the getSince method on the **Call Sheet** and click **Invoke Method**.

The input parameters for the Data object are displayed and can be given values.

13. Click **Execute**.
    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 was loaded in COE using a Dynamic Java Execution step.

Again we can use Simple Mode for the OrderDTO object.

Follow these steps:

1. Select the setCustomer method in the **Call Sheet** and click **Invoke Method**. As expected, the input parameter is a Customer object.

2. Clear the check box in the **Null** column.
   The **Customer** row expands to expose its properties.

All the properties can be edited in this window. The Integer and String properties can be added in the **Value** column. The remaining properties expand to show their properties when you clear the **Null** box for the property. If the property is a single object, it expands to expose its properties. If it is an array or collection, you can add the appropriate number of elements. This graphic shows a snapshot of the editing process.

The locations property has two elements; the types property has three elements. The poAddr object is of type Address, and is expanded exposing simple string properties.

Building Test Steps

A *test step* is an element in the test case workflow that represents a single test action that is performed. Test steps are in two major categories:

- Most test steps act on the system under test, and evaluate the response. Some common examples are testing an Enterprise JavaBean (EJB) method, a web service, or a message through a messaging service provider.
- A second category of test steps performs utility functions, such as data conversion, data manipulation (such as encoding), logging, writing information to files.

Note: DevTest does not support sending I/O streams from test steps, properties, or both.

Both categories of steps go into the building of a test case.

More Information:

- Add a Test Step
- Configure Test Steps

- Add Filters - Assertions - Data Sets to a Step
- Configure Next Step
- Replay to a Step
- Set a Starter Step
- Generate Warnings and Errors

Add a Test Step

To add a test step:

Do one of the following actions:

- Click **Add Step** on the toolbar
- Select **Commands**, **Create a New Step** from the main menu

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 appropriate test step.

More Information:

- Add a Test Step - Example

Add a Test Step - Example

This example adds a step to the multi-tier-combo test case in the examples directory. A new Dynamic Java Execution step will be added to the multi-tier-combo test case, after the Get User step.

Follow these steps:

1. Click **Add Step**.
   A panel that lists the common test steps opens.
   - If some steps are already created in the test case, as in this test case, the panel shows **Steps in Model** on top. The **Steps in Model** list shows all the steps present in the test case.
   - For a new test case, this list 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** menu.

2. Select the main category of the step to be configured (for example, Web/Web Services, Java/J2EE, and Utilities).
   The sub category opens.

3. To add a step to the test case, click the step.
   The step editor for the selected test case opens. Each step type has a different step editor.

4. To add a Dynamic Java Execution step after the Get User step, right-click the Get User step and select the Dynamic Java Execution step.
   The step is added and the step editor for the Dynamic Java Execution step opens.

Step editor for the Dynamic Java Execution step

Add Step Information

Follow these steps:

1. 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.

Step Information editor

2. Enter the following parameters:

   - **Name**
     The name of the step. You can rename the step in this text box.

   - **Think Time**
     The amount of time the test case waits before executing this step. Think time lets DevTest simulate the amount of time it takes a user to decide what to do before taking action. To specify how the time is calculated:

     a. Select the time unit by clicking the appropriate drop-down (millis, seconds, minutes)

     b. Enter a starting value in the **Think Time** field

     c. In the **To** field, enter an end value and a time indicator.
     DevTest picks a random think time in the specified range. For example, to simulate a user think time for a random interval between 500 milliseconds and 1 second, enter "500 millis" and "1 seconds."

   - **Use Global Filters**
     To instruct the step to use global filters, select this box. For more information about filters, see Add a Filter.

   - **Quiet**
     Select this box if you want DevTest to ignore this step for response time events, and performance calculations.

   - **Execute On**
     Specifies the simulator that the step runs on. Specify specific simulators for steps that must run on a specific computer. For example, when reading a log file, run the step on the computer where the log resides.

   - **Next**
     The next step to execute in the test. If the step ends the test case execution, you will not specify a next step. An assertion that fires in this step overrides this value.

Configure Test Steps

To configure a test step:

1. Add the step in the model editor.
   After the step is added, a different test step panel opens in the **Element** tree.

2. Configure the test step by setting the parameters in the configuration elements (assertions, filters, data sets, and so forth).

Details of each test case/step element can be seen by clicking the element arrow next to the configuration elements to expand it.

Step Information panel

- **Step Information**
  The **Step Information** element is the first element in the elements panel and carries the name of the step as its label. In the preceding example, the **Step Information** element is **Get User**. You can change the name of the test step after expanding it. Then set the next step to be executed after the current step is completed. The other options are explained in Add a Test Step.

- **Step Type Information**
  This field has the title of the selected step type (Enterprise JavaBean Execution in our example). Each step is different and has a specific configuration requirement. Therefore, each step has a custom editor to provide the information that is required to run the step and test it (and possibly to get a response also). This custom editor opens up when the element is expanded.

- **Log Message**
  This message appears after any step is added in DevTest and lets you set the log message that appears after the step runs.

- **Assertions**
  The addition and configuration of one or more assertions. In an assertion configuration, you also must set the next step to be executed when the assertion fires.

- **Filters**
  The addition and configuration of filters. Filters are added under the filter element of each test step.

- **Data Sets**
  The addition and configuration of one or more data sets that apply to the test step. The data sets are fired before executing a test step. 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 elements (assertions, filters, and data sets) have a toolbar at the bottom of the **Element** tab.

You can add/delete an element to a step by clicking the icons at the bottom of the individual elements.

- For detailed information about adding filters, see Add a Filter.
- For detailed information about adding assertions, see Add an Assertion.

Add Filters - Assertions - Data Sets to a Step

Add filters, assertions, and data sets under the corresponding element of each step in the right panel.

Add Filters Assertions and Data Sets panel

To add a filter, assertion, or data set, click **Add** on the toolbar.

For more information, see:

- Filters
- Assertions
- Data Sets

Configure Next Step

In a test case workflow, you can assign the "next step" to a selected test step. You can also assign an end step, fail step, or abort step.

- Assign the Next Step
- End Step
- Fail Step
- Abort Step

Assign the Next Step

After executing the selected step in the workflow, it will then go to the defined next step for execution.

You can configure the "next step" to either go to the other steps in the workflow or direct to the following actions:

- End the test
- Fail the test
- Abort the test.

Follow these steps:

1. Click the step for which you want to decide the next step.
2. Right-click and select **For next step** and click the targeted next step.

The workflow in the model editor changes. The information in the **Next** field in the step editor also changes.

You can also end the test, fail the test, or abort the test.

End Step

The End step brings 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. If the execution reaches this step, the entire test case is deemed to have failed. The Fail step is the default for many internal DevTest exceptions (for example, an EB exception). However, assertions can set the Fail step as the next step 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. If this step is reached, the test case is deemed to be aborted (without completion).

Replay to a Step

You can do a quick replay of a test case to any step in the case.

To replay to a step:

1. Click the step and right-click to select **Replay to here**.
2. The case is replayed to the selected step.

Set a Starter Step

You can set any test step to be a starter step in the workflow.

The starter test step then starts the test case workflow.

To set a starter step:

1. Click the step and right-click to select **Set as starter**.

2. The selected step is set as the first step in the workflow. This option is not available to the first step in the test case.

Generate Warnings and Errors

You can configure two other types of tests as next steps.

- Generate Warning
- Generate Error

For an example, we have selected the assertion in the Add User step.

Follow these steps:

1. To open a menu, select a step and right-click on the assertion.

2. Select **If triggered, then**.

3. Select **Generate Warning**.
   The test case will go to this next step (Generate Warning), only when the assertion is triggered.

Generate Warnings on an assertion menu

Generate Warning Step

When the test step fails, DevTest uses an "Ignore" type of step logic that does not raise an alarm or event (the Generate Warning step). The Generate Warning step does not change the test case workflow.

Generate Error Step

Test steps can either pass or fail. When they fail they do not actually fail the test.

To fail the test, the test step sets the test case workflow to execute the "fail" step. This action implicitly makes the step to be considered as failing.

If they explicitly fail, they generate an error and they raise a NODEFAILED event and then they continue the test step.

Creating Test Cases

A *test case* specifies completely how to test a business component in the "system under test." In some cases, a test case specifies the entire "system under test".

A test case is persisted as an XML document, and contains all the information that is necessary to test the component or system. The test cases are created and maintained in DevTest Workstation.

The first step in creating a test case is to create a project. In this project, you can create single or multiple test cases.

More Information:

- Create a Test Case
- Open a Test Case
- Save a Test Case
- Test Cases in Model Editor
- Add Test Steps
- Configure the Next Step
- Branching and Looping in a Test Case
- Import Test Cases
- Response (.rsp) Documents
- Accessing Files in Another Project
- Test Case Toolbar

Create a Test Case

You can create a test case by opening a project or creating a project.

For example, the default project "examples" displays a tree structure with folders for **Configs**, **Data**, **Staging Doc**, **Suites**, **Tests**, and others.

To create a test case:

1. Right-click the **Tests** folder in the **Project** panel and click **Create New Test Case**.

2. In the dialog, browse to the directory where you plan to store the test case, then enter the name of the new test case.
   See Test Cases in Model Editor for more information.

Open a Test Case

To open or view an existing test case:

1. Select **File**, **Open**, **Test Case** from the main menu.
   The recently opened test cases are displayed. The test cases are listed in order of use. The most recently opened is at the top of the list.

2. If the target test case is in the list, select it.
   If the target test case is not in the list, browse to it by clicking **File System**, **Classpath**, or **URL**.

3. Select a file, then click **Open**.
   The selected test case is opened and displayed in the model editor. You can also open an existing test case by double-clicking it from the project panel.
   You are now ready to add new elements (filters, assertions), or change the existing elements in the test case.

Save a Test Case

When you save a test case, DevTest also saves the results of each step in the test case to a response document in the same directory, with an ".rsp" suffix. For more information, see Response Documents.

If a required field in the test step element, filter, assertion, or data set is blank, you cannot save the test case.

To save a test case:

Do one:

- Click **Save** on the toolbar.

- Select **File**, **Save (Sample)** from the main menu. The original test case name is displayed.

Test Cases in Model Editor

When you create or open a test case, the model editor opens.

> Editing the .xml test files outside of the model editor may lead to an unreadable or unusable test case. Issues with hand-edited test case files are not supported by CA Support.

The model editor provides a graphical view of the test case, with all the test steps and elements that are attached to it. For sample examples, you can open the test cases in the **LISA_HOME\examples** directory.

Model Editor page

The model editor has three sections:

- **Project** panel**:** To create a project to create, view, or edit test cases or other documents.
- **Model editor:** To add, edit, delete the test steps that are created for a test case.
- **Element** panel**:** To apply filters, assertions, data sets, or companions to a test case or test step.

Add Test Steps

Steps can be added in several ways.

To add a test step:

Do one of the following steps:

- Select **Commands**, **Create a New Step** from the main menu.
- Click **Add Step** from the **Test Case** toolbar.
- Right-click a step in the workflow and click **Add Step After**. A step is added right after/below that step.

When a step is inserted into an existing workflow, the steps on either side of the inserted step are updated automatically, keeping the workflow linear.

If the workflow contains branches or loops, the next step is not set automatically.

Configure the Next Step

To configure the Next step in the model editor:

1. Select the target test step and right-click to open a menu.
2. Click **For next step** and select the target next step.

Reorder Test Steps in a Test Case

To reorder the steps in the model editor:

1. Select a step and right-click to reorder in a workflow.
2. Select the step to be set as the next step.

You can also drag-and-drop in the steps in the model editor. If the workflow is linear, all the steps are updated automatically. A nonlinear workflow contains branches and loops. When the workflow is nonlinear, the next steps will not be updated. In nonlinear workflows, the next step can be updated in the **Step Information** tab of that step.

Branching and Looping in a Test Case

Branching in a Test Case

Branching in a test case is done using assertions.

Any number of assertions can be applied to a test step. Every assertion has a condition that 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 an Assertion.

Looping in a Test Case

Loops are created when a test step downstream from a specified test step sends control to the test step that started the flow, as in a loop. What is important is the ability to come out of the loops. To break conditionally out of a loop (like a "while" loop in programming), set assertions on a test step that participates in the loop.

Use a data set to achieve a loop that executes a given number of times or until specific data is exhausted (like a "for" or "foreach" loop in programming). A data set is used to assign values to one or more properties a finite number of times. The next step that is executed after the data set is exhausted is specified with the data set definition. You can use this step to break the loop.

For example: if a data set contains 20 rows of users to log in to a system, you can create a loop 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, using the data from a data set. The loop 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 Data Sets.

Import Test Cases

You can import old test cases into the current working project directory.

To import test cases:

1. Right-click the **Tests** folder in the **Project** panel and select **Import Files**.
2. Select the files to be imported into that folder.
3. Click **OK**.
   The process of importing starts and copies all the files (including the files in subdirectories) to the selected folder. All the configurations from the test cases and virtual service models are merged into the project configs.

To import older versions of files (versions older than LISA 5.0):

1. Select **File**, **New**, **Project** from the main menu.
   The **Create New Project** dialog appears.
2. Select a directory that has test cases from the older version.
3. Select the **Base the new project on one of the following** check box.
4. Select the **Create project from existing documents directory** option.
5. Click **Create**.
   This action creates the project with all old version files converted to the new version. If a project with the same name exists, it asks for another name for the new project.

The project creation messages include the auto-convert message for all the test cases and virtual service models that were transformed to meet the requirements for the new version.

After the completion, you can see the following messages:

- Migration of the project
- Configuration change details
- Test case change details

All the imported files are selected.

Response (.rsp) Documents

When you record from a website or interact with a server, the responses are saved to a response document so the information is available later.

The response documents are created and maintained automatically, with no effort on your part, and saved as files with the test case file name and an .rsp extension. Like test case files, the response documents are XML files.

A response document maintains the HTTP response for the following items:

- Each HTTP-based step in a test case
- The response information from web service calls

- JDBC results from a database query

For example, if you ran the HTTP-based steps using the Interactive Test Run (ITR) utility, 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 about using the HTTP responses to create filters, see Filters.

For more information about using the HTTP response to create assertions, see 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, copy the associated response document also so that you do not lose the saved responses.

Response documents are optional, in that they are not necessary to run tests. They exist to give you the ability to view results, view DOM Tree, view the JDBC table, and more.

When you use the replay function, information is read from the response document.

Accessing Files in Another Project

To access files in another project at the same directory level as the current project, use the **LISA_PROJ_ROOT** property.

For example, assume that you have the following projects:

- C:\Lisa\Projects\Project1
- C:\Lisa\Projects\Project2

When you work in Project2, you can access a file in Project1 by specifying the following type of path:

```
{{LISA_PROJ_ROOT}}/../Project1/Data/AccountNumbers.xls
```

Be sure to provide a path that is valid across all your environments.

Test Case Toolbar

The test case tool bar contains most actions you can perform on a test case.

Test case toolbar

Icon	Description
Create a New Step Icon	Create a new step.
Delete a Test Step Icon	Delete a test step.

Set the Starting Step Icon	Set the currently selected step as the starting step in the workflow.
Cut icon	Cut the selected text.
Copy Icon	Copy the selected text.
Clipboard icon	Paste the selected text.
Image of VSE Recorder icon (yellow lightning bolt)	Click to open a menu and create steps by recording a test case in the DevTest browser. You can record a test case with Web Recorder (HTTP proxy).
Selenium Test Step Icon	Click to open a menu and choose to: ▪ Import a JSON script to create a Selenium test step ▪ Export a Selenium test step as a JSON script
Blue Gear Icon (Start the ITR)	Start a new Interactive Test Run.
Stage Quick Test icon	Stage a quick test.
Replay Test to Point Icon	Replay a test to a specific point.
Model Properties icon	Show model properties.
Zoom icon	Reset zoom scale to 1:1.
	Zoom in.

Zoom in icon	
Zoom out icon	Zoom out.
Icon - page with XML symbols	View XML source.

Building Subprocesses

A *subprocess* is a test case that is called from another test case instead of run as a stand-alone test case.

Subprocesses can be used as modules in other test cases, which increases their ability to be reused. You can build a library of subprocesses that can be shared across many test cases.

In a programming language, a subprocess would be referred to as a function or a subroutine.

A test case must be self-contained. The value for all the properties that are used in the test case must come from in the test case. A subprocess expects the test step that runs it to provide some property values (input properties). When the subprocess completes, it makes property values available to the calling step (return properties).

Subprocesses can be nested. A subprocess can call another subprocess.

> **Note:** If a subprocess that calls another subprocess is marked as **Quiet**, all called subprocesses will be Quiet.

You create the steps in the subprocess in the same way you do for a regular test case, with the following differences:

- Mark the test case as a subprocess in the **Test Case Information** tab of the test case (as explained in Create a Subprocess Test Case).
- Do not add data sets as you would in a test case. Instead, the data set must be part of the calling case, and the current values are passed to the subprocess when it is invoked. The exception is when a data set is a part of the subprocess logic itself. In that scenario, do not use a global data set.
- Do not use a configuration file or a companion in the subprocess to initialize any parameters that you expect to be passed from the calling step. For testing purposes, we add these values elsewhere. When the subprocess is called, the calling step passes these values.

Note: The think time parameter in the parent test case (the test case that calls the subprocess) is propagated to the subprocess. To make your subprocess think times run independently of the calling process, set a testExec property with the name **lisa.subprocess.setThinkScaleFromParent** to "false" so that you can decide for each subprocess. For a global override, set **lisa.subprocess.setThinkScaleFromParent=false** in local.properties.

You can build subprocesses from scratch, or you can convert an existing test case into a subprocess.

The Execute Subprocess test step simplifes calling a subprocess test case.

More Information:

- Create a Subprocess Test Case
- Convert an Existing Test Case into a Subprocess
- Subprocess Example

Create a Subprocess Test Case

You can create a subprocess test case and define input and output properties for it.

- Define Subprocess Input Properties
- Define Subprocess Output Parameters

Follow these steps:

1. Create a test case or open an existing one.
2. 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 (without selecting a step). The **Test Case Information** tab opens in the right panel.
3. To make this test case a subprocess, select the **This is a subprocess** box.
   By default, a test case is not designated to be a subprocess.
4. Two new tabs for **Subprocess Input** and **Subprocess Output Parameters** are added.
5. In the **Documentation** tab, provide some detailed documentation of the subprocess.
   This text is visible in any test step that calls the subprocess.
6. When you have finished adding the subprocess steps, configure the input and output properties for the subprocess.

Define Subprocess Input Properties

Follow these steps:

1. Click the **Subprocess Input Parameters** tab.

2. Define the input parameters and the prospective input parameters.
   A list of the parameters that the subprocess requires appears in the **Subprocess Input Parameters** field. Some parameters are added automatically. You can add other parameters if necessary.

3. To add a property, click **Add** at the bottom of this panel.

4. Enter the values for Key (property name), Description, and Default Value.
   The default value is used when you run the subprocess in the Interactive Test Run facility (ITR). This value lets you test the subprocess as though it was a regular test case. These default values are ignored when another test step invokes the subprocess.

5. To remove unwanted properties, click **Delete** .

   - **Prospective Input Parameters (may be required parameters)**
     If DevTest finds a possible input property in the subprocess, the property is listed in this field. If it is a valid input property, use the **Add** button to promote this property to the **Subprocess Input Parameters** list.

Define Subprocess Output Parameters

Subprocess Output (Result) Properties: A list of all the properties that the subprocess sets. A list of the parameters that the subprocess requires appears in the **Subprocess Output Parameters** field. Some parameters are added automatically. You can add other parameters as necessary.

To add a property, click **Add** at the bottom of this panel.

To remove properties that you do not want to be made available to the calling step, click **Delete** .

After the input and return properties are checked, and all the input parameters have default values, you can run the subprocess in the ITR.

Convert an Existing Test Case into a Subprocess

To convert an existing test case into a subprocess:

1. Open the test case and rename it appropriately.

2. Mark the test case as a subprocess in the **Test Case Information** tab of the test case.

3. Remove any data sets that provide the values of properties that are to be input properties of the new subprocess.
   The exception is when a data set is integral to the subprocess logic.

4. Remove any properties from configuration files, or a companion that initializes parameters that are to be input properties of the new subprocess.

5. After the input and output properties are checked and all the input parameters have default values, you can run the subprocess in the ITR.

Subprocess Example

The following example shows a subprocess that is derived from the jms.tst test case in the DevTest examples directory, and a test case that calls it.

A subprocess that was derived from the jms.tst test case in the examples directory, and a test case that calls this subprocess

One data set, order_data, was removed from the original test case. The order_data data set provided the values for **order_num** in the original test case. The property **order_num** is now an input property.

Subprocess Example 2

The following graphic shows a test case with one test step: Execute Subprocess.

Subprocess example 3

Step1 is of the type Execute Sub Process, and it runs the subprocess **jms** (shown previously).

The input property matches, and the calling step has asked for the **lisa.jms-1.rsp** property to be made available after the subprocess has finished executing.

Building Documents

A *staging document* contains the information about how to run a test case.

An *audit document* lets you set success criteria for a test case in a suite.

You can apply metrics and can add events, which are used for monitoring the test case after the test run.

A *suite document* can be used to run a collection of test cases.

More Information:

- Building Staging Documents

- Building Audit Documents
- Understanding Events
- Generating Metrics
- Building Test Suites

Building Staging Documents

A staging document contains the information about how to run a test case.

You create staging documents in DevTest Workstation.

If a test case contains a global data set on the first step that is set to end the test when the data set is exhausted, all test instances for a staged run end. Other staging parameters such as steady state time are overridden.

Local data sets do not end the staged run this way, nor will data sets on steps other than the first step.

Follow these steps:

1. Select **File**, **New**, **Staging Document** from the main menu.
   The **New Staging Doc** dialog appears.
2. Enter the name of the new staging document.
3. Click **OK**.
   The staging document editor appears.
4. In the Base tab, specify basic information about the staging document.
   This information includes the staging document name, the load pattern, and the distribution pattern.
5. In the Reports tab, specify the type of report that you want to create at run time.
6. In the Metrics tab, specify the metrics that you want to record at run time.
7. In the Documentation tab, enter descriptive information about the staging document.
8. (Optional) In the IP Spoofing tab, enable, and configure IP spoofing.
9. (Optional) In the Source View tab, review the XML version of the staging document.
10. Select **File**, **Save** from the main menu.

Staging Document Editor

The Staging Document Editor is where you specify the criteria for running test cases.

The Staging Document Editor contains the following tabs:

- Base: To specify the basic parameters.
- Reports: To select and add reports.

- Metrics: To select metrics and specify your sampling intervals.
- Documentation: To enter descriptive information about the staging document.
- IP Spoofing: To enter the IP spoofing details. IP spoofing allows multiple IP addresses on a network interface to be used when making network requests.
- Source View: To view the XML source of the staging document.

Staging Document Editor - Base Tab

The **Base** tab of the Staging Document Editor describes the basic parameters of a test case:

- The global adjustments to think times
- Duration of the test (elapsed time or number of runs)
- Information to pace tests, such that a specific number of tests complete in a specified time period
- Number of virtual users
- Load patterns for the virtual users
- The distribution patterns for the virtual users

The Base tab of the Staging Document Editor

The **Base** tab is divided into the following panels:

- Upper Panel
- Load Pattern Selection Panel
- Distribution Selection Panel

Upper Panel

The upper panel lets you set the following parameters:

- **Run Name**
  The name of the staging document.

- **Think Time**
  The think time in percentage, for all the test steps in the test case. Each test step can declare a think time in the step information section. Here, you can apply a global change to these think times, as a percentage of their values. For example:

  - To eliminate think times, set the percentage to 0%

  - To cut think times in half, set the percentage to 50

  - To double think times, set the percentage to 200%

- **Enable CA CAI**
  You can select to enable or disable Application Insight.

- **Num Test Executions**
  The number of test executions to complete in a specific time.

- **Per Given Time**
  The time period (wall-clock) in which the tests run.
  Values: h for hours, **m** for minutes, **s** for seconds
  You can specify that 2500 tests complete in 8 minutes.
  The think times are not changed to achieve the required pace.
  DevTest runs without any pause between tests when the test pace cannot be achieved because it is too high. DevTest reports in the log that the test is running at a lower pace than requested.
  DevTest does not add more users if the test pace cannot be achieved because too few virtual users are specified. To estimate the required number of virtual users, see the Optimizer utility.

Load Pattern Selection Panel

The **Load Pattern Selection** panel lets you complete the following actions:

- Set the test duration

- Set the number of virtual users (instances)

- Set the load pattern for the virtual users (if you have multiple virtual users).

For more information, see Load Pattern Selection.

Distribution Selection Panel

The **Distribution Selection** panel lets you distribute virtual users (instances) over your running simulators. For more information, see Distribution Selection.

Load Pattern Selection

The **Base** tab of the Staging Document Editor includes a **Load Pattern Selection** panel.

The load pattern options are as follows:

- **Immediately Ramp**

- **Manual Load Pattern**
- **Run N Times**
- **Stair Steps**
- **Weighted Average Pattern**

> **Note:** Pacing is only supported with the following load patterns:
>
> - **Run N Times**
> - **Immediate Ramp**
> - **Stair Step**

Pacing depends on the ability of the load pattern to estimate the number of steady-state instances. The number of steady-state instances running affects the pace at which steps run. These load patterns are the only ones that can estimate that pace.

Immediately Ramp

The **Immediately Ramp** pattern applies when you are running only a few virtual users (instances) but you want to specify the test duration. You are not concerned with any loading pattern. This pattern starts all virtual users simultaneously.

To configure this pattern, enter the following parameters:

- **Instances**
  The number of virtual users.
- **Max Run Time**
  Select either **No Max** (the test case determines the time) or **Maximum Run Time**. In the latter case, specify the **Maximum Run Time**. This setting overwrites any value that is entered for **Cycles**. You can specify **h** for hours, **m** for minutes, **s** (or no letter) for seconds (default).

The graph at the bottom provides a graphical view of the pattern.

Manual Load Pattern

The **Manual Load** pattern gives you the most control over the loading and unloading of virtual users (instances). This pattern is similar to the **Stair Steps** pattern, but it lets you 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. In each row, you define the time interval and the number of virtual users to add or remove (the **Instances Change** column) for that step. The elapsed time (the **Total Time** column) and the total number of virtual users (**Running Instances** column) are automatically calculated.

In the **Time Interval** column, enter a time followed by **h** for hours, **m** for minutes, **s** (or no letter) for seconds (default).

To add, remove, or change the current order of the steps, use the standard icons from the toolbar at the bottom of the table.

You could have to scroll to see these icons. Alternatively, you can select a row and can use the following shortcuts:

- 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 graph at the bottom provides a graphical view of the pattern.

Run N Times

The **Run N Times** pattern applies when you run only one or only a few virtual users (instances), but you want to specify how many times the test runs. You are not concerned with any loading pattern. This pattern starts all virtual users simultaneously.

To configure this pattern, enter the following parameters:

- **Instances**
  The number of virtual users.
- **Cycles**
  Select between running continuously for the interval set in **Max Run Time**, or running a maximum number of times.
- **Max Run Time**
  Select either **No Max** (the test case determines the time) or **Maximum Run Time**. In the latter case, specify the **Maximum Run Time**. This setting overwrites any value that is entered for **Cycles**. You can specify **h** for hours, **m** for minutes, **s** (or no letter) for seconds (default).

Stair Steps

The **Stair Steps** pattern introduces virtual users (instances) to the system in well-defined steps, instead of 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 the maximum number of virtual users. The number of virtual users to introduce at each step is: **Steady State Instances** that are divided by **Number of Steps**.
- **Ramp Up Time**
  The time period over which virtual users are added to reach the maximum number of virtual users. The time interval between steps is: **Ramp Up Time** that is 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. Because 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 graph at the bottom provides a graphical view of the pattern.

Weighted Average Pattern

The Weighted Average pattern adds and removes virtual users (instances) based on a statistical calculation. DevTest calculates the number of steps, and the time period between steps, as frequently as every second, by honoring a moving weighted average. This distribution will approximate a bell curve distribution. Most virtual users are added within two 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 reach the maximum number of virtual users.

- **Steady State Time**
  The time period of the meaningful test run.

- **Ramp Down Time**
  The time period over which virtual users are removed. Because 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 graph at the bottom provides a graphical view of the pattern.

Distribution Selection

The **Base** tab of the Staging Document Editor includes a **Distribution Selection** panel.

If you use DevTest Workstation, the **Distribution Selection** panel is unnecessary because your virtual users run locally (using a simulator that is part of DevTest Workstation).

If you use DevTest Server, with several simulator servers active, the **Distribution Selection** panel lets you specify how to distribute your virtual users across the simulators.

The distribution options are:

- **Balanced Based on Instance Capacity**

- **Dynamic Simulator Scaling with DCM**

- **Percent Distribution**

- **Round Robin Distribution**

Balanced Based on Instance Capacity

The **Balanced Based on Instance Capacity** distribution requests that DevTest base 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. The default is 255. DevTest dynamically tracks the percent load and adds virtual users to specific simulators to try to keep the load percentage as even as possible. Therefore, the simulator with the lowest percentage load is a candidate for the next virtual user that is 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.

No parameters are required.

Dynamic Simulator Scaling with DCM

The **Dynamic Simulator Scaling with DCM** distribution requests that DevTest automatically determine when to raise capacity to meet the needs of a running test and then automatically expand the lab.

This distribution pattern includes the following parameters:

- **Checkpoint time**
  The interval at which DevTest evaluates whether more simulators are required. Enter the value in seconds (for example, 300s) or minutes (for example, 5m).

- **DCM Dynamic Lab Name**
  If you stage to an existing coordinator and the test determines that it requires more capacity to run the test, this parameter indicates which lab to activate to run more simulators. Include the VLM prefix and the fully qualified lab name.

- **Maximum Expansion**
  The maximum number of simulators to create at the time of expansion. If you have a policy that limits the number of simulators for each user, use this parameter to enforce it. The default value of 0 means unlimited.

During the checkpoint evaluation, DevTest examines the performance of the simulators that are currently running and how many more virtual users must be brought online. If more simulators are required, DevTest starts the process of expanding the lab. When the simulators come online, DevTest starts directing traffic to them.

Percent Distribution

The **Percent Distribution** distributes virtual users (instances) over the simulators, based on the percentages that you specify.

The names of the running simulators (plus local) are available in a drop-down list in the **Simulator Name** column.

Select a simulator and specify a percentage of virtual users for the simulator. Repeat this action for all the simulators to include until you have 100 percent. Use integer values for the percentages.

> **Note:** Although "Auto" appears as a choice in the drop-down list, it is not recommended. "Auto" results in confusing allocations of virtual users because it competes with your explicit distribution choices.

Round Robin Distribution

The **Round Robin Distribution** requests that DevTest control the allocation of virtual users (instances), based on a simple round-robin distribution pattern.

DevTest selects a simulator randomly and adds a virtual user, then goes to another simulator and adds a user, and continues in this fashion as necessary. After all simulators are used, the process continues with the first simulator. This distribution is useful when staging a single, large load test on your system.

No parameters are required.

Staging Document Editor - Reports Tab

The **Reports** tab of the Staging Document Editor lets you specify the report generator that you run for every test case or test suite.

Reports tab of the Staging Document Editor

The following report generators are available:

- **Default Report Generator**
  Captures functional, performance, and metric information and publishes that data to the reporting database referenced by the registry. The **Reporting Portal** uses that database.

- **Load Test Report Generator**
  Designed for load tests with thousands of virtual users. This report captures load metrics but not step-level metrics. If it captured step-level metrics, there would be too much data and the reporting database would slow down the test.

- **XML Report Generator**
  Creates an XML file with all the possible data that can be captured. The captured data can be limited using the report options. To view this report, import the file into the **Reporting Portal**.

Details of the data 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 more fully in Generating Metrics.

Reports can be selected in the following modules and can be seen in the **Report Viewer**:

- Stage a Quick Test
- Building Staging Documents
- Run a Test Suite

After they are requested, they can be viewed and managed later.

When you select a report from the drop-down list, a summary of the report appears in the area below.

According to the selected report, the parameters change. Set the parameters as and where necessary.

The **Reports** tab is divided into the following areas:

- **Right panel**
  The right panel is where you select the report to be added.

- **Attributes**
  The **Attributes** area contains the **Report Generator Type** and the required parameters for the report.

  - **Report Generator Type:** A pull-down menu lists the available report types.

  - **Parameters:** The parameters that are required to set the selected report generator.
    The **Maximum number of errors before test is stopped** field limits the number of errors that the test can have before LISA aborts it. A zero entry means to allow all errors.

- **Left panel**
  The left panel shows the list of added reports. You can add, save, move, and delete reports by using the toolbar at the bottom of the panel.

Staging Document Editor - Metrics Tab

The **Metrics** tab of the Staging Document Editor lets you select the test metrics to record.

Contents

- Add a Metric

- Set an Email Alert
- Sampling Parameters

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.

Metrics tab of the Staging Document Editor

The **Metrics** tab is divided into two sections. The top panel lists the default metrics, differentiated by color code. The bottom panel has the sampling parameters.

You can change the color that is used to represent any metric in the staging document. Click the color and select a new color and then save the staging doc. Change the color before staging the test.

You can add or delete metrics and set email alerts in the top panel.

The following types of metrics are available:

- DevTest Whole-Test Metrics
- DevTest Test Event Metrics
- SNMP Metrics
- JMX Attribute Reader (JMX metrics)
- TIBCO Hawk
- Windows Perfmon Metrics
- UNIX Metrics through SSH

For more information about each metric, see Metrics Descriptions.

Add a Metric

Follow these steps:

1. Click **Add** on the toolbar.
   A dialog to add metrics opens with a list of metrics that can be added.

2. Select the target metric type and click **OK**.
   The **Metric Selection** dialog opens.

3. Select the target subcategory for the metric and click **OK**.
   Subcategories are different for each metric. The newly added metric appears in the list of existing metrics in a different color.

For the descriptions of all the metrics in all categories, and details on how to configure them to include in reports, see Generating Metrics.

Set an Email Alert

To set an email alert on a specific metric:

1. Click the **Set Alert** button corresponding to the metric.
   The **Edit Alert** dialog opens.

2. You can set the acceptable limits for the metric (low and high value), and the details to be sent in an email. Provide the name of your email server and a list of email addresses.
   The email alerts that are added to staging documents store the SMTP password as an encrypted value. Also, if you open an existing staging document and then you resave it, the SMTP password is saved as an encrypted value.
   To add and remove email addresses, use the **Add** and **Delete** buttons at the bottom of the window.

3. When you are finished, click **Close**.
   Notice that the **Set Alert** button is now an **Edit Alert** button on the **Metrics** tab.

> **Note:** For the email alert to work, you also must set the SMTP server-related paths in the lisa.properties file.

Sampling Parameters

The bottom panel has two slider bars that let you set sampling parameters:

- **Sample rate:** This parameter specifies how often to take a sample, or record the value of a metric. The sample rate is specified as a time period, and is the reciprocal of the sampling rate.

- **Samples per interval:** This parameter 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) produces 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).

Staging Document sampling parameters slider

The preceding example uses 5 seconds per sample and 25 samples per interval. Those values produce a metric value every 5 seconds and a summary value every 125 seconds.

The metric values are stored in XML files or database tables to include in reports (see reports).

Staging Document Editor - Documentation Tab

The **Documentation** tab of the Staging Document Editor lets you enter descriptive information about the staging document.

Staging Document Editor - IP Spoofing Tab

The **IP Spoofing** tab of the Staging Document Editor lets multiple IP addresses on a network interface to be used when making network requests.

- IP Spoofing Support
- IP Spoofing Scenarios
- Configuring IP Addresses in Windows

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 some systems such as web applications, this outcome often more closely resembles "real-world" behavior.

IP Spoofing tab of the Staging Document Editor

- **Enable IP Spoofing**
  If selected, IP addresses are spoofed for all supported test steps in a test case.
- **Version**
  - **IPv4:** If selected, IPv4 addresses are spoofed.
  - **IPv6:** If selected, IPv6 addresses are spoofed.

  Either IPv4 or IPv6 must be selected.
- **Selection Algorithm**
  - **Round Robin:** Spoofed IP addresses are selected in a round-robin format.
  - **Random:** Spoofed IP addresses are selected in a random format.

IP Spoofing Support

IP spoofing is supported only for HTTP.

You can configure the following steps to use IP spoofing:

- HTTP/HTML Request
- REST Step

- Web Service Execution (XML)
- Raw SOAP Request

IP Spoofing Scenarios

You can expect these results when IP spoofing is configured in a staging document:

- **A single test case with a single user with a loop:**
  Based on sequential or random order of the IP address list each loop in one cycle of the test gets a different IP address.
- **A single test case with a single user with no loop, so that test starts and ends for every cycle:**
  Use random addressing so that the IP address is different. If you do not select **random**, each user always starts with the first IP address.
- **A single test case with multiple users with a loop behaves as number 1:**
  The test exec or cycle is what maintains the list of IP addresses to use.
- **A single test case with multiple users with no loop must have random selected:**
  Selecting **random** ensures that a random IP address is used when the cycle starts instead of the first in the list.

Configuring IP Addresses in Windows

This section describes how to add IP addresses to a network interface for IP spoofing.

> **Note:** Before you add IP addresses, ensure that you are an administrator.

On Windows, IP address information can be obtained by using the command-line utility **ipconfig**.

To add a single IP address, 192.168.0.201, use the **netsh** command-line utility.

```
netsh in ip add address "Local Area Connection" 192.168.0.201 255.255.255.0
```

To add many IP addresses, use **netsh** in a loop.

For example, the following command adds 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, you can verify the new IP addresses by using the command-line utility **ipconfig /all**.

Staging Document Editor - Source View Tab

The **Source View** tab of the Staging Document Editor shows the XML source of the staging document.

Staging Document Examples

Sample staging documents are provided in the **StagingDocs** folder of the examples project. All of them use the Run N Times load pattern and the Percent Distribution pattern.

You can use these documents as the starting point of your new staging document. Then, rename it to save it under a different name.

- **1User0Think_RunContinuously.stg:** This staging document runs a single virtual user with zero think time. This staging document runs the test continuously, which does not necessarily mean forever.
- **1user1cycle0think.stg:** This staging document runs a single virtual user one time with zero think time.
- **ip-spoofing.stg:** This staging document lets you test IP spoofing support.
- **jboss-jmx-metrics-localhost.stg:** This staging document runs three concurrent virtual users one time for 440 seconds.
- **Run1User1Cycle.stg:** This staging document runs a single virtual user one time with 100 percent think time.
- **Run1User1CyclePF.stg:** This staging document runs a single virtual user one time with 100 percent think time. Application Insight is enabled.
- **Run1User1CycleShowAll.stg:** This staging document runs a single virtual user one time with 100 percent think time. Application Insight is enabled.

Building Audit Documents

An audit document lets you set success criteria for a test case in a suite.

- Event Audits Panel
- Run for Audit Info Panel

An audit document can track:

- Events that must occur or must not occur during a test
- Whether a test takes too little or too much time to complete

You can specify audit documents in the Base tab of the suite document editor. Be sure to select the **Record All Events** check box in the Reports tab.

Audit documents are located in the **AuditDocs** folder of a project. The file extension is **.aud**.

When you create a project, the **AuditDocs** folder contains a default audit document with the name **DefaultAudit.aud**.

> **Note:** When you use audit documents, test step names cannot contain any short description from the audit document that is used in the test. For example, if **Abort** is a short description in the audit document, then you cannot use the word **Abort** in a test step name.

The following graphic shows the audit document editor. The editor contains an **Event Audits** panel and a **Run for Audit Info** panel.

Audit Document Editor

To create an audit document:

1. Select **File**, **New**, **Test Audit** from the main menu.
2. Enter the name of the audit document, and click **OK**.
   The audit document editor opens.
3. To audit events, configure the parameters in the Event Audits panel.
4. To audit the execution time, configure the parameters in the Run for Audit Info panel.
5. Select **File**, **Save** from the main menu.

Event Audits Panel

The **Event Audits** panel lets you specify events that must occur or must not occur during a test.

To add an event:

1. Click **Add** .
2. In the new row, select the event name from the drop-down list in the **Event ID** column. Each row has the following parameters:
   - **Short Desc Contains**
     To filter an event by the value of the short description of the event, enter the keywords from the short description in this column.

- **Must See**
  Select this check box if the event must occur during the test.

- **Must NOT See**
  Select this check box if the event must not occur during the test.

- **Fail Message**
  If this audit fails, a message to log.

3. Add more rows for each event that you want to include.
   If you try to add event audits that logically conflict with each other, the editor displays a warning message.

   You can rearrange rows by clicking **Up** and **Down** .

   You can delete rows by clicking **Delete** .

Run for Audit Info Panel

The **Run for Audit Info** panel lets you audit the execution time.

This panel has the following parameters:

- **Audit Run Time**
  To enable the execution-time audit, select this check box.

- **Minimum Time**
  The minimum time (in seconds) that the test must run.

- **Maximum Time**
  The maximum time (in seconds) that the test can run and still be considered a successful audit. If there is no maximum time constraint, then enter 0.

- **Failure Message**
  If this audit fails, a message to be logged.

Understanding Events

An *event* is a message that is broadcast, informing any interested parties that an action has occurred. Events are created for every major action that occurs in a test case.

An event results every time a step runs, a property is set, a response time is reported, a result is returned, a test succeeds, or fails, and more. You can see every event or filter out the events that are not of interest.

Events are important 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. The following graphic shows the **Test Events** tab.

Response | Properties | Test Events

Timestamp	EventID	Short	Long
13:34:13,596	Step history	ABEFD4BED91028...	
13:34:13,596	Step started	Pay Bill Online	Withdraw money...
13:34:13,596	Property set	lisa_last_pftxn	<removed>
13:34:14,045	Property set	lisa.Pay Bill Onlin...	200
13:34:14,045	Property set	lisa.Pay Bill Onlin...	http://localhost:8...
13:34:14,045	Property set	lisa.Pay Bill Onlin...	Server=Apache-...
13:34:14,046	Step request	Pay Bill Online	POST /lisabank/d...
13:34:14,046	Step target	Pay Bill Online	http://localhost:8...
13:34:14,046	Info message	Pay Bill Online	Requested URL:h...
13:34:14,046	Property set	LISA_COOKIE_loc...	[version: 0][nam...
13:34:14,046	Step response time	Pay Bill Online	446
13:34:14,046	Step bandwidth c...	Pay Bill Online	16558
13:34:14,046	Step response	Pay Bill Online	<!-- jspstart: roo...
13:34:14,071	Property set	lisa.Pay Bill Onlin...	Integrator of type...
13:34:14,071	Property set	lisa.Pay Bill Onlin...	<?xml version="...
13:34:14,071	Pathfinder	Pay Bill Online	
13:34:14,071	Assert evaluated	Pay Bill Online [A...	Assert of type [A...
13:34:14,071	Assert evaluated	Pay Bill Online [Si...	Assert of type [Si...
13:34:14,071	Log message	Will execute the ...	Withdraw money...

Screenshot of the Test Events tab in the ITR

You can observe and filter events in a Quick Test.

Stage a Quick Test events tab

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 about reports, see Reports. For more information about metrics, see Generating Metrics.

> **Note:** Internal to DevTest, a step can also be referred to as a node, explaining why some events have "node" in the EventID.

An EventID, a short value, and a long value characterize an event. EventIDs are keywords that indicate the type of event. The short values and long values contain information about the event. Their contents vary with the type of event.

> **More Information:**
>
> - Event Descriptions

Adding and Viewing Events

You can add and view events:

- Through the ITR
- Through a quick test or staging document

Adding and Viewing Events through the ITR

You can enable some events in the **Settings** tab of the Interactive Test Run (ITR) utility.

Select the **Show CALL_RESULT** and **Show NODERESPONSE** check boxes to view those events.

You can view test events in the ITR by clicking the **Test Events** tab.

Adding and Viewing Events through a Quick Test/Staging Document

When you start a test case through the **Start Test** or **Start Suite** option, the test is staged and relevant graphs appear in the **Perf Stats** tab.

To view the test events, click the **Events** tab.

You can select the **Events to Filter Out** in the left panel or select from the predefined filter sets.

To refresh the test events list, select the **Auto Refresh** check box in the **Events** tab. As the test runs, you see the events that you designated on the **Events** tab.

View Events in Test Suites

You can also observe the events when you stage 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 about reports, see Reports. For more information about metrics, see Generating Metrics.

An event has 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 vary with the type of event.

Note: Internal to DevTest, a step can also be referred to as a node, explaining why some events have "node" in the EventID.

More Information:

- Event Descriptions

Generating Metrics

Metric collection, our own metric calculation method, is an extensible reporting mechanism, and lets you generate reports to various outputs.

Metrics lets you 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 software, a hardware system, or their specifications. Quantitative methods using metrics have proved to be powerful in several areas of computing, and testing is no exception.

Two broad groups of metrics are:

- **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. If a metric is a counter, then it is essentially an ever-increasing number during the test run (that is, number of errors). When a metric is flagged as a counter, it is graphed differently in reports. The value that is graphed is the difference between the current sample and the previous sample.

For most metrics, the type of metric (gauge or counter) is already known. When a metric could be used as either, you can specify the type that you want.

Metrics can be added to the following staging documents and test suite documents. Test monitors and quick tests can generate metrics. Information about specifying and generating metrics for each document or test is available in sections about each editor.

- Quick Tests: Use to monitor the test.
- Staging Documents: Use to include in reports.
- Test Suite Documents: Use to include in reports.
- Test Monitors: Use to monitor tests.

Building Test Suites

You can run or stage a combination of test cases together in DevTest. The collection of test cases that are run together are in a form of a test suite.

You create test suites from DevTest Workstation.

Follow these steps:

1. Select **File**, **New**, **Suite** from the main menu.
   The suite document editor opens.

2. In the Base tab, specify basic information about the suite.
   This information includes the suite name and the suite entries.

3. In the Reports tab, specify the type of report that you want to create at run time.

4. In the Metrics tab, specify the metrics that you want to record at run time.

5. In the Documentation tab, enter descriptive information about the suite.

6. Select **File**, **Save** from the main menu.

Test Suite Editor

When you create a test suite, and when you open an existing test suite, the Test Suite Editor opens.

For more information about the specific tabs on this page, see the following:

- Test Suite Editor - Base Tab
- Test Suite Editor - Reports Tab
- Test Suite Editor - Metrics Tab
- Test Suite Editor - Documentation Tab

Test Suite Editor - Base Tab

The **Base** tab of the Test Suite Editor contains basic information about a test suite.

Base tab of the Test Suite Editor

Top Panel

The top panel of the **Base** tab has basic information about the test suite as a whole.

To configure a suite, you also must enter parameters in all the sub tabs.

Basics Tab

This tab has the following parameters:

- **Suite Name**
  The name of the suite.

- **Startup Test/Teardown Test**
  You can specify a startup test that runs before the suite has begun, and a teardown test that runs after the suite has completed. The startup test and the teardown test are not included in any test statistics. Events in these tests appear in the reports. If the startup test fails, test suite testing does not continue. You can use a MAR info file as a startup or teardown test. The primary asset of the MAR info file must be a test case.

- **List Tests**
  Displays a dialog window with a list of the tests currently included in your suite. Save your suite for this list to be current.

> When adding test cases to a test suite, ensure that the test steps in the test case do not contain the word "fail" in the step name, or the test will fail in the test suite.

Defaults Tab

This tab has the following parameters:

- **Base Directory**
  The name of the directory that is assumed to be the base directory for any individual test that does not contain a complete path. If a test cannot be found elsewhere, DevTest looks in this directory. If you do not specify a base directory, a default is created for you when the suite document is saved.

- **Default Staging Doc**
  If a staging document is not specified for an individual test, the staging document to use.

- **Default Audit Doc**
  If an audit document is not specified for an individual test, the audit document to use.

- **Default Coordinator Server**
  If a coordinator server is not specified for an individual test, the coordinator server to use. The pull-down menu lists the available coordinator servers. This parameter is needed only if you are running DevTest Server.

Run Mode Tab

This tab has the following parameters:

- **Serial**
  Tests will be run one after another in the order they show up in the suite document. This setting is used for functional and regression testing.

- **Parallel**
  Tests are run simultaneously. This setting is used for load and performance testing. You must have enough virtual users to be able to run all tests concurrently.

- **Allow Duplicate Test Runs**
  Lets you select whether to run the same test more than once. If the same test appears in an included suite, a directory, or a directory tree, duplicate tests can occur in a suite.

The parent run mode for the parent suite is automatically applied to child suites.

Bottom Panel

The bottom panel of the **Base** tab shows the types of documents that can be added in the suite and the associated document details.

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 left panel, after all the tests are added and saved to the suite document.

Suite entries are added individually by selecting from the following types. Depending on the selection, the **Entry Details** area changes.

- **Type**
  Select the suite entry type as one of:

  - **MAR Info:** The name of a MAR info file.

  - **Test Case:** The name of a test case.

- **Suite:** The name of a suite document. All tests are extracted from this suite and run as individual tests.

- **Model Archive:** The name of a MAR file.

- **Directory:** The name of a directory that contains tests to be included in the suite. All tests in this directory use the same staging document, audit document, and coordinator server. If new test cases are added to the directory, they are automatically included in this suite.

- **Directory Tree:** The name of a directory that contains tests to be included in the suite. DevTest looks in the named directory and recursively through all subdirectories in the tree. All tests in this directory tree use the same staging document, audit document, and coordinator server. If new test cases are added to the directory tree, they are automatically included in this suite.

For directories and directory trees, you can filter the test cases by name. To filter the results, enter a Regex expression in the **File Regex Filter** field. You can enter multiple filters for a suite by clicking Add on the toolbar and entering new filter information. You can also define filters in properties and use the properties in the **File Regex Filter** field. To display the tests that will be included in your suite, use the **List Tests** button. To see the latest results, remember to save the suite.

The **Active** check box lets you control whether the selected entry runs. For example, you can clear the **Active** check box for one of the test cases in a suite.

Adding a Document Type in a Suite

Depending on the document type that is selected, the fields for entry details change.

As an example, we add a MAR info document. The entry details for the same have changed from the ones that are shown for a directory tree.

- **MAR Info**
  The MAR info name. Enter the name or select from the pull-down menu or browse to the file or directory. When it is selected, click **Open** to open the respective file.

- **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, the default staging document is used. After it is selected, click **Open** to open the staging document. When you have selected a staging document to use in this suite, the name of the staging document appears below the **Staging Doc** field. The name is the same **Run Name** as you see on the editor tab when you are editing the document.

- **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, the default audit document is used. After it is selected, click **Open** to open the audit document.

- **Coordinator Server**
  The name of the coordinator server to use with this entry. Select from the list of available coordinator servers that are listed in the pull-down menu. This parameter is needed only if you are running a DevTest Server.

Click **Add** on the toolbar to add this entry to the list shown in the left panel. You can delete entries by clicking **Delete** . You can rearrange entries by using the **Move Up** and **Move Down** icons.

After you have entered all your test entries, save your test suite document.

Test Suite Editor - Reports Tab

The **Reports** tab of the Test Suite Editor is where you specify the test reports to produce, and the events to capture at the test suite level.

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 more fully in Metrics.

Reports Tab of the Test Suite Editor

The **Reports** tab contains the following panels:

- **Report Generators**
- **Attributes**

Report Generators

The **Report Generators** panel has a list of the reports that have been selected.

To add a report, click **Add** 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, select the report in the list and click **Delete** .

Attributes

The **Attributes** panel lists the available report types.

The following report types are available:

- Default Report Generator
- XML Report Generator

Parameters

- **Record All Events**
  If selected, records all events.
- **Record Properties Set/Referenced**
  If selected, records the set or referenced properties.
- **Record Performance Metrics**
  If selected, records the performance metrics. Use this value for load testing.
- **Record Request/Response/Screenshot**
  If selected, records the request, response, and screenshot.
- **Maximum Number of Errors and Failures Before Test is Stopped**
  Stop the test when this number of errors and failures is reached.
- **Include warnings with errors and failures**
  For the parameter above, include warnings in the count of errors and failures.

Test Suite Editor - Metrics Tab

The **Metrics** tab of the Test Suite Editor is where you select the test metrics to record. This tab is also where you set the sampling specifications for the collection of the metrics.

You can also set an email alert on any metric that you have selected.

Metrics tab of the Test Suite Editor

The **Metrics** tab is divided into two sections.

The top panel shows the default metrics that are added to the suite. You can add more metrics by clicking **Add** on the toolbar.

In the bottom panel, two slider bars enable you to set sampling parameters:

- **Sample rate:** This value specifies how often to take a sample; that is, record the value of a metric. This parameter is specified as a time period, and is the reciprocal of the sampling rate.
- **Samples per interval:** This value specifies how many samples are used to create an interval for calculating summary values for the metric.

Taking sample values every minute (**Sample Rate Per Interval**=1 minute), and averaging every 60 samples (**Samples Per Interval**=60), produces 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 metric values are stored in XML files or database tables for including in reports (see Reports).

Add Metrics

The following metrics categories are available:

- DevTest Whole Test Metrics
- DevTest Test Event Metrics
- JMX Metrics
- SNMP Metrics
- TIBCO Hawk Metrics
- Windows Perfmon Metrics
- UNIX Metrics Via SSH

To add metrics:

1. Click **Add** on the toolbar.
   The **Add Metric** dialog appears.
2. Select the target metric and click **OK**.
3. To configure metrics from the other categories, repeat the procedure.

For full descriptions of all the metrics, in all categories, and details on how to configure them for including in your reports, see Generating Metrics.

Set Email Alerts

Follow these steps:

1. Click the **Set Alert** button corresponding to the metric.
   The **Edit Alert** dialog appears.

2. 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.

3. Add and delete email addresses by using the **Add** and **Delete** icons at the bottom of the window.

4. When finished, click **Close**.

Test Suite Editor - Documentation Tab

The **Documentation** tab of the Test Suite Editor is where you enter the related documentation.

Working with Model Archives (MARs)

The main deployment artifact is a type of file named a *Model Archive (MAR).*

MAR files enable the bundling of the artifacts that are needed to run a test case, suite, virtual service, or CVS monitor. MAR files also make it easier to move artifacts from one project to another.

You can configure MARs from DevTest Workstation or by using the Make Mar command-line utility.

The file extension is **.mar**.

The MAR files contain the following items:

- A primary asset
- All secondary files that are necessary to run the primary asset
- An info file
- An audit file

The MAR files are created from files in a project. After a MAR file is created, it is independent of the project.

Contents of a MAR

MAR files contain one of the following primary assets:

- Test case
- Suite
- Virtual service
- Test case monitor
- Suite monitor

MAR files also contain any secondary files that the primary asset requires.

For example, if the primary asset is a virtual service model, the MAR file also contains a service image.

A MAR file can include the data set for a data-driven virtual service.

In addition, MAR files contain the following files:

- MAR info file
- MAR audit file

You can specify that a MAR file be optimized. When the MAR file is built, only those project files that are required will be added. However, you can also configure an optimized MAR file to include one or more non-required project files.

If a MAR file is not optimized, all the project files will be added.

> **Note:** An archive is typically held in memory. Therefore, the use of optimized archives is highly encouraged.

MAR Info File

MAR info files contain information that is necessary to create a MAR. The file extension is **.mari**.

The information that is specified in a MAR info file depends on the type of primary asset.

Primary Asset	Information Specified
Test case	Test case, configuration file, and staging document
Suite	Suite and configuration file
Virtual service	Virtual service model, configuration file, concurrent capacity, think time scale, and auto restart flag
Test case monitor	All the information specified for a test case, plus the service name, notification email, priority, and run schedule

Suite monitor	All the information specified for a suite, plus the service name, notification email, priority, and run schedule

When you use the stage/deploy related options, a MAR info file is created but not saved.

MAR Audit File

MAR audit files contain metadata about the creation of a MAR. The file extension is **.maraudit**.

The following metadata is included in a MAR audit file:

- The date and time when the MAR was created.
- The name of the computer on which the MAR was created.
- The root directory of the project the MAR was created from.
- The name of the user that created the MAR.

Explicit and Implicit MAR Creation

Two approaches to creating MARs are:

- Explicit
- Implicit

To use the web-based dashboards or command-line utilities to deploy assets, you must explicitly create a MAR. An exception is the Test Runner command-line utility, which you can use to run test cases and suites that are not packaged in a MAR.

If you are staging a test case, staging a suite, deploying a virtual service, deploying a test case as a monitor, or deploying a suite as a monitor from DevTest Workstation, then you use the implicit approach.

Explicit MAR Creation

In the explicit approach, you perform steps to create a MAR info file, which you then use to build the MAR.

Follow these steps:

1. Identify the primary asset to execute.
2. Create a MAR info file.
3. Build the MAR.

Implicit MAR Creation

In the implicit approach, both a MAR info file and a MAR are automatically created.

Follow these steps:

1. Identify the primary asset to execute.
2. Do one of the following actions from DevTest Workstation or by using Test Runner:

- Stage a test case
- Stage a suite
- Deploy a virtual service
- Deploy a test case as a monitor
- Deploy a suite as a monitor

Create MAR Info Files

You can create a MAR info file from an existing test case, suite, or virtual service.

To create a MAR info file from an existing test case:

1. In the **Project** panel of DevTest Workstation, right-click the test case and select **Create MAR Info File**.
   The **Create MAR Info File** dialog opens.
2. Enter a name for the MAR info file in the **Name** field.
3. Select the configuration file for this test case from the **Configuration** field list.
4. Select the staging document for this test case from the **Staging doc** field list.
5. Select the coordinator server for this test case (if applicable) from the **Coordinator server** field list.
6. Click **OK**.
   The .mari file is created in the same folder the test case resides in.

To create a MAR info file from an existing suite:

1. In the **Project** panel of DevTest Workstation, right-click the suite and select **Create MAR Info File**.
   The **Create MAR Info File** dialog opens.
2. Enter a name for the MAR info file in the **Name** field.
3. Select the configuration file for this suite from the **Configuration** field list.
4. Click **OK**.
   The .mari file is created in the same folder the suite resides in.

To create a MAR info file from an existing virtual service:

1. In the **Project** panel of DevTest Workstation, right-click the virtual service and select **Create MAR Info File**.
   The **Create MAR Info File** dialog opens.
2. Enter a name for the MAR info file in the **Name** field.
3. Select the configuration file for this virtual service from the **Configuration** field list.

4. If this virtual service will be part of a virtual service group, enter a group name in the **Group Tag** field. A group tag must start with an alphanumeric character and can contain alphanumerics and four other characters: period (.), dash (-), underscore (_), and dollar sign ($).

5. To indicate the load capacity, enter a number in the **Concurrent capacity** field.
   The default value is 1, but it can be increased to provide more capacity. Capacity is how many virtual users (instances) can execute with the virtual service model at one time. Capacity here indicates how many threads there are to service requests for this service model.

6. Enter the think time percentage (regarding the recorded think time) in the **Think time scale** field.
   To double the think time, use 200. To halve the think time, use 50. The default value is 100.

7. Select or clear the **Start the service on deployment** check box.
   This check box indicates whether the service starts immediately on deployment. The default is to start the service.

8. Select or clear the **If service ends, automatically restart it** check box.
   Selecting this check box keeps the service running even after an emulation session has reached its end point.

9. Click **OK**.
   The .mari file is created in the same folder the virtual service resides in.

Create Monitor MAR Info Files

You can create a Monitor MAR info file from an existing test case or suite.

For more information about monitors, see Monitor with CVS (Continuous Validation Service) .

Follow these steps:

1. In the **Project** panel of DevTest Workstation, right-click the test case or suite and select **Create Monitor MAR Info File**.
   The **Create Monitor MAR Info File** dialog opens.

2. Enter a name for the MAR info file in the **Name** field.

3. Specify the monitor information about the Monitor Info tab.

4. Specify the time schedule information about the Schedule tab.

5. (Test Case) Specify the test case information about the Test Case Info tab.

6. (Suite) Specify the suite information about the Suite Info tab.

7. Click **OK**.
   The .mari file is created in the same folder the test case or suite resides in.

Monitor Info Tab

Complete the following fields:

- **Service Name**
  The name of the service. This name appears in the **CVS Dashboard**.

- **Notify Email**
  The email address to notify.

  To send an email to multiple email addresses, enter an email address for a distribution list in this field.

- **Priority**
  Set the priority to High, Medium High, Medium, Medium Low, or Low.

Schedule Tab

Complete the following fields:

- **Start**
  The date and time when the monitor starts. For the time value, use a 24-hour clock.

- **Stop**
  The date and time when the monitor stops. For the time value, use a 24-hour clock. The stop date and time must be later than the start date and time.
  How often the monitor runs. Depending on the selection, more options could appear.

  - **One Time:** The monitor runs once, at the start date and time.

  - **Every:** Enter the frequency in minutes. The monitor runs every NN minutes. However, if the previous run has not completed, CVS skips the run and try again at the next scheduled time. CVS will not terminate a monitor after it has started.

  - **Daily:** The monitor runs once a day, at the time that is specified in the start time.

  - **Weekly:** Select one or more days of the week. The monitor runs once on each selected day, at the time that is specified in the start time.

  - **Monthly:** Enter one or more days of the month. If you enter multiple days, you must be separate them with spaces. For example, you could enter 1 15 30. If you specify a day that does not occur in a specific month (such as 31) the day is ignored for that month. The monitor runs once on each day, at the time that is specified in the start time.

  - **CRON:** Enter a standard cron specification. Cron is a standard UNIX syntax for specifying time intervals. This approach allows the most flexibility when specifying a run schedule.

Test Case Info Tab

Complete the following fields:

- **Configuration**
  The name of the configuration. If this field is blank, the configuration active in the test case is used.

- **Staging doc**
  The name of the staging document.

- **Coordinator server**
  The name of the coordinator server.

Suite Info Tab

Complete the following field:

- **Configuration**
  The name of the configuration.

Edit MAR Info Files

The MAR info editor lets you change the information that was specified during the creation of the MAR info file.

The editor includes an optimize check box. By default, the check box is selected. When the MAR file is built, only those project files that the primary asset requires are added. However, the editor lets you specify that an optimized MAR file include one or more nonrequired project files.

If you clear the **optimize** check box, then all the project files are added to the MAR file.

To edit MAR info files:

1. In the **Project** panel of DevTest Workstation, double-click the MAR info file.
   The editor opens.
   The information that you can edit in the upper area depends on the type of MAR info file.

2. In the **Notes** area, add any notes about the MAR info file.

3. Select or clear the **optimize** check box.
   If you want the MAR to include all of the project files, then clear the **optimize** check box.
   The **Files to Include** and **Files to Exclude** areas are disabled.

   > **Note:** An archive is typically held in memory. Therefore, the use of optimized archives is highly encouraged. This check box is selected by default.

4. If you want the MAR to include any nonrequired project files, then move the files to the **Files to Include** area.
   The **Files to Include** area lists the required project files. The **Files to Exclude** area lists the nonrequired project files.

5. From the main menu, select **File**, **Save**.

Build MARs

After you have created and edited a MAR info file, you can use the file to build a Model Archive (MAR).

You can save the MAR to a folder that is inside or outside the project directory.

Follow these steps:

1. In the **Project** panel of DevTest Workstation, right-click the MAR info file and select **Build Model Archive**.
   The **Build Model Archive** dialog appears.

2. Navigate to the folder where you want to save the MAR.

3. Click **Save**.

Deploy to CVS

You can deploy a Monitor MAR info file to CVS.

Follow these steps:

1. In the **Project** panel of DevTest Workstation, right-click the .mari file and select **Deploy to CVS**.
   A confirmation message appears.

2. To see the newly added monitor, go to the **CVS Dashboard**.

Make Mar Utility

You can use the Make Mar command-line utility to show the contents of MAR info files (stand-alone or in an archive) or to create model archive files from MAR info files.

This utility is located in the **LISA_HOME\bin** directory.

Options

Each option has a short version and a long version. The short version begins with a single dash. The long version begins with two dashes.

- **-h, --help**
  Displays help text.

- **-s file-name, --show=file-name**
  Shows the contents of a MAR info file.
  If the file name refers to a MAR info file, then it is written out. If the file name refers to an archive, then both the audit and info entries are written out.

- **-c, --create**
  Creates one or more model archives from MAR info files.
  To specify the MAR info file name to build the archive, use the --marinfo argument.
  To specify the name of the archive to create, use the --archive argument.
  If the --source-dir argument is used, then the --marinfo argument is taken as a name pattern to look for in the full directory tree. In this case, the --target-dir argument must be used to note where to place the created archives. This command also implies auto-naming of the created archives.

- **-m mar-info-name, --marinfo=mar-info-name**
  The parameter that specifies the name of the MAR info file to read (if the --source-dir argument is not used) or the MAR info file name pattern to look for (if the --source-dir argument is used). In the latter case, if not specified, it defaults to .mari.

- **-s directory, --source-dir=directory**
  Specifies the directory where the tool searches for MAR info files to make archives from. The full directory tree is searched for files that match the pattern that the --marinfo argument specifies.

- **-t output-directory, --target-dir=output-directory**
  Specifies the directory where archives are written.
  When this argument is used, the created archives are automatically named based on the MAR info file name with a numeric suffix as appropriate to ensure that no files are overwritten.

- **-a archive-file, --archive=archive-file**
  Specifies the name of the archive file to create.
  When this argument is used, the --marinfo argument must specify a single existing MAR info file. The --source-dir and --target-dir arguments are not allowed.

- **--version**
  Prints the version number.

Examples

The following example shows the contents of a MAR info file.

```
MakeMar --show=C:\Lisa\examples\MARInfos\AllTestsSuite.mari
```

The following example creates a model archive named **rawSoap.mar**.

```
MakeMar --create --marinfo=C:\Lisa\examples\MARInfos\rawSoap.mari --
archive=rawSoap.mar
```

The following example creates a model archive for every MAR info file in the **examples\MARInfos** directory. The destination directory **examples\MARs** must exist before you run this command.

```
MakeMar --create --source-dir=examples\MARInfos --target-dir=examples\MARs
```

Running Test Cases and Suites

You have many options for running test cases:

- In DevTest Workstation, you can use the Interactive Test Run (ITR) utility to walk through and verify the steps of a test case.

- In DevTest Workstation, you can run a test case quickly with minimal setup to a specific step in the case.

- In DevTest Workstation, you can run a test case quickly with minimal setup.

- In DevTest Workstation, you can stage a test case. This option requires you to specify a configuration, staging document, and coordinator server. Often, the same test cases are used with different staging documents to perform different tests. That is, you do not have to write a separate test case for functional, regression, and load testing. Instead, a different staging document is prepared using the same test case.

- Test Runner is a command-line utility that lets you run tests as a batch process.

- LISA Invoke API is a REST-like web application that enables you to run test cases and suites with a URL.

- You can schedule tests to run at regular time intervals by using Continuous Validation Service (CVS).

- You can run tests as part of an automatic build process by using Ant and JUnit.

- You can run tests from DevTest Portal.

More Information:

- Interactive Test Run (ITR) Utility
- Stage a Quick Test
- Stage a Test Case
- Run a Test Suite
- Run a Selenium Integration Test Case
- Assumption of Load Testing
- Test Runner
- LISA Invoke API
- REST Invoke API
- Using the HP ALM Plug-in
- HTTP and SSL Debug Viewer
- Manage Test Artifacts

Interactive Test Run (ITR) Utility

The Interactive Test Run (ITR) utility lets you walk through and verify the steps of a test case. With this utility, you can verify that test cases are running correctly before staging them. The ITR runs the test that is in memory, rather than loading a test case document. You can make changes to the test case in real time to alter test properties or the workflow.

For example, you can run a specific test step out of the natural workflow sequence. The real-time changes that you make as the test is running are integrated into the test case. The test case continues to run without requiring a restart. The exception is a global change, such as changing the active configuration. If you execute a step and you get an unexpected result (for example, because of a bad parameter), you can leave the ITR open and edit the step. You can then go back to the ITR and execute the same step again.

The ITR lets you save the current ITR state, and to load an ITR state that was previously saved. This feature lets you communicate an error, in context, to another member of your team. You can send the test case and the saved ITR state to provide the exact actions and the results that you observed.

More Information:

- Start an ITR Run
- Examine the Results of an ITR Run
- Graphical Text Diff Utility

Start an ITR Run

The Interactive Test Run (ITR) utility is run from an open test case in DevTest Workstation.

Follow these steps:

1. Do one:
   - Select **Actions**, **Start Interactive Test Run** from the main menu.
   - Click **ITR** on the toolbar. The icon gives you the option of starting a new ITR run or opening a previous ITR run.

   The **Interactive Test Run** window opens.
2. (Optional) To change any settings, use the Settings tab.
3. Use the Run tab to execute all or part of the test case.

ITR Run Tab

The **Run** tab on the **ITR** window shows the steps that have been executed (in gray) and the next step that will be executed (in green).

In the following graphic, the first two steps (Add User and Get User) have been executed. The third step (Verify User Added) is the next step that will be executed.

Each step has a pull-down menu that contains all the steps in the test case. You can use this menu to change the next step that will be executed. Select a step, and it will replace the current next step with the one of your choice. After you change the step, the workflow will continue from the new step.

You manage the ITR by using the toolbar at the bottom of the **Run** tab.

	Run the next step in the test case. After the step has run, you can examine the results in the right panel. Click the icon again to run the next step that is listed, or select a different step to run as described previously.
	Run all of the steps in the test case. While the test is running, the icon changes into a **Stop** icon. You can stop the test by clicking the **Stop** icon. When the last step has been executed, a dialog indicates that the test is complete.
	Start the test again. This feature is useful if you change your test case and you want to execute it from the beginning.
	Save the current ITR state. In the dialog, enter the name of the save file. The suffix .itr is appended to the name. Subsequent saves overwrite this file.
	Load a saved ITR. Browse to the .itr file that you want to load. The Saved ITR State contains the information that is displayed in the tabs, capturing all the testing performed. To use the Saved State, first open the test case that was used when the original tests were run.
	To add or watch properties, open the property watch window.

ITR Settings Tab

The **Settings** tab lets you control various aspects of the ITR.

You can filter out events and properties from the results tabs. You might not want to see some verbose events. Likewise, you do not always want to clutter the property list with DevTest internal properties.

- **Show CALL_RESULT**
  Include EVENT_CALL_RESULT events in the **Test Events** list.

- **Show NODERESPONSE**
  Include EVENT_NODERESPONSE events in the **Test Events** list.

- **Show Internal Props**
  Include all internal events in the property lists in the **Initial State** and **Post Exec State** tabs.

- **Auto Cleanup Auto-Runs**
  Clean up the Auto Runs automatically.

- **Auto Run Delay (secs)**
  In the Automatically Execute Test mode, the ITR pauses between each step execution. This setting lets you change the pause interval. The ITR does not honor the think time, so this setting lets you add a constant delay between each step execution.

- **Enable CA CAI**
  Controls whether Application Insight is enabled.
  Default: false

Examine the Results of an ITR Run

You can examine the results of the execution of a test step, during the run, between two steps, or at the end of the run.

Select the step in the **Execution History** list. Examine the information in the right panel, which contains the following tabs.

Response Tab

The **Response** tab displays the step response after executing a step.

The display uses the editor appropriate for the response type. For example, the Add User step in the multi-tier-combo test case invokes the HTML/XML response.

Complex Object Editor

The Get User step in the multi-tier-combo test case invokes an EJB that returns an object that is displayed in the Complex Object Editor.

ITR Results Response tab

Some editors include two icons that you can use to save the ITR state and start an external browser .

> **Note:** If the response is XML and it is larger than 5 MB, it is displayed in plain text with no DOM view. This limit can be adjusted with the **gui.viewxml.maxResponseSize** property.

Properties Tab

The **Properties** tab displays the state of the step after execution (Value) and immediately before execution (Previous Value).

To show or hide DevTest internal properties, select or clear the **Show Internal Props** check box in the **Settings** tab.

ITR Results Properties tab

Properties that the step set are highlighted in green. Properties that were modified in the step are highlighted in yellow.

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 show or hide the events, select or clear the **Show CALL_RESULT** and **Show NODERESPONSE** check boxes in the **Settings** tab.

ITR Results Test Events tab

The **Test Events** tab contains the following columns:

- **EventID**
  The name of the event.

- **Timestamp**
  The date and time of the event.

- **Short**
  The short description of the event.

- **Long**
  The long description of the event. The long description can be truncated in the display. To view the full text, click the cell and its full contents are displayed in the **Long Info Field** panel.

You can enter the complete or full description of the event in the **Long Info Field**.

An event is generated on all assertions that are fired or evaluated.

If a warning was generated, the row is yellow.

If an assertion was fired, the row is purple.

If a property was set, the row is orange.

If an error was generated, the test was failed, or the test was aborted, the row is red.

> **Note:** If the step calls a subprocess that contains an assertion, then the **Test Events** tab includes the appropriate event for the assertion. For example, the **Test Events** tab can include an Assertion fired event that occurred in the subprocess.

Graphical Text Diff Utility

A graphical XML diff engine and visualizer are available to compare XML files and graphically display their differences.

- Start the Graphical Text Diff Utility
- Graphical Text Diff Utility Properties

Start the Graphical Text Diff Utility

Follow these steps:

1. Select **Actions**, **Graphical Text Diff** from the main menu.
   The **Graphical Text Diff** window opens. From this panel, you can compare the contents of two XML files.

2. Click **Compare** to start the compare.
   The following window opens, which shows the actual comparison. The **Diff Viewer** tab shows the comparison.

Graphical Text Diff - Diff Viewer tab

To start the Graphical Text Diff utility from the Interactive Test Run (ITR) utility:

1. Select the **Test Events** tab in the ITR.
2. Right-click a table row and select **Select Left Text to Compare**.
3. Right-click a table row to compare with and select **Compare To**.
   The Graphical Text Diff utility opens for comparison.

While in the Graphical Text Diff utility, you can also load the contents by selecting a file.

Follow these steps:

1. Click the **Select Contents** tab in the **Graphical Text Diff** window.
2. To load the file, click **Load from File**.
   The results of the diff are then displayed in the global tray panel component.

Note: You can also set the compare options in the **Settings** tab. For more information about the compare options, see Graphical XML Side-by-Side Comparison.

Graphical Text Diff Utility Properties

The Graphical XML diff utility uses the following properties. These properties can be overridden in **local.properties** or at run time.

- **lisa.graphical.xml.diff.engine.max.differences**
  This property configures the maximum number of differences that are detected before the XML diff algorithm stops. Set to -1 to compute all differences. When there are many differences, this setting causes the XML diff algorithm to finish faster.
  Default: 100
- **lisa.graphical.xml.diff.report.max.linewidth**
  This property configures the maximum width of a line of text in the XML diff results report. If the input XML has long lines of text, this property causes the XML diff results report to consume less memory.
  Default: 80
- **lisa.graphical.xml.diff.report.max.numberoflines**
  This property configures the maximum number of context lines for a difference in the XML diff results report. If the XML that is being compared has different elements that are large, this property causes the XML diff results report to consume less memory.
  Default: 5

Note: In most cases, these values should not be changed. Changing the default values for these properties could result in the Graphical XML diff engine using more CPU or running out of memory, or both.

Stage a Quick Test

DevTest Workstation lets you run a test case quickly with some minimal setup. A quick test runs the test that is in memory, rather than loading a test case document. A quick test uses a simple prebuilt staging specification with few options. The test has minimal instances so it is easy to stage/run, but lacks much of the functionality of a test staged with a staging document as in the case of a proper test case. A quick test lets you select and monitor events and metrics and view a standard performance report.

Follow these steps:

1. Open a test case in DevTest Workstation.

2. Select **Actions**, **Stage a Quick Test** from the main menu.
   The **Stage a Quick Test** dialog opens.
   If you have coordinators and simulators that are attached to a registry, it shows you the coordinator or simulator servers attached.

3. Specify the following parameters:

   - **Run Name**
     The name of the quick test.

   - **Number of Instances**
     The number of concurrent users (instances) to be used. Your license determines the maximum number of users that you can specify.

   - **Stage Instances To**
     The name of the coordinator server where the test is staged.

   - **If test ends, restart it**
     Select this check box to run the test continuously until you stop it manually.

4. Click **OK**.
   The Test Monitor window opens.

Test Monitor Window - Quick Test

When you stage a quick test, the **Test Monitor** window opens in DevTest Workstation. The **Test Monitor** window for a quick test is similar to the **Test Monitor** window for a test case.

At first, the test is staged, but the test has not started running yet.

You can select the metrics and events to monitor during the test run.

The Test Monitor consists of two tabs:

- The Perf Stats Tab displays collected metrics as a function of time. After the test starts, this display cannot be changed.

- The Events Tab displays events as they are recorded during the test run.

If you run the test case and there are failures in the test, then a third tab that is labeled Failures is displayed.

If you run a baseline test case, then the **Consolidated Baseline** tab is also displayed. For more information, see Using Application Insight.

You can limit the size of the failure events by adding the **lisa.coord.failure.list.size** property to the **local.properties** file.

Perf Stats Tab - Quick Test

The **Perf Stats** tab enables you to add the metrics and events that you want to monitor during the test run.

Image of the Perf Stats tab of the Test Monitor window

The **Perf Stats** tab consists of the following panels:

- **Test Metrics Status**
- **Current Interval Metrics**
- **Summary Interval Graphs**

Test Metrics Status

The **Test Metrics Status** panel lists the current metrics being monitored.

Some metrics are shown by default. The metrics are color-coded to help you distinguish between them.

You can add more metrics by clicking **Add** . A pull-down menu displays all the metrics categories that can be added.

The **Test Metrics Status** panel contains the following columns:

- **Color**
  The color coding that is 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 asterisk means use only the event name for the metric.

- **Current Scale**
  The vertical scale that is used on the graphs. You can adjust the graph scale on the Current Interval Metrics graph for any metric. Adjust the scale by double-clicking the **Current Scale** cell and selecting a new value from the drop-down list. You see the metric change scale on the Current Interval Metrics graph.

  > **Note:** The default setting for **Current Scale** is Auto Scale at 100. The number in () shows what Auto Scale has determined the scale value must be to fit all the data on the same 0-100 graph. The value * scale = y coordinate on the graph. If you modify the scale, the current value does not change. However, the calculation of determining the y coordinate used on the graph could yield a different y coordinate. If the coordinate is greater than 100, the Y-axis limits also must grow to accommodate the larger values.

- **Current Value**
  The instantaneous value of the metric.

For detailed information about metrics, see Generating Metrics.

Current Interval Metrics

For each metric in the **Test Metrics Status** panel, the **Current Interval Metrics** panel displays the value at a specified time interval during the run.

To specify the time interval, use the slider at the bottom of the panel. You cannot adjust the value after the run has started.

Summary Interval Graphs

The **Summary Interval Graphs** panel displays the value of each metric in the **Test Metrics Status** panel, averaged over a specified time interval.

To specify the number of samples per interval, use the slider at the bottom of the panel. You cannot adjust the value after the run has started.

Events Tab - Quick Test

The **Events** tab displays the events that are generated during the run.

The **Events** tab consists of the following panels:

- **Events to Filter Out**
- **Simulators**
- **Test Events**

Stage a Quick Test events tab

Events to Filter Out

The **Events to Filter Out** panel enables you to restrict the events that appear during the run. The panel contains a list of events. If the check box for an event is selected, the event does not appear during the run.

The panel includes a drop-down list. The options enable you to filter none of the events, include a predefined set of events, or filter all of the events. The options are as follows:

- **No Filter**
- **Terse Event Set**
- **Common Event Set**
- **Verbose Event Set**
- **Load Test Set**
- **Custom Event Set**
- **Filter All Events**

Note: Under a load test, these UI elements are disabled. For a test case that DevTest determines is a load test, you cannot change the test to send, for example, individual step responses. Sending more than the basic events negates the value of the load test.

Simulators

The **Simulators** panel shows the status of the simulators in use.

Test Events

The **Test Events** panel displays the events. The most recent event appears at the top. The oldest event appears at the bottom.

You can list events in real time by selecting the **Auto Refresh** check box at the bottom of the panel, or you can refresh the list manually by clicking the **Refresh** icon, with the **Auto Refresh** box cleared. Only those events that have not been filtered out are displayed.

For each event, the following information is displayed:

- **Timestamp**
  The time of the event
- **Event**
  The name of the event
- **Simulator**
  The name of the simulator in which the event was generated.
- **Instance**
  The instance ID and run number (separated by a forward slash)
- **Short Info**
  A short piece of information about the event
- **Long Info**
  A long piece of information about the event (if available)

Failures Tab - Quick Test

If there is an error in the staged test run, you see a **Failures** tab at the bottom of the window.

Failures tab on the Test Monitor window

To see a report on any failed test, double-click the **Long Info** field at the right of the window.

Starting and Stopping Quick Tests

With the **Test Monitor** window open, you can start and stop a quick test by clicking icons on the main toolbar.

To start a quick test, click **Play** on the main toolbar.

When the quick test starts, the **Play** icon changes to a **Stop** icon.

To stop a quick test, click **Stop** on the main toolbar.

If you click **Stop** again, you are asked if you want to kill the test run.

- When you stop a test, you are saying "do not start any new instances." No new instances are started and a running test case does not loop back to the beginning. Tests that are running complete all steps, then go to the end step.
- When you kill a test, you are saying "stop all running instances as soon as possible." No new instances will be started, and a running test case does not go to the next step. No end step is run. Reports show that the test is still running, because the test never goes through an end step.

To replay a quick test, click **Replay** on the main toolbar.

To close a quick test, click **Close** on the main toolbar.

Actions Menu

When you run a test and click **Play** to start the test, the following options are added to the **Actions** menu.

- **Pause Metrics Collection**
- **Un-pause Metrics Collection**
- **Save Recent Metrics to XML Document**
- **Save Recent Metrics to CSV Document**
- **Save Interval Data to CSV Document**
- **Save Interval Data to XML Document**
- **Save Recent Events to CSV Document**
- **Stage This Test**
- **Stop Test**
- **Restart Test (Reloads Documents)**

Stage a Test Case

You can run a test case from DevTest Workstation by specifying the configuration, staging document, and coordinator server.

Follow these steps:

1. Open a test case in DevTest Workstation.
2. Select **Actions**, **Stage Test** from the main menu.
   The **Stage Test Case** dialog opens.
3. Select the configuration from the **Configuration** drop-down list.
   If you leave this field blank, the default configuration is used.
4. Select the staging document from the **Staging doc** drop-down list.
5. Select the coordinator server from the **Coordinator server** drop-down list.

   > **Note:** There are two choices for the coordinator: "Coordinator@default" and blank. If you select blank, then the simulation starts on the workstation machine and it will need connectivity to the DevTest database.

6. Click **Stage**.
   The Test Monitor window opens. You can now select the metrics and events to monitor, and then start the test case.

Test Monitor Window - Test Case

When you stage a test case, the **Test Monitor** window opens in DevTest Workstation. The **Test Monitor** window for a test case is similar to the **Test Monitor** window for a quick test.

At first, the test is staged, but the test has not started running yet.

You can select the metrics and events to monitor during the test run.

The Test Monitor consists of two tabs:

- The Perf Stats Tab lets you select metrics, and display them as a function of time.
- The Events Tab lets you select and display events as they occur during the test run.

If you run the test case and there are failures in the test, then a third tab that is labeled Failures is also displayed.

If you run a baseline test case, then the **Consolidated Baseline** tab is also displayed. For more information, see Using Application Insight.

You can limit the size of the failure events by adding the **lisa.coord.failure.list.size** property to the **local.properties** file.

Perf Stats Tab - Test Case

The **Perf Stats** tab lets you add the metrics and events that you want to monitor during the test run.

Image of the Perf Stats tab of the Test Monitor window

The **Perf Stats** tab consists of the following panels:

- **Test Metrics Status**
- **Current Interval Metrics**
- **Summary Interval Graphs**

Test Metrics Status

The **Test Metrics Status** panel lists the current metrics being monitored.

Some metrics are shown by default. The metrics are color-coded to help you distinguish between them.

The **Test Metrics Status** panel contains the following columns:

- **Color**
  The color coding that is 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 asterisk means use only the event name for the metric.

- **Current Scale**
  The vertical scale that is used on the graphs. You can adjust the graph scale on the Current Interval Metrics graph for any metric. Adjust the scale by double-clicking the **Current Scale** cell and selecting a new value from the drop-down list. You see the metric change scale on the Current Interval Metrics graph.

  > **Note:** The default setting for **Current Scale** is Auto Scale at 100. The number in () shows what Auto Scale has determined the scale value must be to fit all the data on the same 0-100 graph. The value * scale = y coordinate on the graph. If you modify the scale, the current value does not change. However, the calculation of determining the y coordinate used on the graph could yield a different y coordinate. If the coordinate is greater than 100, the Y-axis limits also must grow to accommodate the larger values.

- **Current Value**
  The instantaneous value of the metric.

The metrics are based on sampling. For example, the instances metric is the number of virtual users running a test when the sample is taken. The number of instances that are waiting to run tests can be a higher number. For example, if pacing is configured in the staging document, the instances metric shows how many virtual users are running a test. This value can be less than the population of virtual users that are allocated in the staging document. Assume that you have 250 steady-state virtual users. If the instance count is 100, then the other 150 virtual users are waiting because pacing has them on hold.

For detailed information about metrics, see Generating Metrics.

Current Interval Metrics

For each metric in the **Test Metrics Status** panel, the **Current Interval Metrics** panel displays the value at a specified time interval during the run.

To specify the time interval, use the slider at the bottom of the panel. You cannot adjust the value after the run has started.

Summary Interval Graphs

The **Summary Interval Graphs** panel displays the value of each metric in the **Test Metrics Status** panel, averaged over a specified time interval.

To specify the number of samples per interval, use the slider at the bottom of the panel. You cannot adjust the value after the run has started.

Events Tab - Test Case

The **Events** tab displays the events that are generated during the run.

Stage a Quick Test events tab

The **Events** tab consists of the following panels:

- **Events to Filter Out**
- **Simulators**
- **Test Events**

Events to Filter Out

The **Events to Filter Out** panel enables you to restrict the events that appear during the run. The panel contains a list of events. If the check box for an event is selected, the event does not appear during the run.

The panel includes a drop-down list. The options enable you to filter none of the events, include a predefined set of events, or filter all of the events. The options are as follows:

- **No Filter**
- **Terse Event Set**
- **Common Event Set**
- **Verbose Event Set**
- **Load Test Set**
- **Custom Event Set**

- **Filter All Events**

> **Note:** Under a load test, these UI elements are disabled. For a test case that DevTest determines is a load test, you cannot change the test to send, for example, individual step responses. Sending more than the basic events negates the value of the load test.

Simulators

The **Simulators** panel shows the status of the simulators in use.

Test Events

The **Test Events** panel displays the events. The most recent event appears at the top. The oldest event appears at the bottom.

You can list events in real time by selecting the **Auto Refresh** check box at the bottom of the panel, or you can refresh the list manually by clicking the **Refresh** icon, with the **Auto Refresh** box cleared. Only those events that have not been filtered out are displayed.

For each event, the following information is displayed:

- **Timestamp**
  The time of the event
- **Event**
  The name of the event
- **Simulator**
  The name of the simulator in which the event was generated.
- **Instance**
  The instance ID and run number (separated by a forward slash)
- **Short Info**
  A short piece of information about the event
- **Long Info**
  A long piece of information about the event (if available)

Failures Tab - Test Case

If there is an error in the staged test run, a **Failures** tab appears at the bottom of the window.

Failures tab on the Test Monitor window

To see a report on any failed test, double-click the **Long Info** field at the right of the window.

Starting and Stopping Test Cases

With the **Test Monitor** window open, you can start and stop a test case by clicking icons on the main toolbar.

To start a test case, click **Play** on the main toolbar.

When the test case starts, the **Play** icon changes to a **Stop** icon.

To stop a test case, click **Stop** on the main toolbar.

If you click **Stop** again, you are asked if you want to kill the test run.

- When you stop a test, you are saying "do not start any new instances." No new instances are started and a running test case does not loop back to the beginning. Tests that are running complete all steps, then go to the end step.
- When you kill a test, you are saying "stop all running instances as soon as possible." No new instances will be started, and a running test case does not go to the next step. No end step is run. Reports show that the test is still running, because the test never goes through an end step.

To replay a test case, click **Replay** on the main toolbar.

To close a test case, click **Close** on the main toolbar.

Run a Test Suite

A test suite lets you group 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 in the suite still produces 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.

Running a test suite requires connectivity to the DevTest database.

When a suite is included in 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 are ignored.

In DevTest Workstation, you can configure and stage the test suite, monitor the tests while they are running, and view the reports at the conclusion of the test.

Follow these steps:

1. Open the test suite in DevTest Workstation.
2. Select **Actions**, **Run** from the main menu.
   This action opens a menu where you select whether to run locally or to run with a specified registry.

   The **Run Suite Locally** dialog opens.
3. Enter the name of the test suite.
4. Select the configuration from the drop-down list.
5. Click **Stage**.

The Stage Suite Execution window opens and the tests start.

Stage Suite Execution

When you click **Stage** during the process of running a test suite in DevTest Workstation, the **Stage Suite Execution** tab opens and the tests start.

Stage Suite Execution page

The top panel displays the following information:

- The location and name of the test suite document
- The name of the current test

The middle panel displays the Tests Run meter and four thermometers.

- The Tests Run meter shows the number of tests that completed and the total number of tests.
- The thermometers have the following labels: **Passed**, **Failed**, **Aborted**, and **Not Active**. The colors of the thermometers indicate that you have passed the preselected number of cases. The known behavior when you run more than 50 to 100 test cases in a suite is:
  - If the number of tests passed is greater than 50, then the thermometer is orange.
  - If the number of tests passed is greater than 75, then the thermometer is red.
  - If the number of tests passed is greater than 100, then the thermometer is gray.

The lower panel has the following tabs:

- Events Tab: Lists requested events as they occur
- Results Tab: Shows the status of each individual test

Stage Suite Execution - Events Tab

The **Events** tab lists the events that you have selected from the panel on the left of the tab.

Events to Filter Out

The **Events to Filter Out** area contains a list of the available events. Each event has a check box. To select events that you do not want to monitor, use this list.

One approach is to select the events to filter out manually. Another approach is to select one of the following event sets from the drop-down list and then customize the selections as necessary:

- Terse Event Set
- Common Event Set
- Verbose Event Set
- Load Test Set

You can clear all the check boxes by selecting **No Filter**. You can select all the check boxes by selecting **Filter All Events**.

Test Events

The **Test Events** area lists the events as they occur, with the most recent on the top of the list. You can list events in real time by selecting the **Auto Refresh** check box at the bottom of the panel. You can refresh the list manually by clicking the **Refresh** icon, with the **Auto Refresh** check box cleared.

For each event, the following information is displayed:

- **Timestamp**
  The time of the event
- **Event**
  The name of the event
- **Simulator**
  The name of the simulator in which the event was generated.
- **Instance**
  The instance ID and run number (separated by a forward slash)
- **Short Info**
  A short piece of information about the event
- **Long Info**
  A long piece of information about the event (if available)

Stage Suite Execution - Results Tab

The **Results** tab shows the status of the individual tests in the test suite.

The Results tab of the Stage Suite Execution page

Suite Results

The **Suite Results** area displays a list of all the tests in the suite, with icons.

- The green icon indicates that the test has completed and passed.
- The red icon indicates that the test has completed and failed.
- The blue icon indicates that the test is still running.

To display the test monitor for tests that are still running, click **View Test** at the bottom of the panel.

For completed tests, you can select the test name to see a status in the text area to the right. Seeing the status is most useful to see why a specific test failed.

Status Window

The **Status** window displays the status of the test that is selected on the left.

The following test is running:

Stage Suite Suite Results window with test running

The following test has failed:

Suite Results for failed test

A failed test status shows information available about 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 specific selected test:

Stop Test icon	**Stop Test**	Stops the test after it reaches the next logical end/fail. It does not start a new cycle.
Kill Test icon	**Kill Test**	Stops the test immediately after the current step completes.
View Test icon	**View Test**	This tool will work only for currently running tests. This starts the test monitor for the selected test.

The second set of icons manages the test suite itself, or when a suite is running:

Run Suite icon	**Run Suite**	Starts/restarts the suite test.
Close Suite icon	**Close Suite**	Close the **Stage Suite Execution** window.
View Test icon	**View Test**	Starts the test monitor for the selected test.

More Information:

- Use the Registry Monitor

Use the Load Test Optimizer

The Load Test Optimizer appears in the Registry Monitor and lets you 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 runs continuously. More simulated users continue to access it, until a specific target is reached. This target is typically a predefined average response time.

The Load Test Optimizer helps determine how many users the system under test supports. An optimizer can be set on any DevTest metric, such as average response time. An optimizer can also be set on a metric pulled from an external source, such as SNMP, JMX, or Windows Perfmon. The number of users increases at a preset frequency and informs you when a preset metric threshold is achieved. For example, you can increase the number of test instances by 5 instances every 10 seconds until the average response time hits 2 seconds.

To optimize the test, it must be running.

To configure and start an optimizer:

1. In the Registry Monitor, select a test from the running tests list and click **Optimize Test** .
   The **Optimizer** panel opens with the name of the staging document at the top.

2. Configure the following parameters:

   - **Metric**
     The metric to use for the optimization. Select from the pull-down list.

   - **Simulator**
     The simulator that is used to spawn test instances. Select from the pull-down list.

   - **Threshold Low**
     The metric value at which the optimizer reports that the system requires more virtual users. Enter a numeric value.

   - **Threshold High**
     The metric value at which the optimizer reports that the system cannot support any more virtual users. Enter a numeric value.

   - **Increment (#instances)**
     The number of extra virtual users to add to the system at the update frequency. Enter a numeric value.

   - **Update Frequency (millis)**
     The number of milliseconds between increments

3. Click **Start Optimizer** .
   As the optimizer increases the number of virtual users, the number of virtual users display on both the **Optimizers** section and in the Instances column of the **Tests** tab in the Registry Monitor.
   As the optimizer executes, you can change the parameters. To update the optimizer with the changed information, click **Update** .

4. To close the optimizer, click **Close** .

Run a Selenium Integration Test Case

Selenium Integration test steps let you import test scripts for web-based user interfaces from Selenium Builder to DevTest Solutions. Recording these test scripts requires Selenium Builder, which is supported only in Mozilla Firefox. After you import the test to DevTest, you can run the test in Google Chrome, Mozilla Firefox, Apple Safari, or Internet Explorer. You can also run the test in a local or remote browser.

A Selenium subprocess opens a new session by default, and closes that session when it is completed.

To share the session between your main test and your subprocess:

1. In the subprocess test case, add the **selenium.testrun** and **selenium.testrun.params** properties in both Subprocess Input Parameters and Subprocess Output Properties.

2. In the Execute Subprocess Step of the main test case, add these two property names in Parameters to Sub Process with the values from DevTest in {{ }} form, and select both properties in Result Properties.

You can run Selenium Integration tests like any other test cases in DevTest. However, running these tests in a browser other than Firefox requires more prerequisite tasks. For general information about running a test case, see Running Test Cases and Suites.

Note: Although Selenium integration supports the following web browsers: Internet Explorer, Firefox, Chrome, and Safari, it is constrained by the Selenium Web Driver support that is defined at http://www.seleniumhq.org/docs/01_introducing_selenium.jsp#supported-browsers-and-platforms.

Run a Selenium Integration Test on Google Chrome (Local)

This section describes how to run a Selenium Integration test case on Google Chrome on your local computer.

Follow these steps:

1. Download the Selenium Chrome driver and save it to a local directory.

    a. Go to http://www.seleniumhq.org/download/.

    b. Locate the Chrome driver in the **Third Party Browser Drivers NOT DEVELOPED by seleniumhq** section and download it.

2. Add the following properties to the project configuration file you use to run test cases on Chrome.

    - **Key:** selenium.browser.type
      Value: Chrome

    - **Key:** selenium.chrome.driver.path
      Value: The full path to the Chrome driver you downloaded. For example: C:\lisa-se\chromedriver.exe.

    Note: Add these properties to **project.config** before you add them to other project configuration files.

3. Right-click the selected project configuration file in the Project panel and select **Make Active**.

4. Run the test.

Tip: If the Google Chrome browser is installed at the user level; for example, its installation location is something like C:\Users\XYZ\AppData\Local\Google\Chrome\Application\chrome.exe, then when you stage the test case, you could see an error message like "org.openqa.selenium.WebDriverException: unknown error: cannot find Chrome binary."

To avoid this error, ensure that the Google Chrome browser is installed as a system wide installation. It must be installed in a location like C:\Program Files (x86)\Google\Chrome\Application\chrome.exe, so it can be accessed by all accounts.

Run a Selenium Integration Test on Microsoft Internet Explorer (Local)

This section describes how to run a Selenium Integration test case on Internet Explorer on your local computer.

Follow these steps:

1. Download the Selenium 32-bit Windows IE driver and save it to a local directory.

   a. Go to http://www.seleniumhq.org/download/.

   b. Locate the 32-bit Windows IE driver in the **Internet Explorer Driver Server** section and download it.

   > **Note:** Because of a known issue with 64-bit driver performance, we recommend that you install the 32-bit version, even if you use a 64-bit system.

2. Add the following properties to the project configuration file that you use to run test cases on Internet Explorer.

   - **Key:** selenium.browser.type
     Value: IE

   - **Key:** selenium.ie.driver.path
     Value: The full path to the Internet Explorer driver you downloaded. For example: C:\lisa-se\IEDriverServer.exe.

   > **Note:** Add these properties to **project.config** before you add them to other project configuration files.

3. Right-click the selected project configuration file in the Project panel and select **Make Active**.

4. Modify your Internet Explorer security settings.

   a. Open Internet Explorer.

   b. Click Tools, Internet Options.

   c. Click the Security tab.

   d. Verify that the Enable Protected Mode check box set the same (selected or cleared) for the following zones. If this setting is inconsistent, Selenium does not start.

      - Internet

      - Local Intranet

      - Trusted Sites

- Restricted Sites

e. Click OK to save your changes and close the Internet Options window.

5. For Internet Explorer 11 *only*, you must set a registry entry on the target computer so that the driver can maintain a connection to the instance of Internet Explorer that it creates.

   - For 32-bit Windows installations, the key you must examine in the registry editor is:

     ```
     HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Internet
     Explorer\Main\FeatureControl\FEATURE_BFCACHE
     ```

   - For 64-bit Windows installations, the key is:

     ```
     HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Internet
     Explorer\Main\FeatureControl\FEATURE_BFCACHE.
     ```

     Note: the **FEATURE_BFCACHE** subkey may or may not be present. If it is not present, you must create it.

     Important! Inside this key, create a DWORD value named iexplore.exe with the value of 0.

6. Run the test.

Run a Selenium Integration Test on Apple Safari (Local)

This section describes how to run a Selenium Integration test case on Apple Safari on your local computer.

Note: Safari is supported only on a Mac.

Follow these steps:

1. Download the Safari driver and save it to a local directory.

   a. Navigate to http://www.seleniumhq.org/download/.

   b. Locate the Safari driver in the **SafariDriver** section and download it.

2. Start the Safari browser and open the file **SafariDriver.safariextz** or drag-and-drop the file into the browser.

3. When the dialog appears, click **Install** to install the web driver extension.

4. To confirm that the extension has been installed successfully, view the Preferences, Extensions tab, which has the WebDriver extension listed.

5. Close the Safari browser.

6. Add the following property to the project configuration file you use to run test cases on Safari.

   - **Key:** selenium.browser.type
     Value: Safari

   > **Note:** Add this property to **project.config** before you add it to other project configuration files.

7. Right-click the selected project configuration file in the Project panel and select **Make Active**.

8. Run the test.

Run a Selenium Integration Test on a Remote Browser

This section describes how to run a Selenium Integration test case on a remote browser. The remote browser can be Mozilla Firefox, Google Chrome, or Internet Explorer 8.0 or later.

Follow these steps:

1. Download the Selenium Server and save it to a local directory.

   a. Go to http://www.seleniumhq.org/download/.

   b. Locate the Selenium Server stand-alone .jar file in the **Selenium Server (formerly the Selenium RC Server)** section and download it.

2. Verify that the driver for the browser that you want to use is available on the remote computer.

3. On the remote computer, run the following command from a command prompt:

   ```
   java -jar selenium-server-standalone-2.xx.0.jar -role hub
   ```

4. On the remote computer, run the following command from a new command prompt (for Internet Explorer, Firefox, or Chrome):

   ```
   java -jar selenium-server-standalone-2.xx.0.jar -role node -
   hub http://localhost:4444/grid/register -Dwebdriver.chrome.driver=c:\lisa-
   se\chromedriver.exe -Dwebdriver.ie.driver=c:\lisa-se\IEDriverServer.exe
   ```

   For Safari, run the following command from a new command prompt:

   ```
   java -jar selenium-server-standalone-2.xx.0.jar -role node -
   hub http://localhost:4444/grid/register -Dwebdriver.chrome.driver=c:\lisa-
   se\chromedriver.exe -Dwebdriver.ie.driver=c:\lisa-se\IEDriverServer.exe  -
   browser browserName=firefox -browser browserName=chrome -browser
   browserName="internet explorer" -browser browserName=safari
   ```

5. On the local computer, add the following properties to the project configuration file you use for running test cases on the remote browser.

   - **Key:** selenium.broswer.type
     Values: IE, Firefox, Safari, or Chrome

- **Key:** selenium.remote.url
  Value: URL to the remote Selenium Server hub. For example, **http://your_remote_hostname:4444/wd/hub**.

If you plan to run Selenium Integration tests on multiple browsers, create a project configuration file for each browser type. You can then run the tests on multiple browsers by making different configuration files active for each test run. For more information about configuration files, see Configurations.

> **Note:** Add these properties to **project.config** before you add them to other project configuration files.

6. Right-click the selected project configuration file in the Project panel and select **Make Active**.

Run the test.
The test runs on the selected browser on the remote computer.

> **Note:** For more information about using Selenium Server in a Grid configuration, see https://code.google.com/p/selenium/wiki/Grid2.

Run a Selenium Integration Test with a Simulator Service

Starting the simulator as a server is the only way you can actually see the browser come up. Selenium requires interactivity with the desktop, so the only way you can run your Selenium test against a simulator service is to run it headless.

1. Update your PATH System Environment variable to the folder of where your IE and Chrome drivers are on the remote machine in the path. The folder on the remote machine should match what is in your project config.

2. On the simulator service, update the **Properties**. Select the **Log On** tab, check the **Allow the service to interact with desktop** checkbox. You must use a local system as Logon.

3. Restart the Simulator service.
   You will now be able to run your Selenium test against the remote Simulator service.

Assumption of Load Testing

By default, DevTest examines various characteristics of a test case or suite at run time and may determine that you are running a load test. If DevTest determines you are running a load test, it reconfigures automatically.

The automatic configuration can be changed by setting the property **lisa.load.auto.reconfigure** =false.

Reconfiguration for load testing does several things. The most important is limiting the events that are sent from the simulators executing the test. Specifically, the simulators only send events in the LoadTest filter set. StepStarted, Step Response, Step Response Time, CycleStarted, Log, Profile, and other similar events are not sent to the coordinator and therefore not the DevTest component that started the test (DevTest Workstation or Test Runner).

The reason this is done is that with load tests, the overhead of sending the events quickly exceeds the act of creating load. The coordinator quickly becomes overwhelmed, especially with tests that do not have think time.

If DevTest assumes that you are running a load test, messages similar to the following messages appear in the coordinator log file:

```
INFO  com.itko.lisa.coordinator.
CoordinatorImpl - Configuring for load test (vusers >= 150)
INFO  com.itko.lisa.coordinator.CoordinatorImpl - Configuring for load test
INFO  com.itko.lisa.net.RemoteEventDeliverySupport - configuring for load test
```

An extra message can appear suggesting that you turn off Application Insight in the staging document.

If your test is part of a suite, the Test Monitor window usually provides no information. Because load tests can generate events faster than the user interface can keep up, some events are ignored. To see these run-time metrics, change the following property:

```
lisa.load.auto.reconfigure=false
```

To tune the parameters of what CA Application Test thinks is a load test, adjust the following properties:

```
lisa.loadtest.aggressive.detection=false
lisa.coordinator.step.per.sec.load.threshold=100
lisa.coordinator.vuser.load.threshold=150
```

If a staging document has a Default Report Generator, it is assumed NOT to be a load test.

If the number of non-quiet steps per second exceeds 100 or the number of virtual users exceeds 150, then the test is assumed to be a load test.

If you set the **lisa.load.auto.reconfigure** property to true, then any test with a staging document that has 0% think time or a test consisting entirely of steps with 0 think time is considered a load test.

Another result of the load testing assumption is that if load testing has been turned on automatically, an event is generated that documents the change.

Test Runner

The Test Runner command-line utility is a "headless" version of DevTest Workstation with the same functionality, but no user interface. That is, it can be run as a stand-alone application.

Test Runner lets you run tests as batch applications. You give up the opportunity to monitor tests in real time, but you still can request reports for later viewing.

- On Windows, Test Runner is available in the **LISA_HOME\bin** directory as a Windows executable, **TestRunner.exe**.
- On UNIX, Test Runner is available as a UNIX executable, **TestRunner**, and a UNIX script, **TestRunner.sh**.

Test Runner lets you incorporate DevTest tests into a continuous build workflow. Or, use Test Runner with JUnit to run standard JUnit tests in Ant or some other build tool.

Test Runner provides the following options:

```
TestRunner [-h] [[-u username] [-p password] [-r StagingDocument] [-
t TestCaseDocument] [-cs CoordinatorServerName]] | [-s TestSuiteDocument] [-
m TestRegistryName] [-a] [-config configurationFileName] [-Dname=value]
```

To display help information for Test Runner, use the **-h** or the **--help** option.

```
TestRunner -h
```

To display the version number, use the **--version** option.

As Part of an Automated Build

DevTest test cases can be incorporated into an automatic build and test process. DevTest provides the additional software that is required to use Java Ant, and Java JUnit, and an example Ant build script.

DevTest test cases are run and reported as native JUnit tests.

Test Runner can be used in standard JUnit tests using a custom Java class. For more information, see Running DevTest with Ant and JUnit.

More Information:

- Run a MAR with Test Runner
- Run a Test Case with Test Runner
- Run a Suite with Test Runner
- Other Test Runner Options
- Multiple Test Runner Instances
- Test Runner Log File

Run a MAR with Test Runner

To run a Model Archive (MAR) with Test Runner, specify the following option:

- **-mar** or **--mar** name of the MAR file

Example

The following example runs a MAR file named **test1.mar**.

```
TestRunner -mar C:\test1.mar
```

Run a Test Case with Test Runner

To run a single test case with Test Runner, specify the following options:

- **-t** or --**testCase** name of the test case document
- **-r** or --**stagingDoc** name of the staging document

To stage remotely, then also specify the following options:

- **-cs** or --**coordinatorService** name of the coordinator server
  If no **-cs** is supplied to designate a coordinator server to retrieve from the registry, then any coordinator that is specified in the MAR/MARI file is used. If neither coordinator is specified, then any default coordinator that is specified in the registry is used. If there is no default coordinator, then the test runs locally. If " - cs local" is used, then the test runs locally.
- **-m** or **--testRegistry** name of the registry

For more options, see Other Test Runner Options.

Example

The following example runs the multi-tier-combo test case that is located in the **examples** project.

```
TestRunner -t ../examples/Tests/multi-tier-combo.tst -r ../examples/StagingDocs
/Run1User1Cycle.stg -a
```

Run a Suite with Test Runner

To run a suite with Test Runner, specify the following option:

- **-s** or **--testSuite** name of the suite document

No auditing is performed.

To stage remotely, also specify the following option:

- **-m** or **--testRegistry** name of the registry

An important restriction for remote staging project documents is to have a unique name for all projects. This restriction applies not only to project suites, but also remotely staging project test cases that refer to other assets (like data sets) in the project.

For more options, see Other Test Runner Options.

Example

The following example runs the AllTestsSuite suite that is located in the **examples** project.

```
TestRunner -s ../examples/Suites/AllTestsSuite.ste
```

The following example assumes that the registry is running on another computer.

```
TestRunner -s ../examples/Suites/AllTestsSuite.ste -m somecomputer/Registry
```

Other Test Runner Options

This page describes more options that you can use while running a Model Archive (MAR), test case, or suite with Test Runner.

You can use properties when specifying file names. However, in this context, only system properties and properties that are defined in your property files (**lisa.properties**, **local.properties** , and **site.properties**) work.

When the tests are complete, you can view your reports.

Specify ACL Security Information

The **-u** *userName* or **--username**=*userName* and **-p** *password* or **--password**=*password* options pass ACL security information to DevTest. These options are required.

Automatically Start the Test

The **-a** or **--autoStart** option automatically starts the test, so you do not need to press **Enter** after the test has been staged.

Specify the Configuration

The **-config** or **--configFile** option lets you specify the configuration to use for a test run.

Test Runner does not understand DevTest projects, so you must provide the fully qualified path to the configuration file. For example:

```
-config \path\to\lisa\home\examples\Configs\project.config
```

Generate an HTML Report

The **-html** or **--htmlReport** option lets you have Test Runner produce an HTML summary report for a test case. This option cannot be used for suites.

The parameter after this option must be a fully qualified filename. For example:

```
-html \some\directory\MyReport.html
```

Change the Update Interval

The **-u** or --**update** option enables you to change the update interval from the default value of 5 seconds. This interval refers to how often Test Runner writes a status message to the log file.

```
-u 10
```

Runtime Properties

The -D option enables you to add runtime properties to Test Runner execution. This option is useful for adding tags that can be queried in reporting. The format is –D<*name*>=<*value*>.

```
-Dyear=2015
-Dphase=beta
```

Multiple Test Runner Instances

You can stage multiple instances of Test Runner from a single workstation to a DevTest Server.

You need 512 MB for each instance.

Test Runner Log File

Logging output is written to the **trunner.log** file. For information about the location of this file, see Logging.

The logging level that is used is the same as that set in the **LISA_HOME\logging.properties** file.

To change the logging level, edit the **log4j.rootCategory** property in the **logging.properties** file from:

```
{{log4j.rootCategory=INFO,A1}}
```

to

```
{{log4j.rootCategory=DEBUG,A1}}
```

Using the DevTest ALM Plug-in

The plug-in lets you load and run a DevTest test case as a Quality Center test from the HP Quality Center suite. This integration allows you to take advantage of all Quality Center features while harnessing the power of DevTest testing. By loading a DevTest test case into Quality Center, you get a real-time execution of DevTest tests. You also get the full capture of the test results. DevTest tests are executable inside the workflow of Quality Center. The tests report back results to maintain the context and status of the testing process.

Note: See Install the DevTest ALM Plug-in for installation information.

More Information:

- Set up DevTest Tests in HP Quality Center
- Run DevTest Tests in HP Quality Center
- Troubleshooting the DevTest ALM Plug-in

Set up DevTest Test in HP ALM

The plug-in uses VAPI-XP to integrate with HP ALM.

Note: Users must belong to the **QA-Tester** role in HP ALM, or to a role that has the same rights as **QA-Tester**.

You can access the JavaScript and VBScript templates referred to in this procedure by selecting **Start, All Programs, DevTest, Quality Center Plugin**. The templates are functionally equivalent.

The DevTest test can be a three file attachment (test case file, configuration file, and staging file), a MAR file, or a MAR info file.

To link an ALM test to DevTest, using the Test Plan feature of ALM, follow these steps:

1. Create a test.
2. Select **VAPI-XP-TEST** from the Type drop-down and enter the test a name.
3. Click **OK**.
4. Select the scripting language and click **Finish**.
   You add the script later.

5. Once the test is created, click the test.

6. Select the **Test Script** tab of the newly created test.

7. Replace the default contents with the contents of the appropriate script template located at Start>Programs>DevTest>Quality Center Plug-in><<*Script Template>> (JavaScript or VBScript template).

8. After the script template is added, add references to the test, staging, and config files in the Attachments tab.

   ```
   file: ///<drive>:/<path>/<file>.<ext>
   Test Doc: file:///C:/DevTest/examples/Tests/DevTest_config_info.tst
   Staging Doc: file:///C:/DevTest/examples/StagingDocs/Run1User1Cycle.stg
   Config: file:///C:/DevTest/examples/Configs/project.config
   ```

   We recommend that the test, staging, and config files are added as links (URLs) rather than attaching the actual file. As a result, it is not necessary for the changes of the test to be uploaded back to ALM. Click the link icon and then file a properly constructed URL to the file. If the file is on the same system as the ALM instance, it must be in a form similar to following information:

For MAR files and MAR info files, you can use a single file attachment and three file attachments.

In this procedure, you create one or more URL attachments. The following example is a URL for a test case file:

```
file:///C:/DevTest/examples/Tests/rest-example.tst
```

The following example is a URL for a MAR info file:

```
file:///C:/DevTest/examples/MARInfos/rest-example.mari
```

The following graphic displays the **Attachments** tab. The URL for a test case file has been added. You specify the links to the staging and configuration files. The test case requires the test, configuration, and staging files.

Screen capture of Attachments tab with three file attachment

Parameter Substitution

You can change the location of the file for a test to run the same test case with multiple staging and configuration. You create a substitution by applying a single file attachment, MAR file, MAR info file, or three file attachment (test, configuration, and staging) in the **Attachments** tab.

The following substitutions are available:

- **Parameter**
  You configure the parameter substitution in the attachment path from the **Parameters** tab. The values in this tab overwrite the custom and the global properties for a test case.

- **Custom field**
  Custom field parameter substitution only applies to the test plan. Use the **Details** tab to add custom fields.

The following graphic shows an example of a MAR information file with a parameter substitution:

Screen capture of Attachments Tab with MARI Substitution

The substituted parameter is located between the <<< and >>> characters: for example, <<<marifile>>>. After the file is attached, the <<< and >>> characters are encoded to %3C%3C%3C and %3E%E3%3E for example, %3C%3C%3Cmarifile%3E%E3%3E. The absolute file path is converted to the file URL.

The following graphic displays the location of the MAR information file in the **Description** pane.

Screen capture of MARI attachment

The following graphic displays the **Parameters** tab with the name of the substituted MAR information file:

Screen capture of Parameters Tab with MARI defined

Follow these steps to set up ALM:

1. Install HP ALM Connectivity Add-in and register the HP ALM - ALM Client.

    a. From a browser, navigate to your ALM server.

    b. From the Help menu, select **Add-ins page**.

    c. Select **HP Quality Center Connectivity** or **HP ALM Connectivity** and download the add-in to install.

    d. Select **HP ALM Client Registration**, then select **Register HP ALM Client**. These two add-ons are put on the Workstation computer.

2. Create a VAPI-XP test in ALM.

3. Select the **Test Script** tab and replace the default contents with the contents of the JavaScript or VBScript template included in the DevTest Quality Center Plugin menu.

4. Select the **Attachments** tab and add a URL to the test case file, MAR file, MAR info file, or three file attachments.

5. Attach a DevTest test file, staging document, and configuration file to the test plan in ALM.

6. Attach a MAR file to the test plan in ALM.

7. Save the VAPI-XP test.

Run DevTest Test in HP ALM

If the Workstation and registry are on different computers, specify the **lisa.registry.url** property in the **local.properties** file of the Workstation computer. Use this format:

```
lisa.registry.url=tcp://<ipaddress> :2010/Registry
```

After you set up a DevTest test in HP ALM, you can run the test from the Test Plan module or from the Test Lab module. Test Plan is for local testing only.

For test cases, MAR files, and MAR info files, a coordinator and simulator must be running.

To run the test from the Test Plan (debug mode), locate the test and click the **Test Script** tab. From there, you can click the green arrow (Execute Script). The test starts executing, and its output shows up in the output window. You are notified when the test completes.

To run the test from Test Lab:

1. Click the **Test Lab** icon on the right panel.
2. Create a Test Set.
3. Click the Select Tests tab in the top menu bar.
   The Test plan tree opens on the right.
4. Using the Green arrow in the Test Plan tree, add the test to a valid test set.
5. Run the single test or the entire test set.
6. Once the test has completed, check the status of the test from the **Test Set** window or check the history of the test in the **Test Instance Properties** window.

Depending on the structure of the test, a test run shows different results. If the test has multiple cycles that execute, then you see a list of cycle history results and its pass or fail status. If the test has only one cycle, then you see a list of steps for that test.

> **Note:** Subprocess steps show as PASSED or FAILED, but the steps within a subprocess do not appear in ALM.

The following graphic shows the results from a run of the **rest-example** test case in the Test Lab:

DevTestConnection.properties File

The DevTestConnection.properties file contains connection information and is created during the installation of the DevTest ALM plug-in. The file is located in the LISA_HOME\alm-plugin directory. See Install DevTest ALM Plug-in for installation information. You can edit the connection parameters in the file. Setting the parameters in the DevTestConnection.properties file sets the parameters at the global level for tests. All DevTest tests that use the same connection, timeout interval, and log file location overwrites the hardcoded parameters in the plug-in.

> **Note:** We recommend creating a backup of the DevTestConnection.propertes file before editing the file.

If you want to change the connection, timeout, or log settings for a specific test, edit those parameters in the Parameters tab of the test case.

The file contains the following connection values:

- **dt_hostname**
  The name of the host that is hosting the DevTest registry.
- **dt_scheme**
  The protocol HTTP or HTTPS used to connect to the DevTest registry.
- **dt_port**
  The port that is used to connect to the DevTest registry.
- **dt_coordinator**
  The coordinator of the DevTest registry.
- **dt_username**
  The username that is used to connect to the DevTest registry.
- **dt_password_encrypted**
  The encrypted password that is used to connect to the DevTest registry.
  The password is encrypted using the following methods:
  - The DevTestConnection.properties file is created during the installation of DevTest ALM plug-in or
  - Using the InstallationUtility executable to generate a new DevTestConnection.properties file

  The encrypted password can be overwritten with a plain text password by the following methods:

  - - Remove the encrypted password from the DevTestConnection.properties file. Add the clear text password value to the DevTestConnection.properties file.
    - Change the password at the test level in the Parameters tab.

Timeout Values

The following timeout values can be set in the DevTestConnection.properties file or in the Parameters tab of a test:

- **dt_lockWaitTimeout**
  The amount of time in milliseconds to wait to acquire a lock on a DevTest test or run instance.

- **dt_testRunTimeout**
  The amount of time in milliseconds to allow the DevTest test to run.

- **dt_waitTimeout**
  The amount of time in milliseconds to allow data retrieval for DevTest.

Log File Location Values

The log file location values can be set in the DevTestConnection.properties file or in the Parameters tab of a test. Use the **Parameters** tab to change the location of a log file for a specific test or a few tests. For example, if one test fails out of one hundred tests, you can change the location or the name of the log file for only that test. You do not need to review the entire log file of all the tests.

The following log file location values are available:

- **dt_logFile**
  The location of the log file directory.

- **dt_logFileName**
  The name of the log file.

- **dt_logFileRollover**
  The rollover size of the log file in kilobytes.

Troubleshooting the DevTest ALM Plug-in

This page identifies common issues and timeout errors that might occur when using the plug-in.

Common Issues with ALM and DevTest

1. A user running the test might not have permission to Run VAPI-XP Test in ALM Test Lab vs Test Plan. Consult the ALM administrator for a permission issue.

2. For connectivity issues between the Workstation and a remote registry, add this property to your **devtestconnection.properties** file on the Workstation computer: **lisa.registry.url** =tcp://<ipaddress>: 2010/Registry.

3. Ensure that the file attachment has the correct extension.

4. If the test does not run in ALM, verify that the same test runs correctly using the TestRunner Command.

5. You cannot run a DevTest test suite in ALM. You can create a Test Set in ALM Test Plan and then run the whole Test Set.

Run-time Error [-2147467261]: Value Cannot be Null or Parameter Name: Environment Variable for DevTest Not Set: LISA_HOME

LISA_HOME might not have been created after running all of the services.

Symptom:

You receive any of the following errors:

- Run-time Error [-2147467261]: Value Cannot be Null
- Parameter Name: Environment Variable for DevTest Not Set: LISA_HOME

Solution:

1. Verify that LISA_HOME is listed in the Environmental Variable settings.
2. Do one of the following steps:
   - If LISA_HOME is not listed, add it to the Environmental Variable settings.
   - If LISA_HOME is listed, exit the HP ALM client executable from Task Manager and re start the HP ALM client.

Timeout Errors

For messages indicating timeout errors, address the following options by enabling a specific timeout in the HP ALM VBScript or JavaScript:

- Wait Timeout
  The following options set the timeout retrieval of the Test ID and Cycle ID. Increase the timeouts for test cases that might take longer than 30 seconds to run. The default is 30000 milliseconds.

```
Long GetWaitTimeout
```

```
void SetWaitTimeout(long waitTimeoutInMillis)
```

Add the function code in the VBScript or the JavaScript to increase the timeout for extracting the Test ID and Cycle ID. This information is used to extract the test result.

- Lock Wait Timeout
  The following options set the lock acquisition timeout. The default is 10000 milliseconds.

```
long GetLockWaitTimeout()
```

```
void SetLockWaitTimeout(long lockWaitTimeoutInMillis)
```

- Test Run Timeout
  The following options set the timeout for running the test and waiting for a result. The default is 60000 milliseconds.

```
long GetTestRunTimeout()
```

```
void SetTestRunTimeout(long testRunTimeoutInMillis)
```

- Debug Flag
  The following options set the log levels to debug. Set to True for more debug messages.

```
bool IsDebug()
```

```
void SetDebug (bool isDebug)
```

- Response Timeout
  The following options set the timeout for running the test and waiting for a response from the ALM plugin. The default is 100000 milliseconds.

```
long GetResponseTimeout()

void SetResponseTimeout(long ResponseTimeoutInMillis)
```

HTTP and SSL Debug Viewer

The HTTP and SSL Debug Viewer lets you observe the details of HTTP and SSL activity in DevTest Workstation. This feature can be helpful in performing diagnostics.

You access the viewer by selecting **Help**, **HTTP/SSL Debug** from the main menu.

The vertical bar at the left indicates the category of each line:

- The color green is used for HTTP requests.
- The color navy is used for HTTP responses.
- Diagonal stripes are used for SSL. An extra purple bar appears for the SSL handshake summary.

The lines are color coded as follows:

- The color green is used for HTTP request headers.
- The color navy is used for HTTP response headers.
- The color black is used for normal output.
- The color gray is used for unimportant output.
- The color teal is used for interesting output.
- The color magenta is used for important output.
- The color dark orange is used for warning output.
- The color red is used for error output
- The color purple is used for summary output.

The bracketed number at the beginning of each line is a thread identifier. Multiple threads can act on the same connection.

The viewer creates the SSL output by parsing the messages in a debug log.

The SSL output includes a summary of the handshake process. When diagnosing an SSL problem, start by reviewing the handshake summary. If you need more details, review the output that appears before the summary. The viewer might not have all the data that you must have to solve the problem.

You can copy the output and paste it into a separate window as plain text. The colors are not included in the pasted version.

The viewer lets you specify whether to show all the lines, HTTP lines only, or SSL lines only.

SSL Handshake Summary

The SSL output in the HTTP and SSL Debug Viewer includes a summary of the events that took place during the handshake process. When diagnosing an SSL problem, start by reviewing the handshake summary.

> **Note:** This page assumes a basic understanding of SSL or its successor, TLS.

The following graphic shows an example of the summary.

```
[SSL Handshake Summary] Thread [Thread-48]
[SSL Handshake Summary] Acting as a Client
[SSL Handshake Summary]  *†‡ indicates linked optional steps
[SSL Handshake Summary]
[SSL Handshake Summary]  1  RUN                              Client Hello -->
[SSL Handshake Summary]  2  RUN                                           <-- Server Hello
[SSL Handshake Summary]  3* RUN                                           <-- Server Certificate (Public Key)
[SSL Handshake Summary]  4† SKIPPED                                       <-- Request Client Certificate
[SSL Handshake Summary]  5* ASSUMED  Verify and Trust Server Certificate v
[SSL Handshake Summary]  6‡ RUN                                           <-- Server Key Exchange
[SSL Handshake Summary]  7  RUN                                           <-- Server Hello Done
[SSL Handshake Summary]  8† SKIPPED      Client Certificate (Public Key) -->
[SSL Handshake Summary]  9† SKIPPED                                         v Verify and Trust Client Certificate
[SSL Handshake Summary] 10  RUN                     Client Key Exchange -->
[SSL Handshake Summary] 11† SKIPPED      Certificate Verify Confirmation -->
[SSL Handshake Summary] 12  RUN                Client Change Cipher Spec -->
[SSL Handshake Summary] 13  RUN                           Client Finished -->
[SSL Handshake Summary] 14  RUN                                           <-- Server Change Cipher Spec
[SSL Handshake Summary] 15  RUN                                           <-- Server Finished
```

Screen capture of handshake summary

The first line displays the thread name.

The second line indicates whether the SSL debug log that the viewer uses is functioning as a client or a server. If a session has resumed, the second line also displays a corresponding message.

The remaining lines show the steps of the handshake process.

All of the possible steps appear in the summary, even steps that are optional in the handshake protocol. In the optional steps, a symbol appears to the right of the step number. The optional steps that are related to each other are shown with different sets of symbols. For example, an asterisk is used for step 3 and step 5, both of which pertain to the server certificate.

Each step has one of the following statuses:

- RUN: The step was performed.
- SKIPPED: The step was not performed.
- ASSUMED: The step was assumed to have been performed, based on other events that occurred.
- UNKNOWN: The initial status of all steps. If the status did not change, the step was most likely not performed.

Each step includes a brief description of an action that the client or server performed. For example, the first step shows the client sending a hello message to the server. If the action involves a message being sent, a left or right arrow illustrates the direction of the message flow. If the action does not involve a message being sent, a downward-facing arrow appears.

If an SSL problem occurs, the summary provides guidance to help you determine what went wrong. The following example shows the output that appears when a test step attempts to make an https request to a non-SSL port.

```
SEND TLSv1 ALERT:  fatal, description = handshake_failure
javax.net.ssl.SSLHandshakeException: Remote host closed connection during handshake
Ensure that the server is secure (connecting to insecure server over SSL) and that
 you are connecting to the correct port
```

Reports

You determine what data is collected for reports by specifying reporting parameters in one of three areas:

- Quick Tests
- Staging Documents
- Test Suites

You can specify the specific events or metrics you want collected for each test case or test suite you run. You select between three report generators that store reporting data in either a database or an XML file. You also determine how long the reporting data is kept.

After the reports are generated, they can be viewed and managed later, shared with colleagues, or exported to other locations.

When you submit a test case from DevTest Workstation, and the Workstation is not connected to the registry database, no reporting information is saved for that case.

View Reports

View reports from the DevTest Portal. For more information, see View and Navigate Reports.

More Information:

- Report Generator Types
- Changing Reporting Databases

Report Generator Types

You can select the following types of report generators in the **Reports** tab of the Staging Document Editor or the Test Suite Editor.

- Default Report Generator
- Load Test Report Generator
- CA Agile Central Report Generator
- XML Report Generator

Default Report Generator

The default report generator captures functional and metric information and publishes that data to the reporting database referenced by the registry. The reporting portal uses the reporting database.

We do not support this report writer for load testing or performance testing. For load testing, use the Load Test report generator.

> **Note:** When this report generator is selected, CA Application Test will not automatically configure for load testing.

Load Test Report Generator

The Load Test report generator is designed for load tests with thousands of virtual users.

This report captures load metrics but not step-level metrics. If it captured step-level metrics, there would be too much data and the reporting database would slow down the test.

Almost all events are disabled. As a result, the reports contain little information for the run. The main feedback is in the **Test Run** panel of DevTest Workstation.

In the **Parameters** area, you can configure the number of errors at which the test automatically stops. The default value is 100.

The Default report generator is not supported for load and performance testing.

CA Agile Central Report Generator

The CA Agile Center report generator publishes DevTest test results into the CA Agile Central server.

> **Note:** An API key must be created to integrate DevTest with CA Agile Central. See Configure CA Agile Central for complete steps.

In the **Parameters** area of the report generator, use the following option to identify the test case:

- **Build Number Launch State Key**
  The build number, obtained from the launch state with the property key you enter here. Launch state is set before the test starts to run. The "launch of the test" means that you are going to set an extra property, in this case the BUILD_NUM or build number for the system under test. Set the property when staging the test with
  - Test Runner with the –D option
  - invoke with the –0 option
  - DevTest Portal by selecting **Run with Options** and setting the BUILD_NUM property to the build number.

The default value is BUILD_NUM.
The property is saved in the default report and can be searched from the reporting portal, so you have a consistent search in CA Agile Central and in DevTest.

You use CA Agile Central to manage the test cases. The test cases show the build number and whether the tests passed or failed.

The test case ID is assumed to be the prefix of the DevTest test name. The prefix should be delimited by a hyphen (-), an underscore (_), or a space. An example is **TC0001-testName**.

The following video demonstrates how to integrate with CA Agile Central. The CA Agile Central report generator in the video includes a parameter named Test Case Locator, which is no longer available.

XML Report Generator

This report generator creates an XML file with all the possible data that can be captured. The captured data can be limited by using the report options in the Test Suite Editor or Staging Document Editor.

The data in this table can be used for any custom reporting needs.

After your tests, suites, or both are complete, you can view the report data in the DevTest Portal. To export the XML data to a file, see Testing Reports.

Changing Reporting Databases

You can set the way that you configure DevTest to connect to alternate databases in **lisa.properties** or **site.properties**. Set DevTest database components by assigning each component to a defined pool configuration like:

```
lisadb.reporting.poolName=common
lisadb.legacy.poolName=common
lisadb.acl.poolName=common
```

By default all of the DevTest databases share a common Derby database connection pool as defined here.

```
lisadb.pool.common.driverClass=org.apache.derby.jdbc.ClientDriver
lisadb.pool.common.url=jdbc:derby://localhost:1528/database/lisa.db;create=true
lisadb.pool.common.user=rpt
lisadb.pool.common.password=rpt
```

For example, to change the reporting database to connect to Oracle, first define a new database pool configuration.

```
lisadb.pool.mypool.driverClass=oracle.jdbc.OracleDriver
lisadb.pool.mypool.url=jdbc:oracle:thin:@//myhost:1521/orcl
lisadb.pool.mypool.user=rpt
lisadb.pool.mypool.password=rpt
```

Then set the reporting component to use that pool.

```
lisadb.reporting.poolName=mypool
```

When you enter the **Reporting Portal**, you can mouse over the database icon on the upper-right corner of the window to get the details about the database you are using.

Recorders and Test Generators

The DevTest Workstation no-code testing environment allows QA, Development, and others to rapidly design and execute functional, unit, regression, and load tests against dynamic websites (RIAs).

The product can be used to test rich browser and web user interfaces and the many building blocks and data residing below the UI. With DevTest, all the data and implementation layers the team needs to functionally test can be analyzed, invoked, and verified to ensure requirements are met.

More Information:

- Generate a Web Service
- Record a Website

Generate a Web Service

You can create a Web Service (XML) test case. Use the Web Service Execution (XML) 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. For more information about the Web Service Execution step, see Web Service Execution (XML) Step.

Ensure you are running the demo server to use this step.

Generating a web service includes the following steps:

- Create the Web Service (XML) Step
- Create the Web Service Client
- Execute the Test Case

Create the Web Service (XML) Step

Follow these steps:

1. Open the model editor in DevTest Workstation.
2. Right-click in the model editor window and select **Add Steps**, **Web/Web Services**, **Web Service Execution (XML)**.
   A Web Services step is added.
3. Rename the step **addUser** in the **Step Information** area.
4. Double-click the addUser step.
5. The Web Service Execution editor opens.
6. To create an XML document, click **New Document**.

Create the Web Service Client

Follow these steps:

1. Enter the location of the WSDL in the **WSDL URL** field.

   `http://WSSERVER:WSPORT/itko-examples/services/UserControlService?wsdl`

2. Enter **UserControlServiceService** in the **Service Name** field.
   Do not use spaces in the name of the web service.

3. Enter **UserControlServiceService** in the **Port** field.

4. Select the operation to be tested from the **Operation** list.

5. Select the action to be taken on test error: **Abort the Test** from the **On Error** list.

Execute the Test Case

Follow these steps:

1. Click **Execute** .
   The test executes and displays the request and response.

2. To view the request upon execution, click the **Request** tab.

3. To view the response upon execution, click the **Response** tab.

Record a Website

DevTest Workstation provides an HTTP recorder to test a website test case using a proxy recorder.

This helps track your path through the website and automatically creates test steps for each HTTP request that is generated while recording.

> **Note:** Before recording, disable caching or flush the cache in your browser so that the web recorder can get the client request parameter encoding from some server response packets for multibyte encoding systems. If the web recorder cannot capture the response packets, it decodes or encodes the parameter with ISO-8859-1 as the default and the captured parameters may get garbled.

To start the recording:

1. From within a test case, click **Create steps by recording or templating** .

2. Click **Record Test Case for User Interface**, **Web Recorder (HTTP Proxy)**.
   You can also select **Actions**, **Record Test Case for User Interface**, **Web Recorder (HTTP Proxy)** from the main menu.
   This lets you start the browser that is used to record and play back HTTP tests.

- If there is a test case already open in the active tab, you are asked whether you want to replay the tests in the browser.
- If there is no test case open in DevTest Workstation, the **Test Recorder** window opens, where you enter the name of the website to record.

3. Enter the URL for the web page to test.

> **Note:** Entering "localhost," for example: **http://localhost:8080/lisabank/**, does not work with HTTP Proxy recording. For the URL, use the host name or IP address instead of "localhost."

4. Select your preferences from the following:
   - **HTML Responses Only:** Captures only the HTML responses.
   - **Use External Browser:** Opens an external browser window.

Port Usage

By default, the DevTest Proxy recorder uses port 8010 for recording.

If you do not want to use this port, you can override the setting in the **lisa.properties** file with the following property: **lisa.editor.http.recorderPort=8010**.

To start the recording, click **Start Recording** to start the Web recorder, or click **Proxy Settings** to configure the proxy.

The following pages are available.

- Configure Proxy Settings
- Start Recording
- View Recorded Transactions
- View in ITR

Configure Proxy Settings

To open the **Proxy Setting** window, click **Proxy Settings** on the **Test Recorder** window. You can configure the following proxy settings:

- **Use Proxy**
  To use proxy settings, select this option. This option is useful when you want to enter the proxy settings and then you want to use the proxy intermittently.
- **Web Proxy Server**
  Enter the proxy server hostname (server IP) and respective port number.
- **Bypass Web Proxy for these hosts and domains**
  Enter the host name and domains of those servers for which to bypass proxy. Enter a pipe (|) separated list of hosts to be connected to directly and not through a proxy server. An asterisk * can be used as a wildcard character for matching; for example, "*.foo.com|localhost".
- **Secure Web Proxy Server**
  Enter the secure proxy settings.

- **Bypass Web Proxy Server for these hosts and domains**
  Enter the host name and domains for which you need to bypass proxy.

- **Exclude simple host names**
  Select to exclude simple host names (for example, **localhost** and **servername** compared to **192.168.1.1** or **server1.company.com**.)

- **Proxy server Authentication**
  Enter authentication details:

  - **Domain**

  - **User name**

  - **Password**

- **Send Preemptively**
  Select **Wait for Challenge**, **Send Basic,** or **Send NTLM**.

  > **Note:** Typically the proxy server challenges any request when authentication is required. If the proxy server does not send the challenge, setting this field forces the authentication header to be set with the first request.

Click **OK** to save your changes and set the proxy settings.

Start Recording

To start recording the test:

1. Click **Start Recording** in the Test Generator.
   The **Test Recorder** window opens up and displays the web page URL loaded.

   > **Note**: To record Unicode characters, use an external browser.

2. You can test the web page by entering information as a user would.

3. After you are done, click **Stop Recording** at the bottom of the **Test Recorder** window to stop recording your browser activity.
   The **Recorded Elements** window opens.

4. Click **Commit Edits** to complete your recording.

View Recorded Transactions

After you stop recording, all the recorded transactions are shown in the Recorded Elements tab.

All the transactions are listed in the left panel. The Step Details and the Response are seen in the right tab.

Follow these steps:

1. Select the **Response** tab to see the HTML response recorded.

2. Click **Commit Edits** to commit these transactions in the **Test Recorder**.
   The **Parameters In Web Recording** panel opens.

3. Enter any parameters that your test requires.

4. Click **Add to Test and Close** at the bottom of the **Test Recorder** window to add transactions as test steps.
   A test case is created based on your HTTP requests in the DevTest workflow. Each step in the test case represents a recorded HTTP request.

View in ITR

You can view all these transactions again when you run this test case in the ITR.

See the **View**, **Source**, and **DOM Tree** tabs to see more information about the recorded step.

Using BeanShell in DevTest

BeanShell (http://www.beanshell.org/) is a free, open source, lightweight Java scripting language. BeanShell is a Java application that uses the Reflection API to execute Java statements and expressions dynamically. By using BeanShell, you avoid the need to compile class files.

BeanShell lets you 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 a Java class, which is how it is used in the product.

BeanShell is used in several places:

- To interpret property expressions
- As the interpreter framework for the Java Script test step
- As the interpreter framework for the Assert by Script Execution assertion

Using BeanShell Scripting Language

The major difference between BeanShell Java and compiled Java is in the type system. Java is strongly typed, but BeanShell can loosen the typing in its scripting environment. You can, however, impose strict typing in BeanShell if you want.

BeanShell relaxes typing in a natural way that lets you write BeanShell scripts that look like standard Java method code. However, you can also write scripts that look more like a traditional scripting language, such as Perl or JavaScript, while maintaining the Java syntax framework.

If a variable has been typed, then BeanShell honors and checks the type. If a variable is not typed, BeanShell only signals an error if you try 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 lets you declare and then use methods. Arguments and return types can also be loosely typed:

- **Typed**
  int addTwoNumbers(int a, int b){
  return a + b;
  }

- **Loosely Typed**
  add (a,b){
  return a + b;
  }

In the second example, the following works correctly:

```
sumI = add (5,7);
sumS = add("DevTest " , "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()**, and so on: UNIX-like shell commands.
- **print():** Print argument as a string.
- **eval()**: Evaluate the string argument as code.

For more information, see the BeanShell User Guide at http://www.beanshell.org/.

You can also get BeanShell, the source code, and the complete Javadoc at the same place.

Using BeanShell as Stand-alone

BeanShell is available as a stand-alone interpreter so you can try it outside of DevTest. You can download BeanShell from www.beanshell.org. This download is a single small JAR file named 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 a Java class:**

  ```
  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 DevTest

The BeanShell interpreter is used in the Java Script Execution step and the Assert by Script Execution assertion. Both of these elements also expose DevTest Java objects and the current DevTest state (properties). This provides a powerful environment for adding custom functionality. The exposed Java objects can be used to both interrogate and to modify the current state of the test. For example, you can read, modify, and create DevTest properties in your scripts.

As a starting point, become familiar with the **TestExec** class in DevTest. Information about TestExec and many other classes can be found in Using the SDK.

DevTest also uses BeanShell inside property notation when an equal sign is present. For example:

```
{{= new Date()}}
```

This property expression is interpreted using BeanShell.

Using Date Utilities

The **com.itko.util.DateUtils** class includes a number of date utility functions as static methods. These functions all return the formatted date as a string. You can use these functions in parameter expressions or the Java Script 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 2 minutes slow, you can use:

```
=com.itko.util.DateUtils.formatCurrentDate(-120,"yyyy-MM-dd'T'HH:mm:ss.SSSZ")
```

This generates 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 the date that the default Java date formatter generates. If you need a strict RFC 3339 date, you can use the rcf3339 functions:

```
=com.itko.util.DateUtils.rfc3339()
```

This generates 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 samlDate functions are simply helpers so you do not need to remember that format string when using the formatDate APIs.

For more information, see:

http://download.oracle.com/javase/1.5.0/docs/api/java/text/SimpleDateFormat.html

http://tools.ietf.org/html/rfc3339#section-5.6

Class Loader Sandbox Example

The following Java class is a simple example of a class that cannot be run in multithreaded or multiuser 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 in DevTest and create a load test of ten users. If you do not use the class loader sandbox, you see the System.out.println phrases only one time, and the value of s would be incorrect. This result is because all users are running in one class loader. This specific class fails in those circumstances. Presuming that this is the proper function of the application, use support for the class loader sandbox to make this work properly.

When you create the Class Loader Sandbox Companion and DevTest stages your ten users, you see the text phrase in the code appear ten times. DevTest constructs ten separate class loaders and instantiates this class ten times; there are ten separate instances of the class variable "static string s." This capability lets 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 DevTest.
- The POJO has static members.
- You are testing with multiple virtual users.

Generating DDLs

Setting the following properties in **local.properties** creates DDLs for reporting and VSE:

- **eclipselink.ddl-generation=create-tables**
- **eclipselink.ddl-generation.output-mode=sql-script**
- **eclipselink.target-database=Oracle**

To create DDLs for Agent, Application Insight, and CVS, use the following commands:

- **java -jar LisaAgent.jar -ddl oracle** (generate the Oracle DDL for Application Insight)
- **java -jar LisaAgent.jar -ddl mysql** (generate the MySQL DDL for Application Insight)

Using CA Service Virtualization

CA Service Virtualization lets you virtualize software service behavior and model a virtual service to stand in for the actual service during development and testing.

Note: DevTest Portal is a web-based application that provides simpler access to the most commonly used workflows for DevTest products. DevTest Workstation is the main application for advanced users and contains the full range of functionality available within the DevTest products. For a quick summary of the functionality available in the portal, see DevTest Portal Functionality.

Service Virtualization Overview

When developing and testing applications, you often need access to one or more services to verify that the application is working as expected.

These dependent services can become a constraint. For example:

- The service is costly to use.
- The service is available only for limited timeframes.
- The service is needed by multiple teams.
- The service is still being developed.

Service virtualization lets you create a simulated version of a service. The simulated version is referred to as a *virtual service*.

The virtual service does not need to duplicate all the functionality of the actual service.

The potential benefits of service virtualization include faster delivery, higher quality, and cost savings.

The DevTest value proposition is built around the concept of allowing users to create virtual services that emulate the behavior of real systems. The level of emulated behavior is at the discretion of the user. To support this proposition, DevTest implements a variety of functions and features that allow users to execute OS level commands as well as the ability to execute scripted and compiled code. In fact, two key and differentiating features of the product, Magic Strings and Magic Dates, rely on scripting to accomplish their goals.

Exercise caution when consuming resources (assets) provided by third parties.

Product Configurations

CA Service Virtualization has two product configurations:

- **CA Service Virtualization**
  CA Service Virtualization is intended for use cases in development, integration, testing, and user acceptance. Instances of these products service up to ten parallel transactions simultaneously, or approximately ten transactions a second.

- **CA Service Virtualization for Performance**
  CA Service Virtualization for Performance is specifically for performance testing applications, and is more scalable, limited only by the underlying hardware and network.

The default configuration is CA Service Virtualization. For information about how to override the default, see SV Installation and Configuration .

More Information:

- Using DevTest Portal with CA Service Virtualization
- Using DevTest Workstation with CA Service Virtualization

SV Installation and Configuration

This section describes system requirements for CA Service Virtualization and how to install and configure the product.

- System Requirements for CA Service Virtualization
- Software Directory Structure and Files
- Install CA Service Virtualization
- Configure CA Service Virtualization

System Requirements for CA Service Virtualization

The following system resources for CA Service Virtualization are baseline requirements only:

- **CPU:** 2 GHz or faster

- **RAM:** 2 GB or more

- **Disk Space:** 5 GB of free space

- **OS:** Recommended: 64-Bit operating system. Supported: Windows 2008, 7, 8, Linux, Solaris, AIX 6.1 (LISA 5.0 and later)

For larger scale CA Service Virtualization deployments, we recommend the following resources:

- 256 Virtual Service Threads for each VSE instance

- 1 Processor Core and 2GB RAM for each VSE instance

Example: 1,000,000 transactions each day

- 1 thread for each service to support functional tests
- About 6 threads for each service to support virtualization for load and performance tests.
- Eight cores are equal to 2,048 concurrent virtual service threads.
- 16 GB RAM (for DevTest).

For more information about other system requirements, see System Requirements and Prerequisites.

Software Directory Structure and Files

The following list describes the DevTest Solutions directory structure. The directories are in the DevTest root installation folder.

- **bin**
  Contains executables, such as TestRegistry.exe, Workstation.exe, VirtualServiceEnvironment.exe, and VSEManager.exe.
- **DemoServer**
  Contains the DevTest demo server.
- **doc**
  Contains DevTest documentation.
- **examples/VServices**
  Contains examples relating to CA Service Virtualization.
- **tmp**
  Contains the VSE workspace where VSE temporarily stores conversations and stateless transactions.
- **vseDeploy**
  Contains deployed VSMs and related data.

Install CA Service Virtualization

The CA Service Virtualization software is installed with DevTest Server. For information about the installation and configuration of DevTest Solutions, see Installing.

To use CA Service Virtualization, the following processes (or services) must be running:

- Registry (DevTest Registry Service)
- VirtualServiceEnvironment (VSE Service)

As a server level service, CA Service Virtualization can coexist with a registry that has an attached coordinator and simulator. The simulator and coordinator are not mandatory to run CA Service Virtualization.

Configure CA Service Virtualization

To configure CA Service Virtualization properly, complete the following tasks:

1. Set up users and monitor usage.
   Your DevTest Solutions license is for the entire product. The license agreement provides for a given maximum number of concurrent users of the SV Power User user type (among other user types). An administrator grants CA Service Virtualization users various permissions that are associated with the SV Power User user type. Periodically, an administrator generates a Usage Audit Report to monitor compliance with the maximum concurrent usage. See Administering.

2. Set the proxy for localhost.

   When DevTest acts as the HTTP client for VSE, the HTTP traffic from DevTest must be passed to VSE. A common way to pass HTTP traffic is to set VSE as the web proxy. However, proxy use is disabled by default for simple names like "localhost".

   You can override this behavior in the **local.properties** file.

   Follow these steps:

   a. Open the **local.properties** file in the root DevTest installation folder.

   b. Uncomment the **lisa.http.webProxy.nonProxyHosts.excludeSimple** property.

   c. Set the value for this property to false.

   ```
   lisa.http.webProxy.nonProxyHosts.excludeSimple=false
   ```

   d. Save and close the file.

3. Define other settings in the **local.properties** file (optional).

   You can optionally set the following extra configurations in the **local.properties** file:

   - **lisa.vseName**=*VSENAME*
     To rename the VSE server, add this property and change the value where *VSENAME* is the name of the VSE server.

   - One of the following:
     lisa.registryName=*REGISTRY*
     or
     lisa.registryName=tcp://111.666.11.198:2010/REGISTRY

   To connect to another test registry:

   a. Add the **lisa.registryName** property to **local.properties**.

   b. Change the values of *REGISTRY* to the name of the new test registry.

Performance Testing with CA Service Virtualization

Please contact your CA Account team for more details about performance testing with CA Service Virtualization.

More Information

- Install the Database Simulator
- DDLs for Major Databases

Install the Database Simulator

To record JDBC traffic, install the DevTest simulation JDBC driver on the database client. The database client uses the DevTest driver instead of the actual driver.

Note: The JDBC (Driver Based) transport protocol is not supported. This functionality is replaced by agent-based JDBC virtualization.

Follow these steps:

1. Copy the JAR file **lisa-jdbc-sim-*x.y.z*.jar** (where ***x.y.z*** is the DevTest release level) in the LISA_HOME\lib\core directory to the classpath of your database.
   This JAR file contains the DevTest simulation JDBC driver. To help with the use of demo server as the database client, the driver is already copied into the demo server in the DEMO_HOME\jboss\server\default\lib directory.

2. Set the driver class to **com.itko.lisa.vse.jdbc.driver.Driver**. Set the driver class, depending on the style of JDBC that the database client uses:

   - **DriverManager Style**
     In an application server, it is not likely that the database client uses the Java DriverManager to acquire connections. If the application server does, add the following command 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 DevTest driver to the front. Separate the DevTest driver from the rest of the driver class names with a colon (:).

   - **DataSource Style**
     If the database client uses the DataSource style for acquiring a connection (as the Demo Server does), then update its configuration. Where the configuration specifies a data source definition, specify **com.itko.lisa.vse.jdbc.driver.Driver** as the JDBC driver to use, and a connection URL.

3. To achieve a pass through, modify the connection URL so that the DevTest Simulation JDBC driver can identify the real driver information. Format the connection URL as follows:

   ```
   name=value[;name=value...]
   ```

   - ***name***
     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 expects. (It must be defined as the last property so that it can contain semi-colons.)

> **Note:** VSE does not support the Oracle thin driver as a pass through driver. The Oracle thin driver does not provide the full JDBC implementation. If you use Oracle as a database, use other JDBC drivers for virtualization.

For an example of setting the connection URL, see **DEMO_HOME\jboss\server\default\deploy\itko-example-ds.xml**.

The following graphic shows an example of virtualizing the database in a WebMethods environment.

Example of a JDBC adapter configuration

To use both the simulation and Application Insight drivers, make the simulation driver the "outside" driver. Specify the Application Insight class and URL. See the **itko-example-ds.xml** file in the **DEMO_HOME\jboss\server\default\deploy** folder for an example that defines a DevTest data source to JBoss for DevTest demo server.

Other Startup Properties

Regardless of the connection style of the database client, you can add the following properties to the startup command for the database client to affect the simulation driver.

- **lisa.jdbc.sim.require.remote**
  Specifies whether the driver requires an active connection with a DevTest Workstation or VSE Server to run. This method is the best way to have a database client synchronize with VSE to record or play back any startup database activity that the server performs.
  Values:

  - **true:** The driver blocks until there is an active connection with a DevTest Workstation or VSE Server.

  - **false**: The driver does not require an active connection.

  Default: false

- **lisa.jdbc.sim.port**
  Defines the IP port on which the driver listens for connections from a recorder or a running virtual service model.
  Default: 2999

DDLs for Major Databases

To have DevTest generate a DDL for a file, add the following lines to the **local.properties** file:

```
eclipselink.ddl-generation=create-tables
eclipselink.ddl-generation.output-mode=sql-script
eclipselink.target-database=Oracle
```

Starting DevTest with these properties creates the following files that contain the necessary DDL:

- createDDL.jdbc
- dropDDL.jdbc

You can generate DDLs for a different DBMS by changing the target-database value.

More Information:

- EclipseLink Persistence Unit Properties for Session
- EclipseLink Persistence Unit Properties for Schema Generation

EclipseLink Properties for a Session

You can use the following EclipseLink JPA persistence unit properties in a **persistence.xml** file to configure EclipseLink extensions for a session, and as the target database and application server.

- **eclipselink.session-name**
  Defines the name by which the EclipseLink session is stored in the static session manager. Use this option if you must access the EclipseLink shared session outside of the context of the JPA. Use this option also to use a preexisting EclipseLink session that is configured through an EclipseLink **sessions.xml** file.
  Values: A valid EclipseLink session name that is unique in a server deployment.
  Default: EclipseLink-generated unique name.
  Example:

  ```
  persistence.xml file<property value="MySession"
  /> Example: property Mapimport org.eclipse.persistence.config.
  PersistenceUnitProperties;propertiesMap.put(PersistenceUnitProperties.
  SESSION_NAME, "MySession");
  ```

- **eclipselink.sessions-xml**
  Defines the persistence information that is loaded from the EclipseLink session configuration file, **sessions.xml**.
  You can use this option as an alternative to annotations and deployment XML. If you specify this property, EclipseLink overrides all class annotation and the object relational mapping from the persistence.xml, and ORM.xml and other mapping files.
  To indicate the session, set the **eclipselink.session-name** property.

> **Note:** If you do not specify the value for this property, **sessions.xml** file is not used.

Values: The resource name of the sessions XML file.
Example:

```
persistence.xml file<property value="mysession.xml"
/> Example: property Mapimport org.eclipse.persistence.config.
PersistenceUnitProperties;propertiesMap.put(PersistenceUnitProperties.
SESSIONS_XML, "mysession.xml");
```

- **eclipselink.session-event-listener**
  Defines a descriptor event listener to be added during bootstrapping.
  Values: Qualified class name for a class that implements the org.eclipse.persistence.sessions.SessionEventListener interface.
  Example:

```
Persistence.xml file<property value="mypackage.MyClass.class"
/> Example: property Mapimport org.eclipse.persistence.config.
PersistenceUnitProperties;propertiesMap.put(PersistenceUnitProperties.
SESSION_EVENT_LISTENER_CLASS, "mypackage.MyClass.class");
```

- **eclipselink.session.include.descriptor.queries**
  Specifies whether to copy all named queries from the descriptors to the session by default. These queries include the ones that are defined using EclipseLink API, descriptor amendment methods, and others.
  Values:

  - **true:** Enable the default copying of all named queries from the descriptors to the session.
  - **false:** Disable the default copying of all named queries from the descriptors to the session.

  Default: true.
  Example:

```
Persistence.xml file<property value="false"/>Example: property Mapimport org.
eclipse.persistence.config.PersistenceUnitProperties;propertiesMap.put
(PersistenceUnitProperties.INCLUDE_DESCRIPTOR_QUERIES, "false");
```

- **eclipselink.target-database**
  Specifies the type of database that your JPA application uses.
  Values:
  The following values are valid for use in a **persistence.xml** file and for the **org.eclipse.persistence.config.TargetDatabase**:

  - **Attunity:** Configure the persistence provider to use an Attunity database.
  - **Auto:** EclipseLink accesses the database and uses the metadata that JDBC provides to determine the target database. This value applies to the JDBC drivers that support this metadata.
  - **Cloudscape:** Configure the persistence provider to use a Cloudscape database.
  - **Database:** If your target database is not listed here and your JDBC driver does not support the use of metadata that the Auto option requires, configure the persistence provider to use a generic choice.
  - **DB2:** Configure the persistence provider to use a DB2 database.
  - **DB2Mainframe:** Configure the persistence provider to use a DB2 mainframe database.

- **DBase:** Configure the persistence provider to use a DBase database.
- **Derby:** Configure the persistence provider to use a Derby database.
- **HSQL:** Configure the persistence provider to use an HSQL database.
- **Informix:** Configure the persistence provider to use an Informix database.
- **JavaDB:** Configure the persistence provider to use a Java DB database.
- **MySQL:** Configure the persistence provider to use a MySQL database.
- **Oracle:** Configure the persistence provider to use an Oracle Database.
- **PointBase:** Configure the persistence provider to use a PointBase database.
- **PostgreSQL:** Configure the persistence provider to use a PostgreSQL database.
- **SQLAnywhere:** Configure the persistence provider to use an SQLAnywhere database.
- **SQLServer:** Configure the persistence provider to use an SQLServer database.
- **Sybase:** Configure the persistence provider to use a Sybase database.
- **TimesTen:** Configure the persistence provider to use a TimesTen database. You can also set the value to the fully qualified classname of a subclass of the org.eclipse.persistence.platform.DatabasePlatform class.

Default: Auto
Example:

```
Persistence.xml file<property value="Oracle"/>Example: property Mapimport org.
eclipse.persistence.config.TargetDatabase;import org.eclipse.persistence.
config.PersistenceUnitProperties;propertiesMap.put(PersistenceUnitProperties.
TARGET_DATABASE, TargetDatabase.Oracle);
```

- **eclipselink.target-server**
  Specifies the type of application server that your JPA application uses.
  Values:
  The following values are valid for use in the **persistence.xml** file and the **org.eclipse.persistence.config.TargetServer**:

  - **None:** Configure the persistence provider to use no application server.
  - **WebLogic:** Configure the persistence provider to use Oracle WebLogic Server. This server sets this property automatically. Set it only if it is disabled.
  - **WebLogic_9:** Configure the persistence provider to use Oracle WebLogic Server version 9.
  - **WebLogic_10:** Configure the persistence provider to use Oracle WebLogic Server version 10.
  - **OC4J:** Configure the persistence provider to use OC4J.
  - **SunAS9:** Configure the persistence provider to use Sun Application Server version 9. This server sets this property automatically. Set it only if it is disabled.
  - **WebSphere:** Configure the persistence provider to use WebSphere Application Server.

- **WebSphere_6_1:** Configure the persistence provider to use WebSphere Application Server version 6.1.
- **JBoss:** Configure the persistence provider to use JBoss Application Server.
- The fully qualified class name of a custom server class that implements the **org.eclipse.persistence.platform.ServerPlatform** interface.

Default: None
Example:

```
persistence.xml file<property value="OC4J_10_1_3"
/>Example: property Mapimport org.eclipse.persistence.config.TargetServer;
import org.eclipse.persistence.config.PersistenceUnitProperties;propertiesMap.
put(PersistenceUnitProperties.TARGET_SERVER, TargetServer.OC4J_10_1_3);
```

EclipseLink Properties for Schema

You can define the EclipseLink JPA persistence unit properties in a persistence.xml file to configure schema generation.

- **eclipselink.ddl-generation**
  Specifies the Data Definition Language (DDL) generation action to use for your JPA entities. To specify the DDL generation target, see **eclipselink.ddl-generation.output-mode**.
  Values:
  You can use the following values in a persistence.xml file:
  - **none:** EclipseLink does not generate a DDL; no schema is generated.
  - **create-tables**: EclipseLink tries to execute a CREATE TABLE SQL command for each table. When you issue a CREATE TABLE SQL command for an existing table, EclipseLink follows the default behavior of your specific database and JDBC driver combination. In most cases, an exception is thrown, the table is not created, and EclipseLink continues with the next statement.
  - **drop-and-create-tables** - EclipseLink tries to DROP all tables, then CREATE all tables. If EclipseLink encounters issues, it follows the default behavior of your specific database and JDBC driver combination. EclipseLink then continues with the next statement.

  The following values are valid for the **org.eclipse.persistence.config.PersistenceUnitProperties**:
  - **NONE**
  - **CREATE_ONLY**
  - **DROP_AND_CREATE**

  If you use persistence in a Java SE environment and you want to create the DDL files without creating tables, define a Java system property **INTERACT_WITH_DB** and set the value to **false**.
  Default: One of the following:
  - None
  - PersistenceUnitProperties.NONE

  Example:

```
persistence.xml file<property value="create-tables"
/>Example: property Mapimport org.eclipse.persistence.config.
```

```
PersistenceUnitProperties;propertiesMap.put(PersistenceUnitProperties.
DDL_GENERATION, PersistenceUnitProperties.CREATE_ONLY);
```

- **eclipselink.application-location**
  Specifies where EclipseLink writes generated DDL files. Files are written if you set **eclipselink.ddl-generation** to anything other than **none**.
  Value: A file specification to a directory in which you have write access. The file specification can be relative to your current working directory or absolute. If it does not end in a file separator, EclipseLink appends one that is valid for your operating system.
  Default: One of the following:
  "."+File.separator
  or
  <tt>PersistenceUnitProperties.DEFAULT_APP_LOCATION</tt>
  Example:

  ```
  persistence.xml file<property value="C:\ddl\"/>Example: property Mapimport org.
  eclipse.persistence.config.PersistenceUnitProperties;propertiesMap.put
  (PersistenceUnitProperties.APP_LOCATION, "C:\ddl\");
  ```

- **eclipselink.create-ddl-jdbc-file-name**
  Specifies the file name of the DDL file that EclipseLink generates containing SQL statements to create tables for JPA entities. This file is written to the location specified by **eclipselink.application-location** when **eclipselink.ddl-generation** is set to create-tables or drop-and-create-tables.
  Values: A file name valid for your operating system. Optionally, if the concatenation of **eclipselink.application-location** with **eclipselink.create-ddl-jdbc-file-name** is a valid file specification for your operating system, you can prefix the file name with a file path.
  Default: One of the following:

  - createDDL.jdbc

  or

  - PersistenceUnitProperties.DEFAULT_CREATE_JDBC_FILE_NAME

  Example:

  ```
  persistence.xml file<property value="create.sql"
  /> Example: property Mapimport org.eclipse.persistence.config.
  PersistenceUnitProperties;propertiesMap.put(PersistenceUnitProperties.
  CREATE_JDBC_DDL_FILE, "create.sql");
  ```

- **eclipselink.drop-ddl-jdbc-file-name**
  Specifies the file name of the DDL file that EclipseLink generates containing the SQL statements to drop tables for JPA entities. This file is written to the location specified by **eclipselink.application-location** when **eclipselink.ddl-generation** is set to **drop-and-create-tables**.
  Values: A file name valid for your operating system. Optionally, if the concatenation of eclipselink.application-location with eclipselink.drop-ddl-jdbc-file-name is a valid file specification for your operating system, you can prefix the file name with a file path.
  Default: One of the following:

  - dropDDL.jdbc

  or

  - PersistenceUnitProperties.DEFAULT_DROP_JDBC_FILE_NAME

  Example:

```
persistence.xml file<property value="drop.sql"
/>Example: property Mapimport org.eclipse.persistence.config.
PersistenceUnitProperties;propertiesMap.put(PersistenceUnitProperties.
DROP_JDBC_DDL_FILE, "drop.sql");
```

- **eclipselink.ddl-generation.output-mode**
  Defines the DDL generation target.
  Values:
  The valid values for the use in the **persistence.xml** file are:

  - **both:**

Generate the SQL files and execute them on the database.
If **eclipselink.ddl-generation** is set to "create-tables," **eclipselink.create-ddl-jdbc-file-name** is written to **eclipselink.application-location** and then executes on the database.
If **eclipselink.ddl-generation** is set to "drop-and-create-tables," both **eclipselink.create-ddl-jdbc-file-name** and **eclipselink.drop-ddl-jdbc-file-name** are written to **eclipselink.application-location** . Both SQL files execute on the database.

- - **database:**

Execute SQL on the database only (do not generate the SQL files).

- - **sql-script:**

Generate the 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 written to **eclipselink.application-location**. The command does not execute on the database.
If **eclipselink.ddl-generation** is set to "drop-and-create-tables," both **eclipselink.create-ddl-jdbc-file-name** and **eclipselink.drop-ddl-jdbc-file-name** are written to **eclipselink.application-location** . Neither is executed on the database. The following values are valid for the **org.eclipse.persistence.config.PersistenceUnitProperties**:

- - - DDL_BOTH_GENERATION - see **both**
  - - DDL_DATABASE_GENERATION - see **database**
    - DDL_SQL_SCRIPT_GENERATION - see **sql-script**

  Default: The default for Container or Java EE mode is {{database}},
  Example:

```
persistence.xml file<property value="database"
/>Example: property Mapimport org.eclipse.persistence.config.
PersistenceUnitProperties;propertiesMap.put(PersistenceUnitProperties.
DDL_GENERATION_MODE, PersistenceUnitProperties.DDL_DATABASE_GENERATION);
```

> **Note:** Override this setting by containers with specific EclipseLink support. See your container documentation for details. Bootstrap or Java SE mode: **both** or **PersistenceUnitProperties.DDL_BOTH_GENERATION**.

How To Run Multiple VSE Services on a Single Server

You can run multiple VSE services on a single server to restrict user access to virtual services.

There is a limitation on how many VSEs can be created, based on available hardware on the server computer, For the VSE to run, the minimum hardware that is required is 2 CPUs and 4-GB RAM and it supports up to 250 TPS. Based on the available hardware and your TPS requirements, determine the amount of hardware that should be allocated to each instance of VSE and the total number of instances of VSEs you can run on the server computer.

Follow these steps:

1. Copy **VirtualServiceEnvironmentService.exe** to **VirtualServiceEnvironmentService2**.exe. This file is in the LISA_HOME\bin directory (for example, C:\Programs Files\CA\DevTest).

2. Create a new vmoptions file named **VirtualServiceEnvironmentService2.vmoptions** and put the following properties in the file.
   -Dlisa.vseName=name of the new VSE, to distinguish it from others on the same computer.
   -Dlisa.net.8.port=port for the new VSE; must use a different port than 2013.
   -DLISA_LOG=vse2.log (to distinguish between the VSE log files).

3. Add the **VirtualServiceEnvironmentService2.vmoptions** file to the LISA_HOME\bin directory.

4. Repeat for each additional VSE service.

5. To define the service, use the following command.

   ```
   sc create "DevTest VSE2" binpath= "D:
   \CA\DevTest\$LISA_HOME\bin\VirtualServiceEnvironmentService2.exe"
   ```

6. Stop and start the DevTest components.

CA Service Virtualization Concepts

This section describes the main concepts of CA Service Virtualization.

- Basic Workflow
- CA Service Virtualization Components

Basic Workflow

The basic workflow consists of the following steps:

1. Gather transactions from sources such as live traffic recording, request/response pairs, and engineering specifications.

2. Use the transactions to create a virtual service.

3. Deploy the virtual service to a virtual service environment (VSE).

4. Confirm that the virtual service is working correctly.

5. Use the virtual service as a stand-in for the actual service.

6. (Optional) Modify or recreate the virtual service.

CA Service Virtualization Components

The following graphic shows how the components in CA Service Virtualization relate to each other from the perspective of DevTest Workstation.

The Virtual Service Image Recorder creates a virtual service model (VSM) and a service image.

The Service Image Editor and the VSM Editor let you view and edit them.

During virtualization, the VSM and service image load and run on a VSE that runs as a service.

The registry tracks one or more VSEs that are running, and DevTest Workstation uses it to connect to the VSE.

DevTest Portal is used to monitor and control the VSMs and service images that are loaded onto the VSEs.

More Information:

- Live Traffic Recording
- Request/Response Pairs
- Engineering Specifications
- Virtual Service Models
- Service Images
- How Virtualization Works
- VSE Transactions
- Match Tolerance
- Magic Strings and Dates

- Parsing URIs in HTTP Requests

Live Traffic Recording

One way to gather transactions for virtual service creation is to record the traffic between a client and a server.

The process differs for non-messaging systems and messaging systems.

- Non-Messaging Systems
- Messaging Systems

Non-Messaging Systems

The following graphics show that when recording the service image, the virtual service environment (VSE) acts as the pass through mechanism between the client and server. While the VSE passes the requests and responses along, it records the transactions.

Normal Operation

Recording

At the time of virtualization, the VSE responds to the client requests by consulting the recorded transactions.

Messaging Systems

Message-oriented middleware (MOM), or 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 in the MOM.

The types of message destinations are:

- **Queues**
  A publisher adds a message to the queue, and a subscriber pulls messages from the queue, in a "first in, first out" fashion.

- **Topics**
  A publisher publishes a message to a topic, and all subscribers that subscribe to the topic receive the message.

The following graphic shows a simple message-based service. In this scenario, the client adds messages to a queue (**ORDERS.REQUEST**), which the server picks up. The response from the server is in the form of messages added to another queue (**ORDERS.RESPONSE**). The client then picks them up.

Possible variations include:

- Using topics instead of queues

- Multiple responses to a single request, which can possibly target different destinations

CA Service Virtualization aims to virtualize the server. In the recording mode, VSE requires extra proxy destinations (**requestProxy** and **responseProxy** queues in the following graphic) that the client uses instead of their counterparts. The server still listens and posts to the real destinations. VSE acts as a pass through between these proxy and real destinations. VSE records the traffic to create the VSM and the service image that it needs for the virtualization.

Later, when VSE virtualizes the server, it works with the real destinations. VSE does not need the proxy destinations.

Request/Response Pairs

One way to gather transactions for virtual service creation is to provide request/response pairs.

> **Note:** Request/response pairs are also known as RR pairs or R/R pairs.

A request/response pair consists of one request file and one or more response files.

Multiple response files are applicable to messaging scenarios.

- Naming Conventions
- Examples
- Sidecar Files

Naming Conventions

The file names in a request/response pair consist of a prefix, a suffix, and a file extension.

```
<prefix><suffix>.<extension>
```

The prefix is a unique identifier, such as **depositMoney**. You must use the same prefix in all the file names that correspond to a particular transaction.

The suffix of a request file is **-req**.

The suffix of a response file is **-rsp**. If you have multiple responses, add a number to the end of each response suffix.

The file extension can be anything.

The following example has one request and one response:

- **depositMoney-req.txt**
- **depositMoney-rsp.txt**

The following example has one request and three responses:

- **depositMoney-req.txt**
- **depositMoney-rsp1.txt**
- **depositMoney-rsp2.txt**
- **depositMoney-rsp3.txt**

Examples

The Bank v5 and Bank v6 projects in the **LISA_HOME\Projects** directory include examples for the HTTP protocol.

Sidecar Files

A sidecar file contains key/value pairs that are added to the metadata of requests, responses, or both in a virtual service.

When creating a virtual service from request/response pairs, you can specify one or more sidecar files.

A sidecar file is global or specific:

- Global
  Applies to all request or response files.

- Specific
  Applies to an individual request or response file.

The entries in a specific sidecar file take precedence over the entries in a global sidecar file.

A global sidecar file that applies to requests must be named **meta-req.properties**.

A global sidecar file that applies to responses must be named **meta-rsp.properties**.

To name a specific sidecar file, add the string **-meta** to the name of the corresponding request or response file and change the file extension to **.properties**.

The following example has three sidecar files. The **meta-req.properties** and **meta-rsp.properties** files are global. The **depositMoney-rsp2-meta.properties** file is specific.

- **depositMoney-req.txt**

- **depositMoney-rsp1.txt**

- **depositMoney-rsp2.txt**

- **depositMoney-rsp2-meta.properties**

- **depositMoney-rsp3.txt**

- **meta-req.properties**

- **meta-rsp.properties**

The following example shows the key/value pairs that might appear in a sidecar file:

```
Accept-Language=en-US
Content-Type=application/json
```

Engineering Specifications

One way to gather transactions for virtual service creation is to use an engineering specification that describes the service. The engineering specification is sometimes referred to as a *contract*.

For example, assume that you want to virtualize a service that is in the early stages of development. You might not be able to create the virtual service by recording. However, if you have access to a Swagger document that describes the service, you can use this approach.

The following types of specifications are supported:

- **Swagger**
  Supported in DevTest Portal and DevTest Workstation.

- **WSDL**
  Supported in DevTest Portal and DevTest Workstation.

- **WADL**
  Supported in DevTest Workstation.

- **RAML**
  Supported in DevTest Workstation.

Virtual Service Models

A *virtual service model* (VSM) receives service requests and responds to them in the absence of the actual service provider.

You can conceptualize a VSM as a series of steps to be executed when a request is received. The steps create and pass back a response to the request.

A VSM must contain at least one VSE step from the Virtual Service Environment step list in DevTest Workstation to be deployable to a VSE.

When you create a service image, the VSE automatically generates the protocol-specific steps in the VSM. You can modify any of the generated steps from DevTest Workstation. You can also:

- Populate the responses from a Microsoft Excel spreadsheet or by cross-referencing a database table and doing calculations on inputs

- Add some steps of different step types

- Manipulate the request and response steps before proceeding

- Specify the service image to use from the Response Selection step

The VSM defines how behavior patterns get used and queries the service image to determine how to respond. The VSM knows how to navigate the service image.

A VSM must be deployed to the VSE.

Service Images

A *service image* is a recording of the interaction between the client and server as created by CA Service Virtualization. A virtual service model references a service image. The service image is used to deliver the appropriate response to the client in the absence of the server.

- Service Image Contents
- Imported Transactions

Service Image Contents

A service image contains the following sets of information:

- A list of conversations (requests and responses) recorded as a conversation tree

- A list of stateless transactions (requests and responses)

- The responses to send when unknown conversational or stateless requests are encountered

You can visualize a conversation as a series of stateful transactions. However, multiple conversations (from multiple sessions) can be recorded in the same service image. Similar request structures are merged into a single transaction to create a tree, as the following graphic shows.

For example, if multiple users log in to the system with **login()** transactions, all these transactions are merged into a single transaction. But if one user logs in with **login()** and another user logs in with **acquireAuthToken()**, the transactions are not merged.

Imported Transactions

You can import the following XML documents into a service image being recorded:

- **Raw Transactions**
  The XML document represents raw traffic as if coming directly from the network, characterized by a root element of **<rawTraffic>**.
  For an example, see **LISA_HOME\examples\VServices\raw-traffic.xml**.

- **Conversational Traffic**
  The XML document represents traffic that is rearranged into conversations, stateless transaction sets, or both. The traffic is organized into linear lists of transactions, each of which represent a "real" conversation and has a root element of **<traffic>**.
  For an example, see **LISA_HOME\examples\VServices\traffic.xml**.

How Virtualization Works

In the absence of a server, CA Service Virtualization simulates the behavior of the server for its client. The process requires loading the service image that a virtual service model references and running it in the VSE dashboard.

When the virtual service environment (VSE) receives a request, the VSE examines the request and tries to match it to an existing conversational state (session) in the VSE. For example, a cookie ID or some other session identifier can "tie" requests in stateless protocols such as HTTP. The VSE uses the conversational state to determine where in the conversation tree the "current transaction" is, and any other "state" such as a previously submitted authentication token, which might not be part of the current request, but is used in the subsequent response. The user name is an example.

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 it finds a match, it creates a session and the session returns a relevant response. The session is maintained until two minutes after it has last been "seen" by the VSE. You can change this behavior with the **lisa.vse.session.timeout.ms** property. If no conversation starters match, no session is created and the list of stateless transactions is consulted in the order in which 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.

When the VSE finds a previous session and the VSE is in a conversation, the next transaction match depends on many factors. The factors can include navigation tolerance and match tolerance.

If the VSE does not find a matching transaction in the conversational and stateless transactions, it consults the service image again for the type of response to send for an unmatched conversational or stateless request.

VSE Transactions

In CA Service Virtualization, a *transaction* consists of the request that the client sends to the service and the one or more responses that the service sends to the client.

The following graphic shows a transaction that has one request and one response:

A transaction typically has only one response. However, some protocols can return multiple responses from a single request.

- Request
- Response
- VSE Transaction Example
- Synchronous and Asynchronous

Request

The request side of a transaction has the following components:

- **Operation**
  A categorization of similar transactions.

- **Arguments**
  Name/value pairs that are associated with an operation.
  An argument can represent information that the service needs to fulfill the request. For example, the service might need a user name or the amount of money to deposit.

- **Body**
  The payload of the request.

- **Attributes**
  Name/value pairs that contain information about the request.

- **Metadata**
  Name/value pairs that can impact how requests are handled during playback.
  For example, a messaging service might need the name of the queue to which to send the response.

- **Match tolerance**
  A setting that defines the criteria for matching against an inbound request at runtime.

Response

The response side of a transaction has the following components:

- **Body**
  The payload of the response.

- **Metadata**
  Name/value pairs that can impact how responses are handled during playback.

- **Think time**
  A setting that indicates how long to wait before sending the response.

VSE Transaction Example

Assume that an HTTP-based virtual service is created for a service that provides weather information. Let's examine one possible transaction for this virtual service.

The transaction has an operation with the name **GET /api/weather**.

For each request, the service needs the city and the country. Therefore, the transaction has two arguments:

Name	Value
city	Dallas
country	US

The operation and arguments are derived from the request body that the virtual service environment received during the recording of the virtual service:

```
GET http://api.example.com/api/weather?city=Dallas&country=US
```

During playback, the virtual service can receive requests that have different values. For example:

```
GET http://api.example.com/api/weather?city=Seattle&country=US
GET http://api.example.com/api/weather?city=London&country=UK
```

The transaction has no request attributes.

The request metadata contains a number of HTTP-related items. For example:

Name	Value
Host	api.example.com

Name	Value
HTTP-Method	GET
HTTP-Version	HTTP/1.1

The match tolerance is set to Signature.

The response body contains the weather information that was requested:

```
{
    "temp": 88,
    "temp_max": 90,
    "temp_min": 78,
    "wind": 4.1
}
```

The response metadata contains a number of HTTP-related items. For example:

Name	Value
HTTP-Response-Code	200
HTTP-Response-Code-Text	OK
HTTP-Version	HTTP/1.1

The think time is set to 174 milliseconds. This value is based on the behavior that was observed during the recording of the virtual service.

Synchronous and Asynchronous

With most service protocols, including web services, HTTP, and Java, transactions happen synchronously. The request and response are directly related to each other.

With web services and HTTP, the request and response are contained in a single socket connection, and the response typically follows the request. With Java, the request and response are contained in a single thread. The response (the return value) always follows the request (a method call).

Messaging is different because it is asynchronous. The request and response messages do not have to occur in the same socket connection, the same thread, or even the same day. The client sends a request message and continues working while waiting for the response. When the response is sent from the service, the client receives it and matches it to the original request, completing the transaction.

Stateless and Conversational VSE Transactions

VSE transactions are classified as *stateless* or *conversational*.

- Stateless Transactions
- Conversational Transactions
- Conversation Starters

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. Each stateless transaction is independent of the others. For example:

- What is the current weather in Dallas? (Operation: weather; Argument: city)
- What is the current weather in New York? (Operation: weather; Argument: city)
- What will the weather be in Dallas tomorrow? (Operation: weather; Argument: city; Argument: date)

Conversational Transactions

Conversational transactions are *stateful*. Conversations consist of:

- The logical transaction that, if matched, starts a unique session
- The information necessary to create and identify that session

A transaction in a conversation depends on the context that earlier conversations create.

For example, the following conversation involves using an ATM:

1. Connect to the ATM. (Operation: logon)
2. Which account do you want? (Argument: checking)
3. What is my balance? (Operation: balance)
4. Response.
5. Withdraw an amount of money. (Operation: withdraw)
6. How much? (Argument: amount)
7. What is my balance? (Operation: balance)
8. Response.
9. End session. (Operation: logout)

Conversation Starters

The first transaction in a conversation is the *conversation starter*. When a virtual service environment (VSE) receives an incoming request, the VSE reviews the conversation starters to determine whether the transaction means a new conversation is starting.

Because all the transactions in a conversation are related to each other, the VSE must have a way of determining this relationship. This determination is typically made by using some kind of special token such as "jsessionid" or a cookie in web transactions, or a custom identifier in message traffic.

The following types of conversations are supported:

- **Instance-based conversations**
  The protocol layer is responsible for identifying the unique string, which is based on different instances of the client. The string identifies server-side sessions for both recording and playback of the service image.

- **Token-based conversations**
  The VSE generates the token by using a string generation pattern that is stored with the conversation in the service image. After the token is generated, it operates the same as an instance-based conversation. Token-based conversations cannot be automatically inferred during recording. To specify where tokens are found, use the VS Image Recorder.

Navigation Tolerance

Each node in a conversation tree has a *navigation tolerance*. The navigation tolerance plays an important role in the process of finding a match for an incoming request.

When a virtual service that contains at least one conversation receives an incoming request, the virtual service tries to find a match for the request in a conversation.

The starting point for the search is the *current transaction*.

- If a transaction was previously matched in the conversation, that transaction is the current transaction.
- If a transaction was not previously matched in the conversation, the virtual service examines the conversation starter (the top-level node) to see if the incoming request is starting a session. If the result is true, then the conversation starter is the current transaction.

To figure out where in the conversation to search for a match, the virtual service locates the current transaction and checks the navigation tolerance.

The navigation tolerances are:

- **Close**
  The virtual service searches the immediate children of the current transaction.
- **Wide**
  The virtual service searches in the following order:
  1. Immediate children of the current transaction
  2. Immediate children of the siblings of the current transaction ("nephews")
  3. Siblings of the current transaction, or the current transaction itself
  4. The conversation starter of any conversation in the virtual service
- **Loose**
  The virtual service searches in the following order:
  1. Immediate children of the current transaction
  2. Immediate children of the siblings of the current transaction ("nephews")
  3. Siblings of the current transaction, or the current transaction itself
  4. Siblings of the parent of the current transaction ("uncles"), or the parent of the current transaction
  5. Any transaction in the second level of the tree, but only if the current transaction is in the fifth level of the tree or lower

6. The conversation starter of any conversation in the virtual service

The following graphic shows how the navigation tolerance affects the transactions to be searched in a conversation tree. The conversation starter is blue and appears at the top. The current transaction is green and has a check mark. Notice that the current transaction is in the fourth level of the tree.

Some of the procedures for creating a virtual service let you configure the default navigation tolerance for the following categories of transactions:

- All transactions except the last (leaf) transactions
- Last (leaf) transactions

After a virtual service is created, you can change the navigation tolerance for each node from the service image editor in DevTest Workstation.

Unknown Requests

An unknown request occurs in the following situations:

- When there is an active conversation

- When there is not an active conversation
  If there is not an 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 response for unknown stateless requests becomes the reply.

If a request cannot be matched to a follow-on transaction:

- If the navigation tolerance is not Close, the conversation starter transactions are given the chance to satisfy the request.
- If the request is still unmatched and a stateless transaction can produce a reply, then that is sent. The current session continues to remember where it is in its conversational tree.

If that fails, the service image response for unknown conversational requests becomes the reply.

Logical Transactions

A *logical transaction* appears as a single node in a conversation. A logical transaction consists of one Meta transaction with one or more specific transactions.

When a logical transaction appears in the search for a specific request, the Meta transaction is consulted. The Meta transaction determines if this transaction can respond to the request. If the Meta transaction responds to the request, all the specific transactions are asked if they can respond to the request. If none of the specific transactions can respond to the request, the response is taken from the Meta transaction.

This logic can be expressed in 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 (specific transaction matches the given request) {
      return response from the specific transaction
    }
  }

  // No specific transaction found for the given request
  return response from the meta transaction
  }

}
```

Further, a physical Meta transaction carries a list of specific transactions and a list of child Meta transactions. The list of child Meta transactions allows Meta transactions to be structured as a decision tree.

Match Tolerance

Match tolerance defines how the virtual service environment (VSE) decides whether a specific transaction matches the incoming transaction.

- Argument Match Operators
- Meta Transactions and Specific Responses
- How the VSE Selects the Next Response
- How to Debug Match Failures

The levels of match tolerance are:

- **Operation**
  The loosest match tolerance. The operation name of the incoming transaction must match the name of the recorded transaction.

- **Signature**
  The operation name must match and the names of the arguments must match exactly, with no additions or deletions. The order of arguments does not have to be the same.

- **Exact**
  In addition to the signature match, the values for each argument must match the values that were recorded, as defined by the argument match operators.

Argument Match Operators

Every argument in a request has a *match operator*. For an Exact match operation, the operator 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 available match operators are:

- **Anything**
  Always returns true. The virtual service recorder defaults the comparison to **Anything** when it determines that an argument is a date. The incoming value for this argument can be anything. The argument must be present for the signature/META match to work, but the value is ignored and can be blank or null.

- **= 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, 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 must be in the form of a double-braced script expression, {{ }}. This property or expression is evaluated. If the result starts with Y, y, T, t, or ON, then the argument is deemed to have matched. 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 converted to a date before the comparison is done.

Meta Transactions and Specific Responses

When the VSE searches for a conversational match, it only searches Meta transactions. A Meta transaction cannot have a match tolerance of **Exact** (the default is **Signature**). Each Meta transaction has one or more specific responses, which could have any match tolerance (**Exact** is the default).

If none of the specific responses for a Meta transaction match, the response that is specified for the Meta transaction is used.

How the VSE Selects the Next Response

1. If the incoming request is in a conversation, search for a match that is based on the navigation tolerance and other rules explained previously.
2. If nothing is found in the current conversation, look for another conversation (unless navigation tolerance is CLOSE).
3. If there is a conversational match, with a specific response (instead of a Meta match), use the specific response.
4. Otherwise, look for a specific match in the stateless transactions, and if found, use that one.
5. If there was no specific match in the stateless list, but there is a Meta match in the conversational list, use that Meta match.
6. If there was no conversational match at all, but there is a Meta match in the stateless list, use the Meta match.
7. Fail with "no match found," and send back either the "unknown conversational response" or the "unknown stateless response" specified in the service image. The response could be overridden based on the protocol and handlers used.

How to Debug Match Failures

If you get failures to match, open the **LISA_HOME\logging.properties** file and set **log4j.logger.VSE** to **DEBUG** or **TRACE**. This setting causes the VSE to be verbose to the **vse_*xxx*.log** file, where *xxx* is the service image name. The **log4j.logger.VSE** property also tells you exactly what matched and what did not match.

Set the **log4j.logger.VSE** property to **INFO** or **WARN** for production use. Do not leave it set to **DEBUG** or **TRACE** longer than necessary.

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

- Magic Dates

Magic Strings

During the recording phase, VSE examines each request for arguments or parameters. For example, a weather forecasting web service call could include a city name or an airline booking system request could include a flight number. If the flight number is included in the recorded response, it is classified as a *magic string*.

For example, if 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 following example, we recorded a request to set some "state" in the conversation, in this case the flight number (ABC53). The recorder recognized that the string ABC53 was also in the response, so it converted ABC53 to a magic string. The magic string is saved in the service image database as '=request_arg1;/ABC53/.

Example of a magic string in Virtual Service Environment

The {{ }} notation is a standard VSE property substitution syntax. This notation means "substitute the {{ }} text with the runtime value of the first argument of the request." If there is no first argument, use ABC53 as the default.

The practical significance is that if a future client requests flight ZZ99, the virtualized service responds with the correct flight number, ZZ99.

VSE detects magic strings not only in a simple request/response pair. If an argument is ever seen in any subsequent response in a conversation, VSE deems it to be magic and stores the argument value in the conversation state.

By default, VSE does not consider anything in an XML tag for magic strings. This exclusion includes XML tags, attribute names, and attribute values. You can change this behavior by setting **lisa.magic.string.xml.tags=true** in the **local.properties** file, in which case normal magic string rules apply.

In other words, if you have **<foo bar="baz"/>,** none of that is considered for magic strings. By contrast, if the structure is **<foo>bar</foo>**, then "bar" (but not "foo") is considered for magic strings.

The following example is later in the previous 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.

Example of a magic date in Virtual Service Environment

This is the canonical example of conversational state over a stateless protocol, in this case SOAP over HTTP. 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" is included in the response, even though those details were never recorded.

VSE does not regard the seat number, 1 in this case, as a magic string. The seat number was never seen in the original series of requests that led to this response. Nor is it large enough to be regarded as a magic string. A magic string must be at least three 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** file.

The {{ }} property notation is powerful and flexible and is not confined to using magic strings. For example, we could change the "1" in the response in the example to "DataSetValue" and could define a data set in the virtual service model. Then that data set is used to generate the runtime response value. The data set could be a random string generator, a counter, a call to a database, a reference to an Excel spreadsheet row, or anything. }} can execute arbitrary Java code and even generate realistic "everyday" data such as a valid credit card number {{=[:Credit Card:].

More Information:

- Using BeanShell in DevTest

Magic String Exclusions

VSE attempts to identify tokens in identical requests and responses (contingent on specific rules), and turn the values in the response to a "magic string." The value of the magic string varies, based on the values in the request.

However, DevTest has no way of knowing if the values match by design or by accident. This match is most common for the following values:

- Booleans
- The **__NULL** token that the DevTest agent and Application Insight for Java VSE uses. For example:

Request

```
<GetUserRequest>
<userId>lisaitko</userId>
<includeDetails>true</includeDetails>
</GetUserRequest>
```

Response

```
<GetUserResponse>
<userId>lisaitko</userId>
<isActive>true</isActive>
<isEmailVerified>true</isEmailVerified>
</GetUserResponse>
```

The two "true" values in the response have nothing to do with each other, or with the value in the request. In real world scenarios, the number of unwanted "magic strings" that is generated is much greater. One way to avoid this issue is to find and replace these magic strings manually after the service image is recorded.

An easier way, however, is to set the **lisa.magic.string.exclusion** property in **lisa.properties**.

This property lets you specify values that are not candidates for magic string identification. DevTest does not try to correlate these values in the response to values in the request during recording. If necessary, you can still manually edit the service image to add magic strings.

Magic String Case Sensitivity

A *magic string* is a string that in a request argument that is later seen in a response for that conversation. When looking for magic strings, a match is not case-sensitive. Magic string case processing is shown through the following logic:

- If we see "dallas" in the request and "DALLAS" in the response during recording, then we get "austin" during playback, we respond with "AUSTIN".
- If we see "DALLAS" in the request and "dallas" in the response during recording, then we get "AUSTIN" during playback, we respond with "austin".
- If we see "dallas" in the request and "Dallas" in the response during recording, then we get "austin" during playback, we respond with "Austin".
- If we see "dallas" in the request and "DaLLas" in the response during recording, and during playback we see "austin", we send back "austin", not "AuSTin". If you must explicitly deal with this mixed-case response in your service image, alter the response for the meta-transaction to include a callout to your custom Java code such as:

  ```
  {{=MyHelperClass.mixedCase(request_city);}}
  ```

Magic Dates

A powerful date parser scans requests and responses are also scanned during recording. Anything matching a wide definition of date formats is recognized and translated to a *magic date* . In the example that is shown in Magic Strings, the magic date is:

```
{{ =doDateDeltaFromCurrent("yyyy-MM-dd","0D");/2008-12-17}}
```

The use of {{ }} notation is important here.

At run time, this string is translated as "generate a date of the format yyyy-MM-dd that is 0 days from the current date". That is, generate the current date.

If the original recording was taken on 1 February 2009 and the response contained the date 2009-02-10, the magic date string would be:

```
=doDateDeltaFromCurrent("yyyy-MM-dd","10D");/2009-02-10/
```

The **10D** in the magic string means that the VSE generates a date in the response that is 10 days ahead of the current time. Therefore, if the VSE is in playback mode on 12 June 2010, the response contains the string **2010-06-22**.

The valid parameters for date deltas are:

- **D:** Days
- **H:** Hours
- **M:** Minutes
- **S:** Seconds
- **Ms:** Milliseconds

Another variant of magic dates is:

```
doDateDeltaFromRequest
```

Use the **doDateDeltaFromRequest** variant 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 could accept a seating request for a specific flight on a specific day. If that date is seen in the response, the VSE correctly substitutes the date in any subsequent responses.

A more sophisticated example is if the flight request generated a response that detailed a flight crossing the International Date Line. A flight from Los Angeles to Sydney arrives two days later than the departure according to the calendar date even though the flight time is 14 hours. In this example, the response contains something like:

```
doDateDeltaFromRequest("yy-MM-dd", "2D")
```

If VSE processed a similar request for a flight departing LAX on 19-June-2013, it includes the correct arrival date of 21-June-2013 in the response.

> **Note:** You can add any valid date and time formats, including the ones that have zones, to **lisa.properties** for magic date calculations.

If an RR response file has a date with some time zone or time zone offset, then the live response date will have this time zone or time zone offset.

Parsing URIs in HTTP Requests

In DevTest 8.5, significant changes were introduced to the parsing of URIs in HTTP requests. These changes are part of HTTP as a Transport Protocol, rather than to one of the content level data protocol handlers for SOAP or REST. The changes were added to honor the URI specifications as defined by RFC 3986. While understanding the contents of the RFC is beneficial, a list of the changes that were made are included here.

- Reserved characters that are used as delimiters within the path segments of a URI (as defined by RFC 3986 section 2.2 and section 3.3) are now properly honored. These reserved characters include (separated by forward slashes):

  `: / @ / ! / $ / & / ' / ( / ) / * / + / , / ; / =`

- If the first reserved character of a path segment is encountered, everything before it in the segment it is considered part of the path. Everything after it in the segment is parsed for parameters, which are expected to be delimited by the same set of reserved characters.

- Parameters that are parsed from a URI are no longer created as arguments; they are added as metadata instead. If the parameter follows a key/value pair format, a `HTTP-Path-Param-<key>` metadata property is added, where `<key>` is the name of the parameter property; if the parameter does not follow the key/value pair format, a `HTTP-Path-Param-<#>` metadata property is added, where <#> is a unique number to to identify the anonymous parameter.

- Each path segment of the URI is stored as an attribute using the property name pattern of `HTTP-Segment-Attr-<#>`, where <#> is the index of the path segment (starting at 0). This functionality was added to help with any custom parsing that may be required in a DPH.

- URIs are no longer truncated at the first semicolon (or similar element).

How HTTP Requests Are Parsed

Consider the following contrived HTTP request:

```
GET /email/notanemail@nothing.com/hello;world=yes;goodbye=no/ctx
(someProp=someValue)/list?order=ascending&filter=*
```

In DevTest 8.4 and earlier, the URI within the HTTP request was incorrectly truncated at the first occurrence of a semicolon, and everything after it (including other path segments) were treated as part of a path parameter. This resulted in normalized operation of:

```
/email/notanemail@nothing.com/hello
```

In DevTest 8.5 and later, we honor the reserved delimiter characters (as defined previously) and try to parse parameters from the URI as best as we can. Here is the information that is parsed from the request:

Type	Values
Operation	`GET /email/notanemail/hello/ctx/list/`

Arguments	`order = ascending`
	
	`filter = *`

Attributes	`HTTP-Segment-Attr-0 = email`

Type	Values

	`HTTP-Segment-Attr-1 = notanemail@nothing.com`
	
	`HTTP-Segment-Attr-2 = hello;world=yes;goodbye=no`
	
	`HTTP-Segment-Attr-3 = ctx(someProp=someValue)`
	
	`HTTP-Segment-Attr-4 = list`

Metadata	`HTTP-Path-Parm-1 = nothing.com`
	
	`HTTP-Path-Param-world = yes`
	
	`HTTP-Path-Param-goodbye = no`
	
	`HTTP-Path-Param-someProp = someValue)`
	
	`HTTP-Method = GET`
	
	`HTTP-Version = HTTP/1.1`

Detailed Explanation

The operation `GET /email/notanemail/hello/ctx/list/` is the result of normalizing the URI, which makes sure to honor the RFC spec and the reserved delimiter characters. Here is a breakdown of each path segment and why it is parsed the way that it is:

Original Path Segment	Normalized Path Segment	Explanation
`email`	`email`	No delimiters found, so the entire segment is used.
`notanemail@nothing.com`	`notanemail`	The at-symbol is a delimiter, so the **`notanemail`** portion is used to identify the segment, and **`nothing.com`** is treated as an anonymous parameter (because it does not follow the key /value pair format.) This anonymous parameter is stored in the request metadata under the **`HTTP-Path-Parm-1`** property (the 1 means it was the first anonymous parameter found).
`hello;world=yes;goodbye=no`	`hello`	The semicolon is a delimiter, so the hello portion is used to identify the segment, and the rest of the segment is parsed as a set of parameters. **`world=yes`** follows the key/value pair format, so it is stored in the request metadata under the **`HTTP-Path-Param-world`** property. **`goodbye=no`** is delimiter from the previous part of the segment by a semicolon and it also follows the key/value pair format, so it is stored in the request metadata under the **`HTTP-Path-Param-goodbye`** property.
	`ctx`	

Original Path Segment	Normalized Path Segment	Explanation
`ctx (someProp=someValue)`		The left parenthesis is a delimiter; so the **`ctx`** portion is used to identify the segment. **`someProp=someValue)`** follows the key/value pair format, so it is stored in the request metadata under the **`HTTP-Path-Param-someProp`** property. The right parenthesis is part of the parameter value; this is expected because it is considered the same as any other delimiter and there is no assumption that is defined by the RFC for it to be used for closing an open parenthesis.
`list`	`list`	This final path segment contains a query section (delimited by ?), so **`list`** is used to identify the segment. The query parameters **`order`** and **`filter`** are parsed and added as arguments to the request.

Using the REST Data Protocol and Request Data Manager DPH To Create Arguments from Path Segments

It is common for a user to want to extract data from the path segments of a URI to use as arguments for matching purposes. Consider our contrived HTTP request again - it contains a segment that works as an e-mail address:

```
/email/notanemail@nothing.com/hello;world=yes;goodbye=no/ctx
(someProp=someValue)/list?order=ascending&filter=*
```

In order to extract **`notanemail@nothing.com`** into an argument, the **REST Data Protocol** and the **Request Data Manager** DPH must be used together. It is assumed that this process is being performed during the creation of the virtual service (either through a live recording or through request/response pairs). When reaching the step of the recording wizard to select the request and response DPHs to use, make sure that the **REST Data Protocol** is there and add the **Request Data Manager** DPH to the request side data protocols. This should come after the REST Data Protocol.

To configure the REST Data Protocol:

1. By default, the REST DPH is configured with URI rule **`GET /email/notanemail /hello/ctx/list/`** This rule must be deleted because it will not properly match other URIs that contain different text where the e-mail address is expected, such as `GET /email/somethingDifferent@nothing.com/hello;world=yes; goodbye=no/ctx(someProp=someValue)/list? order=ascending&filter=*`

2. Add a new URI rule with the string `GET /email/{EMAILADDRESS}/hello/ctx /list/` This instructs the REST DPH that the second segment in the URI should be stored in the `EMAILADDRESS` request argument.

To configure the Request Data Manager DPH:

This allows us to copy the parsed segments of the URI into the appropriate request arguments. To copy the `HTTP-Segment-Attr-1: notanemail@nothing.com` entry into the `EMAILADDRESS`request argument:

1. Select the first transaction from the list and add a new action to the action list.
2. Select the new action in the action list and verify that the action is set to **Copy.**
3. Expand the **Attributes** node on the right-hand side and select the `HTTP-Segment-Attr-1: notanemail@nothing.com` entry. Once it is highlighted, click the **Source** button. This updates the action to use it as the source.
4. Expand the **Arguments** node on the right-hand side and select the `EMAILADDRESS: notanemail` entry. Once it is highlighted, click the **Target** button. This updates the action so that the source value is copied into the target argument. Do not worry about the UI stating that the argument is `notanemail` - it only says this because that is what the value of the argument is before the Request Data Manager DPH is executed.
5. Select the **Operation** node on the right-hand side. Once it is highlighted, click the **Match** button. This updates the action so that it only is performed against the selected operation.

Your virtual service is now ready for creation. In the generated VSM, you will find the Request Data Manager DPH as the second data protocol on the Virtual HTTPS Listener step.

Using DevTest Portal with CA Service Virtualization

DevTest Portal is a web-based application that provides simpler access to the most commonly used workflows for DevTest products. DevTest Workstation is the main application for advanced users and contains the full range of functionality available within the DevTest products.

This section provides detailed information about using the Portal to virtualize service behavior for supported features.

More Information:

- DevTest Portal
- Create Virtual Services by Recording
- Create Virtual Services from Request/Response Pairs
- Create Virtual Services from Swagger (OpenAPI)
- Create Virtual Services from WSDL
- Edit Virtual Services
- Deploy Virtual Services
- Monitor Virtual Services

Create Virtual Services by Recording

DevTest Portal lets you create a virtual service by recording the traffic between a client and a server.

This procedure assumes that a virtual service environment (VSE) and DevTest Portal are running.

You can suspend work on creating a virtual service and then resume later. The upper right area of the initial screen indicates the number of virtual services that are in progress. To display the virtual services, click **Show List**. To resume creation, click the **Edit** icon in the **Actions** column or click the virtual service name. To remove a virtual service from the list, click the **Delete** icon in the **Actions** column. To return to the initial screen, click **Hide List**.

The following video includes a demonstration of recording with the HTTP transport protocol.

Follow these steps:

1. Select **Create**, **Virtual Service** from the left navigation menu.
2. Provide basic information about the virtual service:
    a. Select the VSE server.
    b. Enter a name for the virtual service.
    c. (Optional) Enter a description of the virtual service.
    d. Select the transport protocol.
3. Click **Continue**.
4. To display a list of previously saved recording configurations, click **List**. You can sort the recordings by name, protocol, or status. You can also delete a recording.
5. Configure the transport protocol and record the traffic by performing the steps in the appropriate page:
    - Virtualize Services by Recording (HTTP or HTTP/S)
    - Virtualize Services by Recording (TCP)
    - Virtualize Services by Recording (ODP)
    - Virtualize Services by Recording (CICS)
6. (Optional) Select one or more data protocols.
7. If you selected at least one data protocol, configure each one.
8. Save the virtual service.

Virtualize Services by Recording (HTTP or HTTP/S)

When you create a virtual service by recording with the HTTP transport protocol, follow the steps in this page to configure the transport protocol and to record the transactions.

The following graphic shows the **HTTP Transport Protocol** panel.

Virtualize Website (HTTP/S) window

The graphic for the recording changes color as the components are validated. In the preceding graphic, the VS Recorder and the server are colored, indicating that they have been validated.

When all the mandatory information that is required for recording is entered, the status of the recording changes from "Draft" to "Ready". The status shows on the blue **Status** button. To get more details for the recording, click **Status**.

Follow these steps:

1. CA Service Virtualization chooses a default host for the VS Recorder. To change the name or IP address of this host where the recorder is running, select another IP address from the drop-down. If there are multiple host IP addresses and you want to record on all those IP addresses, "All" is provided as the first option in the drop-down and is selected by default.

2. To enter or change the port for the VS Recorder, click the lock icon to enable the field for input. Enter a port number that the recorder uses as a listen port of incoming requests.

3. To specify the website to record, enter the URL in the **Enter Target URL** field. The following formats are accepted:

   - <ip>:<port>

   - <hostname>:<port>

   - <server>.<domain>

   As you type the URL, the virtual service environment (VSE) validates the format. Once the field is fully typed and you tab out or enter, the VSE calls the target server to ensure that the server is reachable. You can virtualize applications running on the same machine on which DevTest is running.

4. (Optional) Select or clear the **Do not modify host header parameter received from client** check box.
   If selected, this option instructs the live invocation to forward the host header that is received from the client application to the target server. If cleared, the live invocation regenerates the host header parameter to be **host: <***target host***>:<***target port***>**.

5. (Optional) Select or clear the **De-identify transactions** check box.
   This check box specifies whether to try to recognize sensitive data and substitute random values during the recording. For more information, see De-identifying Data. This option is available in advanced mode.

6. (Optional) Select or clear the **Treat all transactions as stateless** check box.
   This check box specifies whether to treat all recorded transactions as stateless. Usually, leave this option cleared. This option is available in advanced mode.

7. Choose one of the following options:

   - To record without using SSL, leave the **Use SSL** check boxes cleared.

   - To record using SSL, select one or both of the **Use SSL** check boxes. The **Use SSL** check box on the left lets you enter information about the SSL certificate to send to the client. The **Use SSL** check box on the right lets you enter information about the SSL certificate to send to the server.

8. (Optional) If you select one or both **Use SSL** check boxes, complete the following fields:

   - **SSL Keystore File**
     Specifies the name of the keystore file.
     To find a keystore file on the file system, click **Browse**. Keystore files must be in PKCS12 or JKS format.

   - **SSL Keystore Password**
     Specifies the password that is associated with the specified keystore file.

   - **SSL Key Alias**
     Designates an alias for a public key. If there is more than one key in the file, the key alias specifies which key in the file to use.

   - **SSL Key Password**
     Specifies the password that is associated with the specified alias in the keystore file.

9. (Optional) To validate the SSL parameters, click **Validate**. As you enter data into the SSL fields, the system performs progressive validation.

10. Click **Start Recording**.

11. To see transactions as they record, click the orange **Txn** button. To clear the transaction list, either during or after recording, click **Clear**.

12. To work with transactions as they record, click the orange **Txn** button and then the maximize button. You can now select transactions from the list and view the request and response data.
    From the **Body** tab, you can click:

    - **Search**, to search for text in the response data

    - **Format**, to choose to format the response data as Text, XML/HTML, or JSON. You can also choose here to Autoformat the text or to toggle the word wrap.

    - **Maximize**, to maximize this tab for easier viewing

13. When you have finished recording, click **Stop Recording** and click **Next**.

Virtualize Services by Recording (HTTP or HTTP/S) - Enhanced VS Recorder

When you have the Enhanced VS Recorder Preview Feature enabled, you can use an enhanced workflow for recording a virtual service, using version 2 of the DevTest API. This capability is currently limited to the HTTP transport protocol. For more information, see Known Issues.

When you create a virtual service by recording with the HTTP transport protocol, follow the steps in this page to configure the transport protocol and to record the transactions.

The following graphic shows the **HTTP Transport Protocol** panel.

Virtualize Website (HTTP/S) window

The graphic for the recording changes color as the components are validated. In the preceding graphic, the VS Recorder and the server are colored, indicating that they have been validated.

When all the mandatory information that is required for recording is entered, the status of the recording changes from "Draft" to "Ready". The status shows on the blue **Status** button. To get more details for the recording, click **Status**.

Follow these steps:

1. CA Service Virtualization chooses a default host for the VS Recorder. To change the name or IP address of this host where the recorder is running, select another IP address from the drop-down. If there are multiple host IP addresses and you want to record on all those IP addresses, "All" is provided as the first option in the drop-down and is selected by default.

2. To enter or change the port for the VS Recorder, click the lock icon to enable the field for input. Enter a port number that the recorder uses as a listen port of incoming requests.

3. To specify the website to record, enter the URL in the **Enter Target URL** field. The following formats are accepted:

   - <ip>:<port>

   - <hostname>:<port>

   - <server>.<domain>

As you type the URL, the virtual service environment (VSE) validates the format. You can virtualize applications running on the same machine on which DevTest is running.

4. (Optional) Select or clear the **Do not modify host header parameter received from client** check box.
   If selected, this option instructs the live invocation to forward the host header that is received from the client application to the target server. If cleared, the live invocation regenerates the host header parameter to be **host: <***target host***>:<***target port***>**.

5. (Optional) Select or clear the **De-identify transactions** check box.
   This check box specifies whether to try to recognize sensitive data and substitute random values during the recording. For more information, see De-identifying Data. This option is available in advanced mode.

6. (Optional) Select or clear the **Treat all transactions as stateless** check box.
   This check box specifies whether to treat all recorded transactions as stateless. Usually, leave this option cleared. This option is available in advanced mode.

7. Choose one of the following options:

   - To record without using SSL, leave the **Use SSL** check boxes cleared.

   - To record using SSL, select one or both of the **Use SSL** check boxes. The **Use SSL** check box on the left lets you enter information about the SSL certificate to send to the client. The **Use SSL** check box on the right lets you enter information about the SSL certificate to send to the server.

8. (Optional) If you select one or both **Use SSL** check boxes, complete the following fields:

   - **SSL Keystore File**
     Specifies the name of the keystore file.
     To find a keystore file on the file system, click **Browse**. Keystore files must be in PKCS12 or JKS format.

   - **SSL Keystore Password**
     Specifies the password that is associated with the specified keystore file.

   - **SSL Key Alias**
     Designates an alias for a public key. If there is more than one key in the file, the key alias specifies which key in the file to use.

   - **SSL Key Password**
     Specifies the password that is associated with the specified alias in the keystore file.

9. (Optional) To validate the SSL parameters, click **Validate**. As you enter data into the SSL fields, the system performs progressive validation.

10. Click **Start Recording**.

11. To see transactions as they record, click the orange **Txn** button. To clear the transaction list, either during or after recording, click **Clear**.

12. To work with transactions as they record, click the orange **Txn** button and then the maximize button. You can now select transactions from the list and view the request and response data.
    From the **Body** tab, you can click:

- **Search**, to search for text in the response data
- **Format**, to choose to format the response data as Text, XML/HTML, or JSON. You can also choose here to Autoformat the text or to toggle the word wrap.
- **Maximize**, to maximize this tab for easier viewing

13. When you have finished recording, click **Stop Recording** and click **Next**.

Virtualize Services by Recording (TCP)

When you create a virtual service by recording with the TCP transport protocol, follow the steps in this page to configure the transport protocol and to record the transactions.

The following graphic shows the **TCP Transport Protocol** panel.

Create Virtual Service TCP window

When all the mandatory information that is required for recording is entered, the status of the recording changes from "Draft" to "Ready". The status shows on the blue **Status** button. To get more details for the recording, click **Status**.

Follow these steps:

1. CA Service Virtualization chooses a default host for the VS Recorder. To change the name or IP address of this host where the recorder is running, select another IP address from the drop-down. If there are multiple host IP addresses and you want to record on all those IP addresses, "All" is provided as the first option in the drop-down and is selected by default.

2. To enter or change the port for the VS Recorder, click the lock icon to enable the field for input. Enter a port number that the recorder uses as a listen port of incoming requests.

3. Enter the name or IP address of the target host where the server runs in the **Target host** field.

4. Enter the port number of the target host in the **Port** field.

5. Choose one of the following options:

   - To record without using SSL, leave the **Use SSL** check boxes cleared.

   - To record using SSL, select one or both of the **Use SSL** check boxes. TCP recording uses the same SSL certificate for the client and server.

6. (Optional) If you select one or both **Use SSL** check boxes, complete the following fields:

   - **SSL Keystore File**
     Specifies the name of the keystore file.
     To find a keystore file on the file system, click **Browse**. Keystore files must be in PKCS12 or JKS format.

   - **SSL Keystore Password**
     Specifies the password that is associated with the specified keystore file.

   - **SSL Key Alias**
     Designates an alias for a public key. If there is more than one key in the file, the key alias specifies which key in the file to use.

   - **SSL Key Password**
     Specifies the password that is associated with the specified alias in the keystore file.

7. (Optional) To validate the SSL parameters, click **Validate**. As you enter data into the SSL fields, the system performs progressive validation.

8. Select or clear the following check boxes. For more information, see Virtualize TCP.

   - **Treat request as text**
     Specifies whether the request is treated as text.
     If you clear this check box, you must select the request encoding. The drop-down list displays the most common encoding types by default. To see all available options, click **All**. To return to a list of the most common types, click **Subset.**

   - **Treat response as text**
     Specifies whether the response is treated as text.
     If you clear this check box, you must select the response encoding. The drop-down list displays the most common encoding types by default. To see all available options, click **All**. To return to a list of the most common types, click **Subset**.

   - **Request Delimiter is auto detect**
     Specifies whether the request delimiter is auto detected.
     If you clear this check box, you must select the request delimiter that you want to use. Some delimiters require you to enter additional information. For example, if you select **Regular expression matches the record**, you must also specify the regular expression to match.

   - **Response Delimiter is auto detect**
     Specifies whether the response delimiter is auto detected.
     If you clear this check box, you must select the response delimiter that you want to use. Some delimiters require you to enter additional information.

9. (Optional) Select or clear the **De-identify transactions** check box.
   This check box specifies whether to try to recognize sensitive data and substitute random values during the recording. For more information, see De-identifying Data. This option is available in advanced mode.

10. (Optional) Select or clear the **Treat all transactions as stateless** check box.
    This check box specifies whether to treat all recorded transactions as stateless. Usually, leave this option cleared. This option is available in advanced mode.

11. Click **Start Recording**.

12. To see transactions as they record, click the orange **Txn** button. To clear the transaction list, either during or after recording, click **Clear**.

13. To work with transactions as they record, click the orange **Txn** button and then the maximize button. You can now select transactions from the list and view the request and response data.
    From the **Body** tab, you can click:

    - **Searc**h, to search for text in the response data
    - **Format**, to choose to format the response data as Text, XML/HTML, or JSON. You can also choose here to Autoformat the text or to toggle the word wrap.
    - **Maximize**, to maximize this tab for easier viewing

14. When you have finished recording, click **Stop Recording** and click **Next**.

Virtualize TCP

The TCP transport protocol lets you virtualize raw TCP/IP traffic between one or more clients and a server. This transport resembles the HTTP protocol, and uses many of the same features.

During recording, you configure the TCP protocol exactly as you would configure HTTP in Endpoint mode. You give it a socket on which to listen, the target server, and the destination. The protocol forwards data between the client and server, and "listens in" on that data to create VSE transactions.

- Converting Bytes to VSE Transactions
- Creating Conversations
- Out-of-the-Box Delimiters
- Treat Request as Text and Treat Response as Text
- Why Is the Transaction Count Incorrect?

Converting Bytes to VSE Transactions

It can be difficult to know what makes a complete request or response, and how to correlate those requests and responses. The TCP data has no inherent structure. A request delimiter is required to identify requests in the incoming data stream. You can also select a response delimiter, although a response delimiter is not strictly required.

The recorder uses the following rules to determine requests and responses, and to correlate requests and responses together into transactions:

1. Zero or one responses follow a request.
2. Each response is paired with the latest complete request.
3. A request is considered complete when response data arrives, even if the request delimiter indicates that it is not complete.
4. A response is considered complete when request data arrives, even if the response delimiter (if any) indicates that it is not complete.
5. If the response delimiter finds a complete response when response data is received, any extra response data is discarded.

For example:

```
Request 1 - Request 2 - Response A - Partial Request 3 - Response B - Request 4 -
Partial Response C - Request 5 - Response D With Trailing Data
```

This data is parsed into transactions as follows:

- Request 1 with no response
- Request 2 with Response A (from rule 2)
- Partial Request 3 with Response B (from rule 3)
- Request 4 with Partial Response C (from rule 4)
- Request 5 with Response D, but with the "trailing data" dropped (from rule 5)

This system works if the client and server follow a synchronous request/response paradigm, and it handles requests with no responses. The system does not handle asynchronous communication, or one request with multiple responses.

Creating Conversations

By default, it records TCP service images, VSE creates conversations based on your machine name (or IP address) and the outbound socket. Any time your outbound socket changes, a new conversation is started. To change this behavior, select the **Treat all conversations as stateless** check box.

Out-of-the-Box Delimiters

DevTest has the following delimiters:

- **None**
- **Records are terminated with line endings**
  One or more newline characters terminates each request/response. The delimiter itself is not included in the request/response.
  Values:
  - \n
  - \r
  - Unicode characters 0085, 2028, and 2029

- **Records are of fixed length**
  Each request/response is a specific number of bytes in length.

- **Records are equal to the whole data package**
  The response is read until end-of-data is encountered. The entire content received is treated as the response.

- **Records are delimited by specific characters**
  A specific character or sequence of characters, such as a comma, terminates each request /response. The entire sequence must be matched. The delimiter itself is not included in the request/response. This option is similar to importing a CSV file.

- **Delimiters match a regular expression**
  A set of bytes that matches the regular expression terminates each request/response. The delimiter itself is not included in the request/response.

- **Regular expression matches the record (includes delimiter)**
  The characters in the response that match the regular expression are included in the response. Characters that do not match the expression are ignored.

- **SWIFT**
  This is an industry standard protocol used for financial messages.

If the delimiter characters are not included in the request/response (as usually), back-to-back sets of delimiters are ignored.

If you do not specify a delimiter when recording, the recorder attempts to determine the delimiter.

Treat Request as Text and Treat Response as Text

The following check boxes control how the request and response data is parsed.

- **Treat request as text**
  Specifies whether to use the request as the body of the VSE request.
  Values:

  - **Selected:** Sets the entire request as the body text of the VSE request. The first "word" (up to the first space character) is used as the operation name.

  - **Cleared:** Leaves the request body as binary and sets the operation name to OPERATION.

- **Treat response as text**
  Specifies whether to use the response as the body of the VSE response.
  Values:

  - **Selected:** Treats the response body as text and converts any embedded nulls (\0) to spaces.

  - **Cleared:** Leaves the response body as binary.

Why Is the Transaction Count Incorrect?

The transaction count always lags by at least one. When the TCP protocol finds a complete transaction, it caches that transaction until the next transaction starts, because the server may intentionally close the socket after responding. If this happens, the TCP protocol detects it and sets **lisa.vse.close_socket_after_response=TRUE** on the response. This property causes VSE to close the server-side socket after sending that response during playback.

If you do not use a response delimiter, the transaction count lags by another transaction. The TCP protocol does not know that the response is finished until a new request starts. So, the total count can be off by two. The count is correct when the recording finishes.

Virtualize Services by Recording (ODP)

Opaque Data Processing (ODP) allows CA Service Virtualization to virtualize data in sufficient detail when the format of the requests and responses is not known. ODP eliminates the need for a new data handler every time you encounter a new message format.

By recording several requests and corresponding responses, CA Service Virtualization can infer the message structure. CA Service Virtualization is able to correlate bytes in a request to corresponding bytes in a response, giving the same "magic string" behavior that is available in other protocols. CA Service Virtualization is also able to understand the request structure sufficiently to intelligently match new requests that are encountered when the virtual service is played back.

ODP uses a patented algorithm to compare an incoming request to all of the requests in the recorded ODP service image. The closest matching request is selected, and the corresponding response is returned, after performing a dynamic magic string substitution.

The ODP matching algorithm applies entropy-derived weights during the matching process. The entropy weighting process infers which bytes in the message are more important. For example, which bytes correspond to the operation type, as opposed to the rest of the payload, and gives those bytes a greater importance during the matching process. The entropy weighting process works best with larger samples of recorded messages (100 or more) and with a diverse sampling of parameter values.

The ODP matching and response algorithm has been shown to work on both binary and textual message protocols. The best results have been obtained with protocols that use fixed width fields (such as IMS) or delimited fields (such as XML-based protocols). Reasonable accuracy has also been obtained with length-encoded protocol formats (for example, ASN.1), but these formats pose the greatest challenge.

ODP is supported for traffic that can be captured using raw TCP/IP sockets.

The following video explains the concepts of ODP.

How to Use ODP

Use the following instructions for recording a virtual service using ODP.

For more information about prerequisites and preparatory steps, see Virtualize TCP.

To record a virtual service using the ODP transport protocol, enter information in the bottom pane of the **Recording** window.

ODP Recording window

When all the mandatory information that is required for recording is entered. the status of the recording changes from "Draft" to "Ready". The status shows on the blue **Status** button. To get more details for the recording, click **Status**.

Follow these steps:

1. Provide basic information about the virtual service, as described in Create Virtual Services by Recording.

2. CA Service Virtualization chooses a default host for the VS Recorder. To change the name or IP address of the host where the recorder is running, select another IP address from the drop-down. If there are multiple host IP addresses and you want to record on all those IP addresses, "All" is provided as the first option in the drop-down and is selected by default.

3. To enter or change the port for the VS Recorder, click the lock icon to enable the field for input. Enter a port number that the recorder uses as a listen port of incoming requests.

4. Enter the name or IP address of the target host where the server runs in the **Target host** field.

5. Enter the port number of the target host in the **Port** field.

6. Choose one of the following options:

   - To record without using SSL, leave the **Use SSL** check boxes cleared.

   - To record using SSL, select one or both of the **Use SSL** check boxes. TCP recording uses the same SSL certificate for the client and server.

7. (Optional) If you select one or both **Use SSL** check boxes, complete the following fields:

   - **SSL Keystore File**
     Specifies the name of the keystore file.
     To find a keystore file on the file system, click **Browse**. Keystore files must be in PKCS12 or JKS format.

- **SSL Keystore Password**
  Specifies the password that is associated with the specified keystore file.

- **SSL Key Alias**
  Designates an alias for a public key. If there is more than one key in the file, the key alias specifies which key in the file to use.

- **SSL Key Password**
  Specifies the password that is associated with the specified alias in the keystore file.

8. (Optional) To validate the SSL parameters, click **Validate**. As you enter data into the SSL fields, the system performs progressive validation.

9. Select a **Profile Type**. The default selection is **Fixed Width Binary**. Options are:

   - **Fixed Width Binary**
     Binary messages with fields of a fixed length. Examples: CICS, IMS.

   - **Length Encoded Binary**
     Binary messages where fields are variable length. Examples: ASN.1, LDAP.

   - **Text**
     Message structure is textual, fields are variable length. Examples: SOAP, JSON.

   - **Custom**
     Selectable only in advanced mode.

10. Select or clear the following check boxes. For more information, see Virtualize TCP.

    - **Request Delimiter is auto detect**
      Specifies whether the request delimiter is auto detected.
      If you clear this check box, you must select the request delimiter that you want to use. Some delimiters require you to enter additional information. For example, if you select **Regular expression matches the record**, you must also specify the regular expression to match.

    - **Response Delimiter is auto detect**
      Specifies whether the response delimiter is auto detected.
      If you clear this check box, you must select the response delimiter that you want to use. Some delimiters require you to enter additional information.

11. (Optional) Select or clear the **De-identify transactions** check box.
    This check box specifies whether to try to recognize sensitive data and substitute random values during the recording. For more information, see De-identifying Data. This option is available in advanced mode.

12. (Optional) Select or clear the **Treat all transactions as stateless** check box.
    This check box specifies whether to treat all recorded transactions as stateless. Usually, leave this option cleared. This option is available in advanced mode.

13. Click **Start Recording**.

14. To see transactions as they record, click the orange **Txn** button. To clear the transaction list, either during or after recording, click **Clear**.

15. To work with transactions as they record, click the orange **Txn** button and then the maximize button. You can now select transactions from the list and view the request and response data.
    From the **Body** tab, you can click:

    - **Search**, to search for text in the response data
    - **Format**, to select to format the response data as Text, XML/HTML, or JSON. You can also select here to Autoformat the text or to toggle the word wrap.
    - **Maximize**, to maximize this tab for easier viewing

16. When you have finished recording, click **Stop Recording** and click **Next**.
    The configuration window opens.

ODP Configure window

Raw transactions are displayed in the **Transactions/Requests** and **Response** panes.

Select from the **View Data As** drop-down list to select the data format.

Basic options:

- Binary-Hex, UTF-8, US-ASCII, ISO-8859-1, UTF-16BE, UTF-16LE, UTF-16, [All]
  Binary-Hex takes binary data bytes and represents it as hexadecimal. Binary-Hex is the default selection when the recording is created.
  [All] expands the list of options.
  The other selections re-encode the data in **Raw Transactions** and **Clustered Transactions** to whatever charset is selected.

Expanded options:

- [Subset], Binary-Hex, UTF-8, Big5, Big5-HKSCS, CESU-8, EUC-JP, EUC-KR, GB18030, GB2312, GBK, IBM-Thai, IBM00858, IBM01140, IBM01141, IBM01142, IBM01143, IBM01144, IBM01145, IBM01146, IBM01147, IBM01148, IBM01149, IBM037, IBM1026, IBM1047, IBM273, IBM277, IBM278, IBM280, IBM284, IBM285, IBM290, IBM297, IBM420, IBM424, IBM437, IBM500, IBM775, IBM850, IBM852, IBM855, IBM857, IBM860, IBM861, IBM862, IBM863, IBM864, IBM865, IBM866, IBM868, IBM869, IBM870, IBM871, IBM918, ISO-2022-CN, ISO-2022-JP, ISO-2022-JP-2, ISO-2022-KR, ISO-8859-1, ISO-8859-13, ISO-8859-15, ISO-8859-2, ISO-8859-3, ISO-8859-4, ISO-8859-5, ISO-8859-6, ISO-8859-7, ISO-8859-8, ISO-8859-9,

JIS_X0201, JIS_X0212-1990, KOI8-R, KOI8-U, Shift_JIS, TIS-620, US-ASCII, UTF-16, UTF-16BE, UTF-16LE, UTF-32, UTF-32BE, UTF-32LE, windows-1250, windows-1251, windows-1252, windows-1253, windows-1254, windows-1255, windows-1256, windows-1257, windows-1258, windows-31j, x-Big5-HKSCS-2001, x-Big5-Solaris, x-euc-jp-linux, x-EUC-TW, x-eucJP-Open, x-IBM1006, x-IBM1025, x-IBM1046, x-IBM1097, x-IBM1098, x-IBM1112, x-IBM1122, x-IBM1123, x-IBM1124, x-IBM1166, x-IBM1364, x-IBM1381, x-IBM1383, x-IBM300, x-IBM33722, x-IBM737, x-IBM833, x-IBM834, x-IBM856, x-IBM874, x-IBM875, x-IBM921, x-IBM922, x-IBM930, x-IBM933, x-IBM935, x-IBM937, x-IBM939, x-IBM942, x-IBM942C, x-IBM943, x-IBM943C, x-IBM948, x-IBM949, x-IBM949C, x-IBM950, x-IBM964, x-IBM970, x-ISCII91, x-ISO-2022-CN-CNS, x-ISO-2022-CN-GB, x-iso-8859-11, x-JIS0208, x-JISAutoDetect, x-Johab, x-MacArabic, x-MacCentralEurope, x-MacCroatian, x-MacCyrillic, x-MacDingbat, x-MacGreek, x-MacHebrew, x-MacIceland, x-MacRoman, x-MacRomania, x-MacSymbol, x-MacThai, x-MacTurkish, x-MacUkraine, x-MS932_0213, x-MS950-HKSCS, x-MS950-HKSCS-XP, x-mswin-936, x-PCK, x-SJIS_0213, x-UTF-16LE-BOM, X-UTF-32BE-BOM, X-UTF-32LE-BOM, x-windows-50220, x-windows-50221, x-windows-874, x-windows-949, x-windows-950, x-windows-iso2022jp
Binary-Hex takes Base64 encoded data and represents it as hexadecimal.
[Subset] reverts the list of options back to the default values.

Selecting/Expanding:
Each transaction within **Raw Transactions** and **Clustered Transactions** can be selected and expanded. If a request body is too long to be displayed within a single line, it is truncated and can be viewed in expanded mode. If a transaction is small enough to be viewed as one line, it is not truncated, and clicking the Expand “+” icon changes the icon to a “-“ icon but shows nothing underneath the transaction, as no expansion is needed. Only one transaction can be expanded at one time. Expanding a transaction selects that transaction. Selecting another transaction unexpands a previously expanded transaction. Selecting a transaction essentially does a **getResponse**.

Clustered Transactions:
Clustered Transactions view cannot be accessed until the **Create Clusters** button at the bottom of **Raw Transactions** is selected in new recordings. Selecting and clearing **Fixed Width Fields** and **Match on specific transactions in cluster** prompts you to regenerate the clustered transactions because it influences the outcome.

Select the **Perform Data Clustering?** Select this check box to group similar messages into what is named a Data Cluster. The similarity is determined by byte matching algorithms, that would bundle transactions based on request bytes or response bytes or both. The **Perform Data Clustering?** checkbox removes **Clustered Transactions** from view when unchecked.

The **Create Clusters** button at the bottom of the window groups the transactions into different sets named *clusters*. VSE matches the bytes of different transactions and finds some common elements.

Advanced Mode:

- **Estimated Number of Clusters**
  This commonly is parallel to an estimated number of operations we are recording. For optimum results, the default is set to 10.

- **Cluster Distance Method**
  This is the transactions byte-matching algorithm that is used to generate the clusters. By default we use **Response Matching**, which means that we give more preference to matching the response bytes of a transaction while choosing a group. The alternate method is **Request Matching**, which is less efficient.

- **Fixed Width Fields**
  Consider the requests as having fixed-width arguments, as in IMS requests where fields are set to a predefined length.

- **Match on specific transactions in cluster**
  By default, at playback time, we match the incoming requests with the “prototype” transaction of a cluster. A prototype is created when recorded transactions are grouped together, and a meta transaction is created that closely matches the requests in that group. When this check box is selected, we match against the prototype first, then try to find the closed match of the specific transaction in that group. The response sent back is from that specific transaction and not the meta.

Playback Options:

- **Scaling Level**
  Defines how aggressively to analyze the message structure to detect critical fields. **Scaling Leve**l uses an entropy analysis algorithm to predict which bytes are the most critical bytes for matching. A value of 0 turns off analysis. A value of 10 performs the most aggressive analysis. Adjust the value to try alternative matching strategies to get better matches.
  Default: 10

- **Binary Transactions**
  This flag indicates that we should not stretch our incoming transaction to match it to the recorded ones.

- **Use Magic Strings?**
  A check box that determines whether to perform magic string substitution in the response that closely matched the incoming request.

- **Delimiter for Finding Magic Strings** (optional)
  When selected, the delimiter does not become part of the magic string replacement.

- **Minimum Length for Match**
  Defaults to 4.

- **Padding Character** (optional)
  When selected, do not include the padding character when detecting magic strings.

Click **Save Options** to save your changes and continue.

Virtualize Services by Recording (CICS)

Recording CICS programs requires connectivity to a DevTest z/OS LPAR Agent, which must be connected to at least one DevTest CICS Agent. See DevTest Mainframe Bridge Properties for configuration requirements to connect to a DevTest z/OS LPAR Agent.

The following video shows an example of recording.

Follow these steps:

1. In DevTest Portal, click **Create**, **Virtual Service**.

2. Select the VSE server.

3. Enter a name for the virtual service.

4. (Optional) Enter a description of the virtual service.

5. Select the CICS transport protocol.

6. To indicate which CICS programs to record and in which CICS regions the recordings will take place, click **Add Element**.
   The **Program Details** box appears.

Screen capture of Program Details box.

7. To select the program type, use the **Type** list to select the program type.
   LINK indicates recording a CICS LINK command.
   DTP indicates recording CICS Distributed Transaction Processing commands (ALLOCATE, SEND, RECEIVE, and more).

8. Enter the program **Name** of one to eight characters, with no spaces.

9. From the Region list, select a CICS region in which the recording will take place from the list of connected CICS regions, or ANY REGION to record on all connected CICS regions.

10. Select an LPAR from the list.

11. (Optional) Enter filters for the program type.
    For LINK programs:

    User ID is the CICS user ID that the CICS transaction is run under. Only CICS transactions with this user ID are recorded.
    System ID is the CICS SYSID argument that is specified on a Distributed Program LINK. Only CICS LINKs with this SYSID are recorded.
    Transaction ID is the CICS TRANID argument that is specified on a Distributed Program LINK. Only CICS LINKs with this TRANID are recorded.
    EIB Transaction ID is the CICS transaction ID of the CICS transaction in which the CICS LINK is performed. Only CICS transactions with this transaction ID are recorded.

    For DTP programs:

    Connection Name is the four-character connection ID on a CICS ALLOCATE command, or ANY CONNECTION to record all ALLOCATE commands.
    Connection Type is REPLACE or PASS THROUGH. REPLACE indicates that all DTP commands are recorded and virtualized in playback mode. PASS THROUGH indicates that the session ALLOCATE and FREE commands are recorded, but not virtualized in playback mode (allowing the session to be established).
    Transaction ID is the four-character CICS transaction that is invoked on the remote CICS region when the session is established. Only sessions invoking this CICS transaction are recorded.
    EIB Transaction ID is the CICS transaction ID of the CICS transaction in which the CICS DTP commands are performed. Only CICS transactions with this transaction ID are recorded.

When a program is properly defined, the circle in the **Element to Virtualize** pane is green, and the boxes around the fields are green. If there are errors or missing fields, the circle in the **Element to Virtualize** pane is red, and the box around the field with errors is red. In the previous graphic, the **Name** field is required, so the box around that field is red until a program name is entered.

As an alternative to manually entering a program, you may expand the Defined Elements pane and may select one.

Screen capture of Defined Elements pane.

Follow these steps:

1. (Optional) Enter a **Prefix** to limit the search,

2. (Optional) Change the **Max** (maximum) number of programs that are retrieved from each region.

3. (Optional) Filter the display with the **Element**, **Region**, and **LPAR** text boxes.

4. To add a program to be virtualized, select it and click the **Add to Virtualize** button.

The programs listed on this pane are the programs that are defined in the CICS Program Table (PPT) on each CICS region. CICS does not absolutely require all programs to be defined, so to virtualize a program that is not defined, you must enter it manually.

To record:

1. Click **Start Recording**.

2. Run the programs to record.

3. Click **Stop Recording**.

4. Click **Next**.

To configure data protocols:

Request/Response Data Protocol pane

1. From the **Request/Response Data Protocol** pane, if you require copybook processing, select the CICS Copybook Data Protocol and click the **>>** button to add it to the selected protocols.

2. For both the request and response, select a Copybook Bundle. If you do not have any Copybook Bundles defined in the project, you are given the opportunity to create one. The Configuration Details are taken from the copybook bundle, and you can override them on the **Configuration Details** pane.

Configuration Details pane

3. To save the recording, click **Next** and select a Save option.

CA Service Virtualization and SSL

During CA Service Virtualization recording, the SSL Server application is the System Under Test.

Use SSL (Between Recorder and Target Server)

If the **Use SSL** check box between the recorder and server is selected, the recorder communicates with the target server through an SSL connection. By default, the recorder trusts all server certificates.

If the target server requests Client Authentication (two-way SSL), the recorder sends the certificate that is specified in the fields below the **Use SSL** check box. If no certificate information is specified, the recorder uses the values that are specified in the local.properties (the **ssl.client.cert.*** properties) file of the VSE server where recording is performed.

Use SSL Between Client and Recorder

If the **Use SSL** check box between the client and recorder is selected, the client communicates with the recorder through an SSL connection. In this portion of the recording, the recorder is the delegate for the target server and must send back a server certificate when the client initiates a request. The certificate that is sent back to the client is the one specified in the fields below the **Use SSL** check box.

If no certificate information is specified, the recorder uses the default SSL server keystore values that are specified in the local.properties (the **ssl.server.cert.*** properties) file of the VSE server where recording is performed.

Playback of the VSM

If you select **Use SSL to Client**, an SSL handshake occurs between the client application or test case client and the VSM. The keystore that is provided in the VSM Listen step is used as the Server certificate for one-way authentication. There is no SSL handshake between the client and the VSM. The handshake is straight HTTP.

If the Live Invocation step was executed, an SSL handshake occurs between the VSM client to the real server. If the real server requests client authentication, the keystore in the HTTP/S Protocol Live Invocation step is used.

If the keystore contains multiple certificates, CA Service Virtualization uses the first one.

Create Virtual Services from Request/Response Pairs

DevTest Portal lets you create a virtual service from request/response pairs.

If you upload a zip file, the request/response pairs must be in the root of the zip file.

If the request/response pairs that you upload include duplicate file names, only the first duplicate file is processed. The other duplicate files are ignored.

This procedure assumes that a virtual service environment (VSE) and DevTest Portal are running.

You can suspend work on creating a virtual service and then resume later. The upper right area of the initial screen indicates the number of virtual services that are in progress. To display the virtual services, click **Show List**. To resume creation, click the **Edit** icon in the **Actions** column or click the virtual service name. To remove a virtual service from the list, click the **Delete** icon in the **Actions** column. To return to the initial screen, click **Hide List**.

Follow these steps:

1. Select **Create**, **Virtual Service** from the left navigation menu.
2. Click **RR Pairs**.
3. Provide basic information about the virtual service:
    a. Select the VSE server.
    b. Enter a name for the virtual service.
    c. (Optional) Enter a description of the virtual service.
    d. Select the transport protocol.
4. Do one or both of the following actions:
    a. Upload one or more request/response pairs.
    b. Upload a zip file that contains one or more request/response pairs.

    Note: If you created your zip file using the native OSX compress utility, this utility can create an additional _MACOSX folder in the file. This folder prevents DevTest from finding your request/response pairs. You must delete this folder or use a different utility to create the zip file before you can import your file successfully.

    If you are creating a SOAP based virtual service with WS-Security headers that need to be processed, the SOAP messages must be in their original condition (no formatting changes) for signatures to be valid.

5. Click **Continue**.
6. View and edit transactions.
7. (Optional) Select one or more data protocols.
8. If you selected at least one data protocol, configure each one.
9. Save the virtual service.

Create Virtual Services from Swagger (OpenAPI)

DevTest Portal lets you create a virtual service from a Swagger (OpenAPI) 2.0 or 3.0 document.

Assume that you want to virtualize a service that is in the early stages of development. You might not be able to create the virtual service by recording. However, if you have access to a Swagger (OpenAPI) document that describes the service, you can use this approach.

The following graphic shows the **Configure Connection & Resources** dialog, which lets you customize which operations to include.

Screen capture of Configure Connection and Resources dialog.

This procedure assumes that a virtual service environment (VSE) and DevTest Portal are running.

If a **Responses** object in the document has a single HTTP response code that is set to **default**, the response code is mapped to the value 200.

Note: You can create a virtual service from an OpenAPI 3.0 document using the same features that were present in Swagger 2.0. New features that were added as part of OpenAPI 3.0 are not supported.

Note: If you are using the **$ref** keyword to reference a definition, the string value must begin with a relative path, such as "**./**" or "**../**".

You can test this feature with the Forward Cars demo application. Forward Cars includes a Swagger 2.0 document that describes the inventory service. The document is available at the following URL:

```
http://localhost:3500/cars-inventory/swagger.json
```

If Forward Cars is running on another computer, replace **localhost** with the host name or IP address of the other computer.

The Swagger document is also available in the **Data** folder of the **examples** project. The file name is **cars-inventory-swagger.json**.

The following video demonstrates this feature.

Follow these steps:

1. Select **Create**, **Virtual Service** from the left navigation menu.

2. Click **Specification**.

3. Provide basic information about the virtual service:

    a. Select the VSE server.

    b. Enter a name for the virtual service.

    c. (Optional) Enter a description of the virtual service.

4. Specify one or more Swagger documents by using any of the following approaches:

    - Enter a URL that points to a Swagger document.
      The URL must be accessible from the VSE server that you selected.

    - Click **Select File** and select a Swagger document.

    - Click **Select File** and select a zip file that contains one or more Swagger documents.

5. Click **Continue**.
   The **Configure Connection & Resources** dialog appears.

6. You can customize which operations are included. For example, you can remove one of the operations in a set of three. You can also change the MIME type of the request and response. Depending on the contents of the Swagger document, only one MIME type might be available. When you are done with this dialog, click **Finish**.

7. View and edit transactions.

8. (Optional) Select one or more data protocols.

9. If you selected at least one data protocol, configure each one.

10. Save the virtual service.

All query arguments and request header values are included in the generated virtual service regardless of the value of the require flag that is indicated in the Swagger document being parsed.

So, for example, in the following snippet of a Swagger document:

```
{

"name": "patentDocId",

"in": "query",

"description": "patent id to filter the results (eg. 05569676)",

"required": "false",

"type": "string",

},
```

The query argument **patentDocId** has a required flag equal to *false*. Regardless of that condition, the indicated query argument will be included as an argument on the generated virtual service.

All query arguments are included in the virtual service. You must edit the virtual service to remove any query arguments you do not want to include.

Create Virtual Services from WSDL

DevTest Portal lets you create a virtual service from one or more WSDL documents.

Assume that you want to virtualize a service that is in the early stages of development. You might not be able to create the virtual service by recording. However, if you have access to a WSDL document that describes the service, you can use this approach.

Note: If a WSDL document that you specify causes an error message to appear in the portal, check whether the log files can help to diagnose the problem.

The following graphic shows the **Configure Connection & Resources** dialog, which lets you customize which operations to include. The available operations are shown for each combination of service and port.

Screen capture of operation chooser.

This procedure assumes that a virtual service environment (VSE) and DevTest Portal are running.

The following video demonstrates this feature.

Follow these steps:

1. Select **Create**, **Virtual Service** from the left navigation menu.
2. Click **Specification**.
3. Provide basic information about the virtual service:
    a. Select the VSE server.

b. Enter a name for the virtual service.

c. (Optional) Enter a description of the virtual service.

4. Specify one or more WSDL documents by using any of the following approaches:

   - Enter a URL that points to a WSDL document.
     The URL must be accessible from the VSE server that you selected.

   - Click **Select File** and select a WSDL document.
     The WSDL document can use URLs to reference external resources.

   - Click **Select File** and select a zip file that contains one or more WSDL documents.
     The WSDL documents can use URLs to reference external resources.
     The WSDL documents can use relative paths to reference XSD files that are in the same zip file.

   - Click **Select File** and select a WSDL bundle that was created in DevTest Workstation.

5. Click **Continue**.

6. If the **Configure Connection & Resources** dialog appears, you can customize which operations are included for each port in each WSDL document. For example, you can remove one of the operations in a set of three. When you are done with this dialog, click **Finish**.

7. View and edit transactions.

8. (Optional) Select one or more data protocols.

9. If you selected at least one data protocol, configure each one.

10. Save the virtual service.

More Information:

- Tutorial 5: Virtual Service from WSDL

View and Edit VSE Transactions

When you create a virtual service from request/response pairs, WSDL, or Swagger, the **Collect Transactions** pane displays a list of transactions and their requests and responses.

The **Collect Transactions** pane gives you the opportunity to update the transactions. For example, you can add a request/response pair or edit a request body.

If you do not want to make any changes, click **Next** to proceed to the next pane.

The following graphic shows the main portion of the **Collect Transactions** pane.

Screen capture of Collect Transactions pane.

The area above the main portion displays the basic information about the virtual service. You can change the virtual service description and the configuration details of the transport protocol.

The area above the main portion also includes the **Advanced Mode** button. Some of the updates require the advanced mode to be displayed.

- Add, Change, and Delete Transactions
- Update Requests and Responses

Add, Change, and Delete Transactions

You can make various changes at the transaction level.

When you add a request/response pair, WSDL document, Swagger document, or manual transaction, a message lets you undo the action. The message remains until you undo the action or make a change elsewhere in the **Collect Transactions** pane.

Notes:

- You can add a request that does not have a corresponding response. However, the **Next** button is disabled until you add the response.
- If a WSDL document that you specify causes an error message to appear in the portal, check whether the log files can help to diagnose the problem.

Follow these steps:

1. To add a request/response pair, click the **Upload** icon and select the files.
2. To add one or more WSDL documents, do any of the following actions:
   - Enter a URL that points to a WSDL document and click the **Upload** icon.
     The URL must be accessible from the VSE server that you selected.
   - Click the **Upload** icon and select a WSDL document.
     The WSDL document can use URLs to reference external resources.

- Click the **Upload** icon and select a zip file that contains one or more WSDL documents. The WSDL documents can use URLs to reference external resources.
  The WSDL documents can use relative paths to reference XSD files that are in the same zip file.

- Click the **Upload** icon and select a WSDL bundle that was created in DevTest Workstation.

3. To add one or more Swagger documents, do any of the following actions:

   - Enter a URL that points to a Swagger document.
     The URL must be accessible from the VSE server that you selected.

   - Click **Select File** and select a Swagger document.

   - Click **Select File** and select a zip file that contains one or more Swagger documents.

4. If the **Configure Connection & Resources** dialog appears, you can customize which operations are included. For Swagger documents, you can also change the MIME type of the request and response. Depending on the contents of the Swagger document, only one MIME type might be available. When you are done with this dialog, click **Finish**.

5. To add a transaction manually:

   a. Ensure that the advanced mode is displayed.

   b. Click the **Add** icon.

   c. Enter the operation name.

   d. Add the request and one or more responses.

6. To move a transaction up or down, select the transaction and click the **Move Up** or **Move Down** icon. You can also drag a transaction to the new location.

7. To delete a transaction, select the transaction and click the **Delete** icon. You can use the Ctrl key to select multiple transactions.

8. Click **Next**.

Update Requests and Responses

You can make various changes at the request and response level.

Metadata can be viewed and edited only in advanced mode.

Property expressions are allowed in the body and metadata of a request or response. Each property expression is evaluated later at runtime. If the argument operation is set to "Property Expression," the expression should return true or false.

Follow these steps:

1. To edit the request body, make the changes in the **Request** area.

2. To edit the response body, make the changes in the **Response** area.

3. Ensure that the advanced mode is displayed.
4. To add a response:
    a. Click the **Add Response** icon.
    b. Add the body and metadata.
5. To delete a response:
    a. Display the response.
    b. Click the **Delete Response** icon.
6. To view metadata, click the **Metadata** button to the right of the **Request** or **Response** label.
7. To add metadata:
    a. Click the **Add Metadata** icon.
    b. Enter the name and value.
8. To edit metadata:
    a. Select the row that you want to edit.
    b. Make the changes.
9. To delete metadata:
    a. Select the row that you want to delete.
    b. Click the **Delete Metadata** icon.
10. Click **Next**.

Select Data Protocols

When you create a virtual service in DevTest Portal, the **Configure** pane lets you specify which data protocols to use.

The following graphic shows the main portion of the **Configure** pane.

Screen capture of Configure pane.

The area on the left lists the available and selected data protocols for the request side.

The area on the right lists the available and selected data protocols for the response side.

Data protocols that were auto-detected show in the selected lists. All data protocols available for the request and response are displayed. If you set the request or response delimiters to auto detect for a TCP recording, the delimiters that were used for the recording display at the top of this page.

Note: DevTest Portal does not support adding or editing a custom DPH.

Follow these steps:

1. To add a data protocol, select it in the **Available Data Protocols** column and click the » icon. You can add multiple data protocols at the same time.
2. To move a selected data protocol up or down, use the icons to the right of the data protocol.
3. To remove a data protocol, select it in the **Selected Data Protocols** column and click the « icon. You can remove multiple data protocols at the same time.
4. Click **Next**.

Configure Data Protocols

If you select one or more data protocols in the **Configure** pane, the next step is to configure them in the **Configure DPH** pane.

When you are done, click **Next**.

- Configure DPHs
- View Transactions

- View All

Configure DPHs

By default, the selected data protocols appear on the left. A dashed line separates the request data protocols and the response data protocols.

DevTest Portal VS Recorder Configure DPH pane

A green check mark indicates that one of the following items is true:

- The data protocol does not need configuration.
- The data protocol has been configured and saved.

Data protocols that can be or must be configured display their editors when they are selected.

When the editor is opened, a default configuration is displayed. You can make changes to the configuration, or you can accept the default.

To save the configuration, click **Save**.

To save the configuration and move to the next configurable data protocol, click **Save and Next**.

If the data protocol does not need configuration, click **Next**.

View Transactions

If you select **View Transactions**, the transactions appear on the left. Sliders at the top let you select request data protocols and response data protocols. The request and response data for each transaction is also displayed.

DevTest Portal VS Recorder View Transactions tab

You can dynamically apply one or more data protocols by moving a slider. The slider on the left displays request data protocols. The slider on the right displays response data protocols. Selecting **Raw Data** indicates not to apply any request or response data protocols.

As you move the slider, the transactions automatically update to reflect the desired applied data protocols. Moving the slider does not permanently change the transactions; it just gives a preview of what they would look like if that data protocol were applied.

Assume that you have the XML, JSON, and REST data protocols. If you move the slider to the XML data protocol, only the XML data protocol is applied. If you move the slider to the JSON data protocol, the XML and JSON data protocols are applied. If you move the slider to the REST data protocol, all three of the data protocols are applied.

Clicking the **Compare Request Data** and **Compare Response Data** buttons displays a pop-up window that lets you compare transactions side-by-side with data protocol sliders over each side. The sliders work the same way as they do on the **View Transactions** screen. If you compare requests, both sides of the pop-up show request data and both sides show request data protocol sliders. Comparing requests lets you dynamically apply or unapply data protocols and see the effects on the transactions. This display does not permanently change the transactions; it is meant as a preview.

Transactions display on the right side of the pane, with request and response data. When you click **Update Transaction View**, the system applies those data protocols to the recorded transactions. The data protocols, when applied to a transaction, parse requests and responses differently, depending on the protocol. For more information, see Using Data Protocols.

The updated transactions, with parsed requests and responses, are retrieved from the virtual service environment (VSE) server and shown in the "preview" **Transactions** pane on the right of the **Configure** pane.

The **Body** tab under **Response Data** has the following options:

- **Search**
  Search for text in the response.

- **Format**
  Format the response as text, XML/HTML, or JSON. You can also automatically format the text or toggle the word wrap.

- **Maximize**
  Maximize the tab for easier viewing.

View All

To display both the **Configure DPHs** information and the **View Transactions** information, click **View All**.

Select and Configure Data Protocols - Enhanced VS Recorder

This release includes a preview feature that lets you select and configure data protocol information for a virtual service. To use this feature, you must first go to The Sandbox and enable it.

When you create a virtual service in DevTest Portal, the **Configure** pane lets you specify which data protocols to use, and lets you configure them.

- Add a Data Protocol
- Configure a Data Protocol

The following graphic shows the main portion of the **Configure** pane.

Select and Configure Data Protocols pane

Add a Data Protocol

Initially, data protocols that were auto-detected show in the **DPH Stack** tab.

To show all available data protocols, click the plus icon.

To add a data protocol:

1. Select the **Request** or **Response** tab, then click the plus icon. Select from the list of available data protocols and double click or drag the data protocol to the tab.

2. To move a selected data protocol up or down, use the up and down arrow icons on the toolbar.

3. To remove a data protocol, select it and click the **x** icon.

4. Click **Next**.

Configure a Data Protocol

The **DPH Settings** section of the **Configure** pane shows the settings for the data protocol that is selected in the list on the left.

A green check mark indicates that one of the following items is true:

- The data protocol does not need configuration.
- The data protocol has been configured and saved.

Data protocols that can be or must be configured display their editors when they are selected.

When the editor is opened, a default configuration is displayed. You can make changes to the configuration, or you can accept the default.

> You can enlarge any section of the pane by clicking the Maximize icon.

For details on each configurable data protocol, see REST Data Protocol Handler and JSON Data Protocol Handler.

Save a Virtual Service

When you create a virtual service in DevTest Portal, the final step is to save the virtual service. The **Save** pane provides the following options:

- **Save & Close**
  Saves the virtual service and closes the recorder.

- **Save & Deploy Virtual Service**
  Saves the virtual service and deploys it immediately.

- **Save & Edit Virtual Service**
  Saves the virtual service and opens it in edit mode in a new editor tab. This option is not supported for all protocols.

Common Fields

For each option, complete the following fields:

- **Select Project**
  Determines the project where the virtual service is stored. Select a project from the drop-down list. Projects that are listed in the drop-down are projects that exist in the **Projects** directory,

- **Allow duplicate specific transactions**
  Specifies whether to record duplicate specific transactions.
  Default: Cleared
  This option is not supported for all protocols.

- **Create baseline tests**
  Specifies whether to create a baseline test from the virtual service. For more information, see Create Baseline from Virtual Service.

Common Fields (Advanced)

To display more options, enable the advanced mode at the upper right corner of the window. Advanced mode is not supported for all protocols. The following options are added:

- **VSM Style**
  Specifies whether to generate a virtual service model (VSM) that includes the prepare steps.
  Values:

  - **More Flexible:** Include the prepare steps.

  - **More Efficient:** Do not include the prepare steps.

  Default: More Efficient

- **Default Navigation**
  Specifies the navigation tolerance that determines where in the conversation tree a VSM searches for a transaction that follows the specified transaction. Select the default navigation tolerance for all except the last (leaf) transactions.
  Default: Wide

- **Leaf Navigation**
  Specifies the navigation tolerance that determines where in the conversation tree a VSM searches for a transaction that follows the last (leaf) transactions.
  Default: Loose

Deployment-Specific Fields

These fields are specific to the **Save & Deploy Virtual Service** option:

- **Group Tag**
  Specifies the name of the virtual service group for this virtual service. If deployed virtual services have group tags, they are available in the drop-down list. A group tag must start with an alphanumeric character and can contain alphanumeric characters and the following special characters:

  - period (.)

  - dash (-)

  - underscore (_)

- dollar sign ($)

- **Concurrent capacity**
  Specifies a number that indicates the load capacity. *Capacity* is how many virtual users (instances) can simultaneously execute with the VSM. Capacity here indicates how many threads there are to service requests for this service model.

  > **Note:** The **Concurrent capacity** field appears only in CA Service Virtualization for Performance mode.

  VSE allocates some threads equivalent to the total concurrent capacity. Each thread consumes some system resources, even when dormant. Therefore, for optimal overall system performance, set this setting as low as possible. Determine the correct settings empirically by adjusting them until you achieve the desired performance, or until increasing it further yields no performance improvement.
  Out of the box protocols use a framework-level task execution service to minimize thread usage. For these protocols, a concurrent capacity of more than 2-3 per core is rarely useful, unless the VSM has been highly customized.
  For extensions and any VSM that does not use an out of the box protocol, setting a long Think Time may consume a thread during the think time. In these cases, you can increase the concurrent capacity.
  The following formula gives an approximate initial setting in these cases:

  ```
  Concurrent Capacity = (Desired transactions per second
  / 1000) * Average Think Time in ms * (Think Scale / 100)
  ```

  Example:
  Assume that you are using a custom protocol that does not use the framework task execution service to handle think times. You want an overall throughput of 100 transactions per second. The average think time across the service image is 200 ms, and the virtual service is deployed with a 100 percent think scale.

  ```
  (100 Transactions per second / 1000) * 200ms * (100 / 100) = 20
  ```

  In this case, each thread blocks for an average of approximately 200 ms before responding, and during that time is unable to handle new requests. We therefore need a capacity of 20 to accommodate 100 transactions per second. A thread would become available, on average, every 10 ms, which would be sufficient to achieve 100 transactions per second.
  Default: 1

- **Think time scale**
  Specifies the think time percentage for the recorded think time.
  Note: A step subtracts its own processing time from the think time to have consistent pacing of test executions.
  Default: 100
  Examples:

  - To double the think time, use 200.
  - To halve the think time, use 50.

Create Data-Driven Virtual Services

Data-driven virtual services let you use an external data set as a source of request arguments and response data.

Assume that the request arguments and response data are maintained by a team member who would not normally access DevTest Portal or any other part of DevTest Solutions. The team member can continue to maintain the information externally, and can provide the data set to someone who does access DevTest Portal.

The matching process for data-driven virtual services is different from the standard matching process. If the virtual service environment finds a matching signature that is data driven, it searches the data set for a row that matches the inbound request. The matching is done based on the mapping of argument name to column name in the **Request Data** pane. If the virtual service environment finds a matching row in the data set, it sends the response according to the configuration in the **Response** pane.

- Data Set Prerequisites
- Add a Blank Data-Driven Signature
- Convert an Existing Signature to Be Data Driven
- Map Request Arguments to Data Set Columns
- Map Response to Data Set Columns
- Manage Data Sets

Data Set Prerequisites

This release supports the following data set types:

- Microsoft Excel spreadsheet
- Comma-separated value (CSV) file

An Excel file must have the .xlsx extension. The data must be in the first worksheet.

Place the column names in the first row of the spreadsheet or CSV file.

The following graphic shows a sample data set. The first row contains the column names. The subsequent rows contain the actual data.

account_id	account_type	account_balance	name_first	name_last	city	state	response
20-1001	checking	1000	Bruce	Adams	Los Angeles	CA	<env:Envelope xmlns:env=
20-1002	checking	500	Cat	Taylor	Los Angeles	CA	<env:Envelope xmlns:env=
20-1003	savings	3000	Heather	Marley	Los Angeles	CA	<env:Envelope xmlns:env=
20-1004	savings	4000	Joe	Smith	Los Angeles	CA	<env:Envelope xmlns:env=

Screen capture of sample data set.

For each row, the matching logic examines the values from left to right.

> **Tip:** You can help optimize performance by placing the identifying data in the leftmost column or columns.

Add a Blank Data-Driven Signature

To create a data-driven virtual service, you can start by adding a blank signature with an associated data set.

The resulting signature does not have any request arguments. You add the arguments in a later procedure.

The following graphic shows the **Add New Signature** dialog. An Excel file has been selected.

Screen capture of Add New Signature dialog.

Follow these steps:

1. Open a virtual service.
2. Ensure that the advanced mode is displayed.
3. In the **Stateless Signatures** pane, click the **Add** icon and select **Add New Signature**.
4. Enter the operation name.
5. Move the **Data Driven Behavior** slider to the right.
6. Select an Excel file or a CSV file to upload. If the **From Existing** option is available, you can select a previously uploaded file.
7. Click **Done**.
   The blank signature is added.

Convert an Existing Signature to Be Data Driven

To create a data-driven virtual service, you can start with a signature that is not data driven and convert it to be data driven.

> **Note:** The editor also supports converting a data-driven signature to be not data driven. You might want to perform this action if the data set that is being used to drive the data-driven signature is no longer valid or available. Click the **Configure Data Driven** icon, move the **Data Driven Behavior** slider to the left, and click **Done**.

Follow these steps:

1. Open a virtual service.
2. Select an existing signature.
3. Select the **Signature Definition** tab.
4. Move the **Data Driven Behavior** slider to the right.
5. Select an Excel file or a CSV file to upload. If the **From Existing** option is available, you can select a previously uploaded file.
6. Click **Done**.
   The signature is converted to be data driven.

Map Request Arguments to Data Set Columns

After you add a blank signature or convert an existing signature, you configure the request arguments.

Each argument contains a mapping from the argument name to a column name in the data set.

If the data set has columns that are not included in the incoming requests, do not add request arguments for those columns.

The following graphic shows a mapping example. This example uses the sample data set that is shown earlier.

Screen capture of mapping example.

You cannot specify a comparison operator. The only supported operator is the equal sign (=).

Follow these steps:

1. If you added a blank signature, click the **Add Argument** icon in the **Request Data** pane.
2. To change an argument name, click the value in the **Name** column and enter the new name.

3. To change a column name, click the value in the **Mapping** column and select the new column name.

4. Repeat the preceding steps to create more arguments.

5. (Optional) Move the rows up or down. You can also delete rows.
   The request arguments are now mapped to data set columns.

Map Response to Data Set Columns

You also use the data set to create the response. The editor provides the following approaches:

- **Response from Mapping**
  You select the data set column that contains the response.

- **Response from Template**
  You enter the full text of the response. The text can include one or more mappings to the data set.

This template includes a mapping to the **name_first** column in the sample data set:

```
<html>
Hello, {{name_first}}.
</html>
```

Follow these steps:

1. Click the **Add Response** icon in the **Response** pane.

2. To use the mapping approach, do the following steps:

   a. Select the **Response from Mapping** option.

   b. Click **Select Mapping** and select the column that contains the response.

3. To use the template approach, do the following steps:

   a. Select the **Response from Template** option.

   b. Paste the template into the text area.

Manage Data Sets

The **Data Sets** link in the **Manage** section of the left navigation menu lets you access the data sets that have been uploaded.

The data sets also appear in the **All Resources** window.

You can update the contents of a data set. You can also download a data set.

Notes:

- If you deploy a data-driven virtual service and then update the data set, the changes take effect when you redeploy the virtual service.

- You cannot delete data sets from the portal.

The Model Archive (MAR) file for the virtual service includes the data set.

Follow these steps:

1. Select the **Data Sets** link.
2. To update a data set:
    a. Click the **Edit** icon.
    b. Drag or select the updated backing file.
    c. Click **Update**.
3. To download a data set:
    a. Click the **Options** icon.
    b. Select **Download Data Set**.

Examples of Data-Driven Virtual Services

The sample projects include data-driven virtual services that you can review.

- Bank v5 Project: data-driven-http-rest
- Bank v6 Project: data-driven-kioskV6
- Run the Examples

Bank v5 Project: data-driven-http-rest

The **data-driven-http-rest** service virtualizes a movie database.

The service includes a data-driven signature named **GET /3/movie/{URLPARAM0}/**.

The data set is a comma-separated value (CSV) file named **moviesdata.csv**.

The following graphic shows the request arguments and the columns to which they are mapped. The **api_key** and **URLPARAM0** columns are the first two columns in the data set.

Screen capture of request arguments and columns.

The response uses a template.

You can use the **test-data-driven-http-rest** test case in the project to invoke the service. The test case makes three REST calls for rows that are in the data set. The test case then makes a REST call for a row that is not in the data set. The last call verifies that the result contains the string **404 Not Found**.

Bank v6 Project: data-driven-kioskV6

The **data-driven-kioskV6** service virtualizes a bank database.

The service includes a data-driven signature named **addUserObject**.

The data set is a Microsoft Excel file named **ddadduserobject2.xlsx**.

The request arguments are mapped to most of the columns in the data set.

The response is mapped to the **response** column in the data set.

You can use the **test-data-driven-kioskv6** test case in the project to invoke the service. The test case makes seven SOAP calls for rows that are in the data set. The test case then makes a SOAP call for a row that is not in the data set. The last call verifies that the result contains the string **404 Not Found**.

Run the Examples

The following procedure contains the general steps for deploying and testing the virtual service examples.

> **Note:** To review the test steps in the test case, go to DevTest Workstation and open the test case.

Follow these steps:

1. Start the registry.
2. Start a virtual service environment (VSE).
3. Go to DevTest Portal and deploy the virtual service.
4. Start a coordinator.
5. Start a simulator.
6. Go to DevTest Portal and run the test case.
7. Review the results of the test case. For each step, the results include the request and response, and whether the assertion succeeded.

Performance Tuning of Data-Driven Virtual Services

Data-driven virtual services have a set of configuration properties that let you modify the default behavior. You can use these properties with other factors to help optimize performance.

> **Note:** To configure any of the properties, add the property to the **local.properties** file.

The **data.driven.signature.dataset.rowmatcher.timeout.seconds** property specifies the maximum number of seconds that the matching process can take to find a match for an incoming request. The default value is 5. If the timeout is reached, an unknown response is returned and the following message is written to the **vse_*xxx*.log** file:

```
data set row match operation timed out
```

The **data.driven.signature.dataset.batchsize** property specifies the number of rows to retrieve at a time from the data set. The default value is 100.

Any row that matches an incoming request is cached in memory. The matching logic first checks the results cache for any rows that match the incoming request. This technique allows for faster retrieval of repeated requests.

The following properties let you control the behavior of the results cache:

- **data.driven.signature.dataset.resultscache.size**
  Specifies the number of rows that are cached in memory.
  Default: 25

- **data.driven.signature.dataset.resultscache.expiry.minutes**
  Specifies how long a row is kept in the results cache.
  Default: 30

Before the release of DevTest 9.0, CA Technologies conducted performance testing with both Excel files and CSV files as the data set. The following list is a high-level summary of the results:

- Excel files can scale to about 3 MB.

- CSV files can scale to about 15 MB.

- The peak number of transactions per second is 58.

These results are provided as an example of one particular system. The results on your system can be better or worse.

You can help optimize performance by experimenting with various combinations of property settings and other characteristics. These other characteristics include:

- Size of the data set

- Concurrent capacity of the virtual service

- Number of clients

- Heap size of the registry

- Heap size of the virtual service environment

For example, you could increase the timeout to 10 seconds in one performance run. In the next performance run, you could increase the heap size of the virtual service environment to 1024 MB.

Virtualize JDBC Traffic

You can use the DevTest Java Agent with the DevTest Portal to virtualize JDBC-based database traffic. This feature is also known as *agent-based JDBC virtualization*.

> **Important!**
>
> This feature is supported in this release, though we do not encourage users to use the feature without direct guidance from a CA professional.
>
> A CA professional can help evaluate your specific use case and determine the best method to virtualize.
>
> Due to previous accidental usage of this feature, the CA product team has chosen to move it into the Sandbox.
>
> If you are directed by a CA professional to use this feature, you must first enable it in the Sandbox.

The following JDBC drivers and databases are supported:

- Apache Derby
  - Apache Derby 10.8.3.0 JDBC driver to Apache Derby database (derbyclient-10.10.2.0.jar)
- Oracle
  - Oracle 11*g* JDBC driver to Oracle 11*g* database (ojdbc7.jar plus XDB6.jar, ojdbc6-11.1.0.2.jar, and ojdbc5.jar)
  - Oracle 11*g* JDBC driver to Oracle 12*c* database (ojdbc7.jar plus XDB6.jar, ojdbc6-11.1.0.2.jar, and ojdbc5.jar)
  - Progress DataDirect Connect for JDBC 5.1 to Oracle 11*g* database (oracle.jar)
  - Informatica DataDirect JDBC Driver 5.1.2 to Oracle 11*g* database (com.informatica.datadirect-dworacle-5.1.2.HF1.jar)
- IBM DB2:
  - IBM DB2 9.5 JDBC driver to IBM DB2 9.5 database (db2jcc.jar 3.57.82 and db2jcc4.jar)
  - IBM DB2 9.5 JDBC driver to IBM DB2 9.8 database (db2jcc.jar 3.57.82 and db2jcc4.jar)
  - IBM DB2 9.5 JDBC driver to IBM DB2 10.5 database (db2jcc.jar 3.57.82 and db2jcc4.jar)
- Microsoft SQL Server
  - Microsoft JDBC Driver 4.0 to Microsoft SQL Server 2008 R2 database (sqljdbc4.jar)
  - Microsoft JDBC Driver 4.0 to Microsoft SQL Server 2012 database (sqljdbc4.jar)
  - jTDS 1.3.1 JDBC driver to Microsoft SQL Server 2008 R2 database (sqljdbc4.jar)

- jTDS 1.3.1 JDBC driver to Microsoft SQL Server 2012 database (sqljdbc4.jar)

Before you capture the JDBC traffic, install and configure the DevTest Java Agent on the computer where the application makes JDBC calls to the database. The database itself can reside on a separate computer.

When you configure the agent, be sure to perform the following actions:

- Set the capture level for the JDBC protocol to **Full Data**.
- Set the capture level for any protocol above the JDBC protocol to **Counts and Paths** or to **Full Data**.

Important! During the playback phase, start the virtual service *before* you start the system under test.

Note: For information about installing the agent, see DevTest Java Agent. For information about deploying the virtual service, see Deploy Virtual Services.

Capture the JDBC Traffic

The **Virtualize JDBC** window in DevTest Portal includes the following views:

- **Entity View**
- **Conversation View**

This procedure assumes that the **Entity View** is initially displayed.

Note: If you stop a recording and then start a new recording, any queries from the earlier recording are discarded.

Follow these steps:

1. Go to the home page of DevTest Portal.
2. Select **Create**, **Virtual Service** in the left navigation menu.
3. Click **JDBC**.
4. Click **Start recording**.
5. Exercise the parts of the application that make the queries that you want to virtualize. The **Results** pane displays each unique query as it is captured.
6. (Optional) Examine the captured queries.

7. (Optional) Preview the conversations.

8. When you have all the queries and conversations that you are interested in, click **Stop recording**.

9. Click **Create VS Artifact**.

10. Select the project where the virtual service will be created.

11. Click **Create**.

Examine the Captured Queries

During the capture of JDBC traffic, you can examine details about the queries that have been captured.

You can also use the filter capability to narrow the scope of the displayed queries.

> **Note:** Filtering does not affect which queries are virtualized.

Follow these steps:

1. To view an instance of a query, select the query in the **Results** pane.
   The **SQL Query** pane and the **Transactions** pane are populated.

2. To filter the queries, go to the **Databases** section and select the check box for any database that you want to include.

Preview the Conversations

Virtual services can include both conversations and stateless transactions.

During the capture of JDBC traffic, you can preview the conversations in the virtual service that will be generated.

The following graphic shows an example of a conversation.

Screen capture of conversation view.

You can display a miniature view of the overall structure in the upper right corner. This capability is especially helpful for large conversations.

Follow these steps:

1. In the drop-down list, select **Conversation View**.
2. To display a miniature view of the overall structure, click **Show Outline**.
3. To display detailed information about a query in the conversation, click the node.

JDBC Service Image

In agent-based JDBC virtualization, the virtual service that you generate includes a service image.

Each transaction in the service image has an operation and a series of arguments.

The operation conveys the main purpose of the SQL statement. For example:

```
SELECT FROM TRANSACTIONS
```

The arguments can include the list of columns, each major clause of the SQL statement, and parameter values that are bound to the SQL statement. The last two arguments are always the connection URL and the database user.

The following graphic shows a transaction in DevTest Workstation. The operation is **SELECT FROM TRANSACTIONS**. The first argument contains the list of columns. The SQL statement has one parameter value: the account ID.

Screen capture of a transaction

Each transaction in the service image also has an XML document that contains the results of executing the SQL statement. The XML document is located in the **Response** panel of the Service Image Editor in DevTest Workstation.

The results depend on the type of SQL statement:

- If the SQL statement retrieved data, the results contain one or more data sets.
- If the SQL statement updated data, the results contain update counts.

Multiple Data Set Editor

When a transaction is selected in a service image in DevTest Workstation, the **Response** panel lets you view the results of executing the SQL statement.

You can also edit the data that appears. For example, you might want to change the value in a result set to test how the calling code deals with the data returned from the database.

The default editor in the **Response** panel is named **MultiDataSet Document**. This editor consists of the following components:

- **Results** panel
- **Out Params** panel

Results Panel

The **Results** panel can contain any number of result sets and any number of update counts. For example, the following contents are all valid:

- One result set
- One update count
- One result set and one update count
- Three result sets

The following graphic shows an example of a result set.

Screen capture of an example of a result set.

You can add and delete result sets.

In a result set, you can perform the following actions:

- Change a value
- Add a row
- Move a row up or down
- Delete a row
- View and edit column metadata, such as the label and data type

The following graphic shows an example of an update count.

Screen shot of an example of an update count.

You can add and delete update counts.

In an update count, you can change the value. The value must be an integer or a property expression.

Out Params Panel

The **Out Params** panel is used when a stored procedure call has output parameters.

You can add two types of output parameters:

- Simple
- Result set

A simple output parameter has a key and a value.

A result set output parameter has a key and a result set.

The system under test accesses the parameter by an index number or by a string name. You specify the index number or string name in the key.

JDBC Virtual Service Model

In agent-based JDBC virtualization, the virtual service that you generate includes a virtual service model.

The virtual service model has the same listener and responder steps as in Java VSE:

- **Virtual Java Listener**
- **Virtual Java Responder**

To view the protocol configuration information, open the listener step in DevTest Workstation and double-click **JDBC protocol** in the lower right list. The **Protocol Configuration** dialog opens. This dialog contains a set of key/value pairs.

> **Important!** Do not delete any of the rows. If you delete a row, the system under test will not work correctly.

The following graphic shows the **Protocol Configuration** dialog. This example is based on the Apache Derby database.

Screen capture of Protocol Configuration dialog.

The **url_pattern** key specifies the connection URL that triggers virtualization at playback. If the value ends with a semicolon followed by the text **user=** and a database user, the virtualization is restricted to that database user. You can add key/value pairs for additional database users. The key for each user must begin with **url_pattern_**.

The second key is composed of the driver class name and the **connect** method. The value is the name of the connection class.

The third key is composed of the connection class name and the **prepareStatement** method. The value is the name of the prepared statement class.

Notice how the second and third keys create a "chain" of driver to connection to statement.

The keys that begin with **java.sql.DatabaseMetaData** contain name and version information for:

- The database driver
- The database that it was talking to when the SQL traffic was captured

The keys at the bottom contain a mapping of a specific SQL statement and the parameter metadata that the statement represents.

Match Exceptions

The **Throw exceptions on no match** property lets you control what happens when the virtual service model does not have an answer for a SQL statement. By default, the property is enabled.

- If the property is enabled, the agent throws a SQL exception noting the SQL statement that failed to be virtualized. The exception also suggests that you capture more JDBC frames and add them to the service image. This action involves generating new virtualization artifacts and using the service image combine function in either DevTest Workstation or the **ServiceImageManager** command-line tool.

- If the property is disabled, the agent provides appropriate empty results with a best guess as to column definitions.

You can configure this property from the Agents window of the DevTest Portal. The property appears in the **Settings** tab.

Synthetic Parents

Certain scenarios can cause Application Insight to generate a JDBC transaction frame that does not have a parent transaction frame. For example, a startup process might invoke a JDBC call and then exit without doing anything else.

A *synthetic parent* is a parent transaction frame that an agent creates for any transaction frame that does not have a parent.

When a synthetic parent appears in the DevTest Portal, the name is **SYNTHETIC ROOT** and the category is **Synthetic Root**.

The following graphic shows a synthetic parent and a JDBC frame in DevTest Portal.

Screen capture of a synthetic parent and a JDBC frame.

To allow the agent to create synthetic parents, go to the **Settings** tab of the **Agents** window and enable the **Attach Synthetic Parent** property. The property is located in the **Transactions** category.

A related scenario occurs when Application Insight does not generate a JDBC frame because a parent frame is not available. For example, the Java class that makes the JDBC call is not intercepted. When you enable the **Attach Synthetic Parent** property, the agent internally intercepts the Java class and creates both the synthetic parent and the JDBC frame.

If you enable the property by default, the agent might capture unwanted transactions and cause performance issues. Only enable the property when you are certain that it is needed.

Troubleshooting Database Virtualization

This page helps you troubleshoot virtualization issues with some common symptoms and solutions.

First JDBC Call Not Captured

Symptom:

The agent does not capture the first JDBC call. The agent does capture the subsequent JDBC calls correctly.

Solution:

You need to intercept the JDBC caller method. Do the following steps:

1. Create a **rules.xml** file on the agent side.
2. Identify the class that makes the JDBC call.
3. Use the intercept directive to add the method. For example:

   ```
   <intercept class="com.mycompany.JdbcTestDev" delay="false" method="hitDb"
   signature="*" />
   ```

4. Restart the agent.

Transaction Count Does Not Increase

Symptom:

I am trying to virtualize an Oracle database. When I exercise the JDBC transaction in playback mode, the transaction count does not increase in DevTest Portal.

Solution:

Open the log file for the agent. Search for the following exception:

```
java.lang.RuntimeException: com.itko.javassist.NotFoundException: oracle.xdb.
XMLType
   at com.itko.javassist.CtClassType.getClassFile2(CtClassType.java:202)
   at com.itko.javassist.CtClassType.getSuperclass(CtClassType.java:746)
```

If you find the exception, do the following steps:

1. Stop the application and the virtual service.
2. Start the virtual service. Make sure that it is in Running state.
3. Start the test application.
4. Exercise the JDBC transaction again.
   The transaction count should increase.

Alternately, you can download the **xdb6.jar** file and can place it in the classpath of the application.

REST Data Protocol Handler

The REST data protocol handler analyzes HTTP requests that follow the REST architectural style. The data protocol identifies the dynamic parts of the URI strings. The result is a set of rules. During the playback phase, the virtual service environment (VSE) uses the rules to virtualize HTTP requests that invoke the same operations.

If the VSE does not automatically select the REST data protocol, you can select it manually on the data protocol configuration page of the VS recorder, VS creator from RR or spec, and VS editor.

The following graphic shows the REST data protocol configuration UI as it appears in the VS editor.

You can include the HTTP requests in live traffic or in request/response pairs.

You must use the HTTP/S transport protocol for request/response pairs.

You can configure some aspects of the analysis process.

Each rule contains one or more parameters, which represent the dynamic parts. By default, the parameters begin with the text **URLPARAM**.

The following sample rules include the parameters **URLPARAM0** and **URLPARAM1**. The parameters must be inside curly brackets in the rule. After the analyzer generates the rule, you can change the **URLPARAM** identifier.

```
GET /Service/rest/user/{URLPARAM0}
GET /Service/rest/customer/{URLPARAM0}
GET /Service/rest/customer/{URLPARAM0}/order/{URLPARAM1}
```

At playback, the following URI strings match the first rule:

```
GET /Service/rest/user/100
GET /Service/rest/user/101
```

At playback, the following URI string matches the third rule:

```
GET /Service/rest/customer/1234/order/5678
```

Parameters can also be mixed with other constant text if necessary. For example:

```
GET /Service/rest/user/{USERNAME}/format.{type}
```

In this example, the last token could be **format.json** or **format.xml**.

There can be any number of parameters in a token in any position. For example:

```
GET /Service/rest/users/format.{type}.sortorder.{order}.filter.{filter}
```

You can use the REST rules editor that appears after a recording to add or modify rules to look like this depending on your requirements. Rules like this can also be generated from WADL or RAML files.

- Request/Response Pair Format

- Rules Review and Modification

Request/Response Pair Format

This section describes the format of request/response pairs for the REST data protocol.

Request

The request file must include a valid HTTP header. The URL is the first line of the header. The other header lines are optional.

The REST data protocol handler supports all the HTTP methods/verbs: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, and PATCH.

The format of the URL line is:

```
<METHOD><a space character><REST API path><space><HTTP-VERSION>
```

For example:

```
PUT /rest-example/control/users/save HTTP/1.1
```

If the request includes a body, separate the body from the header with a blank line.

The following example shows a request that contains a body in JSON format.

```
PUT /rest-example/control/users/save HTTP/1.1
accept: application/json
content-Type: application/json
Connection: Keep-Alive
User-Agent: LISA

   {
     "user": {
       "emailAddress": "test@test.com",
       "firstName": "first-9",
       "lastName": "last-9",
       "password": "aaaaaaaa",
       "username": "dmxxx-009"
     }
   }
```

You can condense the JSON or XML body on a single line.

```
{ "user": { "emailAddress": "test@test.com","firstName": "first-
9", "lastName": "last-9", "password": "aaaaaaaa", "username": "dmxxx-009"  }}
```

Response

The response file contains an HTTP response code. The file can contain a header and a body. The response code must be the first line in the file, in the following format:

```
<HTTP-VERSION><a space character><HTTP-RESPONSE-CODE>
```

The following example shows a response that has a body.

```
HTTP/1.1 200

"user":{"emailAddress":"lisa.simpson@test.com","firstName":"lisa",
"lastName":"simpson","password":"60fAFoq+W0R4HrLgsfPodkWRw9I=",
"phoneNumber":"","username":"lisa_simpson"}}
```

If the response does not include a response code line, the response code defaults to **200 (OK)**.

Rules Review and Modification

The **Virtual Service Image Recorder**, the **Virtual Service from request/response pairs** interface, and the **Configure** page in the VS editor include a page where you can review and modify the rules that the REST data protocol created.

The left panel (**URI Rules**) displays the rules. When you select a rule, the middle panel (**Matching Traffic**) displays a sample of traffic that matches the rule. To configure the maximum number of rows that display, set the **lisa.protocol.rest.editor.observedtraffic.max** property in the **lisa.properties** file.

To display a list of HTTP requests that do not match the rules, click **Unmatched Traffic**. This list is empty if the rules match all the collected traffic. To configure the maximum number of requests that display, set the **lisa.protocol.rest.editor.unmatchedtraffic.max** property in the **lisa.properties** file.

The URI Rules pane lets you add, update, reorder, and delete the rules.

You can replace the parameters in the rules with more meaningful names. For example:

```
GET /Service/rest/customer/{customerid}/order/{orderid}
```

You can create multiple rules that match the same operation. The rules are matched in the order shown. As a result, a rule that is higher in the list can match when you expect a lower rule to match. To change the order, use the reordering buttons. You can reorder rules, but when they are reloaded, they honor ordering per verb. This is done to ensure a more efficient way of handling matching. For performance considerations, GET rules are considered before DELETE rules because they are more common.

If you delete a rule, click **Unmatched Traffic** to see the effect of removing the rule on the captured traffic.

To change the values of the following configuration properties, open the **Analyzer Properties** panel in the top right.

If you have modified the REST URL rules, and then want to revert to the automatically generated URL rules, you can perform re analysis of the traffic. Click Properties, and when the Properties page displays, click OK without changing any values.

- **Max Changes**
  Defines the maximum number of changes that are allowed for a token before the variability is considered significant enough to generate a rule.
  Think of a URI as a list of "tokens" separated by the "/" character. For example, the URI "GET /rest/user/1234" contains the tokens:

  - GET
  - rest
  - user
  - 1234

  To change the default value, set the **lisa.protocol.rest.maxChanges** property in the **lisa.properties** file.

- **Start Position**
  Defines the position in the URL at which the REST data protocol starts looking for variable tokens.
  The start position is the index of a token in the list of tokens of which a URI is composed. For example, in the URI:

  ```
  GET /service/rest/customers"
  ```

  "GET" is at position 0 and "customers" is at position 3.
  To change the default value, set the **lisa.protocol.rest.startPosition** property in the **lisa.properties** file.

- **Id Identification Regular Expression**
  Defines a regular expression string that the REST data protocol uses to detect resource identifiers in the HTTP requests. To change the default value, set the **lisa.protocol.rest.idPattern** property in the **lisa.properties** file.

- **URL Parameter Prefix**
  Defines the prefix that the REST data protocol uses for the parameters in rules.
  The purpose of this setting is to help the analyzer spot tokens that follow a specific pattern that you know is variable, but which the analyzer may not automatically detect.
  For example, in the URI:

  ```
  GET /rest/user/person-1234-dev
  ```

  You may know that a user ID in the format *person-nnnn-nnn* always follows "user". In this case, you can define a regular expression to detect this pattern directly. In this example, the regular expression would be:

  ```
  person-[0-9]{4}-[a-z]{3}
  ```

  To change the default value, set the **lisa.protocol.rest.parameterBaseName** property in the **lisa.properties** file.

If you change one or more values for these properties, the data protocol reanalyzes the recorded traffic.

JSON Data Protocol Handler

The JSON data protocol handler in the DevTest Portal is used to extract arguments from a JSON payload in the request. Unlike the DevTest Workstation version of the protocol, it does not make any modifications to the request data itself.

For argument extraction, the data protocol takes all leaf nodes in the JSON and converts them to script-friendly names based on their location in the JSON tree. For example, the following JSON data:

```
{
    "menu": {
        "id": "file",
        "value": "File",
        "value t": "File1",
        "value*": "File2",
        "value#": "File3",
        "value@@@": "File4",
        "popup": {
        "menuitem": [
            {
                "value": "New",
                "onclick": "CreateNewDoc()"
            },
            {
```

```
                "value": "Open",
                "onclick": "OpenDoc()"
            },
            {
                "value": "Close",
                "onclick": "CloseDoc()"
            }
        ]
    }
}
```

will have these arguments extracted:

```
menu_id
menu_popup_menuitem_0_onclick
menu_popup_menuitem_0_value
menu_popup_menuitem_1_onclick
menu_popup_menuitem_1_value
menu_popup_menuitem_2_onclick
menu_popup_menuitem_2_value
menu_value
menu_value_t
menu_value_
menu_value_0
menu_value___
```

Edit Virtual Services

DevTest Portal lets you make changes to various components of a virtual service.

You open a virtual service from the **Virtual Services** window of DevTest Portal.

Follow these steps:

1. Go to the home page of DevTest Portal.
2. Select **Manage**, **Virtual Services** in the left navigation menu.
   The **Virtual Services** window opens.
3. Locate the virtual service in the table.
4. Click the virtual service name.

Stateless Transactions View

To view the stateless transactions in a virtual service, open the virtual service and select **Stateless Transactions** from the **View** drop-down list.

The Stateless Transactions view is divided into the following areas:

- Stateless Signatures pane
- Specifics tab
- Signature Definition tab

To display the advanced options, click **Advanced Mode** in the upper right area.

- Stateless Signatures
- Request Data Arguments and Response Data

Stateless Signatures

When the virtual service environment (VSE) tries to find a match for an inbound request in the stateless transactions, the VSE starts by searching for a *signature* that matches the inbound request.

A signature has the following components:

- Operation name
- (Optional) A set of argument names

In a virtual service, an *operation* represents a categorization of similar transactions. An example of an operation is **depositMoney**.

In a virtual service, an *argument* is a name/value pair that is associated with an operation. An example of an argument is the name **amount** and the value **100.00**.

> **Note:** The VSE does not consider the argument values at this stage of the matching process.

Each signature also has an identifier that is unique within the virtual service.

Multiple signatures can have the same operation name. For example:

- Signature A has the **depositMoney** operation and one argument: **amount**.
- Signature B has the **depositMoney** operation and two arguments: **amount** and **date**.

The unique identifiers can help you differentiate signatures that have the same operation name.

Another way to differentiate signatures that have the same operation name is by adding notes.

The following graphic shows the **Stateless Signatures** pane. For each signature, the unique identifier and the operation name appear. The selected signature has the unique identifier **159** and the operation name **addUserObject**.

Screen capture of Stateless Signatures pane.

To display the argument names, select the signature, then select the **Signature Definition** tab.

Request Data Arguments and Response Data

If the VSE finds a matching signature, it then searches for a *specific transaction* that matches the inbound request.

A specific transaction has the following components:

- Operation name
- (Optional) A set of argument names and values, plus other match criteria such as an operator

Each specific transaction also has an identifier that is unique within the virtual service.

The **Request Data Arguments** pane displays the specific transactions for the selected signature.

The following graphic shows a specific transaction. The unique identifier is **162**. The operation name is **addUserObject**. The arguments appear in a table.

Request Data Arguments

162 addUserObject

Name	Operator	Value	Magic String
userObject_accounts_balance_1	=	100000	✓
userObject_accounts_name_1	=	primary	✓
userObject_accounts_type_1	=	CHECKING	✓
userObject_accounts_balance_2	=	555555	✓
userObject_accounts_name_2	=	fun money	✓
userObject_accounts_type_2	=	SAVINGS	✓
userObject_addresses_city	=	Plano	✓
userObject_addresses_line1	=	5800 Granite Pkwy	✓
userObject_addresses_line2	=	Suite 550	✓
userObject_addresses_state	=	TX	

« 1 2 »

10 20

Screen capture of specific transaction.

If the VSE finds a matching specific transaction, it sends the response that is associated with the specific transaction.

The **Response Data** pane contains the response for the selected specific transaction.

The following graphic shows the **Response Data** pane.

Response Data

1 / 1

```
1  <env:Envelope xmlns:env='http://schemas.xmlsoap.org/soap/envelope/'>
2      <env:Header></env:Header>
3      <env:Body>
4          <ns2:addUserObjectResponse xmlns:ns2="http://ejb3.examples.itko.com/">
5              <return>
6                  <accounts>
7                      <balance>{{=request_userObject_accounts_balance_2;/*555555*/
8                      <id>1313094687566321</id>
9                      <name>{{=request_userObject_accounts_name_2;/*fun money*/}}<
10                     <type>{{=request_userObject_accounts_type_2;/*SAVINGS*/}}</t
11                 </accounts>
12                 <accounts>
13                     <balance>{{=request_userObject_accounts_balance_1;/*100000*/
14                     <id>1313094687566198</id>
15                     <name>{{=request_userObject_accounts_name_1;/*primary*/}}</r
16                     <type>{{=request_userObject_accounts_type_1;/*CHECKING*/}}</
17                 </accounts>
18                 <addresses>
19                     <city>{{=request_userObject_addresses_city;/*Plano*/}}</city
20                     <id>id-1313094687566413</id>
21                     <line1>{{=request_userObject_addresses_line1;/*5800 Granite
22                     <line2>{{=request_userObject_addresses_line2;/*Suite 550*/}}
23                     <state>TX</state>
24                     <zip>{{=request_userObject_addresses_zip;/*75024*/}}</zip>
25                 </addresses>
26                 <email>{{=request_userObject_email;/*Demo@itko.com*/}}</email>
27                 <fname>{{=request_userObject_fname;/*DemoFirst*/}}</fname>
28                 <lname>{{=request_userObject_lname;/*DemoLast*/}}</lname>
29                 <login>{{=request_userObject_fname;/*DemoFirst*/}}</login>
30                 <newFlag>true</newFlag>
31                 <phone>{{=request_userObject_phone;/*2145551212*/}}</phone>
32                 <pwd>nU4eI71bcnBGqeO0t9tXvY1u5oQ=</pwd>
33                 <roleKey>{{=request_userObject_roleKey;/*REGULAR*/}}</roleKey>
34                 <ssn>{{=request_userObject_ssn;/*333224444*/}}</ssn>
35             </return>
36         </ns2:addUserObjectResponse>
37     </env:Body>
38 </env:Envelope>
```

Think time spec: 39

Screen capture of Response Data pane.

You can search for text in a response. You can also configure formatting options, such as syntax highlighting and word wrap.

If the VSE does not find a matching specific transaction, it sends the response that is associated with the *default transaction*.

The default transaction appears at the bottom of the **Request Data Arguments** pane. The **Response Data** pane contains the response for the selected default transaction.

Conversation View

To view a conversation in a virtual service, open the virtual service and select the conversation name from the **View** drop-down list.

The following graphic shows part of a conversation. The starter transaction is selected.

Screen capture of a conversation.

Each node in a conversation is a logical transaction.

The upper area of a node displays the operation name.

The main area of a node displays the following information:

- The argument values of the first specific transaction
- The number of specific transactions

To display a miniature view of the conversation, click **Show Outline**. This feature is helpful for viewing large conversations.

To delete a node and any of its children, right-click the node and select **Delete**. You cannot delete the starter transaction.

The **Request Data Arguments** pane and the **Response Data** pane contain detailed information about the selected node. The behavior of these panes is the same as for stateless transactions.

Unknown Transactions

When the virtual service environment (VSE) cannot find a match for an inbound request, it sends one of the following responses:

- Response for unknown conversational request
- Response for unknown stateless request

To view the unknown responses, open the virtual service and select **Unknown Transactions** from the **View** drop-down list.

You can search for text in a response. You can also configure formatting options, such as syntax highlighting and word wrap.

The following graphic shows an unknown response.

Screen capture of unknown response.

Search for Text in a Virtual Service

You can search for text across the following components of a virtual service:

- Request arguments
- Request attributes
- Request metadata
- Response body
- Response metadata
- Response think time
- Signature name
- Signature notes
- Specific name
- Unknown response

The following graphic shows a list of search results. The results are ordered from most significant to least significant.

Screen capture of search interface.

The **Type** column indicates whether the result is part of a stateless transaction or a conversation.

The **Sub Type** column indicates the type of component where the result is located.

The **Operation** column is blank for unknown responses.

The **Snippet** column highlights the location of the text.

If no search options are selected and you enter two or more words in the search field, an OR search is performed.

If no search options are selected, you can do partial word matches. For example, you can search for the letters **pri** in **primary**.

If you receive too many results or too few results, consider adjusting the search options and filters.

The following video shows the search options.

Follow these steps:

1. Open a virtual service.
2. (Optional) Configure the search options:
   - **Whole Word**
     Searches must match the exact word or phrase.
   - **Match Case**
     Searches are case sensitive.
   - **Reg Ex**
     The text in the search field is treated as a regular expression. The **Reg Ex** option is mutually exclusive with the **Whole Word** and **Match Case** options.
3. (Optional) Configure the filters.
4. Enter the text in the search field.
   As you type, suggestions are displayed below the search field.
5. Click a suggestion or press Enter.
   If the search text is found, one or more results appear.
6. Click an entry in the table.
   The editor displays the component where the text is located.

Add a Note to a Signature

You can annotate a signature by adding a note.

For example, assume that a signature represents a scenario that occurs infrequently. You could add a note that provides a description of the scenario.

You could also add a note to indicate that you changed the default settings of a signature.

Follow these steps:

1. Open a virtual service.

2. Right-click a signature and click **Add Note**.

3. Enter text. You can copy and paste within the note.

4. Click the check mark.

Add a Label to a Specific Transaction

You can annotate a specific transaction by adding a label.

The following graphic shows a specific transaction that has a label. The label is highlighted.

Screen capture of a label.

Follow these steps:

1. Open a virtual service.

2. Click the operation name of the specific transaction.
   A text field opens.

3. Type the label and click the check mark.
   The label is displayed to the right of the operation name.

Update a Virtual Service Manually

You can update a virtual service manually from DevTest Portal.

The following reasons are some of the reasons to update a virtual service:

- While recording a virtual service, you forget to invoke an operation.
- The development team adds a new operation to the service that is being virtualized.
- You want to test a proposed change to a service before the change is implemented.

If you need to make extensive updates, consider an alternate approach:

- Combine the virtual service with another virtual service
- Run the virtual service with the execution mode set to **Image Validation**
- Rerecord the entire virtual service

Replace Text in a Virtual Service

You can replace text in a virtual service.

The find portion of this feature is the same as the search feature. For example, you can configure the same search options and filters.

Follow these steps:

1. Click and select **Replace**.
2. (Optional) Configure the search options.
3. (Optional) Configure the filters.
4. Enter the search text in the upper text field.
5. Press Enter or click **Find Next**.
   If the search text is found, one or more results appear.
6. (Optional) Update the search options or the filters.
7. Enter the replacement text in the lower text field.
8. To replace the text in one result, select the result and click **Replace**.
9. To replace the text in all of the results, click **Replace All**.

Add, Change, and Delete Signatures

DevTest Portal lets you perform any of the following actions:

- Add a signature
- Move a signature up or down
- Change the definition of a signature
- Delete a signature

This procedure describes multiple ways to add a signature. Importing a request/response pair can also result in a new signature.

When the virtual service environment (VSE) searches for a signature that matches an inbound request, it starts at the top of the list of signatures and proceeds downward. Therefore, moving a signature up or down affects the order in which matching takes place.

Changes to a signature definition are automatically applied to all of the specific transactions for the signature. For example, if you add the **NewArg** argument, the specific transactions now have this new argument.

By default, matching is case-sensitive. To force the comparison operators to ignore case for an argument, go to the **Signature Definition** tab and clear the check box in the **Case Sensitive** column.

The following graphic shows the **Signature Definition** tab.

Screen capture of Signature Definition tab.

By default, argument values are processed as strings. For example, "10000" is considered less than "9" because "1" comes alphabetically before "9". To force an argument value to be processed as a number, go to the **Signature Definition** tab and select the check box in the **Is Numeric** column.

Date patterns are used to parse date information. The **Date Pattern** column in the **Signature Definition** tab is read only.

Follow these steps:

1. Open a virtual service.
2. To add a signature by copying and pasting an existing signature from the same virtual service:
    a. Select one or more signatures in the **Stateless Signatures** pane.
    b. Click the **Copy** icon.
    c. Click the **Paste** icon.
    d. (Optional) Edit the new signature or signatures.
3. To add a signature by copying and pasting an existing signature from another virtual service:

    > **Warning!** Avoid copying signatures between virtual services that use different data protocol handlers. For example, a signature from an HTTP-based virtual service will not function correctly in a JMS-based virtual service.

    a. Go to the other virtual service.
    b. Select one or more signatures in the **Stateless Signatures** pane.
    c. Click the **Copy** icon.
    d. Return to the current virtual service.
    e. Click the **Paste** icon.

f. (Optional) Edit the new signature or signatures.

4. To add a signature manually:

   a. Ensure that the advanced mode is displayed.

   b. In the **Stateless Signatures** pane, click the **Add** icon and select **Add New Signature**.

   c. Enter the operation name.

   d. Click **Done**.

   e. Select the new signature.

   f. (Optional) Go to the **Signature Definition** tab and add one or more arguments.

   g. Go to the **Specifics** tab and add one or more specific transactions.

5. To move a signature up or down, select the signature and click the **Move Up** or **Move D own** icon. You can also drag a signature to the new location.

6. To change the definition of a signature:

   a. Select the signature.

   b. Go to the **Signature Definition** tab.

   c. Make the changes. You can move an argument up or down. You can add or delete an argument. You can change the settings of the **Case Sensitive** and **Is Numeric** columns. You can change the operation name.

7. To delete a signature, select the signature and click the **Delete** icon. You can use the Ctrl key to select multiple signatures.

Add and Delete Specific Transactions

The DevTest Portal lets you perform any of the following actions:

- Add a specific transaction
- Move a specific transaction up or down
- Change a specific transaction
- Delete a specific transaction

This procedure describes two ways to add a specific transaction. Importing a request/response pair can also result in a new specific transaction.

The procedures for changing a specific transaction are described elsewhere in this section.

Follow these steps:

1. Open a virtual service.

2. To add a specific transaction by copying and pasting an existing specific transaction:

   a. Select one or more specific transactions in the **Request Data Arguments** pane.

b. Click the **Copy** icon.

c. Click the **Paste** icon.

d. (Optional) Edit the new specific transaction or transactions.

3. To add a specific transaction manually, click the **Add** icon in the **Request Data Arguments** pane.

4. To move a specific transaction up or down, select the specific transaction and drag it to the new location.

5. To delete a specific transaction, select the specific transaction and click the **Delete** icon. You can use the Ctrl key to select multiple specific transactions.

Import Request/Response Pairs

When you import a request/response pair, the request is processed against the virtual service.

If the virtual service contains a signature with the same arguments, a specific transaction is added to the signature.

If the virtual service does not contain a signature with the same arguments, a signature is added. The details of the request and response are added as a specific transaction in the signature. A default transaction is also added.

You can specify request and response metadata by including one or more sidecar files.

Note: If the virtual service is based on a transport protocol other than HTTP, this feature has the following prerequisite: Open the corresponding virtual service model in DevTest Workstation and enter the path to the recording session file in the **Documentation** field. A recording session file is one of the following files:

- **VRS file**
  Generated by the Virtual Service Image Recorder in DevTest Workstation.

- **VR2 file**
  Generated by the CA Service Virtualization Recorder in the DevTest Portal.

Follow these steps:

1. Open a virtual service.

2. In the **Stateless Signatures** pane, click **Add** and select **Import Request/Response Pairs**. The **Import Request/Response Pairs** dialog opens.

3. Specify the files that contain the request/response pairs. You can drag the files into the dialog, or you can click a button to select the files from a file system.

4. Click **Done**.

Comparison Operators for Arguments

Each argument in a specific transaction has a comparison operator. The operator indicates how to match the argument value with an inbound request.

By default, the operator is set to the equal sign (=). Therefore, a match occurs if the argument value in the inbound request is the same as the argument value in the specific transaction.

The following mathematical symbols are also supported:

- Not equal to (!=)
- Less than (<)
- Less than or equal to (<=)
- Greater than (>)
- Greater than or equal to (>=)

You can also specify a regular expression or a property expression. You use these operators along with the **Value** column.

For example, assume that any five-digit postal code that begins with **750** is valid. You could set the operator to **RegEx** and set the value to the following text:

```
750\d\d
```

If you want to allow an argument to have any value, set the operator to **Anything**.

The following graphic shows the use of various operators. In this scenario, all of the argument values in the inbound request match the criteria in the specific transaction. For example, the account balance in the inbound request is 5000. The specific transaction indicates that the account balance must be greater than 100.

Inbound Request

Name	Value
AccountType	Checking
AccountBalance	5000
AccountName	Name2

Specific Transaction

Name	Operator	Value
AccountType	=	Checking
AccountBalance	>	100
AccountName	Anything	Name1

Disable and Enable Magic Strings

Each argument in a specific transaction can be classified as a magic string.

When you view a specific transaction in the DevTest Portal, the **Magic String** column indicates whether each argument has been classified as a magic string.

The **Magic String** column is read only. DevTest Workstation provides a way to change the setting. For more information, see Transactions Tab for Stateless Transactions .

Edit Request Attributes and Metadata

DevTest Portal lets you edit request attributes and metadata.

- Edit the Request Attributes
- Edit the Request Metadata

Edit the Request Attributes

In a virtual service, an *attribute* is a name/value pair that contains information about the request.

Assume that you record a virtual service with the HTTP transport protocol. The attributes in the virtual service include the XML namespace and the recorded raw request.

Request attributes are not used in matching.

Default transactions do not have attributes.

Follow these steps:

1. Open a virtual service.
2. Ensure that the advanced mode is displayed.
3. Click the **Attributes** link to the right of the **Request Data** label.
4. (Optional) Click the column heading to sort the request attributes alphabetically ascending. Click again to sort descending.
5. Add, change, and delete attributes as necessary.

Edit the Request Metadata

In a virtual service, the term *metadata* refers to name/value pairs that can impact how requests are handled during playback.

Assume that you record a virtual service with the HTTP transport protocol. The request metadata in the virtual service includes the HTTP method and the content type.

Request metadata is not used in matching.

Default transactions do not have metadata.

Follow these steps:

1. Open a virtual service.
2. Ensure that the advanced mode is displayed.
3. Click the **Metadata** link to the right of the **Request Data** label.
4. (Optional) Click the column heading to sort the request metadata alphabetically ascending. Click again to sort descending.
5. Add, change, and delete metadata as necessary.

> **Tip:** To filter responses on metadata, use the Request Data Manager data protocol when you are recording. On the Request Data Manager screen, use a metadata element and set it as a request argument. So, this filters requests at playback time also.

Update Responses

DevTest Portal lets you update the response that is associated with a specific transaction. You can also add and delete responses.

- Change the Think Time of a Response
- Edit the Response Metadata
- Add and Delete Responses

Change the Think Time of a Response

Each response for a specific transaction includes a field named **Think time spec**. This field specifies how long to wait before sending the response. If the virtual service was created from recording, the initial value is based on the response behavior that was observed during the recording.

To simulate faster performance, decrease the value.

To simulate slower performance, increase the value.

By default, the value is in milliseconds. To specify the unit of time, add a suffix to the number. Case does not matter. The following suffixes are allowed**:**

Suffix	Description	Example
t	milliseconds	50t
s	seconds	10s
m	minutes	2m
h	hours	1h

You can specify a number range. The think time is randomly selected from the range. For example, the value **100-1000** specifies a random think time from 100 to 1000 milliseconds.

If the think time is 10 milliseconds and it takes 5 milliseconds to process the request and find the response, you only incur an extra 5 milliseconds of delay.

If the think time is 0, or is less than the time that it took to process the request, the VSE sends the response as fast as it can.

The unknown responses in a virtual service have the same field.

When you deploy a virtual service, you can configure a related field named **Think time scale**. This field specifies the percentage by which to multiply all of the **Think time spec** fields in a virtual service.

Follow these steps:

1. Open a virtual service.

2. Locate the response for the specific transaction or unknown response.

3. Change the value of the **Think time spec** field.

Edit the Response Metadata

In a virtual service, the term *metadata* refers to name/value pairs that can impact how responses are handled during playback.

Assume that you record a virtual service with the HTTP transport protocol. The response metadata in the virtual service includes the HTTP response code and the content type.

Response metadata is not used in matching.

Follow these steps:

1. Open a virtual service.
2. Ensure that the advanced mode is displayed.
3. Click the **Metadata** link to the right of the **Response** label.
4. (Optional) Click the column heading to sort the response metadata alphabetically ascending. Click again to sort descending.
5. Add, change, and delete metadata as necessary.

Add and Delete Responses

Normally, a specific transaction has only one response. DevTest Portal lets you add responses. However, some protocols ignore any response other than the first response.

Follow these steps:

1. Open a virtual service.
2. Display the specific transaction.
3. To add a response:
    a. Click the **Add Response** button.
    b. Add the body and metadata.
4. To delete a response, display the response and click the **Delete Response** button.

Find the Transactions that Match a Request

Assume that when you send a request to a virtual service, the response is not what you expect.

You can open the virtual service and then enter the request. DevTest Portal indicates which components in the virtual service are a match for the request. The results can help you determine how to fix the problem.

The request must be in text or XML format.

DevTest Portal lets you enter the actual contents of the request, or specify a request file.

If the results include stateless signatures or stateless specific transactions, the order in which they appear is the same order in which they are matched during playback.

The order of conversational results is not significant.

> **Note:** If the virtual service is based on a transport protocol other than HTTP, this feature has the following prerequisite: Open the corresponding virtual service model in DevTest Workstation and enter the path to the recording session file in the **Documentation** field. A recording session file is one of the following files:

- **VRS file**
  Generated by the Virtual Service Image Recorder in DevTest Workstation.
- **VR2 file**
  Generated by the CA Service Virtualization Recorder in DevTest Portal.

Follow these steps:

1. Open a virtual service.
2. Click and select **Find a Match by Request**.
3. Do one of the following actions:
   - Paste the request into the text area and click **Find a Match**.
   - Drag the request file into the box on the right.
   - Click **Select Request File** and select the request file from a file system.
     The **Search Results** table displays any matches that are found.
4. To display the original request, click **View Request**.
5. Click an entry in the **Search Results** table.
   The matching component is displayed.

Example:

```
GET /find/?q=Dallas
```

The ending slash is required. In the example, entering **GET /find?q=Dallas** will not find any data.

Get Virtual Service URLs

The DevTest Portal lets you obtain the URL to the virtual service component that is displayed in the editor. Any user can then open the virtual service to the same component by entering the URL in a browser.

The following graphic shows a selected URL and the **Page URL** icon:

Screen capture of URL and icon.

Follow these steps:

1. Place the mouse pointer over the **Page URL** icon in the upper right area.
2. Copy the selected URL to the clipboard.

Deploy Virtual Services

You can deploy virtual services from DevTest Portal.

The following video shows an example of deploying and monitoring a CICS virtual service.

> Ensure the virtual service port number is not in use. A virtual service that was created by recording or with Application Insight will have the same port number where it was recorded.

Follow these steps:

1. Go to the home page of DevTest Portal.
2. Select **Manage**, **Virtual Services** in the left navigation menu.
   The **Manage Virtual Services** window opens.
3. Locate the virtual service in the table.
4. Click the **Options** button in the **Actions** column and select **Deploy**.
   The **Deploy Virtual Service** dialog opens.
5. Select the VSE Server where you want to deploy.
6. Modify the fields as necessary:
   - **Group Tag**
     Specifies the name of the virtual service group for this virtual service. If deployed virtual services have group tags, they are available in the drop-down list. A group tag must start with an alphanumeric character and can contain alphanumeric characters and the following special characters:
     - Period (.)
     - Dash (-)
     - Underscore (_)
     - Dollar sign ($)
   - **Concurrent capacity**
     Specifies a number that indicates the load capacity. *Capacity* is how many virtual users (instances) can simultaneously execute with the virtual service. Capacity here indicates how many threads there are to service requests for this service model.

> **Note:** The **Concurrent capacity** field appears only in CA Service Virtualization for Performance mode.

VSE allocates various threads equivalent to the total concurrent capacity. Each thread consumes some system resources, even when dormant. Therefore, for optimal overall system performance, set this setting as low as possible. Determine the correct settings empirically by adjusting them until you achieve the desired performance, or until increasing it further yields no performance improvement.
Out of the box protocols use a framework-level task execution service to minimize thread usage. For these protocols, a concurrent capacity of more than 2-3 per core is rarely useful, unless the virtual service has been highly customized.
For extensions and any virtual service that does not use an out of the box protocol, setting a long Think Time may consume a thread during the think time. In these cases, you may need to increase the concurrent capacity.
The following formula gives an approximate initial setting in these cases:

```
Concurrent Capacity = (Desired transactions per second
/ 1000) * Average Think Time in ms * (Think Scale / 100)
```

Example:
Assume that you are using a custom protocol that does not use the framework task execution service to handle think times. You want an overall throughput of 100 transactions per second. The average think time across the service image is 200 ms, and the virtual service is deployed with a 100 percent think scale.

```
(100 Transactions per second / 1000) * 200ms * (100 / 100) = 20
```

In this case, each thread blocks for an average of approximately 200 ms before responding, and during that time is unable to handle new requests. We therefore need a capacity of 20 to accommodate 100 transactions per second. A thread would become available, on average, every 10 ms, which would be sufficient to achieve 100 transactions per second.
Default: 1

- **Think Time Scale**
  Specifies the think time percentage for the recorded think time.

  > **Note:** A step subtracts its own processing time from the think time to have consistent pacing of test executions.

  Default: 100
  Examples:

  - To double the think time, use 200.
  - To halve the think time, use 50.

7. Click **Deploy**.

> **Note:** A virtual service can be in the following states:
>
> -

- **Deployed**
  No service with the name you entered is already deployed. The service is deployed.

- **Redeployed**
  A service with the name you entered is deployed with the same .vsm file as the service you entered. The service is redeployed.

- **Overridden**
  A service with the name you entered is deployed with a .vsm file that is different from the one associated with the service you entered. The application prompts you to override the deployed service.

Monitor Virtual Services

Once a virtual service has been deployed, you can monitor the virtual service from DevTest Portal.

The monitor window also has information about the creation session for each virtual service. The monitor window refers to this category of information as *recordings*.

Select **Monitor, Virtual Service Environments** in the left navigation menu. The virtual service environments (VSEs) that are running appear.

To display the monitor window for a VSE, click the VSE name.

- Monitor Window Layout
- Display Information About Virtual Services and Recordings
- Actions for Virtual Services and Recordings
- Update Virtual Service Properties
- Configure Tracking Data Cleanup

Monitor Window Layout

The **Auto Refresh** check box is in the upper right corner. Initially, the check box is selected.

The **Summary** area shows the virtual services and recordings, with totals for each of their statuses.

The following graphic shows the **Summary** area.

Screen capture of Summary area.

The toolbar shows actions that you can take on the virtual services and recordings. When an icon is dimmed, that action is unavailable for the selected artifact. You can select multiple virtual services or recordings and click the icon to perform actions on all of them.

Start

Stop

Reset Transaction Count

Remove from VSE

Update Capactiy

Update Think Scale

Update Group Tag

Update Auto Restart

Upload and Deploy a Virtual Service to the environment

The table contains the following columns:

- Check box
  Lets you select and clear virtual services and recordings.

- **Actions**
  Displays a list of actions that you can perform on the virtual service or recording.

- **Category**
  Indicates whether each row is for a virtual service or a recording.

- **Name**
  Displays the name of the virtual service or recording.

- **Resource/Type**
  Displays key information about this virtual service resource, such as what type of virtual service it is.
  With virtual services that are HTTP based, the format is **<listen port> : <http | https> : <bind address> : <base path>**. Some virtual services might not have a bind address.

- **Status**
  Identifies the current state of the virtual service or recording.

- **Txn Count**
  Identifies the number of transactions that were recorded.

- **Up-time**
  Indicates how much time has elapsed since the service was started.

- **Errors**
  Displays a red dot to indicate that errors occurred while running the service.

- **Group**
  Displays the group of the virtual service.

- **Execution Mode**
  Displays the execution mode of the virtual service.

- **Capacity**
  Defines how many virtual users (instances) can execute simultaneously with the virtual service model. This field indicates how many threads exist to service requests for this service model. You can update this field while the service is running.

- **Think Scale**
  Displays the think time percentage regarding the recorded think time.

- **Auto-Restart**
  Identifies whether to restart the virtual service automatically if the service stops running.

You can sort on column values (ascending or descending) by clicking the down arrow to the right of the column name. Use this arrow also to select the columns to display on this window.

You can search the **Name**, **Resource/Type**, and **Group** fields by entering criteria in the search box that is located in the column heading.

To select available columns to display or hide, click the filter button at the top, right of the table.

Display Information About Virtual Services and Recordings

To get more information about a virtual service or recording, click the blue i icon in the **Name** field.

For virtual services, you can display:

- **Config Name**
  Identifies the name of the configuration file for the virtual service model.

- **Resource/Type**
  Displays key information about this virtual service resource, such as what type of virtual service it is.

- **Execution Mode**
  Displays the execution mode of the virtual service. To change the execution mode, click the pencil icon.

- **Last Start**
  Displays the date and time this service was last started.

- **Last End**
  Displays the date and time this service was last stopped.

- **Group Tag**
  Displays the name of the virtual service group for this virtual service.
  If deployed virtual services have group tags, those tags are available in the field when you enter a character. A group tag must start with an alphanumeric character and can contain alphanumerics and the following special characters:

  - period (.)

- dash (-)
- underscore (_)
- dollar sign ($)

You must use the Tab or Enter key or mouse click outside the input field of the **Group Tag** field, then click **Save** to update the field. To delete a group tag from a virtual service, double-click the group tab column, then delete the value, then click **Save**.
Use the drop-down beside the Group Tag field to select a group. To clear a selected group tag, click the X. To select all virtual services with no group tag, select *Unassigned.*

- **Status**
  Identifies the current state of the virtual service.
- **Auto-Restart**
  Identifies whether to restart the virtual service automatically if the service stops running.
- **Up-time**
  Indicates how much time has elapsed since the service was started.
- **Transaction Count**
  Displays the number of transactions that VSE recorded after the service started.
- **Current txn/s**
  Displays the number of transactions currently executing.
- **Capacity**
  Defines how many virtual users (instances) can execute simultaneously with the virtual service model. **Capacity** indicates how many threads exist to service requests for this service model. You can update this field while the service is running.
- **Error Count**
  Identifies the number of errors received.
- **Peak txn/s**
  Displays the largest number of transactions that have run concurrently.
- **Think Scale**
  Displays the think time percentage regarding the recorded think time.

For recordings, you can display:

- **Resource/Type**
  Displays key information about this virtual service resource, such as what type of virtual service it is.
- **Last Start**
  Displays the date and time this recording started.
- **Last End**
  Displays the date and time this recording stopped.
- **Up-time**
  Indicates how much time has elapsed since the service was started.

- **Status**
  Identifies the current state of the recording.

- **Transaction Count**
  Identifies the number of transactions that were recorded.

- **Current txn/s**
  Displays the number of transactions currently executing.

- **Error Count**
  Identifies the number of errors received.

Actions for Virtual Services and Recordings

The **Actions** column lets you perform the following tasks:

- **Start**
  Start a virtual service or recording.

- **Stop**
  Stop a virtual service or recording.

- **View session/tracking information** Display session/tracking information for a virtual service. For more information, see View Session and Tracking Information.

- **Inspection View**

  Open a window for an inspector panel that is tailored for virtual services. For more information, see Inspection View.

  > **Note:** If a deployed virtual service shows no transactions, the client is not configured properly. Reconfigure the client to reference the virtual service instead of the real system. If another service is using that port, stop that service or change the port setting to remove the conflict. Show the inspection view for the selected virtual service.

- **Download MAR** Download the model archive (MAR) file for a virtual service.

- **Reset Txn count** Reset the transaction and error counts for a virtual service.

- **Remove from VSE** Remove a virtual service or recording from the environment.

Update Virtual Service Properties

You can update properties for virtual services in the following ways:

- Update one or more properties for a single virtual service

- Update one property for multiple virtual services

Update One or More Properties for a Single Virtual Service

You can update one or more properties for a single virtual service directly from the monitor window.

When you make a change, the **Save** and **Reset** buttons appear.

Follow these steps:

1. Double-click one of the following fields:
   - **Capacity**
   - **Think Scale**
   - **Group**
   - **Auto-Restart**
   - **Execution Mode**

     To update the execution mode, click **Edit**, rather than double-clicking the field.

   The selected field displays in edit mode.
2. Enter or select the desired value.
   The modified virtual service displays in red, indicating an unsaved change.
3. Click **Save** at the top, right of the table listing.
   - If the value you entered is not valid for the selected field, the **Save** button does not display.
   - Clicking **Reset** discards your changes without saving them.
4. Repeat these steps for any other field that you want to update.

> **Note:** You can also click **Information** in the **Name** field to view the details for a virtual service in a separate window. The fields that are listed above display in edit mode. You can make changes in this window and click **Save** or **Reset**.

Update One Property for Multiple Virtual Services

You can update the setting for the **Group**, **Capacity**, **Think Scale**, or **Auto-Restart** for multiple virtual services at once directly from the monitor window.

Follow these steps:

1. Select the check box for each virtual service that you want to update.
2. Click one of the following icons:
   - **Update Capacity**
   - **Update Think Scale**

- **Update Group Tag**
- **Update Auto Restart**

3. Update the setting.
4. Click **OK**.

Configure Tracking Data Cleanup

You can configure tracking data cleanup for a VSE. The values remain in effect until the service is stopped and restarted. When the service restarts, the values reset according to their defaults or the values in a properties file.

The following graphic shows the **Configure Settings** dialog.

Follow these steps:

1. Select **Monitor**, **Virtual Service Environments** in the left navigation menu.
   The VSEs that are running appear.
2. Right-click the VSE name and select **Configure Settings**.
3. Change one or more values.
4. Click **OK**.

Execution Mode

The execution mode is a setting that controls the runtime behavior of a virtual service.

The number of execution modes that are available depends on the steps in the virtual service. For example, if a virtual service does not have a live invocation step, you cannot use the Live Invocation or Learning Mode options.

The execution modes are divided into four categories.

The first category provides virtualized responses:

- **Most Efficient**
  This mode is the fastest mode. It does not execute the routing or tracking steps. This mode also restricts generated event tracking.

- **Stand In**
  The virtual service environment (VSE) responds like it does in Most Efficient mode, except that when it normally sends back an unknown response, it forwards the request to the live system instead.

- **Transaction Tracking**
  Fires more events than Most Efficient and remembers transaction flow through sessions. This transactional information is used to help determine why a specific response was chosen for a specific request. This mode does not perform as efficiently as Most Efficient. This mode does not show live system responses; it only shows the response from the service image.

The second category passes requests to the live system:

- **Live Invocation**
  Uses the live invocation step of the model to determine a response to the current request. Instead of using the response from the virtual service, it accesses the live service to get the response. The target system of the live invocation controls performance. This mode is also known as pass through or live system.

- **Failover**
  The VSE responds like it does in Live Invocation mode, except that when the live system call fails, the VSE responds instead.

The third category allows for updating the virtual service as needed:

- **Learning Mode**
  This mode is like Image Validation mode, but it automatically "heals" or corrects the virtual service to have the new or updated response from the live system. The next request that is passed into the virtual service automatically sees the new response that was "learned". Not only one system is being checked to learn, but both are, and the live system currently prevails.

- **Image Validation**
  Uses both the VSE and the live system to derive a response to the current request. The responses are logged for applying later to the service image using the View Session and Tracking panel. This mode allows a live comparison between the responses that the VSE provides and a corresponding live system and, where differences exist, patches or heals the service image to keep in sync with the live system. This mode is also known as live healing mode. This mode is the least efficient mode.

The fourth category allows for dynamic behavior:

- **Dynamic**
  Enables the model to determine for each request which of the other modes to use. Performance is, therefore, unpredictable. The only requirement is that the VS Routing step is present in the model after the protocol's listen step or steps.

The following table shows which transport protocols support which execution modes.

Transport Protocols	Most Efficient	Transaction Tracking	Live Invocation	Learning Mode	Image Validation	Dynamic	Stand In	Failover
HTTP/S	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
IBM MQ Native	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
IBM WebSphere MQ	Yes	Yes	No	No	No	Yes	No	No
JMS	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Standard JMS	Yes	Yes	No	No	No	Yes	No	No
RabbitMQ	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Java	Yes	Yes	Yes	Yes	Yes	Yes	No	No
TCP	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
CICS (LINK DTP MRO)	Yes	Yes	No	No	No	Yes	No	No
CICS Transaction Gateway (ECI)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
IMS Connect	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SAP RFC via JCo	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
JCo IDoc	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Opaque Data Processing	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Inspection View

The monitoring window in DevTest Portal includes an Inspection View. You can use the Inspection View to observe the matching and request event details for a deployed virtual service. This information can help you understand the behavior of the virtual service.

- Matching
- Request Event Details

Matching

The Matching subview lists the most recent requests that the virtual service processed.

For each request, the **Match Type** column indicates which of the following conditions occurred:

- No match was found for the request.
- The request matched a stateless transaction.
- The request matched a stateful transaction.

The **Match Tolerance** column indicates whether the matching transaction is a specific transaction or a default transaction.

The **Transaction ID** column contains the unique identifier of the matching transaction.

> **Note:** You can use the information in the **Match Type**, **Match Tolerance**, and **Transaction ID** columns to locate the matching transaction in the virtual service editor.

The **Arguments** column contains the actual value of each argument that is associated with the operation.

The following graphic shows an example of requests in the Matching subview. All of the requests that are visible matched a stateless transaction.

Screen capture of requests in the Matching subview.

The **Events** panel displays any events that are associated with the selected request.

The same information is recorded in the **vse_matches.log** file.

Request Event Details

The Request Event Details subview shows the list of inbound requests that caused the virtual service to error out.

When you select a request, the list of VSM steps that executed is shown, with steps containing error events selected. To see the events that occurred during processing for that step, similar to the Interactive Test Run (ITR) utility, select a step.

> **Note:** When a virtual service exceeds 100 transactions for each second, the Property Set and Property Removed events are disabled to allow for greater overall performance.

View Session and Tracking Information

DevTest Portal lets you view session and tracking information for a deployed virtual service. You can also perform model healing.

- Session Viewing
- Model Healing

Session viewing lets you see the behavior of current (or recent) sessions on a VSE server. You can determine why the response for a specific request was given. Session viewing also enables a live comparison between the responses that VSE provides and a corresponding live system.

Model healing differs from learning because learning changes a service image immediately when CA Service Virtualization and live response differences are detected. Healing logs those differences for later review and application to the service image with the **View Session/Tracking** information panel.

Session Viewing

Session viewing is available only for virtual services that run with an execution mode of Transaction Tracking or Image Validation.

To view the session tracking for a selected virtual service, click the **Actions** column for a service and select **View Session/Tracking information**.

The session panel opens to show the recorded session and transactions. If a deployed virtual service has no recorded transactions, the session panel opens with no session information.

Session Information

The **Session Information** area lists all the sessions for the selected virtual service in tabular format. To select or hide the columns that show, click the **Filter** icon at the upper right of the pane.

- **Session Status**
  Displays an icon to indicate the current session status.
  - **Matched**
    Indicates that there is an exact transaction match between the virtual service and the live system. *Matched* also indicates that the response was successfully recorded, and the recorded response does not differ from the live system response.
  - **Body Mismatched**
    Indicates that there is a match between the virtual service and the live system. However, the recorded response body differs from the live system.
  - **Navigation Mismatched**
    Indicates that conversational transaction navigation is different between the virtual service recording and the live system. For example, the response was not found in the virtual service, but the response is played back from the live system.
- **Session ID**
  Identifies the unique ID for each session.
- **Created On**
  Displays the time stamp of the first transaction in that session.
- **Modified On**
  Displays the time stamp of the most recent transaction.
- **Transaction Count**
  Identifies the number of transactions that were recorded after the service started.
- **Client ID**
  (Protocol-specific) For HTTP, displays the endpoint of the client that submitted the transaction.
- **Most Recent Request**
  Identifies the most recent request that came through on the specific session.

Response Information

The **Response Information** area shows the list of transactions for the selected session. When you point the mouse to the colored ball in the first column of this pane, a tooltip identifies the match for that transaction. If you click any specific transaction, the Request and Response tabs compare request/response between the VSE system and the live system.

In the **Response Information** pane, the following icons represent transactions:

- **Matched**
  Indicates a response body and navigation match between the live system and VSE image.

- **Body Mismatched**
  Indicates a response body mismatch with a navigation match.

- **Navigation Mismatched**
  Indicates a navigation mismatch.

- - For conversational transactions:
    Once a transaction is mismatched or goes to the live system, all subsequent transactions are also mismatched and go to the live system. Once a conversation goes to the live system for a conversation, the VSE image lacks the context to switch the conversation back to the VSE image.

  - For stateless transactions:
    Once a transaction is mismatched or goes to the live system, subsequent transactions still match against the VSE image first to find a response. The response is not played back from the live system if the response exists in the VSE image.

Model Healing

Use the **Download Updated VSI** button in the Session Information area to update the service image with live session/stateless transactions. This process is referred to as model healing. You use model healing to remove the disparity between the VSE image and the live system so that the virtual service works correctly. Select the transactions to update by selecting the check box to the left of the transaction, then clicking **Download Updated VSI** in the Response Information area.

Create Copybook Bundles

The Copybook Bundle page makes it easier to create, import, and manage your copybooks and mapping files. This page:

- Allows you to view and edit the code directly for your mapping files.
- Provides automatic error checking and lets you parse your copybooks before using them in a recording.
- Bundles the copybooks and the associated mapping files together.
- Reduces errors during the recording process.

A copybook bundle is a collection of copybooks and the associated mapping file for those copybooks.

> **Note:** The Copybook Bundle page lets you manage your copybook files. However, to apply them to recorded transactions, you must use the existing procedures in the DevTest Workstation. For more information about using the copybook data protocol with a recording, see Copybook Data Protocol and How to Use the Copybook Data Protocol.

- Access the Copybook Bundle Page
- Create a Copybook Bundle
- Import a Copybook
- Create a Copybook
- Import a Mapping File
- Create a Mapping File
- Find Your Saved Copybook Bundle

Access the Copybook Bundle Page

Follow these steps:

1. From DevTest Portal, click **Manage, Copybooks**.
2. Click the name of the copybook bundle that you want to open.
   The Copybook Bundle tab opens.

When searching for a specific copybook bundle, copybook, or mapping file, click **FILTER** to limit your results by name, text, or status.

Create a Copybook Bundle

Follow these steps:

1. From DevTest Portal, click **Create, Copybook Bundle**.
   The Copybook Bundle tab opens.
2. Enter a valid name for your copybook and click **Save** .
   A new tab opens where you can define the details of your new copybook.
3. Import or create the copybooks that you want to include in your copybook bundle.
4. Import or create the mapping file for your copybook bundle.
5. Click **Comment** to add comments or notes to the copybook bundle.
6. Click **Change Start/End Columns** to define start and end columns and code page.
   - **Start Column**
     Copybooks frequently start each line with a line number. This parameter defines the column on which the parser starts when trying to parse a copybook file definition.

- **Value:** A zero-based inclusive index. However, you can think of it as a "normal" one-based exclusive index.
  Default: 6
  Example: If you set this value to 6, the parser skips the first six characters in a line and starts with the seventh character.

- **End Column**
  Occasionally, copybooks contain other reference data at the end of each line. When that happens, the parser must know on which column to stop. If there is no "extra" data at the end of the lines in the file, you can set this number to something greater than the length of the longest line in the file. If this number is greater than the length of a line, the parser stops at the end of the line.
  Value: A zero-based exclusive index. However, you can think of it as a "normal" one-based inclusive index.
  Example: If you set this value to 72, the parser reads the 72nd character in the line and then it stops (without trying to read the 73rd).

- **Codepage**
  The codepage defines the encoding of the recorded transaction.

> Note: The Start Column, End Column, and Codepage values are required for parsing in the Copybook Bundle page. These values are not transferred to the DevTest. You must modify these values in the DevTest if they differ from the default values.

7. Click **Reload** to reload the page and discard unsaved changes.

8. Click **Status** to view the number of copybooks in your bundle by status (Parsed, Error, Not Parsed).

9. Click **Save** at the top of the page to save your copybook bundle.

Import a Copybook

The Copybook Bundle page lets you import an existing copybook that you can then use in your recordings.

Follow these steps:

1. Open an existing copybook bundle, or click **Create, Copybook Bundle** in DevTest Portal.

2. The Copybook Bundle tab opens.

3. Click **Import** in the Copybooks panel and browse to the location of the copybook you want to import.

4. Click **Open**.
   The contents of the selected copybook display in the Selected panel.

5. Click **Parse** to parse the selected copybook.
   The results display in the Parsing Result panel.

   - A green check mark next to the copybook name indicates that the copybook parsed successfully.
   - A red circle with an X indicates that there were errors in the parsing.
   - A black dash indicates that the copybook has not been parsed.

6. Make any desired changes or corrections to the copybook.
7. Import any additional copybooks that you want to include in this bundle.
8. Import or create the mapping file for your copybook bundle.
9. Click **Save** at the top of the page to save your copybook bundle.

Create a Copybook

If you do not have an existing copybook to import, you can create one in the Copybook Bundle page.

Follow these steps:

1. From DevTest Portal, click **Create, Copybook Bundle**.
   The Copybook Bundle tab opens.
2. Click **Add** in the Copybooks panel.
3. Use the text editor in the Selected Copybook.txt panel to enter the content for your copybook.
4. Click **Save** at the top of the page to save your copybook bundle.

Import a Mapping File

The Copybook Bundle page lets you import an existing mapping file. You can associate this file with one or more copybooks.

Follow these steps:

1. Open an existing copybook bundle, or click **Create, Copybook Bundle** in the DevTest Portal.
   The Copybook Bundle tab opens.
2. Click the **Mapping File** tab.
3. Click **Import** and browse to the location of the mapping file you want to import.

4. Click **Open**.
   The contents of the mapping file display.

5. Make any desired changes or corrections.
   For more information about specific fields or sections, see Create a Mapping File.

   > **Note:** Click **Toggle** in the Mapping Entries panel to view the entire mapping file.

6. Click **Save** at the top of the page to save your copybook bundle.

Create a Mapping File

If you do not have an existing mapping file to import, you can manually create one in the Copybook Bundle page.

> **Note:** As you create your mapping file, the Mapping File Entry panel shows your entries in a mapping file format. To see the entire mapping file, click **Toggle** in the Mapping Entries panel.

Follow these steps:

1. From DevTest Portal, click **Create, Copybook Bundle**.
   The Copybook Bundle tab opens.

2. Click the **Mapping File** Tab.

3. Click **Add** in the Mapping Entries panel.

4. Complete the following fields in the Payload Configuration Editor:

   - **Type**
     Specifies the payload type. Select either **Request** or **Response**.

   - **Name**
     Defines a unique name to identify the type of request or response. The name must be unique among the set of payloads of the same type. For example, you can have a request and response with the same name, by you cannot have two requests with identical names.

   - **Match Type**
     The Match Type attribute applies differently to payload definitions with a "Response" type. For example, responses do not contain arguments or operation names. If a payload "Response" sets this attribute to **Argument, Attribute**, or **Operation**, it never matches anything. This is true unless the matching request selects **Defines Response**,

which overrides this attribute. If a payload "Response" sets this attribute to **All**, the argument, attribute, and operation phases of matching are skipped.
Value: One of the following:

- **All:** Try to match in the following order: Argument, Attribute, Meta Data, Operation, then Payload.
- **Meta Data:** Try to match only the specified Meta Data.
- **Payload:** Try to match on the body of the request.
- **Attribute:** Try to match only the specified attribute.
- **Operation**: Try to match on only the operation name.
- **Argument:** Try to match on only the specified argument.

Default: Payload

- - **Key**

The behavior of this attribute depends as follows on the Match Type:

- - - If the Match Type is **operation** or **payload**, the value of this attribute is used for matching.
    - If the Match Type is **argument**, **attribute**, **metaData**, or **all**, the value of this attribute is assumed to be the key of an argument, attribute, or metaData entry.
    - If the Match Type value is **all**, the key value is NOT used for matching against the operationName or the payload body. The value attribute is used instead.

- - **Value**

The behavior of this attribute (and whether it is required) depends as follows on the Match Type:

1. - - If the Match Type is **Operation** or **Payload**, the value of this attribute is ignored and can be excluded.
    - If the Match Type is **Argument, Attribute, or Meta Data**, this attribute is assumed to be the value of an argument, attribute, or meta data entry and is required for matching.
    - If the Match Type is **All**, this attribute is required and behaves as described previously during the argument, attribute, and Meta Data matching phases. During the operation and payload matching phases, however, the value of this attribute is used for matching (as opposed to using the value of the **Key** attribute as is the case if the Match Type is **Operation** or **Payload**).

   - **Defines Response**
     If selected, the response for this request looks for a payload element with an identical name, but of type "Response." This attribute is ignored when the type is Response.
     Default: Cleared.

   - **Header Bytes**
     Click to define the header and footer bytes (optional).

- **Header Bytes**
  Designates the number of bytes to strip from the beginning of the payload. This value defaults to 0 if you do not provide a value. If the attribute is present, the value must be a valid integer.

- **Footer Bytes**
  Designates the number of bytes to strip from the end of the payload. This value defaults to 0 if you do not provide a value. If the attribute is present, the value must be a valid integer.

- **Save Header/Footer**
  Specifies whether the header and footer bytes that were stripped are persisted in the XML version of the request as hex-encoded strings under the rawHeader and rawFooter tags.

  - **Selected:** Persists the stripped header and footer bytes.

  - **Cleared:** Does not persist the stripped header and footer bytes.

  Default: Cleared.

2. In the Section panel, click the section of your entry that you want to associate with one or more copybooks.

   - Use the **Up** and **Down** arrows to rearrange sections.

   - Click **Add** to add more sections.

3. Select the copybooks to associate with the selected section.

   - Click the **Right Arrow** in the **All Copybooks** column to move the selected copybook to the **Added Copybooks** column.

   - To remove a copybook from this mapping file, click the **Left Arrow** in the **Added Copybooks** column to remove the selected copybook.

4. The Copybook Configuration Elements panel let you define more configuration elements for your copybook, if required.
   Complete the following fields:

   - **Key**
     Defines the unique string in the record that identifies the copybook. Technically, this attribute is optional. However, if a key is not provided, it means that that copybook is the only one that will ever get applied to the payload. It is applied over and over until the payload runs out of bytes. If multiple copybook elements have no key, then the first one is always used, unless the max attribute is specified.

   - **Order**
     Defines a hint as to the order in which the records are found in the payload. The numbers that are used are irrelevant, but "later" records in the payload should use a larger integer. Multiple copybooks can be tagged with the same order number, meaning that those records could be in any order. When a record has been found with a specific order number, subsequent searches only search for copybooks with that

order number and greater. You can include copybooks in a group that will never match against the payload. They are ignored. However, this affects performance because each copybook has to be checked.
Default: 0

- **Max**
  Defines the maximum times that a copybook can be applied to the payload. Blank values, 0, negative numbers, nonnumbers, and nonexistent values all mean "no limit".

- **Name**
  Defines a value to override the record name (that is, the root level in the copybook).

  - If you set this value, the generated node in the XML for this copybook uses this name instead of the record name from the copybook definition.

  - If you do not set this value, the default is to look up the record name from the copybook definition and use that.

  - If you set this value to the record name from the copybook definition, the only effect is on readability of this file.

- **Length Field**
  Defines how to split the payload so that the next record search begins in the correct place.
  If this attribute is not present, the processor attempts to determine the length of the copybook from the definition. If, for some reason, it cannot figure out the length, the processor assumes that the rest of the payload applies to this copybook, and ends processing after applying this copybook. The processor ignores this field if it is not an unsigned Display numeric field.

5. Repeat this process for more entries in this mapping file.

   - Use the **Up** and **Down** arrows to rearrange the entries in your mapping file.

   - Click **Delete** to remove the selected entry.

6. Click **Save** at the top of the page to save your copybook bundle.

Find Your Saved Copybook Bundle

When you create a copybook bundle, a folder is created with the name you specified for the copybook bundle. This folder resides in your Projects folder, and it contains the individual copybook and mapping files. To use this bundle to parse recorded transactions, you must use the existing procedures in the DevTest Workstation. For more information about using the copybook data protocol with a recording, see Copybook Data Protocol and How to Use the Copybook Data Protocol.

Manage Copybook Bundles

To manage copybook bundles, access the Copybooks, Mapping File, and Payload Mapping tabs from **Manage, Copybooks**. The Copybooks and Mapping File tabs are documented on the Create Copybook Bundles page.

Map a Payload

The **Payload Mapping** tab lets you see how a copybook maps a particular payload file. Use the **Payload Mapping** tab to debug issues with your copybook bundle.

Payload Mapping tab

Follow these steps:

1. From DevTest Portal, click **Manage**, **Copybook Bundles**.
2. Select a copybook bundle from the list and click the pencil icon to edit it.
3. Click the **Payload Mapping** tab.
4. Click the blue folder icon to import payload data from a file.
5. Select the payload data file from the file system.
   Supported data types are byte data or base 64 string (byte data that has been encoded to byte64 string).
6. (Optional) Select a code page from the Encoding list to change encoding for the file.
7. (Optional) Use the Toggle icon (two horizontal arrows) to decode Base64 files or to undo the last decoding option you chose.
8. (Optional) Use the Copybook Toggle icon to toggle between mapping the payload with the mapping entry or mapping with copybooks.
   To select copybooks from the Available Copybooks list, click the arrow beside them to move them into the Selected Copybooks column.
9. To map the payload data to the selected copybook, click the Map Payload icon.

Create Baseline from Virtual Service

You can quickly create a baseline during or after the creation of a virtual service.

Create Baseline at the Same Time as Virtual Service

This approach is not available for all protocols.

To create a baseline at the same time as a virtual service, select the **Create baseline tests** option in the Save pane.

Create Baseline from Existing Virtual Service

You can create a baseline from virtual services that are created with DevTest 9.0 or later.

The **Create Baseline Test** option is disabled for unsupported virtual services.

When you run the baseline, be sure to select **Run with Options**. Then ensure that the values of WSSERVER and WSPORT in the configuration file are set correctly. In some cases, the default value of WSSERVER is CHANGE_ME.

Follow these steps:

1. Go to the home page of DevTest Portal.
2. Select **Manage**, **Virtual Services** in the left navigation menu.
3. Locate the virtual service in the table.
4. Click the **Options** button in the **Actions** column and select **Create Baseline Test**.

Virtual Service Metrics Reports

Virtual Service Metrics reports contain information about virtual services and the corresponding virtual service environment. The reports can be searched, viewed, managed, shared, and exported to PDF and CVS file. The following reports are available:

- Transaction Daily Counts
- Transaction Hits and Misses
- Virtual Service Forced Delay
- Virtual Service Response Time
- Virtual Service Transaction Per Second
- VSE Lifetime Total Transactions
- VSE Server Availability
- VS Availability

For more information about viewing and manipulating reports, see View and Navigate Reports.

Transaction Daily Counts

The Transaction Daily Counts report displays the number of transactions in a defined time period for the virtual services within a selected virtual service environment. Each virtual service displays as a single bar within the graph. This report allows you to review your system loads by virtual service.

Screen capture of the Transaction Daily Counts report

Transaction Hits and Misses

The Transaction Hits and Misses report displays the number of transactions that were performed on a virtual service for a given time period. The report displays a bar chart with the total number of the following types of transactions:

- **Node Hits**
  A node hit occurs when the node matches a meta or signature response in a service image.

- **Specific Hits**
  A specific hit occurs when the request matches a specific transaction.

- **Misses**
  A miss occurs when the request is unknown and the request does not match a signature or exact match.

This data in this report is limited to a single virtual service.

Click an item in the legend at the bottom of the report to hide or show the corresponding data. For example, click **Misses** to hide that data in the graph. When data is hidden, the legend item turns gray.

Screen capture of the Transaction Hits and Misses report

Virtual Service Forced Delay

The Virtual Service Forced Delay report helps you determine how a virtual service is performing and its impact to system capacity. Negative numbers in the report indicates that your VSE server is out of capacity. To remedy this situation, you should remove models from the virtual service or bring up another VSE.

This report displays the forced delays for a single virtual service within a VSE. It displays the number of milliseconds before the virtual service sent a result. A positive number means that the virtual service deliberately waited to send a result because of the specified think time. For example, if think time was 10 ms and the time to listen and respond was 8 ms, the virtual service waits 2 ms. A negative number means that virtual service took longer to generate the result than the specified think time.

The Virtual Service Forced Delay report displays the maximum, average, and minimum forced delays, in milliseconds, in the form of a line graph. The forced delays are captured according to a defined sampling interval. Each data sample is represented by a square block on the report. The sampling interval defaults to every 5 minutes, but you can modify the interval by setting the **lisa.vse.metrics.sample.interval** property in your local or site properties file. For more information about defining a property, see Custom Property Files.

Click an item in the legend at the bottom of the report to hide or show the corresponding data. For example, click **Average** to hide the data averages in the graph. When data is hidden, the legend item turns gray.

Screen capture of the Virtual Service Forced Delay report

Virtual Service Response Time

The Virtual Service Response Time report establishes the current health of a specific virtual service. This report is useful for determining how different system loads or performance testing impacts the response times for a virtual service.

> **Note:** Response times in this report include think time. This means that each response time includes the amount of time required to process the request and the specified think time.

The Virtual Service Response Time report displays the displays the maximum, average, and minimum response times, in milliseconds, in the form of a line graph. The response times are captured according to a defined sampling interval. Each data sample is represented by a square block on the report. The sampling interval defaults to every 5 minutes, but you can modify the interval by setting the **lisa.vse.metrics.sample.interval** property in your local or site properties file. For more information about defining a property, see Custom Property Files.

Click an item in the legend at the bottom of the report to hide or show the corresponding data. For example, click **Average** to hide the data averages in the graph. When data is hidden, the legend item turns gray.

Screen capture of the Virtual Service Response Time report

Virtual Service Transaction Per Second

The Virtual Service Transaction per Second report establishes the current health of a specific virtual service. This report is useful for determining how different system loads or performance testing impacts the transactions per second for a virtual service.

This report displays the displays the maximum, average, and minimum transactions per second in the form of a line graph. The transactions per second are captured according to a defined sampling interval. Each data sample is represented by a square block on the report. The sampling interval defaults to every 5 minutes, but you can modify the interval by setting the **lisa.vse.metrics.sample.interval** property in your local or site properties file. For more information about defining a property, see Custom Property Files.

Click an item in the legend at the bottom of the report to hide or show the corresponding data. For example, click **Average** to hide the data averages in the graph. When data is hidden, the legend item turns gray.

Screen capture of the Virtual Service Transaction Per Second report

VSE Lifetime Total Transactions

The VSE Lifetime Total Transactions report helps you confirm the consumption of system resources by virtual services. This report displays the historical transaction counts for a VSE and the virtual services running on that VSE for a specified time period. The transaction counts include all transactions in the database for the selected virtual services on a specific VSE for the specified time period.

The first bar on the left side of the report shows the total transaction count for the VSE. The remaining bars show the transaction counts for each virtual service running on that VSE. You can select the VSE and the virtual services that display in the report from the **Refine By** pane to the right of the report data.

Screen capture of the VSE Lifetime Total Transactions report

VSE Server Availability

The VSE Server Availability report displays the availability of a VSE server for a specified time period. This report shows the status of the VSE server, up or down, in a single bar for each summary period. You can chose to summarize the data by day, week, or month in the **Refine By** pane to the right of the report data.

Each color-coded bar shows the uptime and downtime for the VSE server as a percentage of the summary period (day, week, or month). Hover your mouse over each section of a bar to see the uptime and downtime in seconds. Click one of the bars to see additional availability information, by virtual service, in the VS Availability report.

The second page of the report shows the seconds and percentages of uptime and downtime for each summary period in the report in a table format.

Screen capture of the VSE Server Availability report

VS Availability

The VS Availability report displays the availability of the virtual services on a single VSE for a specified time period. This report shows the status of the virtual service, up or down, in a single bar for each summary period. The summary period (day, week, or month) is defined in the VSE Server Availability report.

Each color-coded bar shows the uptime and downtime for the virtual service as a percentage of the summary period. Hover your mouse over each section of a bar to see the uptime and downtime in seconds. The second page of the report shows the seconds and percentages of uptime and downtime for each virtual service in the report.

Screen capture of the VS Availability report

More Information:

- View and Navigate Reports
- Testing Execution Reports
- Testing Performance Reports
- CAI Top N and Metrics Reports
- User Activity Report

Using DevTest Workstation with CA Service Virtualization

DevTest Portal is a web-based application that provides simpler access to the most commonly used workflows for DevTest products. DevTest Workstation is the main application for advanced users and contains the full range of functionality available within the DevTest products.

This section contains the following pages:

- Creating Service Images
- Editing Service Images
- Editing a VSM
- De-identifying Data
- Virtualizing a Service
- Track Transactions

Creating Service Images

The **Virtual Service Image Recorder** generates service images. Service images pretend to be what you recorded (a manipulated or altered version of your recorded raw traffic).

This section contains the following pages:

- Open a Service Image
- Combine Service Images
- Delete a Service Image
- Create a Service Image
- Create a Service Image from Scratch
- Create a Service Image from a WSDL
- Create a Service Image from WADL
- Create a Service Image from RAML
- Create a Service Image from Swagger (OpenAPI)
- Create a Service Image from Layer 7
- Create a Service Image from Request-Response Pairs
- Create a Service Image from PCAP
- Create a Service Image by Recording
- Work with VSMs
- Using Data Protocols

Open a Service Image

The **Virtual Service Image Recorder** generates service images. Service images pretend to be what you recorded (a manipulated or altered version of your recorded raw traffic).

> **Note:** If you have service images from releases of VSE earlier than LISA 6.0, export those service images. Exporting moves them from the database of earlier versions to the file system in which they are stored in the current release. For more information, see Legacy Service Images.

Opening a service image allows you to change the image in the **Service Image Editor**.

Follow these steps:

1. Right-click the service image in the **Project** panel and select **Open**.
   The **Service Image Editor** opens.

2. Review the selected service image and make any changes.

Combine Service Images

Combining service images is useful when you want to add functionality to an existing service image.

Follow these steps:

1. Right-click the service image on the **Project** panel and select **Combine**.
   The **Combine Service Images** dialog opens.

2. Select one or more service images to combine with the original image (the target).

3. To replace what is in the matching target data, select the **Favor source image(s)** check box.
   When you combine service images, each stateless transaction (at the Meta level) from each source service image is matched against each one in the target service image. For each one that matches, the specific transactions from the source are matched against the ones in the target. When no matches are found, the source transactions are added to the target image. When they do match, the transactions must be merged.
   You have the choice of having the source data (for example, response bodies) replace what is in the matching target transaction. You can also leave the target data as-is.

> **Note:** The process for combining conversations is similar. For each conversation in each source service image, the starter transaction is matched against each starter in the conversations of the target. If no starter matches, new conversations are created in the target image. If they do match, the same process for the stateless list is applied to each Meta node in the source conversation tree. The process adds new transactions and merges matching transactions as appropriate.

Delete a Service Image

You can delete a service image in DevTest Workstation.

Follow these steps:

1. Right-click the service image on the **Project** panel and select **Delete**.
   A confirmation dialog asks you to confirm that you want to delete the selected service image.

2. Click **OK**.
   The selected service image is permanently removed from your project.

> **Note:** As a best practice, think of service images as transient and remove unused or outdated service images.

Create a Service Image

To create a service image:

1. Right-click **VirtualServices**, **Images** on the **Project** panel.

2. Select **Create New VS Image**.

Import Raw Traffic

You can only import raw traffic through the VSE Service Image Recorder.

Follow these steps:

1. Create a service image and select **Virtual Service Image Recorder**.

2. Enter the raw traffic file name or select it from the file browser.

3. To load the parameters for the service image from a previously saved recording session, click the blue folder at the bottom corner of the panel.
   You are prompted to browse for a .vrs file with which to load the parameters.

Image of the Virtual Service Image Recorder page, highlighting the Import Traffic field

4. To continue the recording process normally, select a transport protocol.
   When you start the recording process, VSE imports the raw traffic to a new service image.

More Information:

- Create a Service Image from Scratch
- Create a Service Image from a WSDL
- Create a Service Image from Request-Response Pairs
- Create a Service Image from PCAP
- Work with VSMs
- Using Data Protocols

Create a Service Image from Scratch

To create a service image from scratch (manually):

1. Right-click **VirtualServices** on the **Project** panel and select **Create a New VS Image**, **From scratch**.

2. Enter the identification and protocol information, and click **OK**.
   The **Service Image Editor** window opens.

3. Enter the specific parameters and information for the new service image.

Create a Service Image from a WSDL

You can generate a virtual web service image from a WSDL document in the following ways:

- From the Quick Start Menu
- From Create New VS Image

From the Quick Start Menu

This procedure describes how to create a virtual service image from a WSDL document using the UI.

Follow these steps:

1. Select **Create an SI from a WSDL** from the **Quick Start** menu.

2. Enter the name of a WSDL on the **Connection** tab.

   Click **Utilities** to find WSDLs on your system.
   After you enter the WSDL name, the **Service** and **Port** fields are populated. The associated operations are listed on the **Operations** tab. You can select **All** or **None**, or you can use the check boxes to select specific operations to test.

3. Enter the name of the service image to create in the **Save to** field at the bottom of the window.

   Note: If the default service image name is already in use, a warning icon appears.

4. Click the green arrow at the bottom of the window.
   The **Service Image Editor** opens with your service image displayed.

From Create New VS Image

This procedure describes how to create a virtual service image from a WSDL document using the **New VS Image** option.

Follow these steps:

1. Right-click **VirtualServices**, **Images**, on the **Project** panel and select **Create New VS Image**, **From WSDL**.
   The **Virtual Service From WSDL** window opens.

2. Enter a service image name and the name of a VS model file.

3. Accept the default values for the remaining fields on this window.

> **Note:** To load parameters from a previously saved service image, click **Load from File** at the bottom of the window.

4. Click **Next**.
   The **Connection** tab opens.

5. Enter the port number to which the virtual service listens in the **Listen on port** field at the bottom of the window.

6. Add the WSDL to be virtualized in the **WSDL URL** field.
   This entry can be a local file or a URL.

7. Select the service in that WSDL that must be virtualized, in the **Service** field.
   Usually, only one selection is available.

8. Select the operations in that service to virtualize.
   By default, all the operations are selected.

9. Click **Next**.
   The request/response side data protocols options open.

10. Select **Web Services (SOAP)**.
    The Request Side Data Protocols list is prepopulated with Web Services (SOAP) and the XML data protocol handlers. The **Response Side Data Protocols** list is automatically populated with the Delimited Text data protocol.

11. Click **Next**.

12. See Delimited Text Data Protocol for information about how to configure the Delimited Text data protocol. When you have configured the Delimited Text data protocol, click **Next**.
    On the next window, the service image is generated and the wizard is finished.

13. Click **Finish**.

> **Note:** To save the settings on this recording to load to another service image recording, click **Save** above the **Finish** button.

The blank virtual service model is populated with steps.

14. Save the virtual service model.
    The service image that was generated is a stub service. The service image returns correctly formatted responses, but the values are default values.
    The Service Model (VSM) that was saved is the model that is deployed to the Virtual Service Environment.

Create a Service Image from WADL

This procedure describes how to create a virtual service image from Web Application Description Language (WADL).

Follow these steps:

1. Complete one of the following options:

   - Right-click **VirtualServices** on the **Project** panel and select **Create New VS Image**, **From WADL**.

   - From the **File** menu, select **New, VS Image, From WADL.**

   The **Virtual Service From WADL** window opens.

2. Enter a service image name and the name of a VS model file.

   > **Note:** To load parameters from a previously saved service image, click **Load from File** at the bottom of the window.

   For more information about field descriptions, see Basics Tab.

3. Click **Next**.

4. In the **Listen on port field** at the bottom of the window, enter the port number to which the virtual service listens.

5. In the **WADL URL** field, add the WADL to virtualize.
   This entry can be a WADL file on the file system or a URL.

6. Click **Refresh WADL Cache** .

7. In the **Endpoint** field, select the endpoint in the WADL that must be virtualized.

   > **Note**: Usually, only one selection is available.

8. In the **Methods** pane, select the methods in the endpoint to virtualize.
   By default, all the methods are selected. You can click **Select All** or **Select None**, as appropriate.

9. Click **Next**.

10. Add or chain other Data Protocol Handlers as appropriate. By default, the Rest Data Protocol Handler is selected.

11. Click **Next**.
    On the next window, the service image is generated and the wizard is finished.

12. Click **Finish**.
    To save the settings on this recording to load to another service image recording, click **Save** above the **Finish** button.

Note: The parameter lisa.vse.rest.max.optionalqueryparams specifies the maximum number of optional query parameters to process per method in a WADL file. The default is five; any optional parameters after the fifth one are ignored. We recommend that you do not change the value above five. This can result in the number of generated responses growing exponentially after the fifth one.

Create a Service Image from RAML

This procedure describes how to create a virtual service image from RESTful API modeling language (RAML).

A RAML can define message bodies using a combination of Schema and Example properties. The Example property is used for the transaction body in DevTest. Make sure to specify the Example property in a message body, or else the Schema property is used instead.

Important! The Schema property is not interpreted in any way and appears exactly as specified in the transaction body within DevTest.

Follow these steps:

1. Complete one of the following options:
   - Right-click **VirtualServices** on the **Project** panel and select **Create New VS Image**, **From RAML**.
   - From the **File** menu, select **New, VS Image, From RAML.**

   The **Virtual Service From RAML** window opens.

2. Enter a service image name and the name of a VS model file.

   Note: To load parameters from a previously saved service image, click **Load from File** at the bottom of the window.

   For more information about field descriptions, see Basics Tab.

3. Click **Next**.

4. In the **Listen on port** field at the bottom of the window, enter the port number to which the virtual service listens.

5. In the **RAML URL** field, add the RAML of the web service to virtualize.
   You can also select the RAML from the drop-down list, or click **Browse** to locate the RAML from the file system.
   This entry can be a RAML file on the file system or a URL.

6. Click **Refresh RAML Cache** .
   DevTest parses the RAML and populates the Endpoint field and the Methods pane.

> **Note**: If DevTest fails to parse the RAML, a warning icon displays after the Endpoint field. To view the error message, click the warning icon.

7. In the **Methods** pane, select the methods to virtualize.
   By default, all the methods are selected. You can click **Select All** or **Select None**, as appropriate. At least one method is required.

8. Click **Next**.

9. Add or chain other Data Protocol Handlers as appropriate. By default, the Rest Data Protocol Handler is selected.

10. Click **Next**.
    On the next window, the service image is generated and the wizard is finished.

11. Click **Finish**.
    To save the settings on this recording to load to another service image recording, click **Save** above the **Finish** button.

> We support parsing any external files that are referred in the RAML file ***include tag*** only if they are present in the path relative to the RAML file.
>
> **Examples**
>
> 1. If the RAML file is present in **C://%LISA_HOME%/Data/test.raml** and it references an external file external.yaml in its include tag as follows:
>
> traits: !include external.yaml
>
> Then the external.yaml file should be present at **C://%LISA_HOME%/Data/external.yaml.**
>
> 2. If the RAML file is present in **C://%LISA_HOME%/Data/test.raml** and it references an external file present at path dependencies/external.yaml in its include tag as follows:
>
> traits: !include dependencies/external.yaml
>
> Then the external.yaml file should be present at **C://%LISA_HOME%/Data/dependencies/external.yaml.**

> **Note**: The parameter lisa.vse.rest.max.optionalqueryparams specifies the maximum number of optional query parameters to process per method in a RAML file. The default is five; any optional parameters after the fifth one are ignored. We recommend that you do not change the value above five. This can result in the number of generated responses growing exponentially after the fifth one.

Create a Service Image from Swagger (OpenAPI)

This procedure describes how to create a virtual service image from a Swagger (OpenAPI) 2.0 or 3.0 document.

> **Note:** You can create a virtual service from an OpenAPI 3.0 document using the same features that were present in Swagger 2.0. New features that were added as part of OpenAPI 3.0 are not supported.

The Swagger document can contain internal and external references. If you include an external reference, DevTest Workstation must be able to reach it.

You can test this feature with the Forward Cars demo application. Forward Cars includes a Swagger 2.0 document that describes the inventory service. The document is available at the following URL:

```
http://localhost:3500/cars-inventory/swagger.json
```

If Forward Cars is running on another computer, replace **localhost** with the host name or IP address of the other computer.

The Swagger document is also available in the **Data** folder of the **examples** project. The file name is **cars-inventory-swagger.json**.

Follow these steps:

1. Complete one of the following options:

   - Right-click **VirtualServices** on the **Project** panel and select **Create New VS Image**, **From Swagger Specification**.

   - From the **File** menu, select **New, VS Image, From Swagger Specification.**

   The **Virtual Service From Swagger Specification** window opens.

2. Enter a service image name and the name of a VS model file.

   > **Note:** To load parameters from a previously saved service image, click **Load from File** at the bottom of the window.

3. For more information about field descriptions, see Basics Tab.

   > On the Basics tab, you can specify whether or not to check for duplicate transactions. **The Allow duplicate specific transactions** checkbox specifies whether DevTest can respond more than once to the same call, choosing a different response. Round-robin matching only happens if this check box is selected.

4. Click **Next**.

5. In the **Listen on port** field at the bottom of the window, enter the port number to which the virtual service listens.

6. In the **URL** field, enter a URL or browse to the file that contains the document.

7. Click **Refresh Swagger Cache** .
   If the Swagger parsing is successful, the **Endpoint** field and **Methods** pane are populated with information from the Swagger document. You cannot modify the **Endpoint** field.

8. In the **Methods** pane, select the methods in the endpoint to virtualize.
   By default, all the methods are selected. You can click **Select All** or **Select None**, as appropriate.

9. (Optional) Change the MIME type of the request and response.

10. Click **Next**.

11. Add or chain other data protocols for this virtual service image from the Swagger document as appropriate. By default, the REST data protocol is selected.

12. Click **Next**.
    On the next window, the service image is generated and the wizard is finished.

13. Click **Finish**.
    To save the settings on this recording to load to another service image recording, click **Save** above the **Finish** button.

Note: The parameter lisa.vse.rest.max.optionalqueryparams specifies the maximum number of optional query parameters to process per method in a Swagger document. The default is five; any optional parameters after the fifth one are ignored. We recommend that you do not change the value above five. This can result in the number of generated responses growing exponentially after the fifth one.

All query arguments and request header values are included in the generated virtual service regardless of the value of the require flag that is indicated in the Swagger document being parsed.

So, for example, in the following snippet of a Swagger document:

```
{

"name": "patentDocId",

"in": "query",

"description": "patent id to filter the results (eg. 05569676)",

"required": "false",
```

"type": "string",

},

The query argument **patentDocId** has a required flag equal to *false*. Regardless of that condition, the indicated query argument will be included as an argument on the generated virtual service.

All query arguments are included in the virtual service. You must edit the virtual service to remove any query arguments you do not want to include.

The following example
schema:{ example:{ }}is not supported. The format that is supported is:schema: {}examples:{ <mime type>:{ }}

More Information:

- Tutorial 6: Virtual Service from Swagger (OpenAPI)

Create a Service Image from Layer 7

The **Virtual Service From Layer7** option generates a virtual service image from Layer 7.

Prerequisites:

1. Download version 2.4 or 2.5 of the Layer 7 Command-line Migration Tool from the Layer 7 support website.
   Version 2.4 is compatible with CA API Gateway 8.0, 8.1, and 8.2.
   Version 2.5 is compatible with CA API Gateway 8.0, 8.1, 8.2, and 8.3.

2. Extract the JAR file.

   Note: The JAR file contains the **cmt2.jar** file that is required for step 3 of the following procedure. This file can be located anywhere.

3. Set **JAVA_HOME** to a version of JDK 1.7.

Follow these steps:

1. Do one of the following:

   - Select File, New, VS Image, From Layer7

   - Right-click the root node of the project and select Create New VS Image, from Layer7.

   - Right-click the Virtual Services Images folder and select Create New VS Image, Layer7.

The **Virtual Service From Layer7** window opens.

2. Enter the **Layer7 Connection Info** fields.

3. Click the folder icon to locate the cmt2.jar file and click **Get Layer7 Services**.
   The **Layer7 Services** tab displays the available services.

4. Select a service and click **Next**.

5. Enter a service image name and the name of a VS model file.
   Accept the default values for the remaining fields on this window.

   > **Note:** To load parameters from a previously saved service image, click **Load from File** at the bottom of the window.

6. Click **Next**.
   The **Connection** tab opens.

7. Select the operations in that service to virtualize.
   By default, all the operations are selected.

8. Click **Next**.
   The request/response side data protocols options open.

9. Select **Web Services (SOAP)**.
   The Request Side Data Protocols list is prepopulated with Web Services (SOAP) and the XML Data Protocol data protocol handlers. The Response Side Data Protocols list is automatically populated with the Delimited Text Data Protocol.

10. Click **Next**.

11. See Delimited Text Data Protocol for information about how to configure the Delimited Text data protocol. When configured, click **Next**.
    On the next window, the service image is generated and the wizard is finished.

12. Click **Finish**.

    > **Note:** To save the settings on this recording to load into another service image recording, click **Save** , above the **Finish** button.

    The blank virtual service model is populated with steps.

13. Save the virtual service model.
    The service image that was generated is a stub service. The service image returns correctly formatted responses, but the values are default values.
    The service model (VSM) that was saved is the model that is deployed to the Virtual Service Environment.

Create a Service Image from Request-Response Pairs

You can generate a virtual service image from request/response pairs in the following ways:

- From the UI
- From the Command Line

You can use sidecar files to customize the metadata from your request/response pairs.

Create a Service Image from Request-Response Pairs in the UI

This procedure describes how to create a virtual service image from request/response pairs using the UI.

Follow these steps:

1. Right-click the **VirtualServices**, **Images** icon and select **Create New VS Image**, **From Request/Response Pairs**.
   The **Virtual Service From Request/Response Pairs** page opens.

2. Enter a service image name and the name of a VS model file.

3. Accept the default values for the remaining fields on this window and click **Next**.

   Note: To load parameters from a previously saved service image, click **Load from File** at the bottom of the window.

4. Browse the file system for the directory that contains your request/response pairs.

   A transaction is generated for each request/response pair in the directory that you specify. For HTTP/S, the files must contain the entire SOAP envelope and the header.

5. Specify the following information:

   - Transport protocol
   - Appropriate encoding
   - Whether to are use binary request/response pairs

6. Click **Configure**.

   The configuration window for the selected transport protocol opens.

7. Enter the configuration information for the request and click **Finish**.

   The **Virtual Service from Request/Response Pairs** window opens.

8. Click **Configure** again to persist the values you entered.
   If you select a different protocol, that protocol is presented. You can provide the configuration information for it.

9. Click **Next**.
   The **Data Protocols** panel opens, where the request/response pairs were analyzed and appropriate data protocols for the request and response were defaulted.

10. Make any changes or add more data protocols and click **Next**.
    The virtual service request/response pairs for token identification and conversations display. Only stateless transactions are supported from request/response pairs.

11. Click **Next**.

> **Note:** To save the settings on this recording to load to another service image recording, click **Save** above the **Finish** button.

After the processing of the VS image finishes, you can open the service image and the virtual service model.

Create a Service Image from Request-Response Pairs from the Command Line

In addition to creating a service image from request/response pairs using the UI, you can create the image through the ServiceImageManager command line.

Follow these steps:

1. Create a service image in the UI.

2. After the image is created, click **Save** above the **Finish** button, then click **Finish**.
   The settings are saved in a file with a **.vrs** extension.

3. Navigate to the LISA_HOME\bin directory and enter the following command:

   ```
   ServiceImageManager --vrs=recording-session-file --vsi-file=vsi-file --
   vsm_file=vsm-file --record
   ```

   - **recording-session-file**
     Defines the path of the **.vrs** file that you created previously, and

   - **vsi-file**
     Defines the service image file that is created.

   - **vsm_file**
     Defines the virtual service model files that is created.

Create a Service Image from PCAP

If you use packet capture software such as Wireshark to create logs of traffic, VSE can use those logs to create a virtual service image from a packet capture file (PCAP).

Prerequisites:

1. Native library to provide PCAP support for your operating system

   a. For Windows, use **WinPcap** (https://www.winpcap.org/)

b. For Mac OS, Linux, and UNIX systems, use the **libpcap** library (https://www.tcpdump.org/). This is installed by default on Mac OS and most Linux/UNIX distributions.

2. **JnetPcap** (http://jnetpcap.com/) - Java wrapper for interfacing with native PCAP libraries . Download the appropriate binary package for your operating system from http://jnetpcap.com/download.

3. Add the **jnetpcap.jar** from the binary package to the **LISA_HOME\lib** directory and the jnetpcap native library to the **LISA_HOME\bin** directory.
   The native library is different for different operating systems. For Windows, it is jnetpcap.dll.
   The PCAP functionality is now configured.

4. Restart DevTest Workstation (if it is running) to pick up the configuration changes.

Follow these steps:

1. Right-click the **VirtualServices**, **Images** folder and select **Create New VS Image**, **From PCAP** .
   Enter a service image name and the name of a virtual service model file.
   Accept the default values for the remaining fields on this window.

   Note: To load parameters from a previously saved service image, click **Load from File** at the bottom of the window.

2. (Optional) To add documentation about this virtual service, click the **Notes** tab.

3. Click **Next**.
   The data protocol window opens.

4. Click **Next**.

5. To select the packet capture file to be used for input, enter the file name or browse the file system.

6. Select HTTP/S as the transport protocol, and click **Configure**.
   The **Virtual Service from PCAP Transport Protocol Configuration** window opens.

7. Enter the following configuration options:

   - **Listen/Record on port**
     Defines the port on which the client communicates to DevTest.

   - **Target host**
     Defines the name or IP address of the target host where the server runs.

   - **Target port**
     Defines the target port number listened to by the server. Leave this field blank if you will select a Proxy passthrough style.
     Defaults: 80 (HTTP) and 443 (HTTPS)

> **Note: Target host** and **Target port** are important. They determine how DevTest matches packets. Select **Gateway** and enter the IP address and port of the server hosting the service to be virtualized. You could have some network traffic for every computer on the subnet where the capture was performed, depending on how the PCAP capture was done. Given an IP address and port, the data can be filtered from the file for all packets going to or from a specific IP and port. Then those packets are reconstructed to form valid TCP streams, which remove duplicates, and reorders packets in the correct order, and so forth. This work is all done for you by the operating system TCP stack during a real recording. These streams are replayed to the real protocol (HTTP) and as far as the protocol is concerned, the data arrives over a valid TCP stream.

- **Recorder passthru style**
  Specifies how **VS Image Recorder** acts during recording. Select **Gateway**.

- **Use SSL to server**
  Specifies whether DevTest uses HTTPS to send the request to the server.

  - **Selected**: DevTest sends an HTTPS (secured layer) request to the server.
    If you select **Use SSL to server**, but you do not select **Use SSL to client**, DevTest uses an HTTP connection for recording. DevTest then sends those requests to the server using HTTPS.

  - **Cleared**: DevTest sends an HTTP request to the server.

- **Use SSL to client**
  Specifies whether to use a custom keystore to play back an SSL request from a client. This option is only enabled when **Use SSL to server** is selected.
  Values:

  - **Selected:** You can specify a custom client keystore and a passphrase. If these parameters are entered, they are used instead of the hard-coded defaults.

  - **Cleared:** You cannot specify a custom client keystore and a passphrase.

- **SSL keystore file**
  Specifies the name of the keystore file.

- **Keystore password**
  Specifies the password associated with the specified keystore file.

> **Note:** For more information about configuring VSE in a two-way SSL environment, see Virtualizing Two-way SSL Connections.

- **Allow duplicate specific transactions (good for NTLM)**
  Specifies whether to record duplicate specific transactions.

8. Click **Finish** to return to the previous window.

9. Click **Next** to start recording.

Create a Service Image by Recording

The **Virtual Service Image Recorder** generates service images, which pretend to be what you recorded.

To start recording a new virtual service image:

1. Complete one of the following steps:

   - Click **VSE Recorder** on the main toolbar.
   - Right-click the **VirtualServices** node on the **Project** panel and select **Create a VS Image** , **By recording**.

   The Virtual Service Image Recorder opens.

2. Complete the following **Virtual Service Image Recorder tabs**, as appropriate:

   - Basics
   - Data Protocols Window
   - Recording by Transport Protocol (provides instructions for each method of recording virtual service images)

Basics Tab

The **Basics** tab on the first window in the **Virtual Service Image Recorder** wizard lets you enter the name, protocol, and navigation options for the image.

Enter the information as needed in the fields, depending on the protocol you are using. All subsequent windows are protocol-specific. For more information about specific protocols, see Transport Protocols.

VSI Recorder, Basics tab

The **Basics** tab options are:

- **Write image to**
  Specifies a unique service image name.
  The VSI path defaults to the project that was open when DevTest Workstation started. If you change to another project, the VSI path still defaults to the project that was open when DevTest Workstation started.

- **Import traffic**
  Specifies a raw or conversational XML traffic file to import. This field can be blank if no such file exists. If you specify a file, the transactions in the referenced XML document are merged with those from the previous recording.

- **Transport protocol**
  Specifies the transport protocol to use. For more information, see "Virtual Service Image Recorder - Transport Protocols".

- **De-identify**
  Specifies whether to try to recognize sensitive data and substitute random values during the recording. For more information, see "De-identifying Data".

- **Treat all transactions as stateless**
  Specifies whether to treat all recorded transactions as stateless. In most cases, leave this option cleared.

- **Allow duplicate specific transactions**
  Specifies whether DevTest can respond more than once to the same call, choosing a different response. Round-robin matching only happens if this check box is selected. This option is disabled for transport protocols that do not allow duplicate specific transactions.

- **Default navigation**
  Specifies the navigation tolerance that determines where in the conversation tree a VSM searches for a transaction that follows the specified transaction. Select the default navigation tolerance for all except the last (leaf) transactions.
  Default: WIDE

- **Last**
  Specifies the navigation tolerance that determines where in the conversation tree a VSM searches for a transaction that follows the last (leaf) transactions.
  Default: LOOSE

- **Export to**
  Specifies the full path of a file where you want the raw traffic logged. If you specify one, every time a transaction is offered to the recorder (either from the transport protocol during actual recording or from the import process), it is written to this file. A recording session can be captured for importing later and can be reused while data protocol details are being refined.

- **Model file**
  Defines the full path of your virtual service model file for this service image. If you provide a file name in this field, the recorder generates a VSM automatically. The model style is in effect only if you request a VSM.

- **VS Model style**
  Specifies whether to generate a VSM including the prepare steps.
  Values:

  - **More flexible:** Includes prepare steps (leading to a five-step model in case of HTTP/S protocol).

  - **More efficient:** The prepare steps are absent (leading to a three-step model in case of HTTP/S protocol).

  Default: More flexible

  > **Note:** To load parameters from a previously saved service image, click **Load from File** at the bottom of the window.

The **Notes** tab lets you document this service image.

> **Note:** If you import a raw traffic file with the VSE Recorder and then click the back button to return to the first panel, the VSE Recorder reimports the traffic file and processes the transactions again, creating twice the number of transactions.

Additionally, if you complete the process but you do not initially designate a traffic file and progress to the recording panel, then go back to the first panel and select the traffic file; when you go through the **VSE Recorder** again, the recorder processes no transactions.

Data Protocols Window

The second window in the **Virtual Service Image Recorder** wizard lets you enter information about data protocols for the virtual service.

Image of the Data Protocols page on the Virtual Services Image Recorder

The recorder can use the following data protocol handlers. Choosing the appropriate data protocol helps the recorder to analyze the information it records to differentiate the conversations and identify transactions belonging to the conversations. You can chain these data protocol handlers to use them with each other.

- **Auto Hash Transaction Discovery**
  Identifies a message by the hash code of the data. The hash code changes with even a slight change in the data, which effectively makes all requests unique. This protocol is useful if you run the same small set of requests against the service.

- **CICS Copybook Data Protocol**
  Splits the recorded request into its respective container chunks. It then sends each chunk to the Copybook data protocol and aggregates the corresponding XML.

- **Copybook Data Protocol**
  Converts copybook text to XML.

- **CTG Copybook Data Protocol**
  Splits the recorded request into its respective container chunks. It then sends each chunk to the Copybook data protocol and aggregates the corresponding XML.

- **Data De-identifier**
  Tries to recognize sensitive data and substitute random values during the recording. For more information, see De-identifyng Data.

- **Delimited Text Data Protocol**
  Converts delimited strings to XML.

- **EDI X12 Data Protocol**
  Transforms ANSI X12 EDI documents to an XML representation in the body of the request.

- **Generic XML Payload Parser**
  Identifies whether the requests and responses are XML strings. If you use this protocol, you can identify variables in the XML messages that the recorder uses.

- **JSON Data Protocol**
  Converts JSON data to an XML equivalent and converts XML data to JSON format.

- **Request Data Copier**
  Copies data from the current inbound request into the current testing context.

- **Request Data Manager**
  Manipulates VSE requests as they are recorded or played back.

- **REST Data Protocol**
  Analyzes HTTP requests that follow the REST architectural style. The data protocol identifies the dynamic parts of the URI strings. The result is a set of rules. During the playback phase, VSE uses the rules to virtualize HTTP requests that invoke the same operations.

- **Scriptable Data Protocol**
  Provides scripts on the request side, the response side, or both, to process the request or response.

- **SWIFT Data Protocol**
  Converts SWIFT messages to an XML equivalent and converts XML data to SWIFT messages.

- **Web Services Bridge**
  Applies only to the DevTest Travel example. You can ignore this data protocol because it is specific to the example and is not useful in a general case.

- **Web Services (SOAP)**
  Applies to use by a web service client.

- **Web Services (SOAP Headers)**
  Converts SOAP header elements into request arguments.

- **WS-Security Request**
  Strips security from the SOAP Request before sending it along the Virtualize framework and applies security to outgoing SOAP responses.

- **XML Data Protocol**
  Converts an XML document into a proper operation/arguments type of request.

> **Note:** JDBC does not allow a data protocol.

For more information about these data protocols, see "Using Data Protocols".

For more information about using a dynamic data protocol, see "Generic XML Payload Parser".

Transport Protocols

Each available transport protocol is described in the following sections:

- Apache Kafka Transport Protocol
- HTTP/S Transport Protocol

- IBM MQ Native Transport Protocol
- IBM WebSphere MQ Transport Protocol
- JMS Transport Protocol
- Standard JMS Transport Protocol
- RabbitMQ Transport Protocol
- Java Transport Protocol
- JDBC Transport Protocol
- TCP Transport Protocol
- CICS (LINK DTP MRO) Transport Protocol
- CICS Transaction Gateway (ECI) Transport Protocol
- IMS Connect Transport Protocol
- SAP RFC via JCo Transport Protocol
- JCo IDoc Transport Protocol
- Opaque Data Processing Transport Protocol

Apache Kafka Transport Protocol

The following pages contain detailed instructions for recording a service image that uses the Apache Kafka transport protocol:

- Kafka Topic Recorder
- Kafka Proxy Recording

Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Kafka Topic Recorder

The Kafka topic recorder lets you record transactions by listening on request and response topics.

As shown in the following graphic, the recorder sits outside of the application's message flow entirely and simply "listens in" to the message traffic.

The message-oriented middleware (MOM) is the platform on which messages are exchanged.

The live service and the recorder can both listen to the request topic without interfering with each other.

Similarly, the client and the recorder can both listen to the response topic without interfering with each other.

A request channel is composed of the following parts:

- Channel name
- Kafka receive operation
  This operation is what receives messages from the request topic.

A response channel is composed of the following parts:

- Channel name
- Kafka send operation
  This operation is what sends messages to the response topic during playback.
- Correlation scheme

As shown in the following graphic, the VSE service takes the place of the live service during playback. You must shut down the live service so that it does not interfere with the handling of requests.

Follow these steps:

1. In the **Basics** tab of the **Virtual Service Image Recorder**, set the transport protocol to **Apache Kafka**. Be sure to configure the service image and virtual service model.
2. Click **Next**.
   The proxy recorder setup page opens.
3. Change the recorder type from **Proxy Recorder** to **Topic Recorder**.
   The proxy recorder setup page has basic and advanced parameters. To display the advanced parameters, click **PRO** at the top of the editor.
4. Select the topic assets for the **Consume Topic** list and the **Send Topic** list. If the assets have not been created, you can create them from this page. You can also edit assets from this page.
5. (Optional) Click the plus (+) icons to add request channels and response channels.

6. (Optional) Select the **Correlation Scheme Enabled** check box and specify which scheme you want.

7. Click **Next** to start the recording session.
   The recording feedback page opens.
   The table in the **Summary** tab contains a list of each request channel and response channel. Each row includes the channel name, the selected destination asset, the number of messages that have flowed through the channel, and the status.
   The bottom of the page shows the number of pending transactions, the number of sessions, and the total number of transactions.
   You can use the runtime monitor to view the assets that are used in real time. For information about how the runtime monitor works, see Kafka Send Receive Step.

8. Exercise the target service.

9. When the service is complete, click **Next** to finish the recording.

10. Specify any data protocols and click **Next**.

11. (Optional) Save a recording session file.

12. Click **Finish**.
    The service image and the virtual service model are created.

Kafka Proxy Recording

Proxy recording is a common method for recording a messaging application.

The main components of proxy recording are:

- The client
- The service
- The proxy topic
- The live topic
- The VSE recorder

The following graphic shows these components. The message-oriented middleware (MOM) is the platform on which messages are exchanged.

The VSE recorder is inserted into the message flow between the client and the service. Each request message goes through the recorder before it reaches the service. Each response message goes through the recorder before it reaches the client.

A *channel* is the pairing of a proxy topic and a live topic.

The proxy request topic and the live request topic form a request channel.

The proxy response topic and the live response topic form a response channel.

The following graphic shows the request channel and the response channel in a proxy recording.

A proxy recording can have multiple request channels and multiple response channels.

In this procedure, you select an asset for each topic.

Follow these steps:

1. In the **Basics** tab of the **Virtual Service Image Recorder**, set the transport protocol to **Apache Kafka**. Be sure to configure the service image and virtual service model.

2. Click **Next**.
   The proxy recorder setup page opens.
   The proxy recorder setup page has basic and advanced parameters. To display the advanced parameters, click **PRO** at the top of the editor.
   Each channel must have a unique name.

3. (Optional) To generate a virtual service model that operates on live topics instead of proxy topics, display the advanced parameters and set the **VSE Type** field to **Live**.

4. Select the assets for the **Proxy Request**, **Live Request**, **Proxy Response**, and **Live Response** lists. If the assets have not been created, you can create them from this page. You can also edit assets from this page.

5. (Optional) Click the plus (+) icons to add request channels and response channels.

6. (Optional) Select the **Correlation Scheme Enabled** check box and specify which scheme you want. Each response channel can have a different correlation scheme.

7. Click **Next** to start the recording session.
   The recording feedback page opens.
   The table in the **Summary** tab lists each request channel and response channel. Each row includes the channel name, the selected destination asset, the number of messages that have flowed through the channel, and the status.
   You can click a cell in the **Destination** column to see the proxy destination that is associated with the channel.
   The bottom of the page shows the number of pending transactions, the number of sessions, and the total number of transactions.
   If an error prevents the recording session from starting, a dialog opens and you return to the proxy recorder setup page.
   Errors can also occur during the recording session. Errors that are associated with a specific request channel or response channel appear in the **Status** column. The status icon turns red, and a tooltip displays the error message. More general errors appear at the top of the recording feedback page.
   You can use the runtime monitor to view the assets that are used in real time. For information about how the runtime monitor works, see Kafka Send Receive Step.

8. Exercise the target service.

9. When the service completes, click **Next** to finish the recording.

10. Specify any data protocols and click **Next**.

11. (Optional) Save a recording session file.

12. Click **Finish**.
    The service image and the virtual service model are created.

HTTP/S Transport Protocol

The Virtual Service Image Recorder can record using the HTTP/S transport protocol.

- Virtualize Two-way SSL Connections

Follow these steps:

1. To start recording a new virtual service image, complete one of the following steps:

   - Click **VSE Recorder** on the main toolbar.

   - Right-click the **VirtualServices** node on the **Project** panel and select **Create a VS Image , by Recording**.

   The **Virtual Service Image Recorder** opens.

2. Complete the Basics tab as in the following graphic:

3. Click **Next**.
   The next wizard window opens.

4. Enter the port and host information for this step.

   - **Listen/Record on port**
     Defines the port on which the client communicates to DevTest. 8001 is typical, but you can use another port number. Do not enter the http:// or https:// prefix.

   - **Target port**
     Defines the target port number listened to by the server. Leave this field blank if you will select a Proxy passthrough style.
     Defaults: 80 (HTTP) and 443 (HTTPS)

   - **Recorder pass-through style**
     Specifies how the **VS Image Recorder** acts 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 affects how the client connects in the recording mode.

- If **VS Image Recorder** listens in a gateway mode, the client must send HTTP requests directly to the recorder and not to the server. If the client is a browser, the URL contains the host and port of the recorder instead of the host and port of the server.

- If **VS Image Recorder** listens in a proxy mode, the client must specify the recorder host and port as the proxy. If the client is a browser, then the URL contains the host and port of the server. The proxy settings must be set to route the request through the recorder.

- Most of the HTTP clients have a setting for NOT using proxy for localhost. If your **VS Image Recorder** is running on localhost in proxy mode, disable this setting for the traffic to get correctly passed through the recorder.

- **Do not modify host header parameter received from client**
  Specifies whether to pass through the value of the **Host** parameter. This option is only available when recording in Gateway mode. The pass-through option instructs the recorder not to rewrite the Host header parameter when resending traffic to the target endpoint.

  SSL Client (Server-Facing) Settings

  These settings are used to specify whether DevTest uses HTTPS to send requests to the server. The options that are configured here are also applied to the **HTTP/S Live Invocation Step** of the virtual service that is generated by the recording.

  - **Use SSL to server**

    - Specifies whether DevTest uses HTTPS to send requests to the server.

    - If you select **Use SSL to server**, but you do not select **Use SSL to client**, an HTTP connection is used between the client and DevTest for recording. DevTest then forwards those requests to the server using HTTPS.

    - If you do not select **Use SSL to server**, DevTest uses HTTP to send requests to the server.

  - **SSL keystore file**

    - Specifies the name of the client-side keystore file. DevTest uses this keystore when using an HTTPS connection to the server.

    - By default, the **ssl.client.cert.path** property is used for the client-side keystore file. This property can also be selected from the **Defaults** section of the dropdown list.

  - **Keystore password**

    - Specifies the password associated with the specified client-side keystore file.

    - You can specify the password directly using the Password Editor option, or you can use the DevTest Property Reference Editor option to specify a property expression that will be evaluated to provide the password.

    - By default, the property expression **{{ssl.client.cert.pass}}** is used for the client-side keystore password. This property expression can be selected from the **Defaults** section of the dropdown list.

SSL Server (Client-Facing) Settings

These settings are used to specify whether DevTest acts as an SSL server with a custom keystore during recording and playback. These settings are only enabled when you select **Use SSL to server**. The settings that are configured here are also applied to the **HTTP/S Listen Step** of the virtual service that the recording generates.

- **Use SSL to client**
  - Specifies whether DevTest uses a custom keystore when acting as an SSL server during recording and playback.
  - If you select Use SSL to client, an HTTPS connection is used between the client and DevTest for recording. You can also specify a custom server-side keystore file and password that is associated with it.
  - If you do not select **Use SSL to client**, an HTTP connection is used between the client and DevTest for recording. DevTest forwards those requests to the server using HTTPS.
- **SSL keystore file**
  - Specifies the name of the server-side keystore file. DevTest uses this keystore when handling HTTPS connections from the client.
  - By default, the **ssl.server.cert.path** property is used for the server-side keystore file. This property can also be selected from the **Defaults** section of the dropdown list.
- **Keystore password**
  - Specifies the password that is associated with the specified server-side keystore file.
  - You can specify the password directly using the Password Editor option, or you can use the DevTest Property Reference Editor option to specify a property expression that will be evaluated to provide the password.
  - By default, the property expression **{{ssl.server.cert.pass}}** is used for the server-side keystore password. This property expression can be selected from the **Defaults** section of the dropdown list.
- **Enable Client Certificate Authentication**
  - Specifies whether a request should be sent to the client for their certificate during the SSL handshake. The following options determine what to do with the client certificate.
  - Request Client Certificate
    DevTest requests a client certificate during the SSL handshake without requiring that one is sent back.
    This is the default option when client certificate authentication is enabled.
  - Require Client Certificate
    DevTest requires a valid client certificate during the SSL handshake. If the client does not send back a certificate or it is invalid, then DevTest stops the SSL handshake and the connection fails.

> in a two-way SSL environment, see Virtualizing Two-way SSL Connections.

5. Click **Next**.
   The **VS Image Recorder** starts recording the traffic. The assigned port and service target display on this window.

6. To send the requests to the server routed through the **VS Image Recorder** to start recording traffic, use your HTTP client.
   As the VS Image Recorder records transactions, the dynamic display statistics on the lower portion of the window increase. The options and dynamic display statistics include:

   - **Total conversations**
     Displays the number of conversations recorded.

   - **Total transactions**
     Displays the number of transactions recorded.

   - **Clear**
     Clears the list of currently recorded transactions.

7. When you have completed the recording, click **Next** to move to the next step.
   If you click **Next** and no transactions were recorded, an error message appears. Click **OK** to continue recording.

   > **Note:** If the transactions are not recorded, you could 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.

   The **Transactions** tab displays a list of the most recent transactions recorded. On this list of transactions, you can double-click a transaction and can see a dialog showing the content of the transaction.

8. Verify the base path and update it if necessary.

9. To require a bind-to-port step before processing requests, select the **A separate bind-to-port step is required** check box.

10. Click **Next**.

11. Do not select any value for the data protocol on the next window and click **Next**.

12. If no conversations were detected during the recording process, select transactions that start conversations. For token-based conversations, specify where tokens can be found. Use the **Token Identification** area in the **VS Image Recorder**. Select the transactions that start conversations and identify the session tokens. To designate the **getNewToken** Conversation Starter Transaction listed as a conversation starter, select it and click the blue arrow.
    The step components include:

- **Conversation Starter Transactions**
  Lists the transactions that you have selected as conversation starters. To move the transaction to the Remaining Transactions list (if you do not want it to be a conversation starter), select a transaction and click the arrow.

- **Remaining Transactions**
  Lists the recorded transactions. To move the transaction to the Conversation Starter Transactions list, select a transaction and click the arrow.

- **Plus icon**
  Selects all transactions in the list (either Conversation Starters or Remaining) that are like a selected transaction. To move all the selected transactions, use the appropriate arrow button.

- **Conversation count**
  Displays the number of conversations in the recording. As you build conversations, the number increases.

- **Force stateless**
  From the **Remaining Transactions** list, select any transactions that should stay stateless and select the check box. For example, you could 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 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 field identifies which of its responses to look in. In general, 1 is the only option.

- **Look in**
  This field 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 Meta data entries in the response, plus one for the body of the response.

- **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 red rubber stamp 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 **Erase**.

- After you mark a token, you can use the **Search** icon to find similar transactions and mark their tokens. To open a dialog where you can select text (such as XML tags) that bound the conversation token, click **Search**. To specify the leading and trailing text to search for, use this method.

13. Click **Next**.
    During postprocessing, the **VS Image Recorder** displays the processing status. As part of the preparation for writing the .vsi file, the recorder verifies request and response bodies to ensure that, if they are marked as text, that they are text. If they are not, the type is switched to binary.
    The recorder completes postprocessing the recording.

    Note: To save the settings on this recording to load into another image recording, click **Save** above the **Finish** button.

14. Click **Finish** to store the image.

15. Review and save the virtual service model in DevTest Workstation.

Virtualize Two-way SSL Connections

To virtualize a two-way SSL connection, DevTest must have information for both the client-side keystore and server-side keystore.

Note: To use the default DevTest keystore (see **webreckeys.ks** in the installation directory) as the server-side keystore, extract the DevTest certificate from the DevTest keystore and add it to the client truststore. The DevTest certificate is a self-signed certificate and not a certificate that a CA authority issues. This workaround only works when the client accepts self-signed certificates.

The HTTPS recorder resembles the following graphic:

The following diagrams illustrate one-way and two-way SSL virtualization.

IBM MQ Native Transport Protocol

This page describes how to record a service image that uses the IBM MQ Native transport protocol.

> **Note:** This transport protocol is for WebSphere MQ in native mode. If you are using WebSphere MQ in JMS mode, we recommend that you use the JMS transport protocol.

This feature is supported on WebSphere MQ 5.2, 6.0, 7.1, and 8.0.**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Proxy recording is a common method for recording a messaging application.

The main components of proxy recording are:

- The client
- The service
- The proxy queues
- The live queues
- The VSE recorder

The following graphic shows these components. The message-oriented middleware (MOM) is the platform on which messages are exchanged.

The VSE recorder is inserted into the message flow between the client and the service. Each request message goes through the recorder before it reaches the service. Each response message goes through the recorder before it reaches the client.

A *channel* is the pairing of a proxy queue and a live queue.

The proxy request queue and the live request queue form a request channel.

The proxy response queue and the live response queue form a response channel.

The following graphic shows the request channel and the response channel in a proxy recording.

A proxy recording can have multiple request channels and multiple response channels.

In this procedure, you select an asset for each queue.

Follow these steps:

1. In the **Basics** tab of the **Virtual Service Image Recorder**, set the transport protocol to **IBM MQ Native**. Be sure to configure the service image and virtual service model.

2. Click **Next**.
   The proxy recorder setup page opens.

Screen capture of proxy recorder setup page.

The proxy recorder setup page has basic and advanced parameters. To display the advanced parameters, click **PRO** at the top of the editor.
Each channel must have a unique name.

3. Select the queue assets for the **Proxy Request**, **Live Request**, **Proxy Response**, and **Live Response** lists. If the assets have not been created, you can create them from this page. You can also edit assets from this page.

4. (Optional) Click the plus (+) icons to add request channels and response channels.

5. (Optional) Select the **Correlation Scheme** check box and specify which scheme you want. Each response channel can have a different correlation scheme.

6. Click **Next** to start the recording session.
   The recording feedback page opens.

Screen capture of recording feedback page.

The table in the **Summary** tab lists each request channel and response channel. Each row includes the channel name, the selected destination asset, the number of messages that have flowed through the channel, and the status.
You can click a cell in the **Destination** column to see the proxy destination that is associated with the channel.
The bottom of the page shows the number of pending transactions, the number of sessions, and the total number of transactions.
If an error prevents the recording session from starting, a dialog opens and you return to the proxy recorder setup page.
Errors can also occur during the recording session. Errors that are associated with a specific request channel or response channel appear in the **Status** column. The status icon turns red, and a tooltip displays the error message. More general errors appear at the top of the recording feedback page.
You can use the runtime monitor to view the assets that are used in real time. For information about how the runtime monitor works, see IBM MQ Native Send Receive Step .

7. Exercise the target service.

8. When the service completes, click **Next** to finish the recording.

9. Specify any data protocols and click **Next**.

10. (Optional) Save a recording session file.

11. Click **Finish**.
    The service image and the virtual service model are created.

IBM WebSphere MQ Transport Protocol

Note: This transport protocol has been deprecated. Refer to the following for creating new tests and virtual services:

- JMS Client Assets
- JMS Transport Protocol
- IBM MQ Native Transport Protocol

This page describes how to record a service image that uses the IBM WebSphere MQ transport protocol.**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Follow these steps:

1. To start recording a new virtual service image, complete one of the following steps:

   - Click **VSE Recorder** on the main toolbar.
   - Right-click the **VirtualServices** node on the **Project** panel and select **Create a VS Image , by Recording**.

   The **Virtual Service Image Recorder** opens.

2. Complete the Basics tab as in the following graphic:

3. Click **Next**.
   The recording mode selection step opens, with one of the following options selected:

   - **Proxy Mode**
     Proxy mode is the only true recording mode that VSE supports. Proxy mode provides the following options:

- Generate the service from the Live queues
- Generate the service using the Proxy queues

Proxy mode allows the use of proxy destinations to be set up on the message bus. The client application is configured to put messages on those proxy destinations and DevTest records them and forwards to the real destination. The same thing happens on the response side. DevTest is configured to listen on the real response destination, get the message, and forward it to the response proxy destination. This mode can behave differently if you enable temporary destinations, making the response side automated.

- **Import Mode**
  This mode for recording virtual services is available if you specify a raw traffic file in the **Import traffic** field on the initial recording window. In import mode, you can specify extra information about which request and response queues were detected in the raw traffic file. You can also select to skip the request response steps.

4. If you are recording in proxy mode, complete the following fields:

   - **Generate service using the Live queues / Proxy queues**
     Specifies whether to use the live queues or the proxy queues when generating the final virtual service model and image.

   - **Max Pending Transactions**
     Defines the maximum number of running response listeners on each response queue. These response listeners run as long as a transaction is pending. When the number of pending transactions exceeds the maximum, the oldest transaction closes automatically. If you enter 0 in this field, there is no maximum.

   - **Disable Multiple Responses**
     Specifies whether to support multiple responses in each transaction. By default, a transaction stays pending after the first response, waiting for more responses to arrive for the same transaction. When you select this option, the transaction automatically closes after the first response. If there are many transactions coming in quickly, this option can boost performance, especially for MQ.

   - **Correlation**
     Contains the possible schemes for correlating the request and response sides of individual transactions. JMS is asynchronous, which means the requests and responses are received separately. This drop-down list lets you define to the VSE Recorder which request to associate with which response. The **Correlation** field has the following options:

     - **Sequential:** Associates every response with the last request received, chronologically. There is no correlation scheme, so the VSE MQ recorder acquires an exclusive read lock on the live response queue to ensure that another MQ listener cannot take response messages from it.
       When you specify a correlation scheme, such as **Correlation ID** or **Message ID to Correlation ID**, the VSE MQ recorder can assume that any other listener on the live response queue will also use a correlation scheme. If all listeners on the live response queue use correlation schemes, the VSE MQ recorder can keep its responses separate without resorting to an exclusive read lock, so it opens the queue with the shared input flag.

     - **Correlation ID:** The request and the response must have the same Correlation ID.

- **Message ID to Correlation ID:** The request Message ID must be the same as the response Correlation ID.
- **Message ID:** The request and the response have the same Message ID.

5. If you are recording in import mode, complete the following fields:

   - **Client Mode**
     Specifies how to interact with the WebSphere MQ server.
     Values:

     - **Native Client:** A pure Java implementation using IBM-specific APIs
     - **JMS:** A pure Java implementation that is based on the JMS specification. If you want this implementation, the best practice is to use the JMS transport protocol instead of MQ.
     - **Bindings:** This option requires access to the native libraries from a WebSphere MQ client installation. Ensure that the DevTest application run time can access these libraries. In most cases, having these libraries available in the PATH environment is adequate.

   - **Review the queues and transaction tracking mode**
     Specifies whether to skip the request and response steps. If the raw traffic file comes from Application Insight, this option is cleared. Application Insight autodetects queues and transactions. If the raw traffic file does not come from Application Insight, this option is selected by default to let you review destination and tracking settings.

   - **Correlation**
     Contains the possible schemes for correlating the request and response sides of individual transactions. JMS is asynchronous, which means the requests and responses are received separately. This drop-down list lets you define to the VSE Recorder which request to associate with which response. The **Correlation** field has the following options:

     - **Sequential:** Associates every response with the last request received, chronologically. There is no correlation scheme, so the VSE MQ recorder acquires an exclusive read lock on the live response queue to ensure that another MQ listener cannot take response messages from it.
       When you specify a correlation scheme, such as **Correlation ID** or **Message ID to Correlation ID**, the VSE MQ recorder can assume that any other listener on the live response queue will also use a correlation scheme. If all listeners on the live response queue use correlation schemes, the VSE MQ recorder can keep its responses separate without resorting to an exclusive read lock, so it opens the queue with the shared input flag.
     - **Correlation ID:** The request and the response must have the same Correlation ID.
     - **Message ID to Correlation ID:** The request Message ID must be the same as the response Correlation ID.
     - **Message ID:** The request and the response have the same Message ID.

6. Click **Next**.
   The **Destination Info** tab opens.

7. Enter your proxy and live queue names, and select the queue type.
   The **Create Proxy Queue** check box lets you create a temporary queue on the fly to be used as a proxy queue. Select this option if you did not manually create the proxy queue on WebSphere MQ.

8. Click the **Connection setUp** tab.

9. Enter the connection parameters used to connect to the MOM.
   These connection parameters are internally saved.
   The **Advanced** tab at the bottom of the **Connection setUp** tab includes the following subtabs:

   - **Environment**
     Lets you add, delete, and specify values for more MQ environment settings.

   - **MQ Exits**
     Lets you point to MQ exits for Security, Send, and Receive.

10. To define an optional separate set of connection information for when your proxy queues are on a different queue manager from your live queues, click the **Proxy Connection setUp** tab.

11. Define the connection details for the response destinations on which to listen for messages.

12. Define the proxy queue on which the client application will receive these responses.

    - Remember that multiple responses are possible for a request. To register a set of response proxy queues, click **Add** .

    - Select the **Use for Unknown Responses** check box before registering the response queue to use for unknown requests.

    - To define a temporary destination, select the **Use temporary queue/topic** check box. Selecting this option disables the destination name and proxy destination name fields and marks the response listener you are adding as a temporary response. The destination name is blank.
      A "temporary" destination in messaging is a destination that is created on demand for a messaging client. A temporary destination is typically used in request/response scenarios. DevTest supports using a temporary queue for the responses, but only supports one concurrent transaction with a temporary queue at a time.

13. Click **Next**.
    The **Destination List** tab opens.

14. Click the **Current Connection Info** tab and verify that the connection information is correct.

15. The information that is shown on the **Current Connection Info** tab is copied from the connection information that you gave earlier. You might want to change it in a rare case when the response connection information is different.

16. Click **Next** to start recording.
    The names of the queues to which VSE is listening display, and the status is Waiting.
    If you connect to WebSphere MQ and you chose to create the proxy queues on the fly, those proxy queues are created.

17. Run the client that adds messages to the request proxy queue.
    VSE copies the messages to the real request queue. The server acquires those messages from there and sends responses to the response queue or queues. Again, VSE acquires those messages and copies them to the response proxy queue, where the client is listening.
    As the transactions are recorded, the message count increases and the **Total sessions** and **Total transactions** counts increase on the **Virtual Service Image Recorder** window. At the end of recording, all the requests have gone through the same request queue. About half of the responses have returned through temporary queues, and the other half have returned through the nontemporary response queue.
    When the run is complete, you may see the count of messages on the response queues one fewer than you would expect. Because a single request can have multiple responses, VSE will not yet have recognized the last transactions complete. The messages corresponding to the last transaction are therefore not counted.

18. Click **Next** to trigger closure of the last transaction and complete the necessary cleaning. (You would see an intermediate window if you were using a dynamic data protocol.) As part of preparing the recorder for writing out the .vsi file, the recorder verifies request and response bodies to ensure that, if they are text, if so marked. If they are not, the type is switched to binary.
    A Request Data Manager data protocol has been added on the request side. For more information about configuring this protocol, see "Request Data Manager". You can change or add more data protocols.

19. Click **Next**.

    Note: To save the settings on this recording to load into another service image recording, click **Save** above the **Finish** button.

20. Click **Finish** to close the window and store the image.

21. Review and save the VSM that was generated in the main window.

JMS Transport Protocol

The following pages contain detailed instructions for recording a service image that uses the JMS transport protocol:

- Basic Proxy Recording
- Live VSM Type
- TIBCO Monitor Recorder
- JMS Topic Recorder

Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Basic Proxy Recording

Proxy recording is a common method for recording a messaging application.

The main components of proxy recording are:

- The client

- The service
- The proxy queues
- The live queues
- The VSE recorder

The following graphic shows these components. The message-oriented middleware (MOM) is the platform on which messages are exchanged.

The VSE recorder is inserted into the message flow between the client and the service. Each request message goes through the recorder before it reaches the service. Each response message goes through the recorder before it reaches the client.

A *channel* is the pairing of a proxy queue and a live queue.

The proxy request queue and the live request queue form a request channel.

The proxy response queue and the live response queue form a response channel.

The following graphic shows the request channel and the response channel in a proxy recording.

A proxy recording can have multiple request channels and multiple response channels.

In this procedure, you select an asset for each queue.

Follow these steps:

1. In the **Basics** tab of the **Virtual Service Image Recorder**, set the transport protocol to **JMS**. Be sure to configure the service image and virtual service model.

2. Click **Next**.
   The proxy recorder setup page opens.

Screen capture of proxy recorder setup page.

The proxy recorder setup page has basic and advanced parameters. To display the advanced parameters, click **PRO** at the top of the editor.
Each channel must have a unique name.

3. (Optional) To generate a virtual service model that operates on live queues instead of proxy queues, display the advanced parameters and set the **VSE Type** field to **Live**.

4. Select the queue assets for the **Proxy Request**, **Live Request**, **Proxy Response**, and **Live Response** lists. If the assets have not been created, you can create them from this page. You can also edit assets from this page.

5. (Optional) Click the plus (+) icons to add request channels and response channels.

6. (Optional) Select the **Correlation Scheme** check box and specify which scheme you want. Each response channel can have a different correlation scheme.

7. Click **Next** to start the recording session.
   The recording feedback page opens.

Screen capture of recording feedback page.

The table in the **Summary** tab lists each request channel and response channel. Each row includes the channel name, the selected destination asset, the number of messages that have flowed through the channel, and the status.
You can click a cell in the **Destination** column to see the proxy destination that is associated with the channel.
The bottom of the page shows the number of pending transactions, the number of sessions, and the total number of transactions.
If an error prevents the recording session from starting, a dialog opens and you return to the proxy recorder setup page.
Errors can also occur during the recording session. Errors that are associated with a specific request channel or response channel appear in the **Status** column. The status icon

turns red, and a tooltip displays the error message. More general errors appear at the top of the recording feedback page.
You can use the runtime monitor to view the assets that are used in real time. For information about how the runtime monitor works, see JMS Send Receive Step.

8. Exercise the target service.

9. When the service completes, click **Next** to finish the recording.

10. Specify any data protocols and click **Next**.

11. (Optional) Save a recording session file.

12. Click **Finish**.
    The service image and the virtual service model are created.

Live VSM Type

The proxy recorder includes a field called **VSM Type**.

The default value is **Proxy With Live Invocation**. The recorder generates a virtual service model that operates on the proxy queues and supports the live invocation mode.

If you set the value to **Live**, the recorder generates a virtual service model that operates on live queues instead of proxy queues. The virtual service model does not support the live invocation mode.

During playback, the live service must be shut down so there is no interference between the virtual service and the live service.

The **VSM Type** field is an advanced parameter.

The following graphic shows the components. Notice the absence of proxy queues.

TIBCO Monitor Recorder

With every queue on a TIBCO EMS server, a user with administrator privileges can connect to a special monitor topic. While subscribed to this monitor topic, an administrator can receive a copy of every message that is sent over the corresponding queue. This feature allows the VSE recorder to be outside of the message flow of the application entirely and simply monitor the message traffic.

The following graphic shows the main components of TIBCO Monitor recording:

- The client
- The service
- The request and response queues
- The request and response monitor topics
- The VSE recorder

The message-oriented middleware (MOM) is the platform on which messages are exchanged.

- **Monitor Topic (Request)**
  The TIBCO monitor topic that is associated with the request queue for the application. The TIBCO server forwards each message that is sent on the request queue to this topic automatically.

- **Monitor Topic (Response)**
  The TIBCO monitor topic that is associated with the response queue for the application. The TIBCO server forwards each message that is sent on the response queue to this topic automatically.

The TIBCO Monitor Recorder supports TIBCO EMS 6.3.

You cannot use monitor topics during playback to virtualize the service. You can do either of the following actions:

- Create a separate set of queues and configure the client and VSE service to use them.

- Shut down the live service so the VSE service can replace it.

In the JMS connection asset, the admin user must have administrative privileges.

Follow these steps:

1. In the **Basics** tab of the **Virtual Service Image Recorder**, set the transport protocol to **JMS**. Be sure to configure the service image and virtual service model.

2. Click **Next**.
   The proxy recorder setup page opens.

3. Change the recorder type from **Proxy Recorder** to **TIBCO Monitor Recorder**.
   The proxy recorder setup page has basic and advanced parameters. To display the advanced parameters, click **PRO** at the top of the editor.

4. Select the queue assets for the **Receive Destination** list and the **Send Destination** list. If the assets have not been created, you can create them from this page. You can also edit assets from this page.

5. (Optional) Click the plus (+) icons to add request queues and response queues.

6. (Optional) Select the **Correlation Scheme** check box and specify which scheme you want.

7. Click **Next** to start the recording session.
   The recording feedback page opens.
   The table in the **Summary** tab contains a list of each request channel and response channel. Each row includes the channel name, the selected destination asset, the number of messages that have flowed through the channel, and the status.
   The bottom of the page shows the number of pending transactions, the number of sessions, and the total number of transactions.
   You can use the runtime monitor to view the assets that are used in real time. For information about how the runtime monitor works, see JMS Send Receive Step.

8. Exercise the target service.

9. When the service is complete, click **Next** to finish the recording.

10. Specify any data protocols and click **Next**.

11. (Optional) Save a recording session file.

12. Click **Finish**.
    The service image and the virtual service model are created.

JMS Topic Recorder

The JMS topic recorder lets you record transactions by listening on request and response topics.

As shown in the following graphic, the recorder sits outside of the application's message flow entirely and simply "listens in" to the message traffic.

The message-oriented middleware (MOM) is the platform on which messages are exchanged.

The live service and the recorder can both listen to the request topic without interfering with each other.

Similarly, the client and the recorder can both listen to the response topic without interfering with each other.

A request channel is composed of the following parts:

- Channel name
- JMS receive operation
  This operation is what receives messages from the request topic.

A response channel is composed of the following parts:

- Channel name
- JMS send operation
  This operation is what sends messages to the response topic during playback.
- Correlation scheme

As shown in the following graphic, the VSE service takes the place of the live service during playback. You must shut down the live service so that it does not interfere with the handling of requests.

Live invocation mode is not supported. The virtual service model that is generated does not include a passthrough step.

Follow these steps:

1. In the **Basics** tab of the **Virtual Service Image Recorder**, set the transport protocol to **JMS**. Be sure to configure the service image and virtual service model.

2. Click **Next**.
   The proxy recorder setup page opens.

3. Change the recorder type from **Proxy Recorder** to **Topic Recorder**.
   The proxy recorder setup page has basic and advanced parameters. To display the advanced parameters, click **PRO** at the top of the editor.

4. Select the topic assets for the **Receive Destination** list and the **Send Destination** list. If the assets have not been created, you can create them from this page. You can also edit assets from this page.

5. (Optional) Click the plus (+) icons to add request channels and response channels.

6. (Optional) Select the **Correlation Scheme** check box and specify which scheme you want.

7. Click **Next** to start the recording session.
   The recording feedback page opens.
   The table in the **Summary** tab contains a list of each request channel and response channel. Each row includes the channel name, the selected destination asset, the number of messages that have flowed through the channel, and the status.
   The bottom of the page shows the number of pending transactions, the number of sessions, and the total number of transactions.
   You can use the runtime monitor to view the assets that are used in real time. For information about how the runtime monitor works, see JMS Send Receive Step.

8. Exercise the target service.

9. When the service is complete, click **Next** to finish the recording.

10. Specify any data protocols and click **Next**.

11. (Optional) Save a recording session file.

12. Click **Finish**.
    The service image and the virtual service model are created.

Standard JMS Transport Protocol

Note: This protocol has been deprecated. Refer to the following for creating new tests and virtual services:

- JMS Client Assets
- JMS Transport Protocol

This page contains the detailed instructions for recording a virtual service image using the Standard JMS transport protocol.**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Follow these steps:

1. To start recording a new virtual service image, complete one of the following steps:

   - Click **VSE Recorder** on the main toolbar.
   - Right-click the **VirtualServices** node on the **Project** panel and select **Create a VS Image , by Recording**.

   The **Virtual Service Image Recorder** opens.

2. Complete the Basics tab as in the following graphic:

3. Click **Next**.
   The recording mode selection step opens with one of the following options selected:

   - **Proxy Mode**
     Proxy mode is the only true recording mode that VSE supports. Proxy mode provides the following options:

     - Generate the service from the Live queues

     - Generate the service using the Proxy queues

     Proxy mode allows the use of proxy destinations to be set up on the message bus. The client application is configured to put messages on those proxy destinations and DevTest records them and forwards to the real destination. The same thing happens on the response side. DevTest is configured to listen on the real response destination, get the message, and forward it to the response proxy destination. This mode can behave differently if you enable temporary destinations, making the response side automated.

   - **Import Mode**
     This mode for recording virtual services is available if you specify a raw traffic file in the **Import traffic** field on the initial recording window. In import mode, you can specify extra information about which request and response queues were detected in the raw traffic file. You can also select to skip the request response steps.

4. To record in proxy mode, complete the following fields:

   - **Generate service using the Live queues / Proxy queues**
     Specifies whether to use the live queues or the proxy queues when generating the final virtual service model and image.

- **Max Pending Transactions**
  Defines the maximum number of running response listeners on each response queue. These response listeners run as long as a transaction is pending. When the number of pending transactions exceeds the maximum, the oldest transaction closes automatically. If you enter 0 in this field, there is no maximum.

- **Disable Multiple Responses**
  Specifies whether to support multiple responses in each transaction. By default, a transaction stays pending after the first response, waiting for more responses to arrive for the same transaction. When you select this option, the transaction automatically closes after the first response. If there are many transactions coming in quickly, this option can boost performance, especially for MQ.

5. To record in import mode, complete the following fields:

   - **Review the queues and transaction tracking mode**
     Specifies whether to skip the request and response steps. If the raw traffic file comes from Application Insight, this option is cleared. Application Insight autodetects queues and transactions. If the raw traffic file does not come from Application Insight, this option is selected by default to let you review destination and tracking settings.

   - **Correlation**
     Contains the possible schemes for correlating the request and response sides of individual transactions. JMS is asynchronous, which means the requests and responses are received separately. This drop-down list lets you define to the VSE Recorder which request to associate with which response. The **Correlation** field has the following options:

     - **Sequential:** Associates every response with the last request received, chronologically. There is no correlation scheme, so the VSE MQ recorder acquires an exclusive read lock on the live response queue to ensure that another MQ listener cannot take response messages from it.
       When you specify a correlation scheme, such as **Correlation ID** or **Message ID to Correlation ID**, the VSE MQ recorder can assume that any other listener on the live response queue will also use a correlation scheme. If all listeners on the live response queue use correlation schemes, the VSE MQ recorder can keep its responses separate without resorting to an exclusive read lock, so it opens the queue with the shared input flag.

     - **Correlation ID:** The request and the response must have the same Correlation ID.

     - **Message ID to Correlation ID:** The request Message ID must be the same as the response Correlation ID.

     - **Message ID:** The request and the response have the same Message ID.

6. Click **Next**.
   The **Destination Info** tab opens.

7. Enter your proxy and live destination names, and select the destination type.

8. Click the **Connection setUp** tab.

9. Enter the connection parameters used to connect to the MOM.
   These connection parameters are internally saved.
   The **Advanced** tab at the bottom of the **Connection setUp** tab lets you define custom connection properties for the service image.

10. Click **Next**.
    The **Destination List** tab opens.

11. Define the connection details for the response destinations on which to listen for messages.

12. Define the proxy queue on which the client application will receive these responses.

    - Remember that multiple responses are possible for a request. To register a set of response proxy queues, click **Add** .

    - Select the **Use for Unknown Responses** check box before registering the response queue to use for unknown requests.

    - To define a temporary destination, select the **Use temporary queue/topic** check box. Selecting this option disables the destination name and proxy destination name fields and marks the response listener you are adding as a temporary response. The destination name is blank.
      A "temporary" destination in messaging is a destination that is created on demand for a messaging client. A temporary destination is typically used in request/response scenarios. DevTest supports using a temporary queue for the responses, but only supports one concurrent transaction with a temporary queue at a time.

13. Click the **Current Connection Info** tab and verify that the connection information is correct.
    The information that appears on this tab is copied from the connection information that was given earlier. You could change it in a rare case when the response connection information is different.

14. Click **Next** to start recording.
    The names of the destinations to which VSE is listening display.
    The **Pending Transactions** field displays the number of transactions that are stored in the pending transaction buffer waiting for more responses. The transactions are closed either by reaching the maximum pending transaction count or by using the **Disable Multiple Responses** option. As the transactions close, they are moved from the pending transaction buffer to the total transaction buffer.

15. Run the client that adds messages to the request proxy queue.
    VSE copies the messages to the real request queue. The server acquires those messages from there and sends responses to the response queue. Again, VSE acquires those messages and copies them to the response proxy queue, where the client is listens.
    As the transactions are recorded, the message count increases and **Total sessions** and **Total transactions** counts increase on the **Virtual Service Image Recorder** window. At the end of recording, all the requests have gone through the same request queue. About half of the responses have returned through temporary queues, and the other half have returned through the nontemporary response queue.
    When the run is complete, you may see the count of messages on the response queues one less than you expect. Because a single request can have multiple responses, VSE will not yet have recognized the last transactions to complete. Therefore, the messages corresponding to the last transaction are not counted.
    Recommended data protocols are defaulted on this panel. You can add or edit both

request and response side data protocols here. If there are configuration panels for the data protocols you have chosen, they open next. For more information, see Using Data Protocols.

16. Click **Next**.
    The last transaction closes and the necessary cleanup completes.

> **Note:** To save the settings on this recording to load into another service image recording, click **Save** above the **Finish** button.

17. Click **Finish** to close the window and store the image.

18. Review and save the VSM that is generated in the main window.

RabbitMQ Transport Protocol

This page describes how to record a service image that uses the RabbitMQ transport protocol.

Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Proxy recording is a common method for recording a messaging application.

The main components of proxy recording are:

- The client
- The service
- The proxy queues
- The live queues
- The VSE recorder

The VSE recorder is inserted into the message flow between the client and the service. Each request message goes through the recorder before it reaches the service. Each response message goes through the recorder before it reaches the client.

A *channel* is the pairing of a proxy queue and a live queue.

The proxy request queue and the live request queue form a request channel.

The proxy response queue and the live response queue form a response channel.

A proxy recording can have multiple request channels and multiple response channels.

Follow these steps:

1. In the **Basics** tab of the **Virtual Service Image Recorder**, set the transport protocol to **RabbitMQ**. Be sure to configure the service image and virtual service model.

2. Click **Next**.
   The proxy recorder setup page opens.

Screen capture of proxy recorder setup page.

The proxy recorder setup page has basic and advanced parameters. To display the advanced parameters, click **PRO** at the top of the editor.
Each channel must have a unique name.

3. (Optional) To generate a virtual service model that operates on live queues instead of proxy queues, display the advanced parameters and set the **VSE Type** field to **Live**.

4. Select the queue assets for the **Proxy Request**, **Live Request**, **Proxy Response**, and **Live Response** lists. If the assets have not been created, you can create them from this page. You can also edit assets from this page.

5. (Optional) Click the plus (+) icons to add request channels and response channels.

6. (Optional) Select the **Correlation Scheme** check box and specify which scheme you want. Each response channel can have a different correlation scheme.

7. Click **Next** to start the recording session.
   The recording feedback page opens.

Screen capture of recording feedback page.

The table in the **Summary** tab lists each request channel and response channel. Each row includes the channel name, the selected destination asset, the number of messages that have flowed through the channel, and the status.
You can click a cell in the **Destination** column to see the proxy destination that is associated with the channel.
The bottom of the page shows the number of pending transactions, the number of sessions, and the total number of transactions.
If an error prevents the recording session from starting, a dialog opens and you return to the proxy recorder setup page.
Errors can also occur during the recording session. Errors that are associated with a specific request channel or response channel appear in the **Status** column. The status icon turns red, and a tooltip displays the error message. More general errors appear at the top of the recording feedback page.
You can use the runtime monitor to view the assets that are used in real time. For information about how the runtime monitor works, see RabbitMQ Send Receive Step.

8. Exercise the target service.

9. When the service completes, click **Next** to finish the recording.

10. Specify any data protocols and click **Next**.

11. (Optional) Save a recording session file.

12. Click **Finish**.
    The service image and the virtual service model are created.

Java Transport Protocol

The DevTest Java Agent is required to record Java service images. In the LISA Bank application, the agent is installed by default. For any other application, install the agent and start the registry.

For testing, start the demo server, where the agent is installed, or run the LISA Bank application. For running with any other Java application, configure the agent. For more information about the agent, see Agents.

When using multiple recorders in the same workstation to record multiple EJB remote calls concurrently from different classes, the service image and VSM created by each recorder display an aggregated list of classes.

Follow these steps:

1. To start recording a new virtual service image, complete one of the following steps:

   - Click **VSE Recorder** on the main toolbar.
   - Right-click the **VirtualServices** node on the **Project** panel and select **Create a VS Image** , **By recording**.

   The **Virtual Service Image Recorder** opens.

2. Select Java as the transport protocol on the Basics tab.

3. Click **Next**.
   The **Select Java classes to virtualize** window opens.

4. To move selected agents between the following columns, use the provided arrows:

   - **Available Online Agents**
     Lists the online agents that are available to be connected.
   - **Connected Agents**
     Lists contains the online or offline agents that are connected for the virtual service model. The offline agents display in a gray italic font. You can select the agents from the **Available Online Agents** list.

   When you select an agent from either list, the host name and the main class appear.

5. To add an agent that is not in the list, type the agent name in the field above the list of connected agents and click **Add Agent** .
   This mechanism is provided for adding any offline agents that were not previously in the connected list. The agent name cannot be empty or already present in the connected agents list. If the agent name entered is present in the online agents list, it is moved from the available online agents to the connected agents list.

6. To select a class by searching, complete the following steps:

   a. Double-click **Search for Classes**.

   b. Type a search string as a fully qualified name (including the package), using regular expressions.

   c. Click **Search**.

   d. Select a class. Some classes could show up more than once. Select a class only once.

   e. Click the right arrow.

7. To select a class by manually entering the name, complete the following steps:

   a. Double-click **Manually Enter a Class Name**.

   b. Type the fully qualified class name.

   c. Click the right arrow.

8. To select a class from a list of classes that the agent suggests, complete the following steps:

   a. Double-click **Agent Suggestions**.

   b. Click **Retrieve** .

   c. Select a class.

   d. Click the right arrow.

9. To add a protocol to the recording, double-click **Protocols** and select the protocol to record.

10. Shut down the system under test.

11. Click **Next**.
    The **Recording Has Begun** page opens.

12. Restart the system under test.

    Note: Shutting down and restarting the system under test is optional. Restarting helps to ensure that VSE captures any initial traffic that occurs when the system under test initializes.

    After the system under test starts, a list of the most recent transactions displays. To see the content of a selected transaction, double-click it.

13. Click **Next** when the recording completes.
    The **Data Protocols** panel opens with appropriate data protocols prepopulated. You can change or add data protocols. There may be additional panels to configure the data protocols. For more information about configuring data protocols, see Using Data Protocols.

14. When you have configured the data protocols, click **Next**.

    While it prepared to write the .vsi file, the recorder verifies request and response bodies to ensure that (if they are marked as text) they are text. If they are not text, the type is switched to binary.

> **Note:** To save the settings on this recording to load into another service image recording, click **Save** above the **Finish** button.,

15. Click **Finish**.

JDBC Transport Protocol

This page contains the detailed instructions for recording a virtual service image using the JDBC (Driver Based) transport protocol.

> **Note:** The JDBC (Driver Based) transport protocol is not supported. This functionality is replaced by agent-based JDBC virtualization.

For more information about prerequisites and preparatory steps, see Preparing to Virtualize JDBC .

Follow these steps:

1. To start recording a new virtual service image, complete one of the following steps:

   - Click **VSE Recorder** on the main toolbar.
   - Right-click the **VirtualServices** node on the **Project** panel and select **Create a VS Image , by Recording**.

   The **Virtual Service Image Recorder** opens.

2. Complete the Basics tab as in the following graphic:

Image of the Basics tab on the Virtual Service Image Recorder for JDBC transport protocol

3. Click **Next**.
   The endpoint configuration window opens.

4. Set up the simulation host and range of ports (default is 2999), as appropriate.
   JDBC VSE supports multiple endpoints during recording and playback. In the recorder and the JDBC listen step editor, there is a table of **Driver Host**, **Base Port**, and **Max Port**. When **Base Port** and **Max Port** differ, a unique endpoint is created for the base port and the max port and each port between.

5. Click **Next** to connect to the JDBC Simulation server.
   To stop trying to connect, click **Cancel** in the status window.

6. Select the connection URLs and users to be recorded in the **SQL Activity** tab.
   The **SQL Activity** tab shows the ten most recent statements executed (and how many times they ran) for every unique connection URL and database user combination. This display refreshes approximately every five seconds.

   - To put the selected connection string and user in the URL and user fields at the bottom of the panel, click **To URL**.

   - To put the URL in the **URLs to Record** list, click **Add**.

7. Click the **Loaded Drivers** tab.
   The **Loaded Drivers** tab lists the JDBC drivers that are installed and registered in the system under test.

   - Select a driver and click **To URL** to put a regular expression that matches any connection through that driver in the **URL** field at the bottom of the panel.

   - Click **Add** to add the pattern to the **URLs to Record** list.

- **URLs to Record**
  Lists the URLs added from the **SQL Activity** list or the **URL/User entry** fields. To delete an entry, select it and click **Remove**.

- **User**
  An unnamed area below the **URLs to Record** field where you can include a database user with the URL. If you leave this field blank, the recording applies to all users connecting to the URL.

> **Note:** To enable the **Add** button, supply a connection URL in the first field and a user name in the next field.

The bottom section shows the list of connection URL and database user combinations that are recorded. The URL can be a regular expression. This list is typically initially empty unless the state attribute is set to RECORD on a simulation connection URL for DataSource style systems under test.
If a driver is selected from the drivers list, you can use the **To URL** button to copy a generic regular expression that matches the connection URLs for that driver to the **URL** entry field. If the connection URL (top level) node in the activity tree is selected, you can use the **To URL** button to copy the URL and user to the **URL** and user entry fields. Also, to add the connection URL and user directly to the recording list, use the **Add** button to the right of the activity tree.
Connection URLs can be either exact or a regular expression that matches connection requests and can be added to the recording list with or without a database user. The absence of a database user matches any user attempting to connect. If the **Add** button to the right of the URL and user fields is disabled, the recording list probably already covers the URL and user combination.

8. Click **Next** when the recording list contains all the URLs and users to record.
   The **Recording has begun** window opens.
   The options and dynamic display statistics include:

   - **Total conversations**
     Displays the number of conversations recorded.

   - **Total transactions**
     Displays the number of transactions recorded.

   - **Clear**
     Clears the list of currently recorded transactions.

   The number of **Total conversations** and **Total transactions** increases with the number of transactions recorded.

> **Note:** If no transactions are recorded, you could 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 have performance issues, it is possible that the target database is responding slowly. Check the **user.home\lisatmp\tmanager.log** file for debug messages that report the query execution time.
For example:

```
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
```

If the query time exceeds a threshold, warning messages like the following are generated:

```
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_MOD
IFIED, SERVICE_INFO, CREATED_ON, NOTES, UNK_CONV_RESPONSE, UNK_STLS_RESPONSE
 FROM SVSE_TRAFFIC
```

9. Click **Next** when the recording is complete.
   The recorder prepares to write the .vsi file by verifying request and response bodies. The recorder ensures that, if they are marked as text, that they are text. If they are not text, the type is switched to binary.
   The recorder finishes postprocessing the recording.

   > **Note:** To save the settings on this recording to load into another service image recording, click **Save** above the **Finish** button.

10. Click **Finish** to store the image.

11. Review and save the virtual service model in DevTest Workstation.

Virtualize JDBC

This section provides more detail about how to prepare to use the VSE framework to virtualize JDBC-based database traffic. Information about JDBC configuration is also available in Install the Database Simulator. This section documents the various ways that driver-based JDBC virtualization can be combined, and the configuration options supported. This section does not include any detail about how to implement the configuration on any specific app container.

> **Note:** The JDBC (Driver Based) transport protocol is not supported. This functionality is replaced by agent-based JDBC virtualization.

JDBC virtualization only works with Java 1.6; it does not work with Java 1.5.

System-Level Properties

You can specify the following properties in one of the following ways:

- By using system variables (adding a -**Dvar=val** flag to the startup for the system under test, or through another mechanism)

- In a properties file named **rules.properties**, which has to be on the classpath (probably in the same directory as **lisa-jdbc-sim-*x.y.z*.jar**, where ***x.y.z*** is the DevTest release).

If a property exists in both locations, the system property overrides the configuration file.

- **lisa.jdbc.sim.require.remote**
  Determines whether the driver waits on a connection to VSE before processing commands.
  Default: false

- **lisa.jdbc.hijack.drivermanager**
  Determines whether the driver attempts to take over all the database connections from this application.
  Default: false

- **lisa.jdbc.sim.port**
  Defines the default listen port.
  Default: 2999

- **lisa.log.level**
  Specifies the default log level.
  Values:

  - OFF
  - FATAL
  - ERROR
  - WARN
  - INFO
  - DEBUG
  - DEV

  Default: WARN

- **lisa.log.target**
  Specifies where the logs are written.
  Values:

  - stderr
  - stdout
  - any file location

  Default: stderr

VSE Driver (stand-alone)

The system under test can use the VSE driver directly or the VSE driver can be wrapped in a data source such as the Apache BasicDataSource. In such cases, the configuration (other than the username and password) is passed through the URL. The driver properties include:

- **URL**
  Specifies the actual URL that is passed to the underlying driver. This URL must be the last element passed.

- **State**
  Specifies the initial state for the driver.
  Values:
  - watch
  - record
  - playback
- **JdbcSimPort**
  Designates the port on which to listen for connections from VSE. This value overrides the system-level property.
- **Driver**
  Designates the class name of the real driver to use.
- **User**
- **Password**

Specify the classname of the driver as **com.itko.lisa.vse.jdbc.driver.Driver**.

The following example shows the format of the URL. Everything is optional except the driver and the URL.

```
jdbc:lisasim:driver=<real driver class>;state=<initial state>;
jdbcSimPort=<starting port>;url=<real url>
```

> **Note:** The URL element must come last. Everything following "url=" is passed unchanged to the underlying driver. DevTest supports passing extra information to the underlying driver by embedding it in the real URL.

For example, the following URL is appropriate for a Derby connection:

```
jdbc:lisasim:driver=org.apache.derby.jdbc.ClientDriver;state=watch;
jdbcSimPort=4000;url=jdbc:derby://localhost:1527/sample;create=true
```

This URL instructs the VSE driver to:

- Use the Derby client driver
- Start up in "watch" mode
- Use port 4000 to communicate with VSE
- Pass the URL jdbc:derby://localhost:1527/sample;create=true to the real Derby driver

VSE Datasource Wrapping a Driver

VSE Datasource Wrapping a Driver could be a more typical configuration for app containers than using a VSE driver directly. VSE provides a data source that wraps a real driver. Most application containers provide a mechanism for directly specifying properties for the data source.

Specify the classname of the driver as **com.itko.lisa.vse.jdbc.driver.VSEDataSource**.

Datasource Properties

- **Driver**
  Specifies the name of the real driver to use (the normal configuration).
- **URL**
  Specifies the real URL to connect with (cannot start with **jdbc:lisasim**).
- **User**
- **Password**
- **JdbcSimPort**
  Specifies the port on which to listen for connections from VSE. This value overrides the system-level property.
- **CustomProperties**
  Defines a semicolon-separated list of name=value pairs. These properties are parsed and delegated to the wrapped data source, if any. If the first character is not alphanumeric, it is used as the delimiter instead of a semicolon.

VSE Data source Wrapping a Datasource

Wrapping a data source is an unusual configuration, when an application container does not permit a driver to be instantiated directly with Class.forName(). To use it, instead of specifying **driver=<classname of real driver>** (for example, **oracle.jdbc.OracleDriver**), specify **datasource=< classname of real datasource>** (for example, **oracle.jdbc.pool.OracleDataSource**).

Specify the class name for the driver as **com.itko.lisa.vse.jdbc.driver.VSEDataSource**.

Supporting Multiple Endpoints

If one application has multiple JDBC connections you want to virtualize (or if one appserver has multiple applications with different JDBC connections you want to virtualize), specify a different **JdbcSimPort** for each connection. You can specify this endpoint either as a property to the VSE Data source, or as a parameter on the URL for the driver.

Example:

Assume an application uses two JDBC connections, and you want to virtualize both of them. Also assume the application uses drivers directly.

Specify **com.itko.lisa.vse.jdbc.driver.Driver** for each connection, with the appropriate underlying parameters such as driver and URL for each. For the first connection, you could specify **jdbcSimPort=3000** and for the second you could use **jdbcSimPort=3001**. Your configuration could look like:

```
connection1.url=jdbc:lisasim:driver=org.apache.derby.jdbc.ClientDriver;
jdbcSimPort=3000;url=jdbc:derby://localhost:1527/sample;create=true
connection2.url=jdbc:lisasim:driver=org.apache.derby.jdbc.ClientDriver;
jdbcSimPort=3001;url=jdbc:derby://localhost:1527/sample2;create=true
```

Although you must still record separately, you can deploy both services simultaneously, and can start and stop them independently.

TCP Transport Protocol

This page contains the detailed instructions for recording a virtual service image using the TCP transport protocol using the DevTest Workstation. For information about using the DevTest Portal, see Virtualize Services by Recording (TCP).

For more information about prerequisites and preparatory steps, see Virtualize TCP.

Follow these steps:

1. To start recording a new virtual service image, complete one of the following steps:

   - Click **VSE Recorder** on the main toolbar.

   - Right-click the **VirtualServices** node on the **Project** panel and select **Create a VS Image , by Recording**.

   The **Virtual Service Image Recorder** opens.

2. Complete the Basics tab as in the following graphic:

3. Click **Next**.

4. Enter the information for both the client and the server, select your delimiters, and add SSL parameters.

   - **Listen/Record on port**
     Defines the port on which the client communicates to DevTest.

   - **Target host**
     Defines the name or IP address of the target host where the server runs.

   - **Target port**
     Defines the port number of the target host where the server runs.

   - **Treat request as text**
     Specifies whether the request is treated as text. For more information, see Virtualize TCP.

- **Request Encoding**
  Lists the available request encodings on the machine where DevTest Workstation is running. The default is UTF-8.

- **Treat response as text**
  Specifies whether the response is treated as text. For more information, see Virtualize TCP.

- **Response Encoding**
  Lists the available response encodings on the machine where DevTest Workstation is running. The default is UTF-8.

- **Use SSL to server**
  Specifies whether DevTest uses HTTPS to send the request to the server.

  - **Selected**: DevTest sends an HTTPS (secured layer) request to the server.
    If you select **Use SSL to server**, but you do not select **Use SSL to client**, DevTest uses an HTTP connection for recording. DevTest then sends those requests to the server using HTTPS.

  - **Cleared**: DevTest sends an HTTP request to the server.

- **Use SSL to client**
  Specifies whether to use a custom keystore to play back an SSL request from a client. This option is only enabled when **Use SSL to server** is selected.
  Values:

  - **Selected:** You can specify a custom client keystore and a passphrase. If these parameters are entered, they are used instead of the hard-coded defaults.

  - **Cleared:** You cannot specify a custom client keystore and a passphrase.

- **SSL keystore file**
  Specifies the name of the keystore file.

- **Keystore password**
  Specifies the password associated with the specified keystore file.

5. Click **Next** to configure the request and response delimiters that will be used during recording.

6. When your recording is complete, click **Next**.

7. Select a response data protocol if the response is encrypted, compressed, or otherwise encoded.

8. The recorder attempts to detect message delimiters that tell DevTest when it has read a complete request or response. Confirm, or correct, these delimiters on this screen.

   Note: A Request delimiter is mandatory. A Response delimiter must be selected for Live Invocation to be available.

9. The recorder verifies request and response bodies to ensure that, if they are marked as text, that they are text. If they are not, the type is switched to binary.

> Note: To save the settings on this recording to load into another service image recording, click **Save** above the **Finish** button.

The TCP/IP protocol supports a live invocation step. To enable the live invocation step, select a response protocol. If you do not select a response protocol, no live invocation step is included in the VSM.

CICS (LINK DTP MRO) Transport Protocol

This section contains the detailed instructions for recording a virtual service image using the CICS transport protocol.

The following video shows an example of recording.

The section contains the following pages:

- Recording IBM CICS Overview
- Recording IBM CICS Example
- Using the CICS Programs to Virtualize Panel

CICS Overview

To allow recording IBM CICS service images, DevTest has a mainframe bridge that is a part of the registry. The mainframe bridge supplies a single point of contact for all of the DevTest Workstations to the mainframe. The mainframe has an LPAR agent that runs on a z/OS LPAR. The LPAR agent provides a single point of contact to all of the CICS agents.

Each CICS region has a DevTest CICS agent that provides the virtualization capability. The CICS agents communicate with the LPAR agent over sockets. The LPAR agent communicates with the mainframe bridge over sockets. The mainframe bridge provides an ActiveMQ interface through the registry to all of the DevTest clients.

The mainframe bridge can run in the following modes:

- **Client**
  The mainframe bridge initiates the connection to the LPAR agent.

- **Server**
  The mainframe bridge waits for the LPAR agent to initiate the connection.

The lisa.properties file contains properties to enable the mainframe bridge.

CICS Example

This example uses the VSE recorder to record a CICS LINK service image.

When the registry is started, it shows that the mainframe bridge has started in client mode and the bridge is connected to the LPAR agent.

This example demonstrates eliminating a resource dependency. In the example, a CICS application requires a VSAM file for input to fields for account balance information, but the VSAM file is unavailable. This example shows how to virtualize the program that maintains the missing VSAM file, eliminating the resource dependency.

The following graphic shows the CICS transaction (DEM3) that is running. The transaction runs from a 3270 terminal, but it could run from the CICS Transaction Gateway.

Running the transaction initiates DEMOAPP3, which performs a CICS LINK to DEMOINQ. DEMOINQ then runs the following CICS LINKs:

1. A local LINK passes a COMMAREA to DEMOGETN to get the customer name and address from a VSAM.

2. A CICS LINK to CICSB passes a COMMAREA to DEMOGETB. DEMOGETB reads the VSAM file DEMOGETBAL, which is not available in the example.

3. A CICS LINK passes a COMMAREA to DEMOGETP to get the telephone number information.

4. A CICS LINK passes a CHANNEL with CONTAINERS to DEMOLOGI.

The following CICS regions are involved:

- Jobname CICSA, with a system ID of S660 and an applid of CICSTS41
- Jobname CICSB, with a system ID of S650 and an applid of CICSTS32

This example virtualizes DEMOGETB on CICSTS41. The example uses the DevTest agent, running in CICSA, communicating with the LPAR agent, communicating with the VSE server. We first run the CICS transaction to record the CICS LINK to DEMOGETB. We then use that recording to virtualize the CICS LINK.

The **Virtual Service Image Recorder Basics** tab contains the name of the service image (DEMOGETB.vsi), the VSM (DEMOGETB.vsm), and the transport protocol of CICS LINK.

The **CICS Programs to Virtualize** panel displays next. For detailed information about this panel, see Using the CICS Programs to Virtualize Panel.

Recording starts when the **Next** button is clicked. During recording, we run transaction DEM3 to invoke DEMOGETB with a CICS LINK.

Screenshot of a CICS screen running the demo for the example recording.

As the following graphic shows, after it runs once, VSE records one transaction.

Screenshot of VSI recorder with one transaction recorded from the example CICS LINK recording.

On the next panel, VSE adds the CICS Copybook data protocol to both the request and response sides.

The following graphic shows the copybook mapping XML file to use in this example. This file defines the mapping for DEMOGETB and it indicates to use DEMOGETB.txt as the copybook if a request for the program DEMOGETB exists.

payload-copybook-mapping-CICS.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<payloads>
    <!-- DEMOGETB -->
    <payload name="DEMOGETB" type="request" matchType="metaData"
             key="DEMOGETB" value="DEMOGETB" >
       <section name="Body">
            <copybook key="">DEMOGETB.txt</copybook>
        </section>
    </payload>
    <payload name="DEMOGETB" type="response" key="DEMOGETB" value="DEMOGETB">
        <section name="Body">
            <copybook key="">DEMOGETB.txt</copybook>
        </section>
    </payload>
</payloads
```

Copybook mapping XML file contents for the CICS LINK demo recording.

As the following example shows, DEMOGETB.txt contains the COBOL copybook definition:

```
Copybooks/DEMOGETB.txt
   05  DEMOGETB-COMMAREA.
     10  DEMOGETB-RETURN             PIC X.
     10  DEMOGETB-MESSAGE            PIC X(70).
     10  BALANCE-REC.
         15  BALANCE-ACCOUNT-NBR     PIC X(8).
         15  FILLER                  PIC X.
         15  BALANCE-BALANCE         PIC X(14).
         15  FILLER                  PIC X.
         15  BALANCE-AVERAGE         PIC X(14).
```

On the next panel, the folder that has DEMOGETB.txt is specified, and the mapping XML. The codepage is changed to IBMO37.

Screenshot of the Configure How to Parse the Copybook Payloads panel for the CICS LINK recording example.

Click **Next**. The copybook processes the data.

Click **Next** to stop processing and complete the service image recording.

When you open the VSI in the **Service Image Editor**, you can see that the copybook mapped the request data, then converted it to attributes. For example:

Screenshot of the request data arguments for the CICS LINK recording example.

The copybook has also formatted the response. The following graphic shows the response to return when using virtualization:

Screenshot of the response to send back when using virtualization for the CICS LINK recording example.

Example

In this example, the VSAM file DEMOBAL is made unavailable, and program DEMO3 is run. DEMO3 fails because of the unavailability of DEMOBAL.

Screenshot of CICS screen for the CICS LINK example, showing the SUT being unavailable.

Next, the virtual service is deployed.

When the program DEMO3 is run again, it works, even though the VSAM file is still unavailable.

Screenshot of the CICS screen for the CICS LINK recording example, with success having VSE virtualize the SUT.

CICS Programs to Virtualize Panel

Use the **CICS Programs to Virtualize** panel to:

- Explore the CICS regions that are connected to DevTest LPAR Agents
- Explore the programs that are defined to these CICS regions
- Select the programs to virtualize
- Manually add programs to virtualize
- Set the filters to refine the CICS LINK statements, transactions, and CICS users that are virtualized
- Establish a group membership to virtualize CICS programs in a group of CICS regions

The following graphic shows the main areas of the **CICS Programs to Virtualize** panel:

The CICS Programs to virtualize selection panel has two main areas.

Available Programs

The **Available Programs** area lets you explore CICS regions that are connected to DevTest with the DevTest CICS Agent and see the programs that are defined there.

You can explore the known CICS regions and programs by:

- LPAR (z/OS Logical Partition)
- CICSPlex
- User-defined grouping
- Region

VTAM APPLIDs (application IDs) identify the CICS regions.

- **Program Name Prefix**
  Defines the prefix that the recorder uses to filter which programs to display. The **Available Programs** area only displays programs that begin with the specified prefix. Always add a program prefix.
- **Max**
  Defines the maximum number of programs to display from one CICS region.

The CICS Programs to virtualize selection panel with the CICS LINK recording example environment shown.

Clicking LPAR shows the known LPARs (in this case, S0W1).

Clicking the plus sign next to S0W1 shows the known CICS regions in SOW1: CICST242, CICSTS32, CICSTS41, and CICSTS42.

Clicking the plus sign next to CICSTS41 shows the known programs that match the prefix DEMOGET on CICSTS41 (DEMOGETB and DEMOGETN).

The list under SOW1 also contains DEMOGETB and DEMOGETN because they are programs that were found in the LPAR.

The CICS Programs to virtualize selection panel with nodes expanded to show more detail.

Clicking **CICSPlex** shows the known CICSPlexs and reveals DEVPLEX1 and DEVPLEX2.

Clicking the plus sign next to DEVPLEX1 shows the known CICS regions: CICST242, CICSTS41, and CICSTS42.

Clicking the plus sign next to CICSTS41 shows the known programs that match the prefix on CICSTS41 (DEMOGETB and DEMOGETN).

The list under DEBPLEX1 also contains DEMOGETB and DEMOGETN because they are programs that were found in the CICSPlex.

Note: For a CICS region to be a member of a CICSPlex group, you must modify a DevTest CICS agent user exit. For more information, see CICS Agent User Exits.

The CICS Programs to virtualize selection panel with User Groups expanded.

Clicking **User Group** shows the known user groups and reveals GROUP1 and GROUP2.

Clicking the plus sign next to GROUP1 shows the known CICS regions: CICST242, CICSTS41, and CICSTS42.

Clicking the plus sign next to CICSTS41 shows the known programs that match the prefix on CICSTS41 (DEMOGETB and DEMOGETN).

The list under GROUP1 also contains DEMOGETB and DEMOGETN because they are programs that were found in the user group.

Note: For a CICS region to be a member of a user group, you must modify a DevTest CICS agent user exit. For more information, see CICS Agent User Exits.

The CICS Programs to virtualize selection panel with Region expanded.

Clicking **Region** shows the known CICS Regions and reveals CICST242, CICSTS32, CICSTS41, and CICSTS42.

Clicking the plus sign next to CICSTS41 shows the groups that contain this CICS region.

Clicking the plus sign next to S0W1 (the LPAR group of which this region is a member) shows the known programs that match the prefix on CICSTS41 (DEMOGETB and DEMOGETN).

The list under GROUP1 also contains DEMOGETB and DEMOGETN because they are programs that were found in the user group.

The CICS Programs to virtualize selection panel with Program expanded.

When the program information is retrieved by clicking a CICS region, the grouping by Program folder appears.

Clicking **Program** reveals all known CICS programs; DEMOGETB and DEMOGETN in this case.

Clicking the plus sign next to DEMOGETB expands the known CICS regions and groups containing DEMOGETB.

Programs to Virtualize

To add programs, double-click them and they appear in the **Programs to Virtualize** panel. Or, click **Add** on the **Programs to Virtualize** panel to add a program manually.

When a program is listed in the **Programs to Virtualize** panel, double-click it to add filters and groupings.

In this example, program DEMOGETB is virtualized in CICS region CICSTS41. Group S0W1 and LPAR are ignored unless the Any Region check box is selected.

Filters allow you to specify which of the CICS LINKs to DEMOGETB are virtualized.

Screenshot of the Add or Edit Programs window for CICS LINK recordings, with Any Region selected.

Selecting the **Any Region** check box clears the **Region** text box and indicates that DEMOGETB will be virtualized in ALL CICS regions in LPAR group S0W1 (all CICS regions residing in LPAR S0W1).

Screenshot of the Add or Edit Programs window for CICS LINK recordings, with filters entered.

In the previous graphic, filters are used to narrow the programs to virtualize.

- **Link Transaction ID**
  Indicates that CICS LINKs to DEMOGETB are virtualized only if they are coded with TRANSID ('TRN1').

- **Link System ID**
  Indicated that CICS LINKs to DEMOGETB are virtualized only if coded with SYSID('SYSB').

- **EIB Transaction ID**
  Indicates that CICS LINKs to DEMOGETB are virtualized only if executed while running under the transaction DEM3.

- **User ID**
  Indicates that CICS LINKs to DEMOGETB are virtualized only if executed while running under user ID CICSUSR1.

CICS Transaction Gateway (ECI) Transport Protocol

To record CICS Transaction Gateway (ECI) images:

1. Select CICS Transaction Gateway (ECI) as the transport protocol on the **Basics** tab of the **Virtual Service Image Recorder**.

2. Complete the fields on the **Basics** tab and click **Next**.
   The next step in the recorder opens.

3. Enter the port and host information for this step.

- **Listen/Record on port**
  Defines the port on which the client communicates to DevTest.

- **Target host**
  Defines the name or IP address of the target host where the server runs.

- **Target port**
  Defines the target port number on which the server listens.

- **Expect SSL From Clients**
  Specifies whether the recorder expects clients to connect to it using SSL. The related keystore and password, if provided, are used to obtain security material (such as certificates).
  Values:

  - **Selected:** The recorder expects clients to use SSL to connect.

  - **Cleared:** The recorder does not expect clients to use SSL to connect.

- **Initiate SSL To Server**
  Specifies whether the recorder connects to the real system using SSL. The related keystore and password, if provided, are used to obtain security material (such as certificates).
  Values:

  - **Selected:** The recorder uses SSL to connect.

  - **Cleared:** The recorder does not use SSL to connect.

- **SSL keystore file**
  Specifies the name of the keystore file.

- **Keystore password**
  Specifies the password associated with the specified keystore file.

4. Click **Next** to display the recording window.

5. Start your application that communicates with the CTG server and perform the activity you want to record.

6. When you see transactions are recorded on the **Virtual Service Image Recorder** window, click **Next**.
   The data protocols window opens.

7. Click **Next**.
   The conversation starter window opens.

8. Click **Next**.

9. Select the options to display the VSM and VSI, and click **Finish**.
   The Service Image Editor displays the CTG service image, as shown in the following graphic.

Service Image Editor with CTG transactions.

IMS Connect Transport Protocol

This page describes how to record a service image that uses the IMS Connect transport protocol.

- IMS Connect Customized Format Files

To record IMS Connect service images:

1. Before starting the recorder, add the following properties in **local.properties** as necessary:

   - If the system under test sends responses that have a four-byte LLLL length field at the beginning of the response message, set the following property:

     `lisa.vse.protocol.ims.response.includes.llll=true`

   - If the system under test receives requests that have a four-byte LLZZ length field at the beginning of the request message, set the following property:

     `lisa.vse.ims.connect.llzz.request=true`

   - If the system under test sends responses that have a four-byte LLZZ length field at the beginning of the response message, set the following property:

     `lisa.vse.ims.connect.llzz.response=true`

   - If the system under test uses copybooks for parsing the payloads, add the following property:

     `lisa.vse.copybook.unknown.passthrough=true`

   This ensures that the binary responses that represent acknowledgment messages (they do not include any payload data) are bypassed from copybook processing.

2. Select IMS Connect as the transport protocol on the **Basics** tab of the **Virtual Service Image Recorder**.

3. Complete the fields on the **Basics** tab, then click **Next**.
   The next step in the recorder opens.

4. Enter the port and host information:

   - **Listen/Record on port**
     Defines the port on which the client communicates to DevTest.

   - **Target host**
     Defines the name or IP address of the target host where the server runs.

   - **Target port**
     Defines the target port number listened to by the server. Leave this field blank if you will select a Proxy passthrough style.
     Defaults: 80 (HTTP) and 443 (HTTPS)

   - **IMS Format File**
     Defines one of the following:

     - The name of the file that supports IMS connect requests that use the 160-byte header that the IBM sample app uses

     - An 80-byte header.

   The field definitions for these supported 160-byte and 80-byte headers are provided in the **ims.format** file in the LISA_HOME directory.
   To use the IMS Connect support that is included in DevTest by default, leave this field blank.

   - **Character Set**
     Defines how the data is encoded by the application to be virtualized.
     Values:

     - ASCII - CP1252

     - EBCDIC - CP037

5. Click **Next** to display the recording window.

6. Start your application that communicates with the IMS Connect server and perform the activity that you want to record.
   Ensure that the host and port for this application are the host and port for the recorder (typically localhost:8001).

7. When you see transactions have been recorded on the **Virtual Service Image Recorder** window, click **Next**.
   The data protocols window opens.
   By default, the TransactionCode header field (if it exists) is set as the request operation. Also, TransactionCode, DestinationId, and ClientId are added as arguments, if these fields exist in the format file definition.

8. Add data protocols, including the Copybook data protocol if needed, and click **Next**.
   The conversation starter window opens.

9. Click **Next**.

10. Select the options to display the VSM and VSI, then click **Finish**.
    The Service Image Editor displays the IMS Connect service image.

> **Note:** CA Service Virtualization supports IMS Connect recording and playback over SSL. The only restriction is that it only supports one keystore file.

IMS Connect Customized Format Files

If the system under test uses request headers that differ from the ones DevTest supports by default, follow these steps before recording:

1. Create a file **<custom-format>.format** in the **Data** folder of the DevTest project.

2. Edit the file in a text editor and add the following text to it. Provide your field definitions under the RequestUserHeader area.

```
#-----------------------------------------------------------------------------
----------
# The IMS™ request message (IRM) header contains a 28-byte fixed-format
# section that is common to all messages from all IMS Connect client applica
tions
# that communicate with IMS TM.
#-----------------------------------------------------------------------------
----------
RequestHeaderCommon {
    LLLL                        int;                    #total message lengt
h IRM + user data
    IRMFixedHeader {
        LL                      short;                  #IRM_LEN, total leng
th of the header segment including user header portion
        ZZ                      byte;                   #IRM_ARCH
        Flag0                   byte;                   #IRM_F0
        UserExitId              string(8);              #IRM_ID
        NakReasonCode           byte(2);                #IRM_NAK_RSNCDE
        Reserved1               byte(2);                #IRM_RES1
        MessageType             byte;                   #IRM_MessageType
        WaitTime                byte;                   #IRM_TIMER
        SocketConnectionType    byte;                   #IRM_SOCT
        EncodingSchema          byte;                   #IRM_ES
        ClientId                byte(8);                #IRM_CLIENTID
    }
}

#-----------------------------------------------------------------------------
----------
# Format of user portion of IRM some custom header
#
# Following the 4-byte length field and the 28-byte fixed portion of the
# IMS™ request message (IRM) header in IMS Connect client input messages,
# user-written client applications can include a user-
defined section in the IRM.
#
#-----------------------------------------------------------------------------
----------
RequestUserHeader {
    MyFlag1 byte;
 MyFlag2 byte;
 MyFlag3 byte;
 MyFlag4 byte;
 TransactionCode string(8);
 DestinationId string(8);
 LogicalTerminal string(8);
 Miscellaneous byte(20);
```

```
}

#-----------------------------------------------------------------------------
----------
# Format of data segments of request message
#-----------------------------------------------------------------------------
----------
RequestPayloadSegment  {
        LL                      short;
        ZZ                      byte(2);
        Data                    byte(LL:inclusive);
}

#-----------------------------------------------------------------------------
----------
# Format of header for response message. Some responses will have it, some w
on't
#-----------------------------------------------------------------------------
----------
ResponseMessageHeader {
        LLLL                    int;                    # followed by multipl
e ResponsePayloadSegment
}

#-----------------------------------------------------------------------------
----------
# Format of data segments of request message
#-----------------------------------------------------------------------------
----------
ResponsePayloadSegment {
        LL                      short;
        ZZ                      byte(2);
        Data                    byte(LL:inclusive);
}
```

When the file is created and saved in the **Data** folder, it appears in the IMS Format file field in the recorder. You can select the format file for applications that use nondefault request headers.

SAP RFC via JCo Transport Protocol

This page describes how to record a service image that uses the SAP RFC via JCo transport protocol.

Follow these steps:

1. Select SAP RFC via JCo as the transport protocol on the **Basics** tab of the **Virtual Service Image Recorder**.

2. Complete the fields on the **Basics** tab, then click **Next**.
   The next step in the recorder opens.

3. Complete the connection details fields.

   - **Client System Name**
     Specifies a unique name to identify the client SAP system.

   - **Client System Connection Properties**
     The Client System Connection properties file contains connection properties to connect to the destination on the client system. This must be a .properties file in the Data directory of your project and contains properties that are typically found in a . jcoServer file. This file MUST NOT specify jco.server.repository_destination. For more information about supported properties, see the JavaDocs for **com.sap.conn.jco.ext. ServerDataProvider** in the **doc** folder of your installation directory.

- **RFC Repository Name**
  Specifies a unique name to identify the SAP system that has the repository for the RFC template. This name is often the same as the Destination System.

- **RFC Repository Connection Properties**
  Defines the Repository Connection properties file that contains connection properties to connect to the system that has the repository. This must be a .properties file in the Data directory of your project and contains properties that are typically found in a .jcoDestination file. For more information about supported properties, see the JavaDocs for **com.sap.conn.jco.ext.DestinationDataProvider** in the **doc** folder of your installation directory.

- **Destination System Name**
  Defines a unique name that identifies the SAP system on which the RFC executes. This is often the same as the Repository Name.

- **Destination System Connection Properties**
  Specifies the Destination System Connection properties file that contains connection properties with which to connect to the system that has the repository. This must be a .properties file in the Data directory of your project and contains properties that are typically found in a .jcoDestination file. For more information about supported properties, see the JavaDocs for **com.sap.conn.jco.ext.DestinationDataProvider** in the **doc** folder of your installation directory.This can be the same file as the Repository Connection Properties.

- **Function Name**
  Specifies the name of the RFC that DevTest should intercept.

4. Click **Next** to display the recording window.

5. Start the application that communicates with the SAP server and perform the activity that you want to record.

6. When you see transactions are recorded on the **Virtual Service Image Recorder** window, click **Next**.
   The data protocols window opens.

7. Click **Next**.
   The conversation starter window opens.

8. Click **Next**.

9. Select the options to display the VSM and VSI, then click **Finish**.
   The Service Image Editor displays the SAP RFC service image.

JCo IDoc Transport Protocol

The SAP JCo IDoc transport protocol allows you to create virtual services that use the SAP JCo IDoc protocol.

The JCo IDoc protocol supports the creation of a virtual service that records and plays back IDocs sent from one SAP system to another using an RFC destination.

The SAP system that initiates communication and sends IDocs is referred to as the Client SAP system. The IDocs sent are referred to as the Request IDocs.

The SAP system that receives the Request IDocs is referred to as the Server SAP system. The IDocs that the Server SAP system returns are referred to as the Response IDocs.

The following graphic shows the regular flow of IDocs between two SAP systems.

Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

To record JCo IDoc service images:

1. Select **JCo IDoc Protocol** as the transport protocol on the **Basics** tab of the **Virtual Service Image Recorder**.

2. Complete the fields on the **Basics** tab and click **Next**.
   The next step in the recorder opens.

3. Complete the connection details fields.

   - **Client RFC Connection Properties**
     Defines the Client RFC Connection properties file that contains connection properties that VSE uses to register itself under a program ID to an SAP gateway and receive IDocs. The properties should be the same as those specified in a .jcoServer file.

   - **Client RFC Destination Name**
     Specifies a unique name that identifies the RFC destination.

   - **Client System Connection Properties**
     Specifies the Client System Connection properties file that contains connection properties to return IDocs to the client SAP system. These properties should be the same as those specified in a .jcoDestination file that can be used to connect to the client SAP system.

   - **Client System Name**
     Specifies a unique name to identify the client SAP system.

   - **Request Identifier XPath Expressions**
     Specifies the XPath expressions that the protocol uses with the request IDoc XML to generate an identifier. The request identifier XPath expresions can be a single XPath

expression. This identifier is used to correlate a request IDoc to a response IDoc. XPath expressions can also be a comma-separated list of XPath expressions, in which case the resulting values from the multiple expressions are concatenated (separated by dashes) and used as an identifier.

- **Server RFC Connection Properties**
  Specifies a properties file that contains connection properties that VSE uses to register itself under a program ID to an SAP gateway and receive IDocs. The properties should be the same as those specified in a .jcoServer file to start a JCo server program that receives IDocs from the server SAP system.

- **Server RFC Destination Name**
  Specifies a unique name to identify the Server RFC destination.

- **Server System Connection Properties**
  The Server System Connection properties file contains connection properties to return IDocs to the client SAP system. These properties should be the same as those specified in a .jcoDestination file that can be used to connect to the SAP server system.

- **Server System Name**
  Specifies a unique name to identify the Server SAP system.

- **Response Identifier XPath Expressions**
  Defines the XPath expressions that the protocol uses with the response IDoc XML to generate an identifier. The Response Identifier XPath Expressions can be a single XPath expression. This identifier is used to correlate a response IDoc to a request IDoc that was received earlier. XPath expressions can also be a comma-separated list of XPath expressions, in which case the resulting values from the multiple expressions are concatenated (separated by dashes) and used as an identifier.

4. Click **Next** to open the recording window.

5. Start the application that communicates with the SAP server and perform the activity that you want to record.

6. When you see that transactions have been recorded on the **Virtual Service Image Recorder** window, click **Next**.
   The data protocols window opens.
   The XML data protocol is selected on the request-side data protocols, because VSE stores the IDocs as XML. Remove the XML data protocol, as it overrides the operation name that the JCo IDoc protocol sets.

7. Click **Next**.
   The conversation starter window opens.

8. Click **Next**.

9. Select the options to display the VSM and VSI, and click **Finish**.
   The Service Image Editor displays the SAP JCo IDoc service image.

Opaque Data Processing Transport Protocol

Opaque Data Processing (ODP) allows CA Service Virtualization to virtualize data in sufficient detail when the format of the requests and responses is not known. ODP eliminates the need for a new data handler every time you encounter a new message format.

By recording several requests and corresponding responses, CA Service Virtualization can infer the message structure. This means CA Service Virtualization is able to correlate bytes in a request to corresponding bytes in a response, giving the same "magic string" behavior that is available in other protocols. CA Service Virtualization is also able to sufficiently understand the request structure to intelligently match new requests that are encountered when the virtual service is played back.

Underneath, ODP uses a patented algorithm to compare an incoming request to all of the requests in the recorded ODP service image. The closest matching request is selected, and the corresponding response is returned, after performing a dynamic magic string substitution.

The ODP matching algorithm applies entropy-derived weights during the matching process. The entropy weighting process infers which bytes in the message are more important. For example, which correspond to the operation type, as opposed to the rest of the payload, and gives those bytes a greater importance during the matching process. The entropy weighting process works best with larger samples of recorded messages (100 or more) and with a diverse sampling of parameter values.

The ODP matching and response algorithm has been shown to work on both binary and textual message protocols. The best results have been obtained with protocols that use fixed width fields (such as IMS) or delimited fields (such as XML-based protocols). Reasonable accuracy has also been obtained with length-encoded protocol formats (for example, ASN.1), but these pose the greatest challenge.

ODP is supported for traffic that can be captured using raw TCP/IP sockets. The data can also be imported from a PCAP file.

The following video explains the concepts of ODP.

The following video demonstrates recording and playback. One portion of the video shows the DevTest Console, which is no longer part of DevTest.

How to Use ODP

Use the following instructions for recording a virtual service image using ODP.

For more information about prerequisites and preparatory steps, see Preparing to Virtualize TCP.

Follow these steps:

1. Open DevTest Workstation.
2. To start recording a new virtual service image, complete one of the following steps:
   - Click **VSE Recorder** on the main toolbar.
   - Right-click the **VirtualServices** node on the **Project** panel and select **Create a VS Image**, **by Recording**.

   The **Virtual Service Image Recorder** opens.
3. Complete the Basics tab as in the following graphic:

Basics tab on the VS Image Recorder for ODP

4. Click **Next**.

5. Enter the information for both the client and the server, select your encoding, and add SSL parameters.

Client and Server Data for ODP

- **Listen/Record on port**
  Defines the port on which the client communicates to DevTest.

- **Target host**
  Defines the name or IP address of the target host where the server runs.

- **Target port**
  Defines the port number that the server is listening on.

- **Treat request as text**
  Specifies whether the request is treated as text. For more information, see Preparing to Virtualize TCP.

- **Request Encoding**
  Lists the available request encodings on the machine where DevTest Workstation is running. The default is UTF8.

- **Treat response as text**
  Specifies whether the response is treated as text. For more information, see Preparing to Virtualize TCP.

- **Response Encoding**
  Lists the available response encodings on the machine where DevTest Workstation is running. The default is UTF8.

- **Use SSL to server**
  Specifies whether DevTest uses HTTPS to send the request to the server.

  - **Selected**: DevTest sends an HTTPS (secured layer) request to the server.
    If you select **Use SSL to server**, but you do not select **Use SSL to client**, DevTest uses an HTTP connection for recording. DevTest then sends those requests to the server using HTTPS.

  - **Cleared**: DevTest sends an HTTP request to the server.

- **Use SSL to client**
  Specifies whether to use a custom keystore to play back an SSL request from a client. This option is only enabled when **Use SSL to server** is selected.
  Values:

  - **Selected:** You can specify a custom client keystore and a passphrase. If these parameters are entered, they are used instead of the hard-coded defaults.

  - **Cleared:** You cannot specify a custom client keystore and a passphrase.

- **SSL keystore file**
  Specifies the name of the keystore file.

- **Keystore password**
  Specifies the password associated with the specified keystore file.

6. Click **Next** to start recording.

Recorder recording ODP transactions

7. When your recording is complete, click **Next**.
   ODP relies on the request body, rather than operation and arguments, for matching. Therefore, most request data protocols are not appropriate.
   Response data protocols, on the other hand, are appropriate if the response is encrypted, compressed, or otherwise encoded.

Selecting data protocols for ODP recording

8. The recorder attempts to detect message delimiters that tell DevTest when it has read a complete request or response. Confirm, or correct, these delimiters on this screen.

> **Note**: A Request delimiter is mandatory. A Response delimiter must be selected for Live Invocation to be available.

Choosing delimeters for ODP recording

9. After recording, the recorder verifies request and response bodies to ensure that, if they are marked as text, that they are actually text. If they are not, the type is switched to binary.

SSL with VSE Recording

During VSE recording, the SSL Server application is the System Under Test.

- Use SSL to the Server
- Use SSL to the Server and Use SSL to Client
- Playback of the VSM

Use SSL to the Server

If **Use SSL to the Server** only is selected during recorder setup (and **Use SSL to Client** is not selected) the client sends a plain HTTP connection to DevTest but the recorder sends an HTTPS (SSL secured socket layer) request to the server application. In this case, *client* denotes the application or test case sending the request.

By default, DevTest is configured to trust any certificate that is sent back from the server application. You can override this behavior by setting the following properties in the **local.properties** file of the installation directory to configure DevTest with a custom trust store:

- lisa.net **.trustStore**
  The full path to the custom trust store file to use

- lisa.net **.trustStore.password**
  The password for the custom trust store file. This property is automatically encrypted and replaced by the lisa.net **.t rustStore.password_enc** property.

If the SSL server requests Client Authentication (two-way authentication), the recorder simulates a client to complete the two-way client authentication. The SSL client-side keystore information that is specified in the **Use SSL to Server** section of the recording wizard is used to send a certificate to the SSL server. If SSL client-side keystore information was not specified, DevTest uses the following properties in the **local.properties** file of the installation directory:

- **SSL keystore file**
  DevTest determines the custom client-side keystore file to use by looking up the following properties (in order of importance):

  - **ssl.client.cert.path**

  - **lisa.default.keystor**
    If neither of these properties exists, DevTest uses the default keystore file that is located at **{{LISA_HOME}}/webreckeys.ks.**

  - DevTest determines the custom client-side keystore password to use by looking up the following properties (in order of importance):

- **Keystore password**

  - **ssl.client.cert.pass.encrypted**

  - **ssl.client.cert.password**

  - **lisa.default.keystore.pass.encrypted**

  - **lisa.default.keystore.pass**
    If none of these properties exist, DevTest uses the password that is associated with the default keystore file located at **{{LISA_HOME}}/webreckeys.ks.**

If the SSL client-side keystore contains multiple certificates, VSE uses the first one.

Use SSL to the Server and Use SSL to Client

To record, the client or test case sends to the configured listen/record on port. The recorder **Target Host** is the real SSL server and the **Target port** is the SSL port that the SSL server uses (typically, 443 or 8443).

During recording, the SSL handshake is between the client (recorder) and the SSL server. The server sends its certificate and DevTest authenticates it. If the server requests client authentication, the certificate that is in **local.properties** is used. If there is not a valid keystore in **ssl.client.cert.path** and the server requests client authentication, then a bad_certificate situation is returned because there is not a certificate for the client recorder to return to the server.

During recording, there is another SSL handshake is between the client (application or test case) and the server (recorder). The recorder sends the certificate that is based on the SSL server-side keystore information that is specified in the **Use SSL to Client** section of the recording wizard. If SSL server-side keystore information was not specified, DevTest uses the following properties in the **local.properties** file of the installation directory:

- **SSL keystore file**
  DevTest determines the custom server-side keystore file to use by looking up the following properties (in order of importance):

- **ssl.server.cert.path**
  If this property does not exist, DevTest uses the default keystore file that is located at **{{LISA_HOME}}/webreckeys.ks.**

- **Keystore password**
  DevTest determines the custom server-side keystore password to use by looking up the following properties (in order of importance):

- **ssl.server.cert.pass.encrypted**

- - **ssl.server.cert.password**
    If none of these properties exist, DevTest uses the password that is associated with the default keystore file that is located at **{{LISA_HOME}}/webreckeys.ks.**

If the SSL server-side keystore contains multiple certificates, CA Service Virtualization uses the first one.

You can configure the recorder to request Client Authentication (two-way authentication) in the **Enable Client Certificate Authentication** section of the recording wizard. If the **Enable Client Certificate Authentication** check box is checked, DevTest is configured to request a client certificate that is based on one of the following options:

- Request Client Certificate
  - DevTest requests a client certificate during the SSL handshake without requiring that one is sent back.
  - This is the default option when client certificate authentication is enabled.
- Require Client Certificate
  - DevTest requires a valid client certificate during the SSL handshake.
  - If the client does not send back a certificate or it is invalid, DevTest stops the SSL handshake and the connection fails.

Playback of the VSM

If the Listen Step was executed and the **Use SSL to Client** option is selected, an SSL handshake occurs between the client application or test case client and the VSM. The server-side keystore information that is provided is used to send an SSL server certificate. If there is no server-side keystore information that is specified, DevTest uses the properties from the **local.properties** from the installation directory.

If the Live Invocation Step was executed and the **Use SSL to Server** option is selected, an SSL handshake occurs between the VSM client to the real server. If the real server requests client authentication (two-way authentication), the client-side keystore information that is provided is used to send an SSL client certificate. If there is no client-side keystore information that is specified, DevTest uses the properties from the **local.properties** from the installation directory.

Work with VSMs

A *Virtual Service Model (VSM)* is a specialized type of test case that becomes the endpoint of a virtualized service. To start the **Virtual Service Image Recorder**, create a VSM.

Contents

- Create a VSM
- Open a VSM

Create a VSM

Follow these steps:

1. Open an existing project or create a project.
   The project opens, and the left **Project** panel shows the project folder and its subfolders.

2. Right-click the **VirtualServices** subfolder node and select **Create New VS Model**.
   Although you can also create a VS Model in another folder, it is a good practice to create it under the **VirtualServices** folder.
   The **VS Model Editor** window opens.

3. Browse to the location for the new VSM.

4. Enter a unique name for the VSM in the **File name** field. The extension is **.vsm**.

5. Click **Save**.
   The new VSM opens.
   When a VSM is open, the **Commands** menu shows menu items relevant to a VSM.

Open a VSM

Follow these steps:

1. Click **Open** on the toolbar, or select a project from the **Open Recent** project list on the **Quick Start** window.

2. Select the virtual service model from its folder in the **Project** panel.
   Virtual service models typically reside in the **VServices** or **VirtualServices** folder.

Using Data Protocols

A data protocol is responsible for parsing requests.

> **Note:** A data protocol is also known as a data handler, a data protocol handler, or a DPH.

Some transport protocols allow (or require) a data protocol to which the job of creating requests is delegated. As a result, the data protocol has to know the request payload. One transport protocol can have many data protocols. Some transport protocols (for example, JDBC) do not allow a data protocol.

During recording, all transport protocols try to detect the request or response payload and default appropriate data protocols on either the request or response side. The indicated data protocol handlers are added for the following payloads:

- **SOAP:** Web Services (SOAP) data protocol handler
- **XML:** XML data protocol handler
- **HTML:** No data protocol handler
- **Signed SOAP:** A request-side WS-Security Request and a response-side WS-Security Response data protocol handler
- **Encrypted SOAP:** A request-side WS-Security Request and a response-side WS-Security Response data protocol handler
- **SOAP Security:** A request-side WS-Security Request and a response-side WS-Security Response data protocol handler

You can add more data protocols, reorder the defaulted ones, or delete the defaults. If you are creating a service image from request/response pairs, a raw traffic file, or means other than recording and you have specified data protocol handlers, the defaults are not added.

You can chain data protocol data handlers to work with each other. For example, on the request side, you can use the following data protocols together:

- Delimited Text
- Generic XML Payload Parser
- Request Data Manager

Auto Hash Transaction Discovery Data Protocol

The Auto Hash Transaction Discovery data protocol uses the hash code of the data to identify a message. The hash code changes with even a slight change in the data, which effectively makes all requests unique. This data protocol is useful if you run the same small set of requests against the service.

The Auto Hash Transaction Discovery data protocol has a simple purpose. As it is given requests to handle, the standard Java hash code function is applied to the text version of the request body. The resulting hash code value is then added as a new argument in the request named **lisa.vse.auto.hashDiscovery**.

This ability can be helpful, especially in virtualizing messaging, for creating a reasonably unique value to use for identifying conversations.

Because this protocol requires no configuration information, it does not present a window in the recording wizard.

CICS Copybook Data Protocol

The CICS Copybook data protocol splits the recorded request into its respective container chunks. Each chunk is then sent to the Copybook data protocol and aggregated with the corresponding XML. The resulting XML is the aggregated XML of the containers.

This example demonstrates the CICS Copybook data protocol usage.

Follow these steps:

1. Complete the required fields on the **Virtual Service Image Recorder** Basics tab.

2. Click **Next**.

3. Enter an IP address and port, then click **Next**.
   The **Data Protocol selection** window opens.

4. Select the CICS Request Data Access data protocol on the request side.

5. Select the CICS Container Copybook data protocol for both the request and response sides, then click **Next**.
   The **Observed Programs** list contains the CICS containers:

Screenshot for CICS Copybook data protocol Observed Programs screen.

6. Click **Next**.

Screenshot of CICS Copybook data protocol How to Parse screen.

7. Enter the following parameters on the request side definitions:

   - **Copybook file definition folder**
     Specifies the folder for the copybook file definition.

   - **Payload to file definition map path**
     Specifies the path for the payload to file definition map.

   - **Encoding**
     Specifies the encoding information.

8. Select **Convert file definitions to XML** and click **Next**.

Screenshot for CICS Copybook data protocol Conversations screen.

9. Click **Next** to complete the service image.

10. To view the service images, select **View Service Image** and click **Finish**.
    The completed service image opens.

Screenshot for CICS Copybook data protocol completed VSI.

Copybook Data Protocol

The Copybook data protocol allows VSE to convert copybook payloads to and from XML at run time in a way that is transparent to the caller. The ability to display these payloads in XML allows other (standard) VSE mechanisms to provide value. You can use the Generic XML Payload Parser to identify and select request arguments and operations for matching and transaction organization in the service image. Using a standard XML structure allows for magic strings to be processed in the responses so that dynamic data is created in a more usable way.

Terms

To better understand the Copybook data protocol, it is important to understand the following terms:

- **Copybook**
  A hierarchical structure that is used to define the layout of data. The copybook identifies the field names, their lengths, and their relationships to each other.

- **Copybook File Definition**
  A file in plain text format containing a copybook.

- **Field**
  A single element in a copybook. A field encapsulates a name for, the length of, and the data type for a single logical piece of data in a record.

- **Payload**
  A collection of one or more records that VSE identifies in either a single request or a single response.

- **Payload Mapping File**
  Also known as a *Payload Copybook Mapping* or *Payload to File Definition Map*, a payload mapping file is an XML document describing how to identify the copybook or copybooks required to parse a payload. The payload mapping file provides the basic structure for the resulting XML.

- **Record**
  A collection of data with no inherent structural elements. One or more fields making up a single copybook defines a record. A record may not contain data for every field a copybook defines, but the relationship between records and copybooks is always one-to-one. A specific record is defined only by a single copybook, and a single copybook defines only one record.

Prerequisites for Creating a Copybook Virtual Service

To prepare for creating a copybook virtual service, complete the following tasks:

- Gather all of the copybooks that define the records in your payloads. These copybooks must be in a directory in your DevTest project. You can include whatever hierarchy you find helpful inside the directory you use. Typically, you would place the copybook file definition folder somewhere under the **Data** folder of the project.

- Create a payload mapping file.

- Prepare your client for recording, gather request/response pairs, capture a PCAP file of the traffic, or use some other means of gathering the transactions to use for your virtual service.

How to Use the Copybook Data Protocol

You can use the Copybook data protocol with any of the transport protocols that VSE supports. Commonly, the Copybook data protocol handler is used with a messaging protocol such as JMS or MQ or with the CICS transport protocol. A sample copybook application that uses HTTP is available in the demo server.

> **Note:** The Copybook Bundle page in the DevTest Portal makes it easier to create, import, and manage your copybooks and mapping files. For more information, see Create Copybook Bundles.

Selection

Keep the following in mind when selecting the Copybook data protocol:

- Consider using the data protocol handler on both the Request and Response sides. Although you can use the data protocol on only one side, it is not common. This documentation does not cover the single-sided usage in detail.

- The Copybook data protocol changes the Request (or Response) body into XML. The Copybook data protocol does not automatically add arguments to the Request for every field. To do useful things with the Copybook data protocol, you also must include another data protocol handler on the Request side. Most commonly, the Generic XML Payload Parser is the second data protocol handler. However, any data protocol handler that does useful things with XML payloads is acceptable.

Configuration

When you have captured your traffic (or imported from Request/Response pairs, PCAP, or raw traffic files), the **Copybook DPH configuration** window opens.

Enter the following parameters:

- **Copybook file definition folder**
  Designates the folder in your project where you store your copybook file definitions. This folder becomes the base path for relative paths in the Payload Mapping File.

- **Payload to file definition map path**
  Specifies the XML document that serves as the Payload Mapping File for this virtual service.

- **Encoding**
  Specifies a valid Java charset. If provided, this value is used to try to convert the bytes in the payload into text for use in the output XML.
  Default: UTF-8
  To configure the default charset, set **lisa.vse.default.charset** in local.properties.

- **Copybook cache TTL**
  VSE must reference a specific copybook at run time and potentially convert it to XML. The Copybook cache TTL parameter defines how long a cached version of the converted copybook can be kept in memory. When the specified interval expires, the converted copybook is removed from the cache. If the file is needed again, it is read again and reconverted.
  Values:

  - To define the timeout, set Copybook cache TTL to a positive number (in seconds).

- To disable caching, set Copybook cache TTL to 0 or a negative number. VSE reads and parses the files each time that they are needed.

- **Copybook parser column start**
  Copybooks frequently start each line with a line number. This parameter defines the column on which the parser starts when trying to parse a copybook file definition.
  Value: A zero-based inclusive index. However, you can think of it as a "normal" one-based exclusive index.
  Default: 6
  Example: If you set this value to 6, the parser skips the first six characters in a line and starts with the seventh character.

- **Copybook parser column end**
  Occasionally, copybooks contain other reference data at the end of each line. When that happens, the parser must know on which column to stop. If there is no "extra" data at the end of the lines in the file, you can set this number to something greater than the length of the longest line in the file. If this number is greater than the length of a line, the parser stops at the end of the line.
  Value: A zero-based exclusive index. However, you can think of it as a "normal" one-based inclusive index.
  Example: If you set this value to 72, the parser reads the 72nd character in the line and then it stops (without trying to read the 73rd).

- **Validate field lengths**
  Specifies whether to enforce the values set in the Copybook parser column start and Copybook parser column end values. This option is only used in VSE, on the Response side. During recording, the payload is converted to XML and then back to bytes to ensure that it can convert symmetrically. Also, during playback the XML responses are converted back to records/payloads before responding to the caller. In both of those operations, VSE can validate that the value in each field is exactly the length that is specified in the copybook. However, it is not always desirable to have this validation.
  Values:

  - **Selected:** The records in your data align exactly with the length defined in your copybook.

  - **Cleared:** The records in your data do not align exactly with the length defined in your copybook.

  Example: if your record does not contain any data for some fields, they are seen as 0 length, whereas the copybook defines that field for a length greater than 0. If you select this option in that case, VSE fails the validation and reports it as an error.

- **XML elements as request arguments**
  Specifies whether to verify that the requests and responses are XML strings.
  Values:

  - **Selected**: Identifies variables from the XML messages that the recorder uses. This verifies that the requests and responses are XML strings.

  - **Cleared**: Does not verify that the requests and responses are XML strings.

  Verifies that the requests and responses are XML strings. Selecting this option lets you identify variables out of the XML messages that the recorder uses. For more detailed information about identifying variables, see Generic XML Payload Parser.

The same editor is also available on the data protocol filters in the VSM (one on the Request side and one on the Response side). This editor lets you change the configuration after recording, as necessary.

Payload Mapping File

The *payload mapping file* is an XML document that describes how incoming payloads can be matched to corresponding copybooks. The payload mapping file is also known as *payload copybook mapping* and *payload to copybook file definition map*. This file also provides hints for how to structure the resulting XML to provide clarity to the user.

Sample

A sample mapping file can be found here. The sample file contains examples of various configurations and comments describing each of the attributes on each node. This sample file is useful as a reference while creating your copybook mapping files.

Structure

The following example shows the basic payload mapping file structure. For simplicity, this example contains no attributes of values.

```
<payloads>
  <payload>
    <key></key>
    <section>
      <copybook></copybook>
      ...
    </section>
    ...
  </payload>
  <payload>
    ...
  </payload>
</payloads>
```

- **<payloads>**
  The root XML node for the document. The node cannot be repeated.

- **<payload>**
  This element, in its entirety, fully describes a payload that matches it. When a payload is determined to match a <payload> element in this document, the <payload> element is used to describe the incoming payload.

- **<key>**
  (Optional) This option makes the XML document more readable. Everything that you specify on this element you can also specify as attributes on the <payload> element.

- **<section>**
  A logical grouping of copybooks that defines a portion of the payload. If there are multiple <section>s, they are processed in order with no repetition.

- **<copybook>**
  A single copybook that may or may not describe a record in the payload. A <section> can contain multiple <copybook> elements.

- **<payload>**
  A <payload> element can contain the following attributes:

  ```
  <payload name="TEST" type="request" matchType="all" key="reqKey" value="
  reqVal" keyStart="3" keyEnd="6" headerBytes="0" footerBytes="
  0" saveHeaderFooter="false" definesResponse="false">
      ...
  </payload>
  ```

Attribute	Description

	Required /Optional	
name	Required	Defines a unique name to identify the type of request described here. **Name** must be unique among the set of payloads of the same type (that is, request or response).
type	Required	Specifies the payload type. **Value:** One of the following: request response.
matchType	Optional	The matchType attribute applies differently to payload definitions of type "response." For example, responses do not contain arguments or operation names. If a payload definition of type "response" sets this attribute to **argument**, **attribute**, or **operation**, it will never match anything (unless the matching request sets the definesResponse attribute to **true**, which overrides this attribute). If a payload definition of type "response" sets this attribute to **all**, the argument, attribute, and operation phases of matching are skipped. **Value:** One of the following: **argument:** Try to match on only the specified argument. **attribute:** Try to match only the specified attribute. **metaData:** Try to match only the specified metaData. **operation:** Try to match on only the operation name. **payload:** Try to match on the body of the request. **all:** Try to match in the following order: argument, attribute, metaData, operation, then payload. **Default:** payload
key	Required	The behavior of this attribute depends as follows on the matchType: If matchType is **operation** or **payload**, the value of this attribute is used for matching. If matchType is **argument**, **attribute**, **metaData**, or **all**, the value of this attribute is assumed to be the key of an argument, attribute, or metaData entry. If the matchType value is **all**, the key value is NOT used for matching against the operationName or the payload body. The value attribute is used instead.
value	Optional /Required	The behavior of this attribute (and whether it is required) changes depends as follows on the matchType: If the matchType is **operation** or **payload**, the value of this attribute is ignored and can be excluded.

		If the matchType is **argument**, **attribute**, or **metaData**, this attribute is assumed to be the value of an argument, attribute, or metaData entry and is required for matching. If the matchType is **all**, this attribute is required and behaves as described previously during the argument, attribute, and metaData matching phases. During the operation and payload matching phases, however, the value of this attribute is used for matching (as opposed to using the value of the **key** attribute as is the case if the matchType is **operation** or **payload**).
keyStart	Optional	Designates the position where the search for the key should start in the payload (1-based index). The search is **inclusive** of this value. If you set keyStart to 1, the search starts with the first byte. If you do not set keyStart, the entire payload is searched and keyEnd is ignored. If you set keyStart to a number less than 1, the value is treated as if the value was 1. If you set keyStart to a number greater than the length of the payload, the entire payload is searched and keyEnd is ignored. If you set keyStart equal to keyEnd or greater than keyEnd, the entire payload is searched. If keyEnd minus keyStart is less than the length of the key, the entire payload is searched.
keyEnd	Optional	Designates the position where the search for the key should end in the payload (1-based index). The search is **exclusive** of this value. If you set keyEnd to 3, it searches only bytes 1 and 2. If you do not set keyEnd, the search starts at keyStart and ends at the end of the payload. If this value exceeds the length of the payload, the search stops at the end of the payload. If this value is equal to or less than keyStart, the entire payload is searched. If keyEnd minus keyStart is less than the length of the key, the entire payload is searched.
headerBytes	Optional	Designates the number of bytes to strip from the beginning of the payload. This defaults to 0 if you do not provide a value. If the attribute is present, the value must be a valid integer.
footerBytes	Optional	Designates the number of bytes to strip from the end of the payload. This defaults to 0 if you do not provide a value. If the attribute is present, the value must be a valid integer.
saveHeaderFooter	Optional	

		Specifies whether the header and footer bytes that were stripped are persisted in the XML version of the request as hex-encoded strings under the **rawHeader** and **rawFooter** tags. **Values:** true: Persists the stripped header and footer bytes. false: Does not persist the stripped header and footer bytes. **Default value:** false.
definesResponse	Optional	If **true** then the response for this request will look for a payload element with an identical name but of type **response**. This attribute is ignored when the type is **response**. **Default value:** false.

You can use the optional <key> element to replace the key-related attributes. The example <payload> element that is shown previously could, optionally, be written as follows for readability:

```
<payload name="TEST" type="request" headerBytes="0" footerBytes="
0" saveHeaderFooter="false" definesResponse="false">
       <key matchType="all" value="reqVal" keyStart="3" keyEnd="6">reqKey</key>
       ...
   </payload>
```

- **<section>**
  Example: The following example is a sample section element with all attributes on it and a description of the attributes.

```
...
   <section name="Body">
       ...
   </section>
   ...
```

Attribute	Required	Description
name	Required	Defines the name of a grouping XML element in the XML output version of the payload. One or more converted copybook elements will be present beneath it.

- **<copybook>**
  Example: The following example is a sample copybook element with all attributes on it and a description of the attributes.

```
...
<copybook key="cpk" order="1" max="1" name="TESTRECORD" length-field="SOME-ID"
>TESTIN.CPY.TXT</copybook>
...
```

Attribute	Required	Description
key	Required	Defines the unique string in the record that identifies the copybook. Technically, this attribute is optional. However, if a key is not provided, it means that that copybook is the only one that will ever get applied to the payload. It will be applied over and over until the payload runs out of bytes. If multiple copybook elements have no key, then the first one will always be used, unless the max attribute is specified.
order	Optional	

		Defines a hint as to the order in which the records are found in the payload. The numbers used are irrelevant, but "later" records in the payload should use a larger integer. Multiple copybooks can be tagged with the same order number, meaning that those records could be in any order. When a record has been found with a specific order number, subsequent searches only search for copybooks with that order number and greater. You can include copybooks in a group that will never match against the payload. They are just ignored. However, this affects performance because each copybook has to be checked. **Default:** 0
max	Optional	Defines the maximum times that a copybook can be applied to the payload. Blank values, 0, negative numbers, non-numbers, and non-existent values all mean "no limit".
name	Optional	Defines a value to override the record name (that is, the root level in the copybook). If you set this value, the generated node in the XML for this copybook uses this name instead of the record name from the copybook definition. If you do not set this value, the default is to look up the record name from the copybook definition and use that. If you set this value to the record name from the copybook definition, the only effect is on readability of this file.
length-field	Optional	Defines how to split the payload so that the next record search begins in the correct place. If this attribute is not present, the processor attempts to determine the length of the copybook from the definition. If, for some reason, it cannot figure out the length, the processor assumes that the rest of the payload applies to this copybook, and ends processing after applying this copybook. The processor ignores this field if it is not an unsigned Display numeric field.

Copybook Matching Logic

You can determine how the matching logic works based on the descriptions of the XML elements and arguments. However, to provide a clearer picture, this section explores matching fully.

When VSE receives a payload, matching occurs in the following phases:

- Payload
- Section
- Copybook
- Finalizing

Payload

Payload is the first phase of matching that happens when VSE receives a new payload. This phase determines which <payload> element from the payload mapping file corresponds to the payload received. VSE tries to match a <payload> element by processing the elements in the payload mapping file in order, from top to bottom. The first match wins.

> **Note:** If no payload elements match the payload, it is treated as an "unknown payload." The content is HEX encoded, and it is wrapped in a generic XML structure. If necessary, unknown payloads are automatically converted back to bytes during playback.

For the Request payloads:

1. If the type attribute of this element is "request," proceed. Otherwise, this element does not match.

2. If the matchType is argument, look for a request argument with a key that matches the key attribute from this element and a value that matches the value attribute from this element. If one is found, this payload element matches.

3. If the matchType is attribute, look for a request attribute with a key that matches the key attribute from this element and with a value that matches the value attribute from this element. If one is found, this payload element matches.

4. If the matchType is metaData, look for a request metaData entry with a key that matches the key attribute from this element and with a value that matches the value attribute from this element. If one is found, this payload element matches.

5. If the matchType is operation, verify whether the operation name for the request matches the key attribute from this element. If so, this payload element matches.

6. If the matchType is payload, search in the boundary that is specified by this element's keyStart and keyEnd attributes for the value in the key attribute. If the key is found in those bounds, the payload element matches.

7. If the matchType is all, Steps 2 through 6 are processed in order, stopping when a match is found. The only variation is that in Steps 5 and 6, the value attribute is used in place of the key attribute.

For the Response payloads (during recording):

1. If the previously seen request payload element set the definesResponse attribute to **true**, immediately return the payload element with the same name as the request element, but with a type of response. If no such element is found, there is no match.

2. If the previously seen request payload element did *not* set the definesResponse attribute to true:

    a. If the type attribute of this element is response, proceed. Otherwise, this element does not match.

    b. If the matchType is metaData, look for a response metaData entry with a key that matches the key attribute from this element and with a value that matches the value attribute from this element. If one is found, this payload element matches.

    c. If the matchType is payload, search in the boundary that is specified by this element's keyStart and keyEnd attributes for the value in the key attribute. If the key is found in those bounds, the payload element matches.

d. If the matchType is all, complete Steps b and c in order, stopping when a match is found. The only variation is that in Step c, the value attribute is used in place of the key attribute.

For the Response payloads (during playback):

During playback, no matching is done for responses. Instead, during recording, whatever match was determined is saved in the Metadata of the Response object. Then, during playback, when that Response object is returned, the processor picks up the value from the Response Metadata, finds the payload element with that name (and type="response") and uses it. If no such element exists, it is considered an unrecoverable error.

Section

After the payload element is identified, the processor reviews each section element in order, from top to bottom. No matching is done at this level. After the processor has fully processed a section (that is, when none of the copybooks in the section match), it proceeds to the next section and does not revisit previous sections.

Copybook

The final phase of matching is to determine which copybook in this section applies first, second, and so on, until all the records in the payload are processed. This phase is the most complex matching phase. The process is:

1. The processor reviews all of the copybook elements in the section and sorts them by the number that is specified in the order attributes.

2. The list of copybook elements with the lowest order number is checked first (in the order the file specifies them). The remaining payload is searched for the value in the key attribute for this copybook element and the index where it is found (if at all) is saved. The remaining payload is searched for the value in the key attribute for this copybook element and the index where it is found (if at all) is saved.

3. After all copybooks in the lowest order list have been checked, if any matched, the one found earliest in the payload is used.

4. If a copybook matched:

    a. If the matching copybook has matched this payload previously (an earlier record in the payload), the max attribute is checked to see if it can be applied again. If the max has been reached, this is not considered a match and the next matching copybook is used.

    b. If the max has not been reached, this copybook is applied to the payload and the number of bytes specified by the copybook is consumed. If there is a length-field attribute on the copybook, the value of that field in the record is used to determine how many bytes to consume and the processor returns to Step 2.

5. If a copybook did not match, the list of copybooks with the next lowest order number is checked using the same rules as Steps 2, 3, and 4. After the processor decides that no copybooks in an order number list match, it does not try to process that order number list (for this section) again for this payload.

6. After all lists are checked and no matches are found (or the payload runs out of bytes), the processor proceeds to the next section.

Finalizing

If bytes are left over in the payload after all sections for a payload have been processed, they are truncated.

CTG Copybook Data Protocol

The CTG Copybook data protocol splits the recorded request into its respective container chunks. The processor then sends each chunk to the Copybook data protocol and aggregates it with the corresponding XML. The resulting XML is the aggregated XML of the containers.

This example demonstrates the CTG Copybook data protocol usage.

Follow these steps:

1. Complete the required fields on the **Virtual Service Image Recorder** Basics tab.
2. Click **Next**.
3. Enter an IP address and port, then click **Next**.
   The **Data Protocol selection** window opens.
4. Select the CTG Copybook data protocol for both the request and response sides and click **Next**.
5. Click **Next**.

Screenshot of CICS Copybook data protocol How to Parse screen.

6. Enter the following parameters on the request-side definitions:
   - The folder for the copybook file definition.
   - The path for the payload to file definition map.

- The encoding information.

7. Select **Convert file definitions to XML** and click **Next**.

Screenshot for CICS Copybook data protocol Conversations screen.

8. Click **Next** to complete the service image.

9. To view the service images, select **View Service Image** and click **Finish**.
   The completed service image opens.

Data De-Identifier Data Protocol

In cases where the transport layer cannot "see" the data to de-identify, VSE provides a data protocol handler. For example, consider the use of gzip in SOAP over HTTP or messaging. If VSE is given a gzipped SOAP request (or response), then data de-identification cannot be done at this level. In that case, the recorder can be configured with a data protocol to perform the necessary gunzip operation. To handle de-identification after the SOAP body has been converted to plain text, add the data de-identification protocol.

VSE typically tries to de-identify data at the lowest possible level. The **De-identify** check box on the first window of the VSE recording wizard requests that the transport protocol try to de-identify as soon as possible. However, the payload for a request is sometimes opaque to the transport protocol. Consider HTTP messages in which the payload is compressed, such as with gzip or FastInfoSet. In this case, the transport protocol cannot apply the de-identification rules. This data de-identifier data protocol is built to address such situations.

Add the data de-identifier data protocol to the request side, response side, or both of a data protocol chain. Add the protocol after another protocol that converts the opaque payload into something that is readable. This data protocol is only intended for recording, not playback. As each request or response is presented to the protocol handler, all de-identification rules that are specified in **LISA_HOME\de-identify.xml** are applied.

This protocol requires no configuration information and so does not present a window in the recording wizard.

Delimited Text Data Protocol

The Delimited Text data protocol lets you parse delimited textual data into arguments and values. This protocol takes the body of the request or response, parses it, and then replaces the body with an XML representation of the parsed data. For example:

```
“name1=val1;name2=val2”
```

becomes:

```
<name1>val1</name1><name2>val2</name2>
```

Follow these steps:

1. Complete the Basics tab of the **Virtual Service Image Recorder.**
2. Select the **Delimited Text Data Protocol** on both the request and response sides.
3. After the recording completes, select the format for your delimited text request:
   - **Name/Value Pairs**
     Defines the delimiter between the name/value pairs and the delimiter between the name and value. For example, for the following data:

     ```
     "name1=val1,name2=val2,name3=val3"
     ```

     the delimiter between the pairs is a comma, and the delimiter between the name and value is an equal sign.
   - **List of Values**
     Defines the delimiter between the values.You can use two types of lists:
     - List of values separated by a textual delimiter.
       For example, for the following data:

       ```
       "val1,val2,val3,val4"
       ```

     the delimiter is a comma.The parameters are named positionally.
   - List of values separated by new line characters, such as a carriage return or line feed. The parameters are named positionally.
   - **Fixed Width**
     Defines (as a whole number) the width of the data field. The parameters are named positionally.
   - **RegEx Delimited**
     Defines a regular expression that locates the values. For example, for the following data:

     ```
     "xxxx123xxx456xx789"
     ```

     a RegEx of \d\d\d finds “123”, “456”, and “789” as values and discard the rest.The parameters are named positionally.
   - **Line Delimited**

4. Complete the following fields.

   - **Field Names Path**
     Defines the location of a field names document, which is a line-delimited document that specifies an ordered list of the names of fields. By default, the fields are named **value1**, **value2**, **value3**, and so on. To specify different names for those XML elements, use a field names document.

   - **Delimiter Type**
     Defines the delimiter type used to separate name/value pairs and lists of values. Text or hex delimiters are automatically selected based the specified delimiter.
     Values: Any alphanumeric characters, and the following:

     - **\r** (carriage return)
     - **\n** (newline)
     - **\t** (tab)

     You can also use hexadecimal notation to specify delimiters.

     > **Note:** You can only use delimiters that are valid XML 1.0. Specifying non-printable control characters makes the service image unusable.

   - **XML Elements as request arguments**
     Specifies whether to add parameters and associated values to requests as arguments.
     Values:

     - **Selected:** Automatically adds the parameters and associated values to the request as arguments.
     - **Cleared:** You must use the Generic XML Payload Parser (or a similar protocol) to select which values become arguments.

     > **Note:** This check box has no effect when the Delimited Text data protocol is used on the response side.

5. Click **Next**.
   The name/value pairs are represented in XML. Here, you can double-click a transaction to show the contents of that transaction.

6. Configure the delimiters for the response side in the same way you did for the request side.
   You can see the virtual service image and the payload that is converted to XML when the processing completes.

> **More Information:**
>
> - Generic XML Payload Parser Data Protocol

EDI X12 Data Protocol

The EDI X12 data protocol transforms ANSI X12 EDI documents into an XML representation in the body of the request. The DPH also creates a VSE request operation consisting of the EDI document type (for example, 835) combined with the specific document version (for example, 004010) and a series of VSE request arguments. These request arguments are the flattened representation of the XML body. They are formed by combining the XML elements, which are separated by _ for tags or @ for each tag attribute for the argument name and the element value for its value.

For example, if the following ANSI X12 EDI 850 document:

```
ISA*00* *00* *ZZ*0011223456 *ZZ*999999999 *990320*0157*U*00300*000000015*0*P*~$
GS*PO*0011223456*999999999*950120*0147*5*X*003040$
ST*850*000000001$
BEG*00*SA*95018017***950118$
N1*SE*UNIVERSAL WIDGETS$
N3*375 PLYMOUTH PARK*SUITE 205$
N4*IRVING*TX*75061$
N1*ST*JIT MANUFACTURING$
N3*BUILDING 3B*2001 ENTERPRISE PARK$
N4*JUAREZ*CH**MEX$
N1*AK*JIT MANUFACTURING$
N3*400 INDUSTRIAL PARKWAY$
N4*INDUSTRIAL AIRPORT*KS*66030$
N1*BT*JIT MANUFACTURING$
N2*ACCOUNTS PAYABLE DEPARTMENT$
N3*400 INDUSTRIAL PARKWAY$
N4*INDUSTRIAL AIRPORT*KS*66030$
PO1*001*4*EA*330*TE*IN*525*VN*X357-W2$
PID*F****HIGH PERFORMANCE WIDGET$
SCH*4*EA****002*950322$
CTT*1*1$
SE*20*000000001$
GE*1*5$
```

the resulting VSE request object contains:

Operation: 850-00340

Arguments: A subset of these name/value pairs follows:

```
interchange_sender_address <null>
interchange_sender_address@Id   0011223456
interchange_sender_address@Qual ZZ
interchange_receiver_address <null>
interchange_receiver_address@Id 999999999
interchange_receiver_address@Qual   ZZ
interchange_group_transaction_segment_element_1 00
...
```

Body: Structured as follows. The XML document has been formatted in this document for readability. The request body does not contain formatting elements such as line ends and indentation.

```
<?xml version="1.0" encoding="UTF-8"?>
<ediroot>
    <interchange Standard="ANSI X.12" Date="990320" Time="0157" StandardsId="
U" Version="00300"
        Control="000000015">
        <sender>
            <address Id="0011223456 " Qual="ZZ"/>
        </sender>
        <receiver>
            <address Id="999999999 " Qual="ZZ"/>
        </receiver>
        <group GroupType="PO" ApplSender="0011223456" ApplReceiver="
999999999" Date="950120"
            Time="0147" Control="5" StandardCode="X" StandardVersion="003040">
```

```
<transaction DocType="850" Name="Purchase Order" Control="000000001">
    <segment Id="BEG">
        <element Id="BEG01">00</element>
        <element Id="BEG02">SA</element>
        <element Id="BEG03">95018017</element>
        <element Id="BEG06">950118</element>
    </segment>
    <loop Id="N1">
        <segment Id="N1">
            <element Id="N101">SE</element>
            <element Id="N102">UNIVERSAL WIDGETS</element>
        </segment>
        <segment Id="N3">
            <element Id="N301">375 PLYMOUTH PARK</element>
            <element Id="N302">SUITE 205</element>
        </segment>
        <segment Id="N4">
            <element Id="N401">IRVING</element>
            <element Id="N402">TX</element>
            <element Id="N403">75061</element>
        </segment>
    </loop>
    <loop Id="N1">
        <segment Id="N1">
            <element Id="N101">ST</element>
            <element Id="N102">JIT MANUFACTURING</element>
        </segment>
        <segment Id="N3">
            <element Id="N301">BUILDING 3B</element>
            <element Id="N302">2001 ENTERPRISE PARK</element>
        </segment>
        <segment Id="N4">
            <element Id="N401">JUAREZ</element>
            <element Id="N402">CH</element>
            <element Id="N404">MEX</element>
        </segment>
    </loop>
    <loop Id="N1">
        <segment Id="N1">
            <element Id="N101">AK</element>
            <element Id="N102">JIT MANUFACTURING</element>
        </segment>
        <segment Id="N3">
            <element Id="N301">400 INDUSTRIAL PARKWAY</element>
        </segment>
        <segment Id="N4">
            <element Id="N401">INDUSTRIAL AIRPORT</element>
            <element Id="N402">KS</element>
            <element Id="N403">66030</element>
        </segment>
    </loop>
    <loop Id="N1">
        <segment Id="N1">
            <element Id="N101">BT</element>
            <element Id="N102">JIT MANUFACTURING</element>
        </segment>
        <segment Id="N2">
            <element Id="N201">ACCOUNTS PAYABLE DEPARTMENT</element>
        </segment>
        <segment Id="N3">
            <element Id="N301">400 INDUSTRIAL PARKWAY</element>
        </segment>
        <segment Id="N4">
            <element Id="N401">INDUSTRIAL AIRPORT</element>
            <element Id="N402">KS</element>
            <element Id="N403">66030</element>
        </segment>
    </loop>
    <loop Id="PO1">
        <segment Id="PO1">
            <element Id="PO101">001</element>
            <element Id="PO102">4</element>
            <element Id="PO103">EA</element>
            <element Id="PO104">330</element>
            <element Id="PO105">TE</element>
            <element Id="PO106">IN</element>
            <element Id="PO107">525</element>
```

```
                        <element Id="PO108">VN</element>
                        <element Id="PO109">X357-W2</element>
                    </segment>
                    <loop Id="PID">
                        <segment Id="PID">
                            <element Id="PID01">F</element>
                            <element Id="PID05">HIGH PERFORMANCE WIDGET</element>
                        </segment>
                    </loop>
                    <loop Id="SCH">
                        <segment Id="SCH">
                            <element Id="SCH01">4</element>
                            <element Id="SCH02">EA</element>
                            <element Id="SCH06">002</element>
                            <element Id="SCH07">950322</element>
                        </segment>
                    </loop>
                </loop>
                <loop Id="CTT">
                    <segment Id="CTT">
                        <element Id="CTT01">1</element>
                        <element Id="CTT02">1</element>
                    </segment>
                </loop>
            </transaction>
        </group>
    </interchange>
</ediroot>
```

Generic XML Payload Parser Data Protocol

The Generic XML Payload Parser data protocol identifies that the requests and responses are XML strings. Using this protocol, you can identify variables out of the XML messages that the recorder uses.

> **Note:** For the Delimited Text Data Protocol and Copybook Data Protocol, this functionality is enabled by selecting the **XML Elements as request arguments** check box on the data protocol configuration windows.

Identifying Conversations and Transactions

The quality of the recorded service image is directly dependent on how much information VSE has to identify the conversations and the individual transactions in them. Specifically, VSE needs help differentiating the transactions from each other:

- **As part of a conversation:** Identifying some unique ID representing a session.
- **As an individual operation:** Identifying some information specific to the semantics of the operation. For example, distinguishing an "order creation" operation from an "order listing" operation.

VSE internally knows many patterns for which to search. For example, for HTTP virtualization (if the server is a Java server) VSE sets a cookie that contains the value of a special variable that is named as sessionid. The sessionid variable uniquely identifies the session. VSE can use the variable to distinguish different sessions.

However, VSE sometimes needs further help, which can be supplied as follows:

- In the HTTP recording, VSE lets you identify tokens.

- For the JMS protocol, you could identify whether VSE uses a JMS correlation ID, custom JMS headers, or all JMS headers for this identification.

- VSE lets you specify a dynamic data protocol that lets you get meaningful information from the message itself. The following section describes this technique.

Using the Generic XML Payload Parser is a technique to help VSE inspect the body of the recorded messages (payload) and extract meaningful information from them to help identify the transactions. This technique can be the only way to get meaningful conversation information, especially for opaque protocols such as the WebSphere MQ native protocol.

Note: When using both the Generic XML Payload Parser and the Delimited Text data protocols, add the Delimited Text data protocol before the Generic XML Payload Parser. Otherwise, the request never appears as parsed in the recorder.

Follow these steps:

1. To use a dynamic data protocol, in the **Data Protocols** tab of the **Virtual Service Image Recorder**, select **Generic XML Payload Parser**, if the payload is a well-formed XML.

2. Go through the rest of the steps, up to the cleanup step.
   After the cleanup step, the following window opens:

Screenshot of the Generic XML data protocol Create XPaths screen.

   By default, no starter transactions are identified, so DevTest does not know which data to review to identify the conversations. However, the **Other Transactions** list shows the recorded transactions.

3. To see the XML payload in the **Content** area, click the first transaction.

XML data protocol Create XPaths screen, marked up.

The XML tab in the **Content** area shows the classic XML view. The **DOM Tree** tab shows the payload in the tree format.

> **Note:** This portion of the window is similar to the panel that appears when you create an XML XPath filter. As described in XML XPath Filter, you can use the **lisa.xml.xpath.computeXPath.alwaysUseLocalName** property to ignore the namespace.

4. To identify a specific node as a parameter for VSE, click the node on the **DOM Tree** tab. The **XPath Query** box displays the query that corresponds to the selected node. To evaluate the XPath query on the current payload, click .
   The result of the evaluation is shown under the heading **Filter Run Results**.

5. To add this XPath query to the parameter list, click **Add** .
   A menu displays where you can set an operation, an argument, a meta parameter, or the request body. For example, use the menu to select an embedded SOAP document to the request body so that the WS SOAP protocol can be used to parse the document.

Screenshot of the Generic XML data protocol Create XPaths screen, marked up

6. Click **Protocol Control Info.** DevTest remembered the XPath string that you identified and gave it an argument name. You can rename the argument. Also from this panel, use the **Save** and **Restore** buttons to save your list of XPaths to a file, or to load a list of XPaths from a file. By saving and loading, you can easily copy and paste from one XML Payload Parser to another.

Screenshot of the Generic XML data protocol Create XPaths screen, with Set Argument option selected.

Likewise, you can pull other variables from this transaction or others. It is not required for all transactions to have the specific argument. At the time of processing, if a specific argument is not present in the payload then it is ignored.
If you scroll to the right on the **Protocol Control Info** panel, you see an **NS** column in which you can add or edit namespace definitions. To locate and import an XML file from the file system and use the namespace definitions from that file, use the **Browse** button.

7. Double-click a transaction in the **XML** tab and you can see a dialog showing the content of the transaction.

8. When you are satisfied with your choice of variables, click **Next**.
   You are directed to the post-processing window.

JSON Data Protocol

The JSON data protocol is used to convert JSON data to an XML equivalent and to convert XML data to JSON format.

When the JSON data is converted to XML, you can use the other data protocols that support XML. For example, you can use the Generic XML Payload Parser data protocol.

The results of the data protocol handler are stored in the body of the Request, Response, or TransientResponse object that was given to it. Typically that object is stored in **lisa.vse.request** or **lisa.vse.response**.

For example, the data protocol converts the following JSON data:

```
[
    {"organizationType":"W",
     "parentOrganizationType":"S",
     "parentUnitNbr":777,
     "positionName":"S",
     "positionType":"S",
     "positionTypeId":56,
     "unitNbr":433,
     "l":
        [{"Roles":["P","P1","P2","S","C","AC","ACF","ACM"]}],
     "groupKey":"P",
     "groupName":"P"
    }
]
```

to the following XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <element class="object">
        <groupKey type="string">P</groupKey>
        <groupName type="string">P</groupName>
        <l class="array">
            <element class="object">
                <Roles class="array">
                    <element type="string">P</element>
                    <element type="string">P1</element>
                    <element type="string">P2</element>
                    <element type="string">S</element>
                    <element type="string">C</element>
                    <element type="string">AC</element>
                    <element type="string">ACF</element>
                    <element type="string">ACM</element>
                </Roles>
            </element>
        </l>
        <organizationType type="string">W</organizationType>
        <parentOrganizationType type="string">S</parentOrganizationType>
        <parentUnitNbr type="number">777</parentUnitNbr>
```

```
        <positionName type="string">S</positionName>
        <positionType type="string">S</positionType>
        <positionTypeId type="number">56</positionTypeId>
        <unitNbr type="number">433</unitNbr>
    </element>
</root>
```

If the JSON contains a one-element array, the data protocol converts it to an object with "element" as the key. For example, the data protocol converts the following JSON data:

```
[{"question1":"answer1","question2":"answer2"}]
```

into the following XML:

```
{"element":{"question1":"answer1","question2":"answer2"}}
```

The JSON data protocol handler reorders the key/value pairs in an object so they are alphabetical by key name.

In order to work correctly, the JSON data protocol handler requires the JSON data to be formatted correctly.

Note: Recording JSON works properly if the application type is set correctly to application/json, text/json, or text/javascript. If the wrong application type is set, recording fails.

Request Data Copier Data Protocol

The Request Data Copier data protocol lets you copy data from the current inbound request into the current testing context. This capability is useful to more easily examine request information when custom behavior is required before the VSE response lookup step in the VS model.

Screenshot of the Request Data Copier data protocol copy request screen.

Request Data Manager Data Protocol

The Request Data Manager data protocol lets you alter VSE requests during recording or playback.

On the **Data Protocols** window, select Request Data Manager for a data protocol. When your recording is complete, the following window opens:

Screenshot of the Request Data Manager data protocol Transactions screen

Fundamentally, this protocol lets you apply a list of actions against a request. You can add the following actions in the **ActionList** section of the window:

- **Copy**
  Copies a piece of data in the request to another part of the request.

- **Move**
  Moves a piece of data in the request to another part of the request.

- **Delete**
  Deletes (or clears) a piece of data from the request.

- **Keep**
  Keeps a piece of data in a request while deleting anything else in that group for the data value.

All actions can be applied to the request operation, any argument, attribute, or metadata entry, or the request body. For example, when virtualizing Java (which ends up with XML documents as arguments), you can move or copy the value of an argument into the request body, so other data protocols can process the argument.

Note: The **Keep** action is most meaningful for arguments, attributes, and metadata. If you specifically keep a value from one of these three groups, any other value in that group that is not referenced by any action in the list for the data protocol is deleted. If you keep one argument, other arguments are removed unless they were the target of a move or copy. This technique is a good way to remove meaningless arguments.

Each action can also be limited to apply only to requests whose operations match a specified regular expression.

In the recording wizard or the model editor containing a Request Data Manager data protocol, add a **Keep** or **Delete** action, or both. Select **argument/attribute/meta data** and specify a regular expression to match as the name. You must also change the cell that reads *named* to *matches*. When the data protocol is run, it keeps or deletes all items in the argument, attribute, or metadata list with a name that matches the pattern. Leaving the operation matching pattern for an action empty affects all requests.

From this list of transactions, double-click a transaction to open a dialog showing the content of the transaction.

Screenshot of the Request Data Manager data protocol Transactions details screen.

Use the Request Data Manager Data Protocol to set JMS and MQ Message Properties

After the recording is finished, use the **Request Data Manager** window to add the targeted JMS message properties to the request arguments. The JMS message properties can be found under the request Meta Data with a **msg.** prefix for standard JMS properties, such as correlation ID, and a **msg.props.** prefix for custom message properties. To copy a property to the request arguments, select **argument** from the drop-down list:

Screenshot of the Request Data Manager data protocol Create actions against VSE requests screen.

WebSphere MQ works the same way.

To set the stateful session key instead of an argument, select **session key** from the drop-down instead.

You can use a single Request Data Manager data protocol to set any number of arguments and the session key simultaneously.

REST Data Protocol

REST is a way of calling web services using the HTTP protocol.

For more information, see REST Data Protocol Handler.

Scriptable Data Protocol

The Scriptable data protocol is available for situations where you need a small amount of processing on the request, the response, or both. Sample scripts are written in BeanShell. Your script can be written in any scripting language that CA Application Test supports.

To specify a language other than BeanShell for the script, enter the language on the first line of the script.

Values:

- applescript (for OS X)
- beanshell

- freemarker
- groovy
- javaScript
- velocity

Format:

`%language%`

Default: beanshell

To use additional scripting languages, see "Enabling Additional Scripting Languages."

When you specify a Scriptable data protocol on the request side, the following window opens.

Screenshot of Scriptable Data Protocol request side.

Two scripts are available on the response side: one for recording and one for playback. You can use the **Response - Recording** script to convert the recorded response into a format that VSE can process. Then, after it is processed, you can use the **Response - Playback** script to return it to the format that the System Under Test expects.

You can add your own script to perform any actions you want on the request, the response, or both.

SWIFT Data Protocol

The SWIFT data protocol performs the following actions:

- Converts a SWIFT message to an XML representation
- Converts the XML representation of a SWIFT message back to its native format

For example, the data protocol converts the following SWIFT messages:

```
{1:F01BANKDEFMAXXX2039063581}{2:O1031609050901BANKDEFXAXXX89549829458949811609N}{3:
{108:00750532785315}}{4:
:16R:GENL
:20C::SEME//YOUR REFERENCE
:16S:GENL
:16R:SETDET
:22F::SETR//TRAD
:16R:SETPRTY
:97A::SAFE//YYYY
:16S:SETPRTY
:16S:SETDET
-}
```

to the following XML:

```
<message>
<block1>
   <applicationId>F</applicationId><serviceId>01</serviceId>
   <logicalTerminal>BANKDEFMAXXX</logicalTerminal>
   <sessionNumber>2039</sessionNumber>
   <sequenceNumber>063581</sequenceNumber>
</block1>
<block2 type=\"output\">
   <messageType>103</messageType>
   <senderInputTime>1609</senderInputTime>
```

```
   <MIRDate>050901</MIRDate>
   <MIRLogicalTerminal>BANKDEFXAXXX</MIRLogicalTerminal>
   <MIRSessionNumber>8954</MIRSessionNumber>
   <MIRSequenceNumber>982945</MIRSequenceNumber>
   <receiverOutputDate>894981</receiverOutputDate>
   <receiverOutputTime>1609</receiverOutputTime>
   <messagePriority>N</messagePriority>
</block2>
<block3>
   <BLOCK3_108>00750532785315</BLOCK3_108>
</block3>
<block4>
   <GENL>
      <GENL_20C>:SEME//YOUR REFERENCE</GENL_20C>
   </GENL>
   <SETDET>
      <SETDET_22F>:SETR//TRAD</SETDET_22F>
      <SETPRTY>
         <SETPRTY_97A>:SAFE//YYYY</SETPRTY_97A>
      </SETPRTY>
   </SETDET>
</block4>
</message>
```

If some fields in the SWIFT message include a date, the data protocol reformats the date in a format that DevTest can identify as a magic date candidate.

The following example shows the fields that the data protocol checks for dates:

```
:98A::SETT//19911130
:98C::TRAD//20140117125901
:98E::PREP//20091107093238,02/N0230
:32A:870902JPY3520000,
:30:640123
```

The data protocol converts these lines to the following XML:

```
<BLOCK4_98A>:SETT//1991-11-30</BLOCK4_98A>
<BLOCK4_98C>:TRAD//2014-01-17 12:59:01</BLOCK4_98C>
<BLOCK4_98E>:PREP//2009-11-07T09:32:38.020-0230</BLOCK4_98E>
<BLOCK4_32A>1987-09-02 JPY3520000,</BLOCK4_32A>
<BLOCK4_30>2064-01-23</BLOCK4_30>
```

The service image contains the dates it represents as magic dates.

SWIFT Conversations

To generate a session key for SWIFT transactions, the SWIFT data protocol extracts information from the message body. The protocol uses the following rules to extract the SWIFT message reference (mesg_ref) and deal reference (deal_ref), then combines them to form the session key.

1. If field 70E is found in the form **:SPRO///xxxREF/<mesg_ref>/<deal_ref>**, they are extracted from here.

2. If field 20C is found in the form **:RELA//<mesg_ref>**, and a second field 20C is found in the form **:TRRF//<deal_ref>**, they are extracted from here.

3. If field 26H is found in the form **<msg_ref>/<deal_ref>**, they are extracted from here.

The data protocol applies these rules in order. If one of them is satisfied, it creates the session key in the form **<message ref>/<deal ref>**. If none of the rules is satisfied, the data protocol treats the transaction as stateless and it does not create a session key.

Web Services Bridge Data Protocol

The Web Services Bridge data protocol exists specifically for the DevTest Travel example. This protocol is specific to the example and is not useful in a general case. Outside of the DevTest Travel example, you can ignore this protocol.

Web Services (SOAP) Data Protocol

The Web Services SOAP data protocol converts a SOAP document in XML form to a proper operation/arguments type of request.

For each request presented to the data protocol, it tries to parse the text form of the request body as an XML document. If the body is a valid XML document and the top-level element is **Envelope**, then the data protocol looks for both **Header** and **Body** child elements.

If the SOAP envelope contains a header element, the data protocol looks for a **ReplyTo** element in the header. Then, under the **ReplyTo** element, the data protocol checks for an **Address** element. If an address element is present, the value is set in the metadata list for the current VSE request under the name **lisa.vse.reply.to**.

If the SOAP envelope contains a body element, then the tag name for the first child of the body element becomes the operation name for the VSE request. If the operation cannot be determined, then nothing occurs for the current request.

If the operation element contains any XML attributes, these attributes are added to the attribute list for the current VSE request.

After it adds the XML attributes to the list, the data protocol examines the entire tree of XML elements under the operation element. All elements that do not have child elements are added to the VSE request as arguments. The name of each argument is constructed by using all parent element tags (up to the operation element) to ensure uniqueness. If the same name appears more than once, then a numeric suffix is added. The numeric suffix indicates an array style structure and ensures the uniqueness of the specific argument.

Finally, the original XML document (the request body) is copied to a new attribute in the VSE request named **recorded_raw_request**.

This data protocol handler requires no configuration information and so does not present a page in the recording wizard.

Web Services (SOAP Headers) Data Protocol

The SOAP Headers data protocol converts elements from the header of a SOAP message into arguments for requests in a virtual service image. This data protocol is compatible with any transport protocol that generates SOAP messages (typically HTTP/S).

To use the SOAP Headers data protocol, generate a VS Image either by recording or from request /response pairs. You typically use the HTTP/S transport protocol. Add the SOAP Headers data protocol as a Request Side data protocol. This data protocol does not require any extra configuration.

This data protocol can be used with the SOAP data protocol to process both SOAP headers and the SOAP body.

Arguments are named based on the structure of the XML.

Example

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">
<wsse:UsernameToken>
<wsse:Username>username</wsse:Username>
<wsse:Password>password</wsse:Password>
</wsse:UsernameToken>
</wsse:Security>
<n1:ServiceControl xmlns:n1="http://localhost:8080/examples.xsd">
<n1:VersionID>2.0</n1:VersionID>
<n1:Asynchronous>
<n1:ReplyRequiredIndicator>false</n1:ReplyRequiredIndicator>
<n1:PassThroughData>
<n1:Key>InteractionID</n1:Key>
<n1:Value>444831</n1:Value>
</n1:PassThroughData>
</n1:Asynchronous>
</n1:ServiceControl>
</soapenv:Header>
...
```

This example would be parsed into elements that are named as follows:

- Security_UsernameToken_Username
- Security_UsernameToken_Password
- ServiceControl_VersionID
- ServiceControl_Asynchronous_ReplyRequiredIndicator
- ServiceControl_Asynchronous_PassThroughData_Key
- ServiceControl_Asynchronous_PassThroughData_Value

Duplicate elements are named with _1, _2, and so on, appended to the name.

WS - Security Request Data Protocol

The WS-Security Request data protocol supports SOAP messages that include WS-Security headers. This data protocol can strip any security from the SOAP Request before sending it along the Virtualize framework. The WS-Security Request data protocol then applies security to outgoing SOAP responses.

When recording a web service with WS-Security headers, add a WS-Security Request (Request Side) data protocol (typically before a Web Service SOAP data protocol) and a WS-Security Response (Response Side) data protocol.

Before you record, you are presented with a set of configuration panels:

- One for the WS-Security Request data protocol
- Two for the WS-Security Response data protocol

Request Data Protocol

For the Request data protocol, configure the handler to process a Request message that the client sends. Fill in the Receive actions that are used to decode and validate the headers. This configuration is used both for recording and playback.

Options available for **Receive (Response) messages** are:

- **Decryption**
- **Signature Verification**
- **SAML Verifier**
- **Username Token Verifier**
- **Timestamp Receipt**
- **Signature Confirmation**

Options available for **Send (Request) messages** are:

- **Timestamp**
- **Username Token**
- **SAML Token**
- **Signature Token**
- **Encryption**

To use encryption, enter the following information:

- **Keystore file**
  Specifies the keystore file to use for encryption.
- **Keystore Type**
  Specifies the keystore type.
  Values:
- Java Key Store
  - Personal Information Exchange (PVKS #12)
- **Keystore password**
  Defines the password associated with the specified keystore file.
- **Keystore alias**
  Designates an alias for a public key.
- **Alias Password**
  Defines an alias password for PKCS#12 files. Leave this value blank or enter the same value you specified for **Keystore Password**.

The **WS-I BSP Compliant** check box indicates whether to verify that you comply with the WS-I Basic Security Profile (including using InclusiveNamespaces and CanonicalizationMethod in SignedInfo).

To validate your keystore information, click **Verify**.

WS-Security Request data protocol receive options for processing Request messages during recording and playback

Response Data Protocol

For the Response data protocol, configure the handler to process Response messages returned from the live service during recording and Response messages that are returned from the VSM. During the recording phase, you must process the Response message as the client would.

Select the **Add Timestamp** check box.

- **Time-To-Live (sec)**
  Defines the lifetime of the message in seconds. To avoid including an Expires element, enter **0** .

- **Use Millisecond Precision in Timestamp**
  Specifies whether to output the timestamp in milliseconds.

> **Note:** Some web services (for example .NET 1.x/2.0 with WSE 2.0) do not comply with standard timestamp date formatting, and do not allow milliseconds. For these web services, clear the **Use Millisecond Precision in Timestamp** check box.

During playback, you must process the message as the server would send it. The SOAP message from the VSM has no Security headers and this configuration applies the Security header.

After the recording completes, a virtual service model is created. In that model, a data protocol filter is attached to the HTTP/S Listen step for the WS-Security Request data protocol.

Update any security configuration information for playback. For example, if your WS-Security settings change on the service, you can update them here instead of rerecording the virtual service.

Virtualize Data Protocol Execution Shell filter parameters

In the VSM, a filter is also added for the WS-Security Response data protocol onto the HTTP/S Response step.

Any security configuration information for playback can be updated for the response message.

To save your security settings to a file, or to load a saved file containing security settings, use **Load** and **Save** .

XML Data Protocol

The XML data protocol converts an XML document into a proper operation/arguments type of request.

This data protocol handler works exactly like the Web Services (SOAP) data protocol. The data protocol requires no configuration information and so does not present a window in the recording wizard.

The XML data protocol is commonly used for high-speed processing of XML documents. It extracts every tag in the document into a uniquely named argument. This can make matching difficult to debug (and slow) so it is common to see the data protocol that is deployed with a Request Data Manager data protocol, which is defined to "keep" only certain useful or significant values from the request. The XML Data Parser does not extract arguments that are based on the attributes of the tags it encounters.

Consider using the Generic XML Parser, which uses XPATH expressions to extract individual elements. Using XPATH has a significant cost, but the parser is more flexible, so it is worth experimenting with both.

Editing Service Images

The **Virtual Service Image Recorder** generates service images. Service images pretend to be what you recorded (a manipulated or altered version of your recorded raw traffic).

If you have service images from releases of VSE earlier than LISA 6.0, export those service images. Exporting moves them from the database of earlier versions to the file system where they are stored in the current release. For more information, see "Legacy Service Images".

Opening a service image allows you to change the image in the **Service Image Editor**.

Follow these steps:

1. Right-click the service image in the project panel and select **Open**.
   The **Service Image Editor** opens.

2. Review the selected service image and make any changes.

> **Note:** You can also access the editor from the **Response Selection** step in the VSM by clicking **Open** next to the name of the service image.

Legacy Service Images

Beginning with LISA 6.0, the product does not store service images in a database. If you must use LISA 5.0 service images in LISA 6.0 and later:

1. Export them using LISA 5.0.

2. Import them using the **Import** item on the project tree shortcut menu in the later LISA version.

> **Note:** A service image exported from LISA 5.0 has an extension of **.xml**. To make this exported service image into a valid LISA 6.0 or later service image, change the extension to **.vsi**.

Service Image Tab

The following fields are available on the **Service Image** tab of the **Service Image Editor**:

- **Image name**
  Identifies the name of the current service image.

- **Created on**
  Identifies the date and time when the service image was created.

- **Last modified**
  Identifies the date and time when the service image was last modified.

- **Notes**
  Displays documentation about the service image.

- **Approximate memory usage**
  Identifies the estimated memory that the service image requires.

- **Response for Unknown Conversational Request**
  Displays the body, metadata, and think time for a response to an unknown conversational request in playback.

- **Response for Unknown Stateless Request**
  Displays the body, metadata, and think time for a response to an unknown stateless request in playback.

Edit Responses for Unknown Requests

Use the arrow to zoom the panels to their largest size. To return to the original size, use the same icon.

Complete the following **Response** panel fields as appropriate:

- **Body**
  Contains the response to be returned for unknown stateless requests during playback.

- **Response Meta Data**
  Displays a toolbar from which you can add, move, or delete key/value pairs as necessary.

- **Think time spec**
  Defines the amount of think time that is required, in milliseconds. The *think time* is the time that is taken before sending the response to a request.
  Values: A number or number range, in milliseconds.
  If you enter a range, the application selects the think time randomly selected from that range.
  You can specify time measurements by adding a suffix to the numbers. Case does not matter.
  The valid suffixes include:

  - **t:** milliseconds
  - **s:** seconds
  - **m:** minutes
  - **h:** hours

  Default: 0
  Examples:

  - 100 specifies a think time of 100 milliseconds.
  - 100s specifies a think time of 100 seconds.
  - 100-1000 specifies a random think time from 100 to 1000 milliseconds.
  - 10t-5s indicates a random think time between 10 milliseconds and 5 seconds.

> **Note:** A step subtracts its own processing time from the think time to have consistent pacing of test executions.

Customize the Response Editor

To customize the response editor, click at the lower right corner of the panel.

Response editor body tab

Options on the response editor menu include:

- **Set reference protocol to**
  No Specific Protocol or **JDBC (Driver based)**.

The title bar on the **Body** tab of the Response Editor shows what category of response has been returned. In the previous graphic, it indicates a Text response. Other categories of response payloads are:

- XML
- JSON
- String
- Large String
- Huge String
- Graphic Image
- Raw Bytes

To use a different editor or to change the type of payload, click the gear icon on the upper right corner of the panel. The menu builds dynamically to only show other potentially valid editors, based on the editor you currently have selected.

When the text body editor is set to **Text**, right-click the Response Body XML to format the response text.

Screenshot of text body editor set to text

If the response payload is detected to be EDI data, it is converted to XML on the **XML** tab.

Response editor Body tab, Raw EDI subtab

Transactions Tab

You can access the **Transactions** tab from the **Service Image Editor**. This tab gives information about both stateless and stateful (conversations) transactions, with slightly different components for each. This section describes viewing the general components of the **Transactions** tab that are the same for both types of transactions.

To select between viewing conversations or stateless transactions, select either **Conversation by number** or **Stateless Transactions** from the drop-down list.

More Information:

- Transactions Tab for Stateless Transactions
- Transactions Tab for Conversations
- Conversation Editor

Transactions Tab for Stateless Transactions

When viewing a stateless transaction on the Transactions tab, you can see the components that are shown in the following graphic.

Transactions Tab for Stateless Transactions, marked up.

- **A:** To view and edit stateless transactions, use the **Stateless Transactions** list. To add, move, or delete stateless transactions, use the toolbar at the bottom of the pane.
- **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. To add, move, or delete stateless transactions, use the toolbar at the bottom of the pane.

- **C:** To view and edit transaction requests and response data for either Specific or Meta transactions, which are selected from the **Transactions** area, use the **Transaction Basics** area. You can select a match style for the specific transaction here.

- **D:** The **Request Data** panel shows the stateless requests.

- **E:** The **Response** panel shows the response to the stateless requests.

> **Note:** If you add or change several transactions, click . The magic string and date variables are created for you. Existing magic strings and variables are not modified.

Transaction Basics Editor

To view and edit transaction data for specific or meta transactions, use the **Transaction Basics** editor. Select a specific transaction or **META** from the **Transactions** list.

The **Transaction Basics** editor lets you specify the following information:

- **Match Style**
  Values:
  - **Signature**
  - **Operation**
- **Operation**
  Defines the operation that is selected.
- **Allow duplicate specific transactions**
  Specifies whether to allow DevTest to respond more than once to the same call, choosing a different response.
  Values:
  - **Selected**: DevTest can respond multiple times to the same call, with different responses. Round-robin matching only happens if this check box is selected. For round-robin to work, each response must be under a specific request. With multiple responses for one request, round-robin matching does not occur.
  - **Cleared**: DevTest can respond only once to a specific call.

Request Data Editor

The **Request Data** editor allows you to update the data that is associated with a request.

Request Data Editor

Arguments

VSE uses the operation name and arguments to look up a matching response for an incoming transaction.

Adding and Removing Arguments

The META transaction is a template for the specific transactions. For more information, see Logical Transactions. Therefore, arguments can be added to and removed from the META transaction, and these changes apply to all specific transactions in that logical group. Arguments cannot be directly added to or removed from specific transactions.

Modifying Arguments

- **Name**
  Defines the name of the argument, which is parsed from the request. In most cases, this field should not be modified. It can only be changed at the META transaction level, and these changes propagate to the specific transactions.

- **Name in Session**
  Defines a value that the application automatically generates when magic strings are identified. The value can be referenced in the current (or later) responses using the {{ }} notation. If this field is not empty, the incoming value is stored in the session using the specified name. This value should not typically be modified.

- **Comparison Operator**
  Defines an operator to use in the matching logic. By default, for a specific transaction, all arguments are expected to match exactly. To change this and create more flexible matching logic, modify the comparison operator. See Argument Match Operators for the definitions of the comparison operators.

- **Magic String**
  Specifies whether to include the specified value as a candidate for magic strings.
  Values:

  - **Selected**: If this check box is selected and you click **Regenerate magic strings**, the specified value is a candidate for magic strings. Selecting this check box does not override the rules for identifying magic strings. You cannot use it to force something to be a magic string; it is still subject to the VSE magic string properties in the **lisa.properties** file.

  - **Cleared**: If this check box is not checked and you click **Regenerate magic strings**, the specified value is excluded as a candidate for magic strings. If something was used as a magic string and you did not want the magic string substitution to happen, clear this check box and select **Regenerate magic strings**.

- **Date Pattern**
  Defines the pattern by which the application interprets incoming and specified values as dates. This value is automatically generated and should not typically be modified.
  This pattern is a Java date and time pattern. It is a strict interpretation, so the values must match the pattern precisely. If both (or either) values cannot be parsed as dates, then the argument is deemed not to have matched. If both values can be parsed as dates, they can then be compared as dates using the following operators:

  - =

  - !=

  - <

- <=
- >
- >=

You can still use "Anything," "Regular Expression," and "Property Expression." If you use "Regular Expression," the incoming value is treated as a string.

- **Case Sensitive**
  Specifies whether matching is case-sensitive.
  Values:
  - **Selected**: All matching is case-sensitive.
  - **Cleared:** The comparison operators ignore case.

  Default: Selected.

- **Is Numeric**
  Specifies whether argument values are processed as strings or numbers.
  Values:
  - **Selected**: The application processes argument values as numbers.
  - **Cleared**: The application processes argument values as strings. That means "10000" is considered less than "9" because "1" comes alphabetically before "9".

  Default: Cleared.

Mass Change

To perform a mass change of request arguments, click **Mass Change** . The **Change Request Arguments** dialog appears.

Change Request Arguments dialog

To specify mass changes, complete the fields on the **Change Request Arguments** dialog as appropriate, and click **Update**.

Attributes

To add, edit, move, and delete key/value pairs, use the **Attributes** tab.

Meta Data

To add, edit, move, and delete Meta data key/value pairs, use the **Meta Data** tab.

Match Script Editor

To insert a sample match script for your information, right-click the **Match Script** panel. You can also switch the match script on or off by selecting or clearing the **Do Not Use the Script** check box.

To designate the scripting language, use the language drop-down on the lower right of the pane.

- **Language**
  Designates the scripting language to use.
  Values:
  - Applescript (for OS X)
  - Beanshell
  - Freemarker
  - Groovy
  - JavaScript
  - Velocity

Default: Beanshell

To hide or display line numbers, the editor toolbar, and the editor status bar, right-click on the left side of the **Match Script** panel, then select the appropriate options from the short-cut menu. The following graphic shows all options that are displayed.

Match Script Editor Stateless

A match script defines how VSE decides whether a specific transaction matches the incoming one. To receive a match that is based on the specific condition, write BeanShell scripts performing appropriate actions.

For example:

```
/* always match name=joe */
ParameterList args = incomingRequest.getArguments();
if ("joe".equals(args.get("name"))) return true else return
defaultMatcher.matches();
```

You do not need to specify a match tolerance level or match operator for the match script to work. The match is found based on the condition in the match script.

By default (with no match script) an inbound request is matched against a service image request by comparing operations, arguments, or both to come to a true/false "Do they match?" answer. A match script simply replaces this logic with whatever logic makes sense and must still come to the true/false "Do they match?" answer.

The script can use the default matching logic. Inside the script, use the expression, "defaultMatcher.matches()". This expression returns a true or false using the VSE default matching logic.

The match script is similar to a scripted assertion. Basically, it is a regular BeanShell script but with the following additional variables preloaded for you (and the usual properties and **testExec** variable):

- **com.itko.lisa.vse.stateful.model.Request sourceRequest** (the recorded request)

- **com.itko.lisa.vse.stateful.model.Request incomingRequest** (the live request coming in)
- **com.itko.lisa.vse.RequestMatcher defaultMatcher** (you can default to this variable)

Return a Boolean value from the script; true means a match was found.

If there is an error evaluating the script, VSE deliberately ignores the error and defaults to the regular matching logic. If you do not think your script is being run, review the VSE log file.

A good way to add logging and tracing into your match scripts is to embed calls to the VSE matching logger. The VSE matching logger produces the messages in the **vse_xxx.log** file, where *xxx* is the service image name. For example:

```
import com.itko.lisa.VSE;
VSE.info(testExec, "short msg", "a longer message");
VSE.debug(testExec, "", "I got here\!\!");
VSE.error(testExec, "Error\!", "Some unexpected condition");
return defaultMatcher.matches();
```

If you log messages at *INFO*, later when the production settings are applied to the **logging.properties** file, the log level is WARN and your messages appear as a DevTest test event (a "Log Message" event).

Tips from **logging.properties**:

- To simplify debugging, keep a separate log for VSE transaction match/no-match events.
- Change INFO to WARN or comment out the following line for production systems:

  ```
  log4j.logger.VSE=INFO, VSEAPP
  ```

- The INFO value typically reports every failure to match.

Match Script Editor Toolbar

Match Script editor toolbar

The Match Script editor toolbar lets you perform the following functions:

Returns you to the last edit that was made

Finds the next occurrence of the selected text

Finds previous occurrence

Finds next occurrence

Toggles the highlight search

Shifts the current line to the left four spaces

Shifts the current line to the right four spaces

Inserts comments slashes (//) at the cursor position

Removes the comments slashes (//)

Response Data Editor

To view and edit the response information, use the Response Data editor.

- To edit the expected response for a transaction, use the **Response Body** area.
- To add, order, delete, and navigate through responses, use the toolbar as described in the Match Script Editor Toolbar.
- To enlarge the **Response Data Editor** panel, click .
- To customize the Response Data editor, click , as described in Customize the Response Editor.
- To inspect the Validation Results, view the XML schema source, and see the error log, use the buttons at the bottom of the panel.

Response Data Editor stateless

Edit the **Think Time Spec** field as necessary.

Response Data Editor Meta Data tab

To add, edit, move, and delete key/value pairs, use the **Meta Data** tab.

Tip: You can only add properties if the response is text. You cannot add properties to a binary response.

Transactions Tab for Conversations

A stateful transaction (a transaction with conversations) has the following components.

Service Image Editor - Transactions tab

- **A:** The **Conversations** list shows all the conversations in the service image. To view and edit a conversation, select it. A conversation consists of one or more logical transactions.
- **B:** The Conversation Tree editor displays the logical conversation that is selected in the **Conversations** list. The conversation is displayed in either a graph node tree view or a standard tree view.
- **C:** To view and edit specific transactions or the Meta data for transactions in a selected logical transaction, use the **Transactions** list. To add, move, or delete stateless transactions, use the toolbar at the bottom of the pane.

- **D:** To view and edit transaction requests and response data for the Specific or Meta transactions, selected in the **Transactions** area, use the Transaction Basics area. The fields are dependent on the selected transport and data protocols. For more information about the **Transaction Basics** area, see the Service Image Editor Transactions Tab for Stateless Transactions.

- **E:** To enter the data for conversational requests during playback, use the Transaction Request Data pane.

- **F:** To enter and edit a script to return actions that are based on specified matching conditions, use the Match Script Editor.

- **G:** To view and edit the response content, think time, and key/value pairs for the specific or Meta transaction, use the Transaction Response Data pane.

Toggle Display Pane

Service Image Editor Transactions tab

You can see details about a conversation by clicking **Toggle Display** on the **Transactions** tab toolbar. From this pane, you can display and edit the following values:

- **Type**
  Specifies the transaction type.
  Values:

  - INSTANCE

  - TOKEN

- **Token Pattern**
  Required for token-based conversations. Clicking the question icon provides examples of string generator patterns.

- **Pattern Example**
  Displays an example of the specified Token Pattern.

- **Starter Transaction**
  Defines the starter transaction for the conversation.

> **Note:** If you add or change several transactions, return to the **Basic Info** tab and click **Regenerate Magic Strings and Data Variables**. The magic string and date variables are created for you. Existing magic strings and variables are not modified.

Conversation Editor

To view and edit recorded transactions or create transactions manually, use the **Conversation** editor. 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 the Graph and Tree views, you can perform the following actions from the editor toolbar, shortcut menus, or the view itself.

- Conversation Editor Toolbar
- Conversation Editor Graph View
  - Node Display Status
- Conversation Editor Tree View
  - Search and Replace Action
  - Test from Here Action
  - Highlight Navigation
  - Viewing Close Navigation
  - Viewing Wide Navigation
  - Viewing Loose Navigation
  - Restructure Conversation

Conversation Editor Toolbar

The **Conversation Tree editor** toolbar varies slightly, depending on the display.

Graph View Tools

Tree View Tools

Conversation Editor toolbar - Tree View

> **Note:** If a tool appears dimmed, it cannot be used with the selected transaction.

The **Conversation Editor** toolbar contains the following commands:

Tool	Icon	Description
Toggle Display	Toggle display icon	Toggles the display of the panel for managing a list of conversations. You can specify name, type, token pattern, pattern example, or starter transaction.
Create New Transaction	Create new transaction icon	Adds a new transaction.
Delete Selected Transaction	Delete icon	Deletes a selected transaction.
Up Arrow	Move up icon	Tree View: moves the selected node earlier in the sibling list.
Down Arrow	Move down icon	Tree View: moves the selected node later in the sibling list.
Right Arrow	Move right icon	Graph View: moves the selected node later in the sibling list.
Left Arrow	Move left icon	Graph View: moves the selected node earlier in the sibling list.
Regenerate	Refresh icon	Regenerates magic strings and date variables for all transactions.
View Navigation		Opens a menu to select menu navigation highlights for stateful transactions (conversations). You can select the following: ▪ No navigation highlight ▪ Highlight on transaction's tolerance

	View navigation icon	▪ Highlight as if close ▪ Highlight as if wide ▪ Highlight as if loose
Toggle	Toggle icon	Toggles the display of transaction IDs for debugging.
Match	Match icon	Pulls a match description from the clipboard and highlights the relevant information in the service image.
Zoom	Zoom icon	Opens a zoom menu.
Display	Display icon	Toggles the conversation display to a tree display.
Display	Display icon	Toggles the conversation display to a graph display.
Zoom Panel	Zoom panel icon	Zooms this panel to its largest size.

Conversation Editor Graph View

The Conversation Editor displays nodes according to status. It has the following components:

Conversation Editor graph view components labelled.

- **A:** Toolbar. For more information, see Conversation Editor Toolbar.
- **B:** A standard node.
- **C:** A selected node.
- **D:** A collapsed node. Child nodes are not displayed. To view child nodes, expand the node.
- **E:** A disabled node. This node and related child nodes (not displayed) are ignored during run time.
- **F:** The shortcut menu.

Node Display Status

You can display nodes in the following styles:

- Full size
- Compact (default)
- Condensed

In each style, a row of black dots at the bottom shows how many specific transactions belong to the node. In the condensed style, hollow dots indicate the number of arguments to the request. In the following graphic, the first node is displayed full size, the second is compact, and the third is condensed.

Conversation Editor Tree View

The tree view displays the same information as the graph view more compactly. It displays nodes according to status, and has the following components:

Conversation Editor tree marked up

- **A:** Toolbar. For more information, see the Conversation Editor Toolbar.
- **B:** A standard node.
- **C:** A selected node.
- **D:** A disabled node. This node and related child nodes (not displayed) are ignored during run time.

- **E:** The shortcut menu.

Search and Replace Action

From the right-click menu, you can select the **Search and Replace** action to change values in the conversation or the transaction.

Conversation Editor - Search and Replace dialog

Scope lets you specify whether the search and replace extends to:

This conversation (the default)

- This transaction
- This transaction and children
- The entire service image.

You can also use the check boxes to specify which pieces of the transactions to include.

Test from Here Action

To test for an expected response from a selected transaction in both graph and tree views, use the Conversation Editor.

Follow these steps:

1. In the Conversation Tree editor, right-click a transaction.
2. Select **Test from here** from the shortcut menu.
3. The **Create Test Request** window opens.

Create Test Request window

4. Enter the unique operation name and add argument key/value pairs, as appropriate.

5. Click **Test**.
   As the following graphic shows, the result displays in blue with the path in yellow.

Test From Here results, tree view

> **Note:** If the test is not successful, the application displays the following error:
>
> "No transaction matching the request follows the selected one in this conversation.

6. Click **OK** to continue.

7. Click **Close**.

Highlight Navigation

To view navigation possibilities in a conversation that is based on the selected navigation tolerance, use the **Conversation Tree**.

Select a transaction and click **View Navigation**.

The default menu selection is **No Navigation Highlight**, which means no transactions are highlighted.

If you select a transaction and click the **Highlight on transaction's tolerance** option, the transaction and its associated transactions are highlighted according to the navigation tolerance defined for the selected transaction. Options are:

- Close
- Wide
- Loose

> **Note:** If you use the drop-down list on the top right of the editor to change the navigation tolerance for the current transaction, that selection overrides the menu. To return the highlighting to the "correct" state in this case, reselect that navigation highlighting option.

Viewing Close Navigation

Select a transaction and click **View Navigation**. The transactions searched in the selected transaction subtree are highlighted in red.

Viewing Close Navigation, graph view and tree view

Viewing Wide Navigation

Select a transaction and click **View Navigation**. The transactions that are searched in the selected transaction subtree and sibling subtrees are highlighted in blue.

Viewing Wide Navigation, graph view and tree view

Viewing Loose Navigation

Select a transaction and click **View Navigation**. The transactions searched in the selected transaction subtree, sibling subtrees, and parent are highlighted in green. A full conversation restart is also possible.

Viewing Loose Navigation, graph view and tree view

Restructure Conversation

In some cases, you want to restructure a conversation by dragging and dropping a transaction and its subtree, if it has one. You restructure conversations in both the graph and tree views.

Follow these steps:

1. Drag a transaction to the transaction that you want to be the new parent transaction.

Restructuring a conversation by dragging transactions

As you drag the transaction, it displays the symbol for "not possible."
When it is possible to drop the transaction, the "not possible" symbol disappears.

Restructuring a conversation by dropping a transaction

2. Drop the transaction on the appropriate parent transaction.
   The transaction and any transactions in its subtree move below the new parent.

Service Images for Messaging Transport Protocols

This page describes the specific characteristics of service images that are created using the JMS transport protocol, the IBM MQ Native transport protocol, and the RabbitMQ transport protocol.

By default, the operation name for the transaction is set to the request channel name that is defined in the request step in the virtual service model.

The queue and connection data in the metadata is for informational purposes only. The data is not used at playback time.

The following items apply to the request metadata:

- The metadata properties that are not grouped with a prefix contain the queue and connection information for the proxy request queue.
- The **channel.name** metadata property must match the request channel name defined in the request and live invocation steps in the virtual service model. This property is the only required metadata property on the request side. You can change the virtual service model's request queues as needed, as long as the request channel name remains the same.
- The metadata properties that start with **msg** are the properties from the original request message.
- The metadata properties that start with **liveRequest** contain the queue and connection properties for the live request queue.

The following items apply to the response metadata:

- The metadata properties that are not grouped with a prefix contain the queue and connection information for the proxy response queue.
- The **channel.name** metadata property must match a response channel name defined in the respond and live invocation steps in the virtual service model. The value specifies the response channel name to be used to send the response to the client. If there is more than one response, the responses can specify different channel names.
- The metadata properties that start with **msg** are used to reconstruct the response message and its properties. For service images that are created using the JMS transport protocol, the only required property is **msg.type**. The **msg.type** property indicates the type of message to send.
- The metadata properties that start with **liveResponse** contain the queue and connection properties for the live response queue.

Editing a VSM

A Virtual Service Image recording creates a Virtual Service Model (VSM) with six steps or eight steps, depending on the option chosen (**More Flexible** or **More Efficient**). Sometimes you would like to edit the VSM by editing generated steps or adding more steps.

You can skip this section unless you must edit a VSM or you must create a VSM without a recorder.

A VSM is a specialized test case and therefore editing or creating a VSM is similar to editing or creating a test case. Many types of steps can be added to a VSM. You can add a step that does not appear in this menu; however, some other step types can make a VSM undeployable.

Access the menu of step types from the VSM Editor by selecting **Add a new step**, **Virtual Service Environment** from the menu.

This section contains the following pages:

- Virtual Service Router Step
- Virtual Service Tracker Step
- Virtual Conversational/Stateless Response Selector Step
- Virtual HTTP/S Listener Step
- Virtual HTTP/S Live Invocation Step
- Virtual HTTP/S Responder Step
- Virtual JDBC Listener Step
- Virtual JDBC Responder Step
- Socket Server Emulator Step
- Messaging Virtualization Marker Step
- Compare Strings for Response Lookup Step
- Compare Strings for Next Step Lookup Step
- Virtual Java Listener Step
- Virtual Java Live Invocation Step
- Virtual Java Responder Step
- Virtual TCP/IP Listener Step
- Virtual TCP/IP Live Invocation Step
- Virtual TCP/IP Responder Step

- Virtual CICS Listener Step
- Virtual CICS Responder Step
- CICS Transaction Gateway Listener Step
- CICS Transaction Gateway Live Invocation Step
- CICS Transaction Gateway Responder Step
- IMS Connect Listen Step
- IMS Connect Live Invocation Step
- Virtual IMS Connect Responder Step
- Steps for Messaging Transport Protocols
- JCo IDoc Listener Step
- JCo IDoc Live Invocation Step
- JCo IDoc Responder Step
- JCo RFC Listener Step
- JCo RFC Live Invocation Step
- JCo RFC Responder Step

Virtual Service Router Step

The Virtual Service Router step routes a request from a virtual service listen step to the response selector step and the protocol-specific live invocation step, or both. The decision is made based on the current execution mode for the running model.

Follow these steps:

1. Complete the following fields as described:

   - **Live invocation step**
     Select the step for the live invocation from the list.

   - **If Environment Error**
     Specifies the action to take or the step to go to if the test fails because of an environment error.
     Default: Abort the test.

   - **When in dynamic mode, determine real mode using**
     Select **Subprocess** or **Script**.

2. To test your parameters for the step, click **Test**.

Virtual Service Tracker Step

Use the Virtual Service Tracker step to track responses in a running virtual service and (optionally) validate against a live system. This validation allows for easier service model debugging and model "healing".

Complete the following fields as described:

- **Image Response**
  Select the Image response file.

- **Live Response**
  Select the Live response file.

- **If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

Default: Abort the test.

Virtual Conversational/Stateless Response Selector Step

You could consider the Virtual Conversational/Stateless Response Selector step as the main step in any VSM. It reviews a specified service image and selects an appropriate virtual response for a specific request. Because there can be multiple responses for a request, the responses are always shown as a list. The step is typically created when you record and virtualize some form of service traffic.

Complete the following fields as described:

- **Service Image Location**
  Select from the drop-down list of available service images to associate with this step. When you have chosen a service image here, view it or edit it by clicking **Open** and it opens in a new tab.

- **Request property name**
  To define the property to review for the inbound request, set the property name. The property name is usually the response from the previous step.

- **Format step response as XML**

  The VSE framework expects Respond steps to accept one of the following:

  - A response object
  - A list of response objects
  - An XML document that represents either

  Note: If this check box is cleared, the step produces a list of response objects. The step produces the list, even if it contains only one response.

  Default: The step response is formatted as XML.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

Virtual HTTP/S Listener Step

Use the Virtual HTTP/S Listener step to simulate an HTTP server, including SSL support. The step listens for incoming HTTP requests and converts them to a standard virtual request format. The URL parameters are decoded using UTF-8 charset and are added as Virtual Request Arguments.

The default name for the Virtual HTTP/S Listener step is **Virtual HTTPS Listener<***portnumber***>**. You can rename the step at any time.

Complete the following fields as described:

- **Listen port**
  Enter the port on which DevTest listens for the HTTP/S traffic.

- **Bind address**
  Enter the local IP address on which connections can come in. With no bind address specified, the listen step accepts connections on the specified port regardless of the NIC (or the IP address) on which it comes in.

- **Bind only**
  To acquire the network resource and move to the next step, select this check box. A second **Listen** step that does not use the Bind only option is required. This option enables the model to listen on a port (the application queues requests until a listen step consumes them). The model performs 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 to client**
  Specifies whether DevTest uses a custom keystore when acting as an SSL server during recording and playback.

  - If you select **Use SSL to client,** an HTTPS connection is used between the client and virtual service. You can also specify a custom server-side keystore file and password that is associated with it.

  - If you do not select **Use SSL to client**, an HTTP connection is used between the client and virtual service.

- **SSL keystore file**

- Specifies the name of the server-side keystore file. DevTest uses this keystore when handling HTTPS connections from the client.

- By default, the **ssl.server.cert.path** property is used for the server-side keystore file. This property can also be selected from the **Defaults** section of the dropdown list.

- **Keystore password**

- Specifies the password that is associated with the specified server-side keystore file.

- You can specify the password directly using the Password Editor option, or you can use the DevTest Property Reference Editor option to specify a property expression that will be evaluated to provide the password.

- By default, the property expression **{{ssl.server.cert.pass}}** is used for the server-side keystore password. This property expression can be selected from the **Defaults** section of the dropdown list.

- **Enable Client Certificate Authentication**
  Selecting this check box allows a request to be sent to the client for their certificate during the SSL/TLS handshake.
  The following options determine what to do with the client certificate.

- **Request Client Certificate**
  The SSL server requests a client certificate without requiring it. This is the default option when you enable client certificate authentication.

- **Require Client Certificate**
  The SSL server requires a valid client certificate.

- **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. The listen step that is associated with the queue (by base path) processes the request.

- **Format step response as XML**

  The VSE framework expects Respond steps to accept one of the following:

  - A response object
  - A list of response objects
  - An XML document that represents either

  > **Note:** If this check box is cleared, the step produces a list of response objects. The step produces the list, even if it contains only one response.

  Default: The step response is formatted as XML.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

Virtual HTTP/S Live Invocation Step

Use the Virtual HTTP/S Live Invocation step to make a real HTTP call to real server in the context of a virtualized HTTP service. This step is typically created by recording and virtualizing some form of HTTP traffic. The step performs the real request, which is based on the current VSE request in use.

The default name for the Virtual HTTP/S Live Invocation step is **Virtual HTTPS LiveInvocation<** *portnumber* **>**. You can rename the step at any time.

Complete the following fields as described:

- **Target server**
  Enter the name of the server to which the request is made.

- **Target port**
  Enter the name of the port on which the request is made.

- **Replacement URI**
  To replace the full URI in a GET/POST request, enter a new target path field. You can provide the URI as a DevTest property. This field can be blank, in which case the URI from the live request is used.

- **Do not modify host header parameter received from client**
  If selected, this option instructs the live invocation to forward the host header that is received from the client application to the target server. If cleared, the live invocation regenerates the host header parameter to be **host: <***target host***>:<***target port***>**.

- **Use SSL to server**
  Specifies whether to send an HTTPS request to the live system.
  If you select **Use SSL to server**, an HTTPS connection is used between the virtual service and the live system.
  If you do not select **Use SSL to server**, an HTTP connection is used between the virtual service and the live system.

- **SSL keystore file**

- Specifies the name of the client-side keystore file. DevTest uses this keystore when using an HTTPS connection to the server.

- By default, the **ssl.client.cert.path** property is used for the client-side keystore file. This property can also be selected from the **Defaults** section of the dropdown list.

- **Keystore password**

- Specifies the password that is associated with the specified client-side keystore file.

- You can specify the password directly using the Password Editor option, or you can use the DevTest Property Reference Editor option to specify a property expression that will be evaluated to provide the password.

- By default, the property expression **{{ssl.client.cert.pass}}** is used for the client-side keystore password. This property expression can be selected from the **Defaults** section of the dropdown list.

- **Format step response as XML**
  The VSE framework expects Respond steps to accept one of the following:

  - A response object

  - A list of response objects

  - An XML document that represents either

  > **Note:** If this check box is cleared, the step produces a list of response objects. The step produces the list, even if it contains only one response.

  Default: The step response is formatted as XML.

- **Bad Response Codes**
  Defines a failure response from the system.
  Format: A comma-delimited list of three-character codes, with each character being either a numeric or the letter x (wildcard).
  Example: 4xx,5xx

- **VSE Lookup Step**
  For a live invocation step to support failover execution mode, it must know the step that is used to look up VSE responses so that it can redirect the VS model to the correct step when necessary. This field contains a list of steps in the VS model. Select the standard VSE Response Lookup step. That allows the live invocation step to move to the VSE response lookup step when necessary.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

Note: The Virtual HTTP/S Live Invocation step supports the **lisa.http.timeout.socket** and **lisa.http.timeout.connection** properties to control the client sockets used. The **lisa.vse.http.live.invocation.max.idle.socket** property controls how long an idle client socket waits before it is too old to use. This property defaults to 2 minutes.

See Local Properties File for more information about using the proxy properties: **lisa.http.webProxy.ssl.host**, **lisa.http.webProxy.ssl.port**, **lisa.http.webProxy.host.account**, and **lisa.http.webProxy.host.credential**.

Virtual HTTP/S Responder Step

Use this step with the Virtual HTTP/S Listener step to transmit responses to HTTP requests produced by the listener. The step uses a virtual response as the reply to the corresponding request using the HTTP/S protocol.

You can create this step by recording and virtualizing HTTP traffic.

Complete the following fields as described:

- **Responses list property name**
  Specifies the name of the property in which to look for the response to send.

- **If Environment Error**
  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **Conversational Model Properties**
  Enter a property, then click **Add**.

  To delete a property, select it from the list and click **Remove**. The listed properties are associated with the current conversation session, which makes the values available to downstream conversational requests.

Virtual JDBC Listener Step

Use the Virtual JDBC Listener step to control the simulation of JDBC database traffic. The step manages the communication with the simulation driver that is embedded in the database client.

The default name for the Virtual JDBC Listener step is **Virtual JDBC Listener<***portnumber***>**. You can rename the step at any time.

Complete the following fields as described:

- **Endpoint Information**
  Set up the simulation host and range of ports (the default is 2999) as appropriate. JDBC VSE supports multiple endpoints during recording and playback. The endpoint information is a table that contains:

  - **Driver Host**
  - **Base Port**
  - **Max Port**

  When **Base Port** and **Max Port** differ, a unique endpoint is created for each port between **Base Port** and **Max Port**, inclusive.

- **Format step response as XML**

  The VSE framework expects Respond steps to accept one of the following:

  - A response object
  - A list of response objects
  - An XML document that represents either

  > **Note:** If this check box is cleared, the step produces a list of response objects. The step produces the list, even if it contains only one response.

  Default: The step response is formatted as XML.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **Connect/Disconnect**
  To connect to the JDBC simulator, click **Connect**. If connected, click **Disconnect** to end the connection. You can use this button to validate the connection information.

- **Installed and Initialized JDBC Drivers**
  Lists the JDBC drivers that are installed and initialized in the database client.

- **Current SQL Activity**
  Identifies the current SQL activity in the database client.

Virtual JDBC Responder Step

The Virtual JDBC Responder step takes the result of a JDBC data call as selected by a conversational response selection step. The step then sends the result to the simulation driver that is embedded in the database client.

Complete the following fields as described:

- **Responses list property name**
  Specifies the name of the property in which to look for the response to send.

- **If Environment Error**
  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **Conversational Model Properties**
  Enter a property, then click **Add**.

  To delete a property, select it from the list and click **Remove**. The listed properties are associated with the current conversation session, which makes the values available to downstream conversational requests.

Socket Server Emulator Step

Use the Socket Server Emulator step to simulate any text-based server socket (typically HTTP). The Socket Server Emulator step supports listening, responding, and binding. The Socket Server Emulator step is low level. If you use this step, you must verify that the block of text that the respond step sends is fully HTTP-compliant.

> **Note:** When you use the Socket Server Emulator step in response mode, the text to go out **must** result in a valid HTTP response message.

Complete the following fields as described:

- **Process mode**
  From the list, select the process mode.
  Values:

- - Full Process

  - Asynchronous setup: Acquire the network resource and move to the next step

- Listen Only
- Respond Only

Default: Full Process

- **Listen port**
  Enter the port on which DevTest listens for the HTTP/S traffic.

- **Bind address**
  Enter the local IP address on which connections can come in. With no bind address specified, the listen step accepts connections on the specified port regardless of the NIC (or the IP address) on which it comes in.

- **Close immediately**
  This option instructs the step to do the configured work and then immediately clean up its network resources. Select this check box to use design-time testing of this step.

- **Use SSL**
  Select this check box to simulate a secure HTTPS website. 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, then 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. The listen step that is associated with the queue (by base path) processes the request.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **Record terminator**
  For a socket emulator that simulates a record-based service, enter the character to mark 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**
  When in a process mode that sends a response and the response is to be a valid HTTP response, this option verifies:

  - The HTTP headers in the response text are correctly formatted
  - (If necessary) The **Content-Length:** HTTP response header is present and correct.

This check box only verifies that the line separators are HTTP-compliant and that the **Content-Length** header is present and accurate. However, to work completely, the message must already be a well-formed HTTP message.
Default: Selected

- **Listener status**
  Indicates whether the listener is running.

- **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**
  Displays the Last/Original Request, which is used only at design time. This tab displays the last request that the step received.

The default name for the Socket Server Emulator step is **Socket Server Emulator <***portnumber***>**. You can rename the step at any time.

Messaging Virtualization Marker Step

Use the Messaging Virtualization Marker step to designate that a message-based test case is designed for use in the Virtual Service Environment. If the test case listens or responds through JMS, add this step to the virtual service model to verify that it can be deployed to VSE.

Note: You do not need to use this step with the JMS transport protocol.

Compare Strings for Response Lookup Step

The Compare Strings for Response Lookup step reviews an incoming request to a virtual service. The step then determines the appropriate response in a 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, and others.

Complete the following fields as described:

- **Text to match**
  Enter the text against which the criteria are matched. This value 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**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **Store responses in a compressed form...**
  Specifiy whether to compress the responses in the test case file.
  Default: Selected

- **Case Response Entries**
  Add, move, and delete entries.

- **Enabled**
  Specifies whether to enable or ignore an entry. Clear this check box to ignore an entry.
  Default: Selected when you add 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 from 1000 milliseconds through 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.
  From the **Criteria** list, select a setting.

- **Compare Type**
  Select an option from the list:

  - Find in string
  - Regular expression
  - Starts with
  - Ends with
  - Exactly equals

  Default: Find in string

- **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, then select a different setting from the **Response** list.

- **Criteria**
  You can update the criteria string for an entry.

- **Response**
  You can update the step response for an entry.

Compare Strings for Next Step Lookup Step

Use the Compare Strings for Next Step Lookup step to review an incoming request and determine the appropriate next step. You can match incoming requests using partial text match and regular expression, and others.

Each matching criterion specifies the name of the step to which to transfer if the match succeeds.

Complete the following fields as described:

- **Text to match**
  Enter the text against which the criteria are matched. This value 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**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **Next Step Entries**
  Add, move, and delete entries.

- **Enabled**
  Specifies whether to enable or ignore an entry. Clear this check box to ignore an entry.
  Default: Selected when you add 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 from 1000 milliseconds through 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.
  From the **Criteria** list, select a setting.

- **Compare Type**
  Select an option from the list:

  - Find in string
  - Regular expression
  - Starts with
  - Ends with

- Exactly equals

Default: Find in string

- **Next Step**
  From the list, select the step to go to if the match is found.

- **Criteria**
  You can update the criteria string for an entry.

Virtual Java Listener Step

Use the Virtual Java Listener step to handle virtualized JVM calls, such as calls to an EJB or other remote system. The step listens for the method calls that the DevTest Java Agent intercepts, and converts them to a standard VSE request.

- **Available Online Agents**
  Lists all the online agents that are available to connect.

- **Connected Agents**
  Lists all the online or offline agents that are connected for the virtual service model. The offline agents display in a gray italic font.

Connecting Agents

You can select agents from the **Available Online Agents** list. Select the agent or agents and click the right arrow button. The agent moves to the connected agents list.

When you select an agent from either the **Available Online Agents** list or the connected agents list, the information bar below the list displays host and Main Class information for the agent.

To add an agent manually to the connected agents list, use the **Add Agent** field above the **Connected Agents**.

The agent name cannot be empty or already present in the connected agents list. If the agent name entered exists in the online agents list, it is moved from the online agents to the connected agents list.

If an existing agent is not in the **Available Online Agents** list, type it in the dialog and click **Add** .

This mechanism is primarily provided for adding offline agents that were not previously in the connected list.

Disconnecting Agents

To disconnect an agent, select the agent from the connected agents list and click the left arrow button.

Selecting Classes and Protocols

To search for classes, select the **Search for Classes** arrow and enter a class name. These classes are entered as fully qualified names (including package), using regular expressions. To select classes, select the class name on the list and select the right arrow to move the class into the right pane. Some classes appear more than once; it is only necessary to select a class once for it to be virtualized.

To enter a class manually, select the **Manually Enter a Class Name** arrow. Enter the name of the class. To move the class in the right pane, select it and click the right arrow. To retrieve a list of classes that the DevTest agent suggested for virtualization, select the **Agent Suggestions** arrow and click **Retrieve** .

To add a protocol to the recording, select the **Protocols** arrow. From the list of available protocols, select any to record and click the right arrow to move them in the right pane.

To view or change the configuration information for a protocol, double-click any row with three dots (...) to the right of the protocol name. The **Protocol Configuration** window opens, and you can update the parameters.

Virtual Java Live Invocation Step

To record the Virtual Java Live Invocation step:

Complete the following fields as described:

- **Format step response as XML**

  The VSE framework expects Respond steps to accept one of the following:

  - A response object
  - A list of response objects
  - An XML document that represents either

  Note: If this check box is cleared, the step produces a list of response objects. The step produces the list, even if it contains only one response.

  Default: The step response is formatted as XML.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

Virtual Java Responder Step

Use this step with the Virtual Java Listener step to provide responses for virtualized JVM calls.

Complete the following fields as described:

- **Responses list property name**
  Specifies the name of the property in which to look for the response to send.

- **If Environment Error**
  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **Conversational Model Properties**
  Enter a property, then click **Add**.

  To delete a property, select it from the list and click **Remove**. The listed properties are associated with the current conversation session, which makes the values available to downstream conversational requests.

Virtual TCP/IP Listener Step

Use the Virtual TCP/IP Listener step to simulate TCP/IP connections to a server application. The step listens for incoming TCP/IP traffic and converts it to a standard VSE request.

The default name for the Virtual TCP/IP Listener step is **Virtual TCP/IP Listener<***portnumber***>**. You can rename the step at any time.

Complete the following fields as described:

- **Listen port**
  Enter the port on which DevTest listens for the TCP/IP traffic.

- **Bind address**
  Enter the local IP address on which connections can come in. With no bind address specified, the listen step accepts connections on the specified port regardless of the NIC (or the IP address) on which it comes in.

- **Use SSL to server**
  Specifies whether to send an SSL (secured layer) request to the server.
  Values:

  - **Selected**: Sends an SSL request to the server. You can select **Use SSL to server** and not select **Use SSL to client.** In that case, a plain TCP connection is presented for recording, but those requests are sent to the server using SSL.

  - **Cleared**: The application does not send an SSL request to the server.

- **Use SSL to client**
  This option is only enabled if **Use SSL to Server** has been selected. Check whether we can play back an SSL request from the client using a custom client keystore. When you specify **Use SSL to client**, you are allowed to specify a custom keystore and a passphrase. If these values are entered, they are used rather than the hard-coded defaults.

- **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, then click **Verify**.

- **Treat request as text**
  Specifies whether the application processes the request as text.
  Values:

  - **Selected**: The application processes the request as text.

  - **Cleared**: The application does not process the request as text.

- **Request encoding**
  When **Treat request as text is selected**, specifies the request encoding.

- **Request Delimiter**
  Specifies the delimiter that defines the end of the request.
  Values:

  - **None**
  - **Records are terminated with line endings**
  - **Records are of fixed length**
  - **Records are equal to whole data package**
  - **Records are delimited by specific characters**
  - **Delimiters match a regular expression**
  - **Regular expression matches the expression (includes delimiter)**
  - **SWIFT**

- **Format step response as XML**

  The VSE framework expects Respond steps to accept one of the following:

  - A response object
  - A list of response objects
  - An XML document that represents either

  > **Note:** If this check box is cleared, the step produces a list of response objects. The step produces the list, even if it contains only one response.

  Default: The step response is formatted as XML.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

Virtual TCP/IP Live Invocation Step

Use the Virtual TCP/IP Live Invocation step to make a real TCP/IP call to a real server in the context of a virtualized TCP/IP service. The step is typically created by recording and virtualizing some form of TCP/IP traffic. The step performs the real request, which is based on the current VSE request in use.

The TCP/IP Live Invocation Step attempts to reuse any existing connection to the live system when multiple requests are made within a certain time period. This time period (in seconds) is controlled by the **lisa.vse.tcp.live.invocation.max.idle.socket** property; by default this time period is two minutes (120 seconds).

If you do not want to reuse the existing connection and instead use a new connection for each request, set the **lisa.vse.tcp.live.invocation.max.idle.socket** property to zero seconds:

lisa.vse.tcp.live.invocation.max.idle.socket=0

The default name for the Virtual TCP/IP Live Invocation step is **TCP Protocol Live Invocation<** *portnumber* **>**. You can rename the step at any time.

Complete the following fields as described:

- **Target port**
  Enter the name of the port on which the request is made.

- **Target server**
  Enter the name of the server to which the request is made.

- **Use SSL to server**
  Specifies whether to send an SSL (secured layer) request to the server.
  Values:

  - **Selected**: Sends an SSL request to the server. You can select **Use SSL to server** and not select **Use SSL to client.** In that case, a plain TCP connection is presented for recording, but those requests are sent to the server using SSL.

  - **Cleared**: The application does not send an SSL request to the server.

- **Use SSL to client**
  This option is only enabled when you select **Use SSL to Server**. **Use SSL to client** specifies whether the application can play back an HTTPS request from the client using a custom client keystore.
  Values:

  - **Selected**: You can specify a custom keystore and a passphrase. If you enter these values, the application uses them instead of the hard-coded defaults.

  - **Cleared**: The application cannot play back an SSL request from the client using a custom client keystore.

- **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, then click **Verify**.

- **Treat response as text**
  Specifies whether the application processes the response as text.
  Values:

  - **Selected**: The application processes the response as text.

  - **Cleared**: The application does not process the response as text.

- **Response encoding**
  When **Treat response as text is selected**, specifies the response encoding.

- **Response Delimiter**
  Specifies the delimiter that defines the end of the response.
  Values:

  - **None**
  - **Records are terminated with line endings**
  - **Records are of fixed length**
  - **Records are equal to whole data package**
  - **Records are delimited by specific characters**
  - **Delimiters match a regular expression**
  - **Regular expression matches the expression (includes delimiter)**
  - **SWIFT**

- **Format step response as XML**

  The VSE framework expects Respond steps to accept one of the following:

  - A response object
  - A list of response objects
  - An XML document that represents either

  > **Note:** If this check box is cleared, the step produces a list of response objects. The step produces the list, even if it contains only one response.

  Default: The step response is formatted as XML.

- **VSE Lookup Step**
  For a live invocation step to support failover execution mode, it must know the step that is used to look up VSE responses so that it can redirect the VS model to the correct step when necessary. This field contains a list of steps in the VS model. Select the standard VSE Response Lookup step. That allows the live invocation step to move to the VSE response lookup step when necessary.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

Virtual TCP/IP Responder Step

To use the Virtual TCP/IP Responder step:

Complete the following fields as described:

- **Responses list property name**
  Specifies the name of the property in which to look for the response to send.

- **If Environment Error**
  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **Conversational Model Properties**
  Enter a property, then click **Add**.

  To delete a property, select it from the list and click **Remove**. The listed properties are associated with the current conversation session, which makes the values available to downstream conversational requests.

Virtual CICS Listener Step

The Virtual CICS Listener step simulates a CICS LINK server.

For information about the fields on this panel, see Using the CICS Programs to Virtualize Panel.

Virtual CICS Responder Step

The Virtual CICS Responder step is used with the Virtual CICS Listener step to transmit responses to requests produced by the listener. The step takes a virtual response and uses it as the reply to the corresponding request.

Complete the following fields as described:

- **Responses list property name**
  Specifies the name of the property in which to look for the response to send.

- **If Environment Error**
  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

To populate the **Conversational Model Properties** list, use the **Add** and **Remove** buttons.

CICS Transaction Gateway Listener Step

Use the CICS Transaction Gateway Listener step to simulate a CTG server, including SSL support. The step listens for incoming CTG requests and converts them to a standard virtual request format.

Complete the following fields as described:

- **Listen on port**
  Enter the port on which DevTest listens for the CTG traffic.

- **Bind address**
  Enter the local IP address on which connections can come in. With no bind address specified, the listen step accepts connections on the specified port regardless of the NIC (or the IP address) it comes in on.

- **Bind only**
  To acquire the network resource and move to the next step, select this check box. A second Listen step that does not use the **Bind only** option is required. This option enables the model to listen on a port (the application queues requests until a listen step consumes them). The model performs 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.

- **Expect SSL From Clients**
  If you select this check box, the recorder expects clients to connect to it using SSL. The related keystore and password, if provided, are used to obtain security information (such as certificates).

- **SSL keystore file**
  Specifies the name of the keystore file.

- **Keystore password**
  Specifies the password associated with the specified keystore file.

- **CTG Protocol version**
  Specifies the version of the CTG protocol to use.

- **Locale**
  Defines the locale that represents the language and country code that are reported to the CTG client during the initial protocol handshake.

- **JVM text**
  Describes the JVM on the mainframe.

- **Server class**
  Defines a string that the application reports to the client. This field is not typically used.

- **Client app ID**
  Defines a string that the application reports to the client. This field is not typically used.

- **Has security**
  Specifies to the CTG client whether user authentication is required.

- **Enable server ping**
  When a CTG client connects to a server, that server starts sending "ping" messages to the client regularly to verify the connection. This is not strictly necessary where VSE is used in a testing environment, but selecting this option causes the VSE CTG server to emulate those "ping" messages.

- **Format step response as XML**

  The VSE framework expects Respond steps to accept one of the following:

  - A response object

  - A list of response objects

- An XML document that represents either

> **Note:** If this check box is cleared, the step produces a list of response objects. The step produces the list, even if it contains only one response.

Default: The step response is formatted as XML.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

CICS Transaction Gateway Live Invocation Step

Use the CICS Transaction Gateway Live Invocation step to make a real CTG call to a real server in the context of a virtualized CTG service. The step is typically created by recording and virtualizing some form of CTG traffic. It performs the real request based on the current VSE request in use.

Complete the following fields as described:

- **Target server**
  Enter the name or IP address of the target host where the CTG server runs.

- **Target port**
  Enter the number of the port on which the CTG server listens.

- **Initiate SSL to the target server**
  If selected, sends SSL (Secure Socket Layer) request to the server.

- **SSL keystore file**
  Specifies the name of the keystore file.

- **Keystore password**
  Specifies the password associated with the specified keystore file.

- **Format step response as XML**

  The VSE framework expects Respond steps to accept one of the following:

  - A response object

  - A list of response objects

  - An XML document that represents either

> **Note:** If this check box is cleared, the step produces a list of response objects. The step produces the list, even if it contains only one response.

Default: The step response is formatted as XML.

- **VSE Lookup Step**
  For a live invocation step to support failover execution mode, it must know the step that is used to look up VSE responses so that it can redirect the VS model to the correct step when necessary. This field contains a list of steps in the VS model. Select the standard VSE Response Lookup step. That allows the live invocation step to move to the VSE response lookup step when necessary.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

CICS Transaction Gateway Responder Step

Use the CICS Transaction Gateway Responder step with the CICS Transaction Gateway Listener step to transmit responses to CTG requests that the listener produces. This step uses a virtual response as the reply to the corresponding request using the CTG protocol.

You can create this step by recording and virtualizing CICS Transaction Gateway traffic.

Complete the following fields as described:

- **Responses list property name**
  Specifies the name of the property in which to look for the response to send.

- **If Environment Error**
  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **Conversational Model Properties**
  Enter a property, then click **Add**.

  To delete a property, select it from the list and click **Remove**. The listed properties are associated with the current conversation session, which makes the values available to downstream conversational requests.

IMS Connect Listen Step

Use the IMS Connect Listener step to respond to IMS Connect requests by the IMS Connect virtual service.

Enter the following fields as described:

- **Listen/Record on port**
  Defines the port on which the client communicates to DevTest.

- **Target host**
  Disabled for the Listen step.

- **Target port**
  Disabled for the Listen step.

- **IMS Format file**
  Disabled for the Listen step.

- **Format step response as XML**

  The VSE framework expects Respond steps to accept one of the following:

  - A response object
  - A list of response objects
  - An XML document that represents either

  Note: If this check box is cleared, the step produces a list of response objects. The step produces the list, even if it contains only one response.

  Default: The step response is formatted as XML.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

IMS Connect Live Invocation Step

To use the IMS Connect Live Invocation step:

Complete the following fields:

- **Listen/Record on port**
  Not applicable for the Live Invocation step.

- **Target host**
  Specifies the name or IP address of the target host where the server runs.

- **Target port**
  Specifies the target port number listened to by the IMS server.

- **IMS Format file**
  To use the IMS Connect support that is included in DevTest by default, leave this field blank.

- **Format step response as XML**
  The VSE framework expects Respond steps to accept one of the following:

  - A response object
  - A list of response objects
  - An XML document that represents either

> **Note:** If this check box is cleared, the step produces a list of response objects. The step produces the list even if it contains only one response.

Default: The step response is formatted as XML.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **VSE Lookup Step**
  For a live invocation step to support failover execution mode, it must know the step that is used to look up VSE responses so that it can redirect the VS model to the correct step when necessary. This field contains a list of steps in the VS model. Select the standard VSE Response Lookup step. That allows the live invocation step to move to the VSE response lookup step when necessary.

Virtual IMS Connect Responder Step

Use the Virtual IMS Connect Responder step with the IMS Connect Listener step to transmit IMS Connect responses.

You can create this step by recording and virtualizing IMS Connect traffic.

Complete the following fields:

- **Responses list property name**
  Specifies the name of the property in which to look for the response to send.

- **If Environment Error**
  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **Conversational Model Properties**
  Enter a property, then click **Add**.

  To delete a property, select it from the list and click **Remove**. The listed properties are associated with the current conversation session, which makes the values available to downstream conversational requests.

Steps for Messaging Transport Protocols

The following steps are specific to virtual service models that are created using the JMS transport protocol:

- **JMS VSE Listen**

- **JMS VSE Respond**

- **JMS VSE Live Invocation**

The following steps are specific to virtual service models that are created using the IBM MQ Native transport protocol:

- **IBM MQ Native VSE Listen**
- **IBM MQ Native VSE Respond**
- **IBM MQ Native VSE Live Invocation**

The following steps are specific to virtual service models that are created using the RabbitMQ transport protocol:

- **RabbitMQ VSE Listen**
- **RabbitMQ VSE Respond**
- **RabbitMQ VSE Live Invocation**

Each step has basic and advanced parameters. To display the advanced parameters, click **PRO** at the top of the editor.

Listen Step

The **Listen** step listens for incoming requests and converts them to standard VSE requests.

The **Receive** tab contains the list of receive operations.

The **Channel Name** field defines the request channel name. The value must match the operation name that is defined in the service image.

To disable and reenable individual request channels, use the **Enabled** check box. At least one request channel must be enabled.

The **ReplyTo Mappings** tab is applicable to the segregated messaging scenario. This type of scenario occurs under the following conditions:

- A set of queues is accessible to DevTest.
- Another set of queues is not accessible to DevTest.
- The application runs on the non-accessible queues.
- Messages are automatically forwarded word for word to the DevTest-accessible queues automatically.

Each mapping consists of the following items:

- The response-side channel name
- The client reply-to destination used by the client

Respond Step

The **Respond** step sends one or more response messages in the VSE response.

The **Channel Name** field defines the response channel name. The value must match the channel.name metadata property that is defined in the service image.

To disable and reenable individual response channels, use the **Enabled** check box.

Live Invocation Step

The **Live Invocation** step sends requests to the live service.

The **Live Request Send** tab contains the list of operations to use for sending a live request.

The **Channel Name** field defines the request channel name. The value must match the channel name in the **Listen** step.

To disable and reenable individual request channels, use the **Enabled** check box. At least one request channel must be enabled.

The **Live Response Receive** tab contains the list of operations to use for receiving live responses.

The **Channel Name** field defines the response channel name. The value must match the channel name in the **Respond** step.

To disable and reenable individual response channels, use the **Enabled** check box. At least one response channel must be enabled.

The **Timeout** parameter and the **Maximum Responses** parameter are among the criteria that the step can use to determine whether to leave the live invocation loop.

The **ReplyTo Mappings** tab is applicable to the segregated messaging scenario. This type of scenario occurs under the following conditions:

- A set of queues is accessible to DevTest.
- Another set of queues is not accessible to DevTest.
- The application runs on the non-accessible queues.
- Messages are automatically forwarded word for word to the DevTest-accessible queues automatically.

Each mapping consists of the following items:

- The response-side channel name
- The service reply-to destination that must be sent to the service for its response to come through the live response queue on the response channel

Virtual Service Model Flows

During normal operation, the execution flow of the virtual service model is like any other VSE service.

The following graphic uses an overlay to illustrate the flow. Notice that the **Live Invocation** step is not used.

Screen capture of normal operation.

1. The **Listen** step receives a request message and converts it into a VSE request.
2. The **Virtual Service Router** step routes the flow to the response selection step.
3. The **VS Image Response Selection** step selects a matching transaction from the service image and produces a VSE response.
4. The **Respond** step sends one or more response messages in the VSE response.
5. Return to Step 1.

During live invocation, the execution flow of the virtual service model is more complicated.

The following aspects of asynchronous messaging make live invocation difficult:

- A single request can have multiple responses.
- Responses can take any amount of time to be returned.
- In general, it is impossible to determine when all of the responses have been received.

As a result, the **Live Invocation** step and the **Respond** step are run in a loop.

The following graphic uses an overlay to illustrate the flow.

Screen capture of live invocation.

1. The **Listen** step receives a request message and converts it into a VSE request.
2. The **Virtual Service Router** step routes the flow to the live invocation step.
3. The **Live Invocation** step forwards the request to the live service.
4. The **Live Invocation** step starts listening on every live response queue.
5. The **Live Invocation** step receives a single response from any of the live response queues and converts it into a VSE response.
6. The **Respond** step returns one response message to the client.
7. Return to Step 5 and repeat until the **Live Invocation** step determines that the transaction is complete. Any of the following conditions can cause the **Live Invocation** step to make this determination:
   - The timeout is set and it passes without receiving another response from any of the live response queues. The default value is 30 seconds. The timeout is an advanced parameter that can be set in the VSE recorder or in the **Live Invocation** step.
   - The maximum number of responses is set and that number has been reached. The default value is 1, which indicates that the loop can recur only once. The maximum number of responses is an advanced parameter that can be set in the VSE recorder or in the **Live Invocation** step.
   - The virtual service model is running close enough to capacity that it needs the current model execution thread to break out of its transaction and handle a new request. If the timeout and the maximum number of responses have not been set, this action is the only way for the model to break out of waiting for live responses and loop back to the **Listen** step.
8. The **Live Invocation** step constructs a final VSE response that contains all of the response messages that went through the loop. The step loops a final time to the **Respond** step.
9. The **Respond** step does not send any messages on the final loop. Instead, the **Respond** step completes the typical VSE state cleanup tasks.
10. Return to Step 1.

When the virtual service model is in Failover mode, it acts like Live Invocation mode most of the time. However, when the Live Invocation step fails to receive a response from the service, the virtual service model fails over to the service image to generate a response.

1. The Listen step receives a request message and converts it into a VSE request.
2. The Virtual Service Router step routes the flow to the live invocation step.
3. The Live Invocation step forwards the request to the live service.
4. The Live Invocation step starts listening on every live response queue.
5. The Live invocation step fails to receive a single response from any of the live response queues. This scenario can happen because of a connection failure while listening on any of the response queues, or a failure to receive a response within the specified timeout.
6. Execution goes to the VS Image Response Selection step, which generates a full VSE response that contains one or more response messages.
7. The Respond step sends one or more response messages in the VSE response.
8. Go to Step 1. Each transaction can follow the Live Invocation flow or the Failover flow, depending on what happens in Step 5.

When the virtual service model is in Stand In mode, it acts like Normal mode most of the time. However, if the service image returns the unknown response, the virtual service model switches to Live Invocation mode and gets the response from the live service. This mode is basically the opposite of Failover mode.

1. The Listen step receives a request message and converts it into a VSE request.
2. The Virtual Service Router step routes the flow to the response selection step.
3. The VS Image Response Selection step fails to find a matching transaction from the service image.
4. Execution goes to the Live Invocation step. The Live invocation step forwards the request to the live service.
5. The Live invocation step starts listening on every live response queue.
6. The Live Invocation step receives a single response from any of the live response queues and converts it into a VSE response.
7. The Respond step returns one response message to the client.
8. Go to Step 4 and repeat until the Live Invocation step determines that the transaction is complete.
9. The Live Invocation step constructs a final VSE response that contains all of the response messages that went through the loop. The step loops a final time to the Respond step.
10. The Respond step does not send any messages on the final loop. Instead, the Respond step completes the typical VSE state cleanup tasks.
11. Go to Step 1. Each transaction can follow the Normal flow or the Stand In flow, depending on what happens in Step 3.

JCo IDoc Listener Step

Use the JCo IDoc Listener step to respond to JCo IDoc requests from the JCo IDoc virtual service. To create this step, record and virtualize JCo IDoc traffic.

Complete the following fields:

- **Client RFC Connection Properties**
  Defines the Client RFC Connection properties file that contains connection properties that VSE uses to register itself under a program ID to an SAP gateway and receive IDocs. The properties should be the same as those specified in a .jcoServer file.

- **Client RFC Destination Name**
  Specifies a unique name that identifies the RFC destination. The name must be unique for each service deployed on a same VSE, especially in the case when two or more services have the same name, but different Client RFC/System Connection Properties selected.

- **Client System Connection Properties**
  Specifies the Client System Connection properties file that contains connection properties to return IDocs to the client SAP system. These properties should be the same as those specified in a .jcoDestination file that can be used to connect to the client SAP system.

- **Client System Name**
  Specifies a unique name to identify the client SAP system. The name must be unique for each service deployed on a same VSE, especially in the case when two or more services have the same name, but different Client RFC/System Connection Properties selected.

- **Request Identifier XPath Expressions**
  Specifies the XPath expressions that the protocol uses with the request IDoc XML to generate an identifier. The request identifier XPath expresions can be a single XPath expression. This identifier is used to correlate a request IDoc to a response IDoc. XPath expressions can also be a comma-separated list of XPath expressions, in which case the resulting values from the multiple expressions are concatenated (separated by dashes) and used as an identifier.

- **Response Identifier XPath Expressions**
  Defines the XPath expressions that the protocol uses with the response IDoc XML to generate an identifier. The Response Identifier XPath Expressions can be a single XPath expression. This identifier is used to correlate a response IDoc to a request IDoc that was received earlier. XPath expressions can also be a comma-separated list of XPath expressions, in which case the resulting values from the multiple expressions are concatenated (separated by dashes) and used as an identifier.

- **Format step response as XML**

  The VSE framework expects Respond steps to accept one of the following:

  - A response object
  - A list of response objects
  - An XML document that represents either

  Note: If this check box is cleared, the step produces a list of response objects. The step produces the list, even if it contains only one response.

Default: The step response is formatted as XML.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

JCo IDoc Live Invocation Step

To use the JCo IDoc Live Invocation step:

Complete the following fields:

- **Client RFC Destination Name**
  Specifies a unique name that identifies the RFC destination.

- **Server RFC Connection Properties**
  Specifies a properties file that contains connection properties that VSE uses to register itself under a program ID to an SAP gateway and receive IDocs. The properties should be the same as those specified in a .jcoServer file to start a JCo server program that receives IDocs from the server SAP system.

- **Server RFC Destination Name**
  Specifies a unique name to identify the Server RFC destination.

- **Server System Connection Properties**
  The Server System Connection properties file contains connection properties to return IDocs to the client SAP system. These properties should be the same as those specified in a . jcoDestination file that can be used to connect to the SAP server system.

- **Server System Name**
  Specifies a unique name to identify the Server SAP system.

JCo IDoc Responder Step

The JCo IDoc Responder step is used with the JCo IDoc Listener step to transmit JCo IDoc responses. To create this step, record and virtualize JCo IDoc traffic.

This step has no parameters.

JCo RFC Listener Step

Use the JCo RFC Listener step to respond to JCo RFC requests from the JCo RFC virtual service. To create this step, record and virtualize JCo RFC traffic.

Complete the following fields:

- **Client System Name**
  Specifies a unique name to identify the client SAP system.

- **Client System Connection Properties**
  The Client System Connection properties file contains connection properties with which to connect to the destination on the client system. This must be a .properties file in the Data directory of your project and contains properties that are typically found in a .jcoServer file.

This file MUST NOT specify **jco.server.repository_destination**. See the JavaDocs for **com.sap.conn.jco.ext.ServerDataProvider** in the **doc** folder of your installation directory.for more information about the supported properties.

- **Format step response as XML**
  If you select this check box, the step response (the incoming request) is serialized as XML and can be manipulated as text. This is deserialized at the response lookup step. Choosing this option slows the virtual service considerably.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

JCo RFC Live Invocation Step

To use the JCo RFC Live Invocation step:

Complete the following fields:

- **Destination System Name**
  Defines a unique name that identifies the SAP system on which the RFC executes. This is often the same as the Repository Name.

- **Destination System Connection Properties**
  Specifies the Destination System Connection properties file that contains connection properties with which to connect to the system that has the repository. This must be a . properties file in the Data directory of your project and contains properties that are typically found in a .jcoDestination file. For more information about supported properties, see the JavaDocs for **com.sap.conn.jco.ext.DestinationDataProvider** in the **doc** folder of your installation directory.This can be the same file as the Repository Connection Properties.

- **Format step response as XML**
  If you select this check box, the step response (the response from the live system) is serialized as XML and can be manipulated as text. This is deserialized at the respond step. Choosing this option slows the virtual service considerably.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test

- **VSE Lookup Step**
  For a live invocation step to support failover execution mode, it must know the step that is used to look up VSE responses so that it can redirect the VS model to the correct step when necessary. This field contains a list of steps in the VS model. Select the standard VSE Response Lookup step. That allows the live invocation step to move to the VSE response lookup step when necessary.

JCo RFC Responder Step

The JCo RFC Responder step is used with the JCo RFC Listener step to transmit JCo RFC responses. You can create this step by recording and virtualizing JCo RFC traffic.

Enter the following parameters:

- **Responses list property name**
  Specifies the name of the property in which to look for the response to send.

- **Conversational Model Properties**
  Enter a property, then click **Add**.

  To delete a property, select it from the list and click **Remove**. The listed properties are associated with the current conversation session, which makes the values available to downstream conversational requests.

De-identifying Data

De-identification (formerly called *desensitization*) means attempting to recognize sensitive data and substituting random, but validly formatted, values for that data during recording. Use data de-identification when you do not want to use real customer data as your test data.

- Dynamic De-identification
- Static De-identification
- Data De-Identifier Data Protocol

Dynamic De-identification

Dynamic de-identification occurs at the transport layer. De-identification is invoked by enabling the **De-identify (transport layer)** check box on the **Basics** tab of the **Virtual Service Recorder**.

This de-identification program uses filters that ensure that sensitive information is never written to disk during the recording phase. The **de-identify.xml** file in the DevTest home directory configures data de-identifiers to recognize known patterns such as credit card numbers. The file replaces the live data with realistic but unusable replacements. The file uses Regex pattern matching to recognize and find sensitive data. This file is parsed each time that the recorder is started.

You can use the built-in TestData string generation patterns as replacement data options. TestData provides 40,000 rows of test data, including replacement data for some common data types: names, addresses, telephone numbers, and credit card numbers.

You can customize these preset patterns to create your own. We recommend a regular expression toolkit such as RegexBuddy. RegexBuddy lets you paste in your recorded payload and interactively highlights Regex matches as you fine-tune the Regex.

Matches are processed in the order they exist in the file, so put your more specific matches first.

To avoid text escaping issues (especially with Regex), you must enclose <regex> and <replacement> child text in a CDATA element.

Static De-identification

Static data de-identification involves manually searching and replacing data in an existing service image.

To access the **Search and Replace** menu, right-click a node in the service image and select **Search and Replace**.

Specify a specific string to replace with another, then indicate the scope of the change. Click **Replace** to run the search and replace function for the areas you selected.

Data De-Identifier Data Protocol

The Data De-Identifier data protocol handler allows the application of de-identification rules when another data protocol is required to "unopaque" a request or response body. For example, when an HTTP message body is gzipped.

For more information, see Data De-Identifier Data Protocol.

Virtualizing a Service

Virtualization is the process by which VSE responds to the client in the absence of the server. Virtualization uses the VSM and the service image.

Starting the Virtual Service Environment

The Virtual Service Environment (VSE) must be started for the virtualization to run. The VSE must register with the DevTest registry.

To start the VSE, select **Programs**, **DevTest**, **Virtual Service Environment** from the **Start** menu.

A window opens to create a VSE named **lisa.VSEServer** and connects to the registry instance.

> **Note:** You can minimize the **Virtual Service Environment** window, but do not close this window.

If you installed DevTest Windows system services, you can use the system service to start VSE.

You can also use the following command to start a named VSE from a command prompt:

[LISA_HOME]\bin\VirtualServiceEnvironment.exe -n *VSEName* **-m** *RegistryName*

- **VSEName**
  Specifies the name of the VSE
- **RegistryName**
  Specifies the name of an existing registry

Running Multiple Virtual Service Environments

To run multiple VSE servers on one computer, add the **-p** *port* command-line option while running **VirtualServiceEnvironment.exe**, where **port** specifies the port number. The port number differs for different VSE instances.

The **maxvirtualservices** property in your license limits the number of virtual services you can run.

Using VSE Manager to Set up the VSE

VSE Manager lets you change your virtual service environments.

For more information about VSE Manager, see VSE Manager Commands.

Session Viewing and Model Healing

Session viewing lets you see the behavior of current (or recent) sessions on a VSE server. You can determine why the response for a specific request was given. Session viewing also enables a live comparison between the responses that VSE provides and a corresponding live system. Where differences exist, the application can use *model healing* to patch or heal the VSE service image to keep it in sync with the live system.

Session viewing is available only for virtual services that run with an execution mode of Transaction Tracking or Image Validation. Model healing is available only for virtual services that run in Image Validation mode.

Model healing differs from learning because learning changes a service image immediately when CA Service Virtualization and live response differences are detected. Healing logs those differences for later review and application to the service image with the **View Session and Tracking** information panel.

You can manage session viewing and model healing with a panel that is accessible while editing a virtual service model.

Session Information tab in the VSE Console

The session panel is divided into two panes: **Session Information** and **Response Information**. Each pane includes color-coded status indicators that show the status of the session and the response. The Session Information and Response Information sections each describe these status indicators in the context of the different panes. In general, the different colors of the status icon indicate the following:

- **Green:** Indicates there is an exact transaction match between the virtual service and the live system. Green also indicates that the response was successfully recorded, and the recorded response does not differ from the the live system response.

- **Yellow:** Indicates a match between the virtual service and the live system. However, the recorded response body differs from the live system.
- **Red:** Indicates conversational transaction navigation is different between the virtual service recording and the live system. For example, the response was not found in the virtual service, but the response is played back from the live system.
- **Gray:** None

Session Information Pane

The **Session Information** pane lists all the sessions for the selected virtual service in tabular format. To reorder the table or to select or clear the columns that show, click the arrow at the right of each column heading.

- **Session Status**
  Displays a color-coded icon to indicate the current session status.
  - **Green:** If the recorded response matches the live system response(green), then the session status is also green.
  - **Yellow:** If the recorded response differs from the live system (yellow), then the session status is also yellow.
  - **Red:** If a response is red, then the session status is also red.

  Once the session status is red, it remains red even if subsequent responses match successfully. This indicates that at least on new response is from the live system.
- **Session ID**
  Identifies the unique ID for each session.
- **Created On**
  Displays the timestamp of the first transaction in that session.
- **Modified On**
  Displays the timestamp of most recent transaction.
- **Txn Count**
  Identifies the number of transactions that were recorded after the service started.
- **Client ID**
  (Protocol-specific) For HTTP, displays the endpoint of the client that submitted the transaction.
- **Most Recent Request**
  Identifies the most recent request that came through on the specific session.

Response Information Pane

The **Response Information** pane shows the list of transactions for the selected session. When you point the mouse to the colored ball in the first column of this pane, a tooltip identifies the match for that transaction. If you click any specific transaction, the request and response tabs appear at the bottom of the pane. These tabs compare request/response between the VSE system and the live system.

In the **Response Information** pane, the following icons represent transactions:

- **Green:** Indicates a response body and navigation match between the live system and VSE image.
- **Yellow:** Indicates a response body mismatch with a navigation match.
- **Red:** Indicates a navigation mismatch.
  - For conversational transactions:
    Once a transaction is red or goes to the live system, all subsequent transactions are also red and go to the live system. Once a conversation goes to the live system for a conversation, the VSE image lacks the context to switch the conversation back to the VSE image.
  - For stateless transactions:
    Once a transaction is red or goes to the live system, subsequent transactions still match against the VSE image first to find a response. The response is not played back from the live system if the response exists in the VSE image.

The **Update** buttons are used to update the service image with live session/stateless transactions. This process is referred to as model healing. You use model healing to remove the disparity between the VSE image and the live system so that the VSM works correctly. When you click **Update**, the session marked with a red ball changes to a gray ball. This session is now tracked.

- The **Response Information Update** button updates the service image for the selected transaction.
- The **Session Information Update** button lets you select multiple sessions that display in the **Service Information** pane and update them all simultaneously.

Track Transactions

In VSE, you can track the transactions that are done through the ITR facility.

When using a virtual service in the ITR, set the execution mode in the configuration file that the project uses.

Follow these steps:

1. Select **Actions**, **View tracked responses** from the ITR run.
2. To track transactions of a specific virtual service, set the execution mode in the active configuration file by setting the **lisa.vse.execution.mode** property to TRACK.
   A new window in the editor lists the tracked transactions.

VSE Manager - Manage and Deploy Virtual Services

VSE Manager is an Eclipse plug-in that lets you deploy and manage DevTest virtual services.

VSE Manager is supported on Eclipse release 4.3 and later. Be sure to run Eclipse with a Java 7 JVM.

The following video demonstrates how to install, configure, and use the plug-in.

Install VSE Manager

To install VSE Manager, follow the standard Eclipse procedure for adding new software.

Follow these steps:

1. In Eclipse, select Help, Install New Software.
   The Install dialog opens.

2. Select Add to add the VSE Provisioning Platform (p2) repository URL:

   `http://www.itko.com/downloads/eclipse/9.5/updates/`

3. Select the VSE Manager UI feature, then click Next.
   The Install Details window opens.

4. Review the install details, then click Next.
   The Review Licenses window opens.

5. Review and accept the licenses, then click Finish.

Configure VSE Manager

Before you use VSE Manager, you must configure the VSE Manager page in the Eclipse Preferences dialog.

The following graphic shows the Preferences dialog with the VSE Manager page selected.

Preferences dialog

Notes:

- Set the **Registry URL** field to an **https** URL. This type of URL requires the enabling of HTTPS communication with the DevTest Console, as described in Using HTTPS Communication with the invoke APIs.

- Be sure to enter values in the **User Name** and **Password** fields. Leaving these fields blank can result in the appearance of the Eclipse Password Required dialog, which is not the correct dialog for VSE Manager. If the Password Required dialog does appear, click Cancel instead of entering a user name and password.

Follow these steps:

1. To configure the registry connection settings, enter the URL for the VSE Webserver.
2. If necessary, enter your user name and password.
3. (Optional) To save the order of columns and their associated widths when closing and reopening the view, select the **Save and restore**... check box.
4. (Optional) To have the view automatically refresh the content approximately every 10 seconds, select the **Update the current status**... check box.
5. To finish and close the dialog, click **OK**.

Import a LISA Project

Follow these steps:

1. From the main menu, open the Eclipse Import wizard.
2. From the CA LISA category, select LISA Project.
3. Select a LISA project and enter its name in the **Project Name** field.
4. Click **Finish**.
   The project imports.

Use the VSE Manager View

VSE Manager View panel

The VSE Manager view supports the following actions:

- Drag-and-drop to rearrange the columns of the view.
- Select the columns to hide or view with the drop-down list.

- Start, stop, or undeploy the selected virtual service.
- Drag-and-drop from an Eclipse-based product and from the native file system to deploy new virtual services to VSE in the MAR file format.

Properties View Integration

When a virtual service is selected in the VSE Manager view and the Properties view is open, you can view and update more information about the selected service.

VSE Manager properties - integration view

With this view, you can:

- Control automatically starting the service when the VSE server is started.
- Change the capacity of a deployed virtual service.
- Change the think scale for the requests to the virtual service.

You can drag a MAR file to the VSE Manager to deploy it.

VSE Commands

This section contains the following pages that detail the available VSE commands:

- VSE Manager Command - Manage Virtual Service Environments
- ServiceManager Command - Manage Services
- ServiceImageManager Command - Manage Service Images
- VirtualServiceEnvironment Commands

VSE Manager Command - Manage Virtual Service Environments

To manage virtual service environments, use the VSE Manager command-line tool. We recommend that you use the command-line tool for LISA 6.0 and later implementations. For more information, see Service Manager Commands.

VSE Manager is included as a tool under the bin directory in DevTest Server.

The VSE Manager command has the following format:

```
VSEManager [--registry name | --vse name | --username=name |--password=name ]
[--status [vs-name | all ] | --deploy archive-file | --redeploy archive-file | --
update vs-name [capacity capacity] [thinkscale scale] [auto-
restart [ON|OFF] [grouptag group tag] | --remove vs-name | --start vs-name | --set-
exec-mode vs-name --mode exec-mode | --stop vs-name | --help | --version
```

- **--registry *name***
  Sets the name of the DevTest registry through which to work.

> **Note:** The --registry parameter must be defined before a command is run. Otherwise, the command will, by default, run on the local registry. If there is no local registry, the command will not run.

- **--vse *name***
  Sets the name of the virtual service environment with which to work.

- **--username=*name***
  Sets the ID with which to authenticate to ACL.

- **--password=*name***
  Sets the password that is associated with the specified --username value.

- **--status [*vs-name* | all]**
  Prints the status of a specific virtual service or of all virtual services in the current environment.

- --**performance[on | off]**
  Specifies the performance mode for the VSE.

- **--deploy *archive-file***
  Deploys a new virtual service to the current environment. The specified archive file must have a virtual service model as its primary asset.

- **--redeploy *archive-file***
  Redeploys the specified virtual service to the current environment.

- **--update *vs-name* [capacity *capacity*] [--thinkscale *scale*] [--auto-restart [ON|OFF] [--grouptag *group tag*]**
  Updates the capacity, think scale, group tag, auto-restart setting, or any combination of these parameters for the named virtual service.

- **--remove *vs-name***
  Removes the named virtual service from the current environment.

- **--start *vs-name***
  Starts the named virtual service in the current environment.

- **--set-exec-mode *vs-name* --mode [DYNAMIC | VALIDATION | LIVE | TRACK | EFFICIENT | LEARNING | STAND_IN | FAILOVER]**
  Sets the execution mode for the named virtual service in the current environment.

- **--stop *vs-name***
  Stops the named virtual service in the current environment.

- **--help**
  Displays help text.

- **--version**
  Prints the VSE version number.

Example: Deploy Virtual Service

This example deploys a new virtual service.

```
VSEManager --username=admin --password=admin --deploy C:\Weather.mar
```

Example: Check Status

This example prints the status of all virtual services in the current environment.

```
VSEManager --username=admin --password=admin --status all
```

```
Name        Resource            Status   Auto  Capacity  Think  Txns  Errors  Exec
Mode       Up Time  Group
----------  ------------------  -------  ----  --------  -----  ----  ------  ----
----------  -------  -----
Calculator  1165 : http :  : /  Running  Yes   1         100    0     0       Most
Efficient  0:11:21
Weather     1225 : http :  : /  Running  Yes   1         100    0     0       Most
Efficient  0:00:44
```

ServiceManager Command - Manage Services

Use the ServiceManager command-line tool to monitor, reset, and stop DevTest services. The commands apply to any DevTest registry, simulator, or coordinator, and to the VSE server. The Service Manager is included under the bin/ directory in DevTest Server.

For more information, see ServiceManager.

ServiceImageManager Command - Manage Service Images

To import transactions (raw or session) into a new or existing service image, use the VSE ServiceImageManager command-line tool. The tool can also combine two or more service images. The ServiceImageManager is located in the LISA_HOME\bin directory.

Control recording in one of the following ways: interactive, timed, or disconnected.

- **Interactive:**
  Use the interactive style when you specify only the --record argument. When the recorder is ready, it waits for you to click **Enter** at the console before starting the recording process. The recorder then waits for **Enter** again to stop the process.

- **Timed:**
  Use the timed style when you specify the --record and --stop= (and, optionally, the --start=) arguments.

- **Disconnected:**
  Use the disconnected style when you specify the --record and --port= arguments. This sets up a listener that the --signal argument uses to pass control signals to the recorder. After a recorder is initiated with the disconnected style, the ServiceImageManager tool, in a separate process, can be used to control it by sending start and stop signals over the port. This control requires specifying only the --signal= and --port= arguments.

All three styles require the --vrs= and --si-file= arguments. Optionally, for the interactive and disconnected styles, specify --go to skip waiting for a start signal and begin recording immediately.

To import, specify the raw or session traffic transaction file to import after the **--import** argument. To control how the import happens, use the transport, request, and response data protocols and navigation tolerances. To use the **- vrs=** argument with the protocol or tolerance arguments, specify it before the others. The **- vrs=** argument resets parameters to their initial state. To specify the name of the file containing the service image in which to import, use the **--si-file** argument. Specify at least the following arguments:

- **--import=**

- **--si-file=**

- Either **--vrs=** or **--transport=**

You can also specify these arguments:

- **--request-data=**
- **--response-data=**
- Either **--non-leaf=** or **--leaf=**

To combine two or more service images, specify the target service image after the **combine** argument. The file name that **--combine=** defines is the target VSI and other files that are listed are the sources. To control how like transactions combine and list the file names of the source images to combine into the target image, use **favor**. Specify at least the following argument:

--combine=

You can also specify the following argument:

--favor=

List the source files for the combine operation.

> **Note:** Arguments with values that contain spaces (such as the names of protocols) must be enclosed in quotation marks. For example: **"--import=my file.xml"** and **-i"my file.xml"**

The ServiceImageManager command has the following format:

- **--help -h**
  Displays help text.
- **--record -d**
  Records the transactions into a service image file.
- **--vrs=*recording-session-file* -v *recording-session-file***
  The **vrs** parameter specifies the recording session file that contains all the configuration information for the recording or import operation.
- **--go -G**
  The **go** parameter specifies that for interactive or disconnected styles, the wait for a start signal is bypassed.
- **--start=*time-spec* -S *time-spec***
  The **start** parameter indicates that the recorder starts recording after the specified interval.
- **--stop=*time-spec* -E *time-spec***
  The **stop** parameter indicates that the recorder will stop recording after the specified interval. This interval is relative to the time at which the recorder started recording, and is specified in milliseconds.
- **--port=*port-number* -P *port-number***
  The **port** parameter specifies the port to be used for recorder control. When used with **--record**, this command notes the port on which the recorder listens for control. When used with **--signal**, it notes the port to which the start/stop signal is sent.

- **--bind=*bindaddress* -b** *bindaddress*
  The IP address on which to listen.

- **--signal=start|stop -g start|stop**
  The **signal** parameter specifies the signal to send to a recorder in a different process. This command requires the **--port=** argument.

- **--import=*raw/traffic-file* -i *raw/traffic-file***
  Imports the specified raw or session traffic XML document into a service image file

- **--transport=*protocol* -t *protocol***
  The **transport** parameter specifies the transport protocol to use during an import operation.
  Values:

  - HTTP/S
  - IBM MQ Series
  - JMS
  - Standard JMS (Deprecated)
  - Java
  - TCP
  - JDBC (Driver based)
  - CICS LINK <DPL>, DTP <MRO, LU6.1, LU6.2>
  - CICS Transaction Gateway <ECI>
  - IMS Connect
  - SAP RFC via JCo
  - JCo IDoc Protocol
  - Opaque Data Processing

- **--request-data=*protocol* -r *protocol***
  The **request-data** parameter specifies the request-side data protocol to use during an import operation.
  Values:

  - Web Services (SOAP)
  - Web Services (SOAP Headers)
  - Web Services Bridge
  - WS-Security Request
  - Request Data Manager
  - Request Data Copier
  - Auto Hash Transaction Discovery

- Generic XML Payload Parser
- Data De-identifier
- Copybook Data Protocol
- Delimited Text Data Protocol
- Scriptable Data Protocol
- DRDA Data Protocol
- XML Data Protocol
- CICS Copybook Data Protocol
- CTG Copybook Data Protocol
- EDI X12 Data Protocol
- JSON
- SWIFT Data Protocol
- REST Data Protocol

- **--response-data=*protocol* -R *protocol***
  The **response-data** parameter specifies the response-side data protocol to use during an import operation.
  Values:

  - WS-Security Response
  - Data De-identifier
  - Copybook Data Protocol
  - Delimited Text Data Protocol
  - Scriptable Data Protocol
  - DRDA Data Protocol
  - CICS Copybook Data Protocol
  - CTG Copybook Data Protocol
  - JSON
  - SWIFT Data Protocol

- **--non-leaf=CLOSE | WIDE | LOOSE -n CLOSE | WIDE | LOOSE -n *tolerance***
  The **non-leaf** parameter specifies the default navigation tolerance to assume for any non-leaf conversation nodes created.
  Default: WIDE

- **--leaf=CLOSE | WIDE | LOOSE -l CLOSE | WIDE | LOOSE -l *tolerance***
  The **leaf** parameter specifies the default navigation tolerance to assume for any leaf conversation nodes created.
  Default: LOOSE

- **--config=*config-file* -C *config-file***
  In Recording mode, specifies a configuration file to use.

- **--si-file=*vsi-file* -s *vsi-file***
  The **si-file** parameter specifies the name of the service image file to which to import transactions. If this file does not exist, the application creates it. Otherwise, the imported transactions are merged with the existing service image.

- **--vsm_file=*vsm-file* -m *vsm_file***
  The **vsm_file** parameter specifies the name of the virtual service model file to create during a recording. The vsm file needs to be created within a DevTest project folder.

- **--combine=*target-vsi-file* -c *target-vsi-file***
  Combines one or more service image files into the named service image file. Unless the **favor** argument is specified, the source service images are favored.

- **--favor=source | target -f source | target**
  The **favor** parameter specifies how to combine like transactions when combining service images.
  Values:

  - **source:** Like transactions (and other data) update the target side from the source side.

  - **target:** Like transactions (and other data) leave the target side unchanged.

- **--version**
  The **version** parameter prints the VSE version number.

The ServiceImageManager command has two error codes:

1 - If an argument exception occurs (exit status bad param).

2 - If other general failure occurs.

Examples

ServiceImageManager has broad functionality, so review the documentation to find the proper arguments. Here are few scenarios that ServiceImageManager can be used for:

Making a Recording

```
ServiceImageManager  -d -v c:/myRecording.vrs -C c:/recording.config  -s c:
/generatedImage.vsi  -m c:/generatedModel.vsm -P 3366 -G
```

This command means:

- **-d**
  Do a recording.

- **-v**
  Build a VRS file (recording configuration file), This file can be generated from the VS Recorder.

- **-c**
  Use a configuration file (optional).

- **-s/-m**
  Name the generated assets.

- **-P**
  Designate the port that the service Image listens to for a command (not the recording port, which is given in the VRS file).

- **-G**
  Start recording now.

After you issue this command, the recorder will be running and you can execute the transaction. To stop the recording, run ServiceImageManager again with the following arguments:

```
ServiceImageManager.exe -g stop –P 3366
```

This command means:

- **-g**
  The action to take: in this example, "stop." Another option would be to remove the **–g** from the argument and start recording later by executing **–g start** to start recording.

- **-P**
  Designate the port that the service Image listens to for a command (not the recording port, which is given in the VRS file).

All the recording configurations such as port, host, and DPH are configured from the VRS file that we generated from the VSE recorder.

Generate a VSI/VSM from a Raw Traffic File

Use the same command except remove the **–d** and add **–I**. For example:

```
ServiceImageManager -v c:/myRecording.vrs -C c:/recording.config  -s c:
/generatedImage.vsi  -m c:/generatedModel.vsm –i c:/rawTarffic.xml
```

Import Raw Traffic Without a VRS File

```
ServiceImageManager.exe -i c:/rawTarffic.xml. -s c:/generatedImage.vsi  -m c:
/generatedModel.vsm -t “HTTP/S” -r "JSON"
```

This command imports the raw traffic and processes it as HTTP/S data. The command then processes the data as JSON (JSON DPH). The model that is generated is an HTTP model with a JSON DPH.

VirtualServiceEnvironment Commands

Use the VirtualServiceEnvironment command to manage the VSE application. The executable is located in the [LISA_HOME]\bin directory.

This command has the following format:

```
VirtualServiceEnvironment [--help|-h] [--name=name|-n=name] [--registry=registry-
spec|-m=registry-spec] [--labName=lab-name|-l=lab-name] [--port=port-number|-P=por
t-number] [--version]
```

- **--help -h**
  Displays help text.

- **--name=*name* -n *name***
  Defines the service name for the environment server.
  Default: The system property **lisa.vseName**. The default value for the system property is **VSEServer**.

- **--registry=*registry-spec* -m *registry-spec***
  The registry to which to connect.
  Example:

  ```
  -m=tcp://localhost:2010/registry1
  ```

- **--labName=*lab-name* -l *lab-name***
  Specifies the name of the lab to use.
  Default: Default

- **--port=*port-number* -P *port-number***
  Specifies the service port to which the VSE publishes. It is the same as specifying the port as part of the **name** argument.

- **--username=username -u username**
  Specifies the DevTest security username.

- **--password=password -p password**
  Specifies the DevTest security password.

- **--version**
  Prints the VSE version number.

Java Agent VSE Properties

The configuration properties for the DevTest Java Agent include some properties that apply to CA Service Virtualization.

You can configure these properties from the Agents window of DevTest Portal. The properties appear in the **Settings** tab.

- **Startup mode**
  Specifies the virtualization mode at startup>
  Values:

  - **Passthrough**

  - **Recording**

  - **Playback**

- **Shallow recording**
  Specifies to perform callbacks only on top level (in the stack frames) virtualized methods.

- **Maximum graph size**
  Defines the maximum size (in bytes) of a serialized-to-XML object graph.

- **Reply timeout**
  Specifies the maximum interval that is allowed for a VSE reply at playback. This value should exceed any think time in your tests.

- **Omit by reference**
  Specifies whether to record or replay arguments that reference and void methods modify.
  Values:

  - **Selected:** Do not record or replay arguments that reference and void methods modify. Select this check box if you are not virtualizing any void methods that change the arguments state. No void methods go to VSE and return immediately to enhance performance.

  - **Cleared:** Record or replay arguments the reference methods and void methods modify.

- **Cache responses**
  Specifies whether to cache response objects key by their string representation to bypass unmarshalling.

  - **Values:**

  - **Selected:** Cache response objects key by their string representation. When enabled, this property still causes a request to VSE. When the XML payload is received back in the agent, instead of deserializing it, we try to look for the object in a cache. The key is essentially an XML representation of the response. This setting is helpful to improve performance if responses are not in a mutable state. Selecting this property can increase memory usage.

  - **Cleared:** Do not cache response objects key by their string representation.

- **Enable JIT**
  VSE jit means model/image logic is partially (or entirely) cached in the agent.

- **JIT threshold**
  Defines how many method invocations cause VSE jitting to occur.

Using Application Insight

The Application Insight feature captures live application traffic and uses this data to create visual paths that let you drill down into and analyze business transactions and architecture.

This section describes how to view and manipulate the application data paths, automate the creation and maintenance of virtual services, and simplify the production and management of test cases and suites.

- Getting Started with Application Insight
- Agent Configuration
- Analyzing Business Transactions
- Using the Shelf
- Create and Manage Tickets
- Creating Virtual Services
- Creating Baselines
- Create Data Sets
- Create Request/Response Pairs from Transaction Frames
- Document Transactions
- Native MQ Application Insight
- Application Insight Command-Line Tool

- VS Traffic to CA Application Insight Companion
- DevTest Extension for Google Chrome
- CAI Top N and Metrics Reports
- Application Insight for CICS Walkthrough
- Integrate with CA API Management

Getting Started with Application Insight

The Application Insight feature is designed to enhance the process of delivering applications to market.

The following roles are the intended users:

- Architects
- Developers
- Quality assurance engineers
- Data architects
- Release engineers

Application Insight captures the data that flows through an application. Application Insight uses this data to generate paths that can be viewed in DevTest Portal. From DevTest Portal, users can perform the following tasks:

- Drill down into business transactions and analyze abnormal behavior
- Generate architecture diagrams for cross-functional team collaboration
- Generate baseline test cases for regression testing
- Generate virtual services
- Isolate a defective component and generate the test cases and virtual services for reproducing the defect

Users can also create defect tickets from the user interface of an application. The tickets can be managed in DevTest Portal.

To view a tutorial, see Application Insight Tutorial.

Application Insight Prerequisites

To review the system requirements for Application Insight and other DevTest components, see System Requirements.

You can use the following methods to add transactions to the Application Insight database:

- Install and configure the DevTest Java Agent for a Java-based application
- Run the DevTest extension for Google Chrome
- Run Native MQ Application Insight

The DevTest Java Agent is installed by default in the following demo applications:

- Forward Cars
- LISA Bank

Once you add transactions to the database, you can view and analyze them in DevTest Portal.

DevTest Portal can monitor agents from the same release or from an older release. DevTest Portal cannot monitor agents from a newer release.

If the agents and the broker are on different computers, ensure that the system times are synchronized. Otherwise, unexpected behavior can occur. For example, it can take a long time to refresh the list of agents in DevTest Portal.

Large Payloads

If you want the agent to capture large payloads in a short period of time, be sure to increase the following settings:

- Java heap size of the broker
- Maximum buffer memory

For information about how to update the Java heap size of the broker, see Memory Settings.

The **Maximum buffer memory** property is a broker property that you can configure from the **Agents** window of DevTest Portal. The name of this property in the **rules.xml** file is **lisa.broker.transaction.max.buffer.mem**.

An example of a large payload is 15 MB.

Without these changes, you might receive a **java.lang.OutOfMemoryError** exception in the broker.

Application Insight Supported Platforms

Application Insight is supported on the following platforms:

- IBM WebSphere Application Server 7.0
- IBM WebSphere Application Server 8.5
- JBoss Application Server 4.2
- JBoss Application Server 7.3
- WildFly 8.2
- JBoss Fuse 6.2
- Jetty 8
- Jetty 9.1
- Jetty 9.2

- Oracle WebLogic Server 10.3
- Oracle WebLogic Server 12.1.1
- SAP NetWeaver 7.3
- SAP NetWeaver 7.4
- TIBCO BusinessWorks 5.*x*
- TIBCO Enterprise Message Service 6.3
- webMethods Integration Server 9.0
- webMethods Integration Server 9.5
- webMethods Integration Server 9.6

For a reference to various components and features of Application Insight and when they are supported, see Application Insight Support Matrix.

Mule ESB Support

The DevTest Java Agent can capture transactions that pass through Mule ESB.

Place the agent on both the client side and the server side.

On the client side, set the capture level of the HTTP Client protocol to **Full Data**.

On the server side, set the capture level of the RMI and HTTP Server protocols to **Full Data**.

Download DevTest Java Agent Files

From the DevTest Portal home page, you can download a zip file of all the DevTest Java Agent files. Downloading the files from the UI is a convenient and secure way to transfer files between systems.

Note: Some browsers or corporate security can block the download. Set the browser to allow pop-ups or accept the download when prompted.

To download the **agent.zip** file, click the down arrow next to your user name and select **Download Agent.**

More Information:

- Java Agent Installation
- Install the Pure Java Agent
- Install the Native Agent

Business Transactions and Transaction Frames

Business transactions and transaction frames are key concepts in Application Insight.

A *business transaction* is a representation of how a user request is serviced by an application. Each transaction encompasses a code path in which one or more servers are run as the result of a client request.

A *transaction frame* encapsulates data about a method call that the DevTest Java Agent intercepted. The data includes such information as:

- The name of the method
- The name of the class to which the method belongs
- The arguments that were passed to the method
- The value that the method returned
- Log4j messages that are written
- Exceptions that are thrown by the Java classes

Each transaction frame has a unique identifier and a category. Typically, the category represents a protocol or operating environment. Examples include EJB, JDBC, JMS, REST, SOAP, and WebSphere MQ. A transaction frame is a node in the transaction path. Each node represents a step in the overall business transaction and the code or business logic that executed. A collection of transaction frames shows the order of the code that executed to fulfill the business transactions.

Each transaction contains a hierarchical set of transaction frames. The *root transaction frame* is the top-level frame in the hierarchy. Each transaction has a unique identifier.

Transactions are also referred to as *paths*. A *transaction path* is a visual representation that shows the flow of how the user request is serviced. For example, the path reveals all the front-end and back-end services that a transaction contacts and the relationship between the services.

The following graphic displays a transaction path in DevTest Portal.

Screen capture of transaction path

SAP NetWeaver Notes for Application Insight

The following protocols are supported for viewing paths, generating baselines, and generating virtual services:

- EJB (remote interface of session beans only)
- JMS
- REST
- Web service

The DevTest Java Agent includes the **transaction.sap.initial.context.factory** property. This property defines the initial context that provides client access to the JNDI Registry Service of AS Java as a name service provider. The default value is **com.sap.engine.services.jndi. InitialContextFactoryImpl**. To change the value, add the property to the agent element of the **rules.xml** file. For example:

```
<agent guid="0" name="A1">
   <property comment="xxx" key="transaction.sap.initial.context.factory", "value="
xxx"/>
</agent>
```

The JNDI URLs for SAP J2EE containers are captured with a port of 50004, which is the default port for the P4 protocol. If the application is using a nondefault port, you might need to change the baseline test to reflect the nondefault port.

Artifact generation for JNDI-based protocols is not supported with the new lookup format, which uses key-value pairs (for example, **ejb:/key1=value1, key2=value2...**). The lookup format that uses the full JNDI name is supported.

For the EJB protocol, the IP address that is captured is the SAP internal server IP address. If you perform tests from an external server using an external public IP address, you might need to change the baseline test configuration in DevTest Workstation before you run the test to reflect this IP address.

When you capture web service traffic, the server URL includes the string **localhost** instead of the computer name or IP address. Before you run a generated baseline or virtual service, open the baseline or virtual service and locate the occurrence of **localhost**. The string might be defined as a property in the active configuration. Change the string **localhost** to the computer name or IP address.

Do not set the capture level for the RMI protocol to Full Data. Setting the capture level to Full Data can cause performance issues.

Agent Configuration

Perform agent configuration tasks like setting capture levels and upgrading an agent from the **Agents** window in DevTest Portal. To open the **Agents** window, select **Settings, Agents** from the left navigation menu.

The **Agents** window contains the following components:

- **Agents pane**
  This pane contains a broker and one or more agents. The broker appears at the top. The broker and each agent displays the current status. You enable or disable the transaction capture, delete offline agents, and upgrade agents to the latest version from this pane. Agent groups can be added, configured, and managed. When a broker and agent are different versions, the hat icon and version number of the agent is orange. When you select an agent or broker, the right pane of the **Agents** window displays detailed information.

- **JVM Performance pane**
  The performance details display at the top, right pane.

- **Adjust Captures for Protocols pane**
  This pane contains the protocols for agent or group that is capturing data. When a group is selected in the **Agents** pane, the performance and count information do not display. This pane is not available when the broker is selected in the **Agents** pane.

- **Information tab**
  This tab contains detailed information about the broker or agent.

- **Settings tab**
  This tab contains the configuration properties for the broker, agent, or group.

CICS Agents

The **Agents** window contains the following components for CICS agents:

- **Agents pane**
  This pane contains a broker and one or more agents. The broker and each agent displays the current status. The broker appears at the top. You enable or disable the transaction capture, delete offline agents, and upgrade agents to the latest version from this pane. When you select an agent or broker, the right pane of the **Agents** window displays detailed information.

- **Agent Properties pane**
  This pane contains the configuration properties for the CICS agent.

- **Virtualized Programs pane**
  This pane displays all the transactions for example, CICS LINK and DTP that are deployed to the CICS agent.

More Information:

- View the agent status
- View and change the VSE mode
- Filter agents
- Stop and start agent transaction capture
- View performance data
- Configure the capture levels for an agent
- View agent and broker information

- View broker notifications
- View and update configuration properties
- Upgrade an agent
- Delete an offline agent
- Create and manage agent groups

Agent Status Indicator

In the left pane of the **Agents** window, each agent has a flag to indicate the status information.

The following list describes the possible states:

- **Green:** The agent has no issues.
- **Yellow:** Exception thrashing or flooding.
- **Orange:** CPU or garbage collection thrashing.
- **Blink between orange and red:** Heap or perm gen exhaustion.
- **Red:** JVM alarm state or deadlock.
- **Gray:** The agent is offline.

The agent log files can provide more information about why the status has become yellow, orange, or red.

VSE Mode Indicator

In the left pane of the **Agents** window, each agent has an indicator for the status of the VSE functionality.

The following list describes the possible states:

- The agent is in passthrough mode.
- The agent is in record mode.
- The agent is in playback mode.
- The agent is in validate mode.

To view the VSE mode, click **Show more** «. To change the state, click the down arrow next to **VSE Mode** and select an option.

Note: VSE mode is not supported for CICS agents.

Filter Agents

Use filter options to narrow down the list of agents that display in the **Agents** window. From the filter pane, sort and filter agents and specify search text criteria.

Agents are sorted and filtered using the following options:

- Name
- Groups
- Status
- Type (not available for filtering agents)
- Monitoring
- VSE Mode

Follow these steps:

1. Select **Settings, Agents** from the left navigation menu.
   The **Agents** window opens with the **Agents** pane in the left pane. By default, the filter pane is hidden.
2. Click **View filters** .
   The filter pane opens.
3. Click **Turn on filters** to enable sort and filter criteria.
4. To sort agents, select an option from the **Sort by** list.
5. To filter agents, select an option from the **Filter by** list.
6. Use the up and down arrows to list the agents in ascending or descending order in the left pane.
7. To search agents by name, enter the search criteria in the **Enter Search Text** field.
8. (Optional) To add more filters, click **Add Filter Type** .
   An additional filter displays in the filter pane.
9. To remove a filter, click **Remove Filter Type** .
10. To refresh the list of agents that display, click **Refresh**
11. To reset the filters to the default settings, click .

12. To hide the filter, click **Hide filters** .

Stop and Start Agent Transaction Capture

The **Agents** window lets you control whether an agent is capturing transactions.

Use the camera toggle button to start and stop agents from capturing transactions. When the camera button is blue, the transaction capture is enabled for the agent. A grey camera button indicates that the transactions capture is disabled.

> **Note:** CICS agent capture is enabled from the Agents Properties pane in the Agents Window. See Configure Capture Levels for Agents to set the CICS agent capture level.

Follow these steps:

1. Select **Settings, Agents** in the left navigation menu.
   The **Agents** window displays.
2. Click **Show more** .
3. To start the agent transaction capture, click to start capturing data.
4. To stop the agent transaction capture, click to stop capturing data.

View Performance Data

You can view a set of performance graphs for an agent or broker.

> **Note:** Performance data is not available for some agents for example, CICS agents.

The graphs display the following information:

- CPU usage
- Memory usage
- I/O throughput
- Transactions per second

The I/O throughput graph is not applicable to the broker.

The following graphic displays the performance graphs for an agent.

Screen capture of agent performance graphs.

Follow these steps:

1. Select **Settings, Agents** in the left navigation menu.
   The **Agents** window opens.

2. In the **Agents** pane, select an agent or broker.
   The performance data displays in the right pane.

3. (Optional) Click a performance graph to view a legend.

4. To turn off the polling or auto refresh, clear the **Auto Refresh** check box.

View CICS Virtualized Programs

The Virtualized Programs pane displays all the transactions that are deployed to the CICS agent. For example, transactions can be CICS LINK and Distributed Transaction Processing (DTP). The virtualized programs are listed with the link and system ID, program name, type of transaction, and the mode information.

Programs that are orphaned by the agent can be removed from the CICS agent.

Follow these steps:

1. Select **Settings, Agents** from the left navigation menu.
   The Agents window opens.

2. Select a CICS agent in the left pane of the **Agents** window
   The Virtualized Programs pane displays in the right pane.

3. To remove an orphaned program, select the checkbox of the program and click **Reset program**.

Configure Capture Levels for Agents

Each protocol that the DevTest Java Agent can capture has a *capture level*.

The following capture levels are available:

- **Counts**
  The agent collects only count information. This level has the least impact on the system under test.

- **Counts and Paths**
  The agent collects transaction frames, but not full requests or responses.

- **Full Data**
  The agent collects all data. This level has the most impact on the system under test.

The default capture level is **Counts**.

The following graphic shows the **Adjust Captures for Protocol** pane, which lists the protocols and their capture levels.

Screen capture of Adjust Captures for Protocol pane

We recommend the following workflow:

1. Start with all the protocols in **Counts** mode.

2. Exercise the application and view the number of counts.

3. Decide which protocols you are interested in and increase the capture levels accordingly.

If you want to create tickets, ensure that the capture level for the **HTTP Server** protocol is set to **Full Data**.

If you want to create baselines or virtual services, ensure that the capture levels for the appropriate protocols are set to **Full Data**.

If the count is more than one thousand, the text **999+** displays. The tooltip for the icon shows the exact number.

> **Note:** Setting different capture levels is not supported for queue-based client and server communication, for example, WebSphere MQ and JMS.

Follow these steps:

1. Select **Settings, Agents** in the left navigation menu.
   The **Agents** window opens.

2. Select an agent or group in the **Agents** pane.

3. Select the check box for one or more protocols. Selecting more than one protocol lets you change the same capture level for several protocols at once. To select all protocols, select the **Select All** check box.

4. Select a capture level from the **Select a capture level** drop-down list.

5. To switch between displaying all protocols and displaying only the protocols whose count is greater than 0, select the **Show counts only** check box.

6. To reset all the counts to 0, click **Reset All Counts** .

7. To refresh the counts and capture levels, select the **Auto Refresh** check box.

View Agent and Broker Information

The following categories of information about an agent or broker are available from the **Agents pane, Information** tab:

- **VM Information**
  The **VM Information** tab displays basic information about the agent and the virtual machine in which it is running. The values are read only.
  If a broker is selected, the agent name is the user name, followed by an ampersand, followed by the computer name.
  The main class is the Java class that contains the main method that was invoked for the agent.
  The PID is the process ID in which the agent is running.

- **VM Properties**
  The **VM Properties** tab displays the system properties and the environment variables. The values are read only.
  The information cannot be displayed for an agent that is offline.

- **Environment Variables**
  The **Environment Variables** tab displays the system environment variables on the computer where the agent is running.

Follow these steps:

1. Select **Settings, Agents** in the left navigation menu.
   The **Agents** window displays.

2. In the **Agents** pane, select an agent or broker.

3. Select the **Information** tab from the right pane.

View Broker Notifications

The **Agents** window displays notifications about brokers that have critical error messages. The messages display in a table from the **Notifications** pane. You can set the total number of messages to display in the **Show Total** field. By default, the number of messages is set to 100. The maximum number of messages that can display is 500. Search the table by time and message.

Follow these steps:

1. Select **Settings, Agents** from the left navigation menu.
   The **Agents** window opens.

2. Select a broker from the left pane.

3. From the Notifications pane, click **Refresh** to get the latest message count.

4. (Optional) Select the **Auto Refresh** check box to refresh automatically the notification information every ten seconds.

5. Click **Expand Panel** to view the details of the notifications.

6. (Optional) Use the **Time** and **Message** search field to find specific error messages.

7. (Optional) Edit the value in the **Show Total** field by entering a number from one to 500. Use the arrows to increase or decrease the number in increments of ten.

View and Update Properties

Configuration properties for an agent or broker are viewed and updated from the Agents window.

The properties are grouped into categories.

Follow these steps:

1. Select **Settings, Agents** in the left navigation menu.
   The **Agents** window displays.

2. In the left portion, select an agent or broker.

3. Click the **Settings** tab.

4. Change the value of one or more properties.

5. To discard any changes that have not saved, click **Revert**.

6. To save the changes, click **Save**.

CICS Agent Properties

You can view the properties for a CICS agent. The properties are read-only and are grouped into the following categories:

- CICS Info
  Configuration information from the CICS region the agent is running in.

- EIB
  CICS agent (CICS transaction) Execute Interface Block configuration information.

- GWA
  CICS agent Global Work Area. This area is used to communicate with the various CICS exits used by the agent.

- General
  Information about the version, number of specific table entries, transaction ID, and task ID for an agent.

Follow these steps:

1. Select **Settings, Agents** in the left navigation menu.
   The **Agents** window displays.
2. (Optional) Select the **Auto Refresh** check box.
3. In the left portion, select a CICS agent.
4. Select a property in the **Agent Properties** pane and review the information.

Configure Agent Logging

The Java agent log files are configured from DevTest Portal.

Two categories of configuration properties appear in the Portal:

- **Logging**
  Logging statements that an application produces and that the DevTest Java Agent captures.
- **Level**
  Logging statements that the DevTest Java Agent produces.

Follow these steps:

1. Select **Settings, Agents** in the left navigation menu.
   The **Agents** window displays.
2. In the left portion, select an agent, agent group, or broker.
3. Click the **Settings** tab.
4. Select the **Logging** category.
5. Configure one or more of the following settings:
   - **Maximum log size**
     Sets the maximum size of an agent or broker log file before it gets archived. You can set the unit of measurement to GB, MB, or KB.
   - **Maximum log archives**
     Sets the maximum number of archived log files to keep.
   - **Java logging level**
     Sets the minimum level at which logging statements are captured. This setting is not applicable to the broker. For more information about the levels, see the Java API documentation for the **java.util.logging.Level** class.
   - **Agent log**
     Sets the location of the agent or broker log files.
6. Expand the **Logging** category and select **Level**.

7. Configure the logging level for one or more agent components.

8. Save the changes.

Manage Java Intercepts

Java intercepts are managed from the Intercepts tab in the Agents window.

The following graphic shows the Intercepts tab:

Screen capture of Intercepts tab

Use the search box in the left pane to filter the list of interfaces and classes.

> **Note:** When entering the filter criteria in the search box, you must type a minimum of three letters.

When an interface or class is selected in the left pane, the right pane displays the methods.

A blue icon in the Action column indicates that the agent is intercepting a method. The icon is gray when the agent is not intercepting a method.

To enable or disable the interception of a method, click the icon in the Action column.

To view the number of classes that are capturing interceptions, select the **Show counts of capture enabled classes** checkbox. Click the folder in the left pane to display the count.

De-identifying Data in Application Insight

Application Insight provides support for *de-identification*, which is the replacement of sensitive data with alternate values.

Examples of sensitive data include:

- Credit card numbers
- Social Security numbers
- Email addresses

By default, de-identification is disabled.

You can enable this feature from DevTest Portal. Open the **Agents** window and select an agent. In the **Settings** tab, go to the **General** section and select the **De-identify Transaction Data** property.

The **de-identify.xml** file in the **LISA_HOME** directory lets you configure what qualifies as sensitive data. For more information, see De-identifying Data.

> **Note:** These operations are CPU intensive. Avoid using this feature on an agent that has high traffic or large payloads.

When assembling transactions, the broker checks the de-identification property for each agent. If the value is true, the replacement is performed and the sensitive data is not saved to the database.

Be aware of the following limitations:

- With dummy data in the transaction request, baselines might not be able to reproduce the exact test step.
- With dummy data in the transaction response, baseline assertions might fail.
- With dummy data in the transaction request, the virtual service environment (VSE) might not be able to find the matched response if you re-run the same test steps. As a workaround, open the service image in DevTest Workstation and set the match style to **Signature**.
- With dummy data in the transaction response, the VSE might return a response that is different from the original response.

Upgrade an Agent in DevTest Portal

From the **Agents** window, you can upgrade agents to the latest version in DevTest Portal. When a broker and agent are different versions, the hat icon and version number of the agent is orange. The button appears next to the agent when a new version is available. Move the pointer over to see the version information.

> **Note:** Not all agents can be upgraded in DevTest Portal. Native agents and agents using JRE 1.5 and **LisaAgent2.jar** must be manually upgraded.

Only pure agents that have the following requirements can be upgraded:

- JRE 1.6 or later

- Agent must be started from the **InsightAgent.jar** file.

Follow these steps:

1. Save the latest **LisaAgent.jar** file to the **LISA_HOME\lib\core\agentupgrade** folder.
2. Select **Settings, Agents** from the left navigation menu.
   The Agents window opens.
3. From the Agents pane, click .
   The following icon displays when the update is successful:
4. Restart the agent to complete the upgrade process.

Upgrade an Agent Manually

If an agent is not eligible to be upgraded in DevTest Portal, use this procedure to upgrade the agent.

Follow these steps:

1. Stop the agent.
2. Replace the existing **LisaAgent.jar** file with the new version.
3. Start the agent.

Delete an Offline Agent

You delete an offline agent from the **Agents** pane of the **Agents** window. An offline agent has a gray status flag. When an agent is deleted, all transactions and frames of the agent are removed from the database. The transactions and frames do not display in Analyze Transactions and in the shelf.

Follow these steps:

1. Select **Settings**, **Agents** from the left navigation menu.
   The **Agents** window displays.
2. (Optional) To search for an agent, click **View filters** and click **Turn on filters** to use the filter criteria.
3. Right-click the offline agent and select **Delete**.
   A confirmation dialog displays.
4. Click **OK**.

Create and Manage Agent Groups

From the **Agents** window, you create agent groups and apply settings to configure multiple agents simultaneously. Agent groups are added, configured, and managed in the agents pane. Each group displays a count of all agents that are in the group.

> **Note**: The mainframe group is automatically created for CICS agents. Group settings are not available for the group. CICS agents cannot be removed from the mainframe group.

- Create Agent Group
- Rename Agent Group
- Delete Agent Group
- Move Agent to a Group
- Remove Agent from a Group
- Copy Agent and Group Settings

Create Agent Group

Use the agents pane to create an agent group. A group can have multiple agents or no agents.

To add a group, click **Add group** , enter a name for the group, and click . The group is added in the agents pane.

After you create a group, you add agents and apply settings from the agent or agent group.

Rename Agent Group

The name of an agent group can be changed except the default group.

Follow these steps:

1. Right-click the agent group and select **Rename**.
2. Enter a name and click .

Delete Agent Group

Agent groups can be deleted except the default group. Selecting a group to delete automatically selects all the agents within that group. The agents within the group are not deleted but moved into the default group. See Delete an Offline Agent for information about deleting an offline agent.

Follow these steps:

1. Select the check box for the group or multiple groups to delete.
   The Actions pane displays.
2. Select **Delete Group**.
3. When a group is selected and one or more agents from another group are selected, select **Delete Group and/or Remove Agent from Group**.
4. When one or many agents from different groups are selected, select **Remove Agent from Group**.
5. Click **OK**.

6. Click **Continue**.
   The group is removed.

Move Agent to a Group

An agent can only belong to one group. If an agent is not defined to a specific group, the agent is listed within the default group. When agents are added to a group, the group-level settings are applied by default. The group or agent settings can be applied to the agent. You move an agent to a group or arrange an agent within a group using the drag-and-drop method. Use the Actions pane to move multiple agents to a group.

Follow these steps:

1. To move an agent, select and drag the agent into a group.

2. To move multiple agents, perform the following steps:

   a. Select the check box of multiple agents.
      The Actions pane displays.

   b. Select **Move Agents**, select a group from the list, and click **OK**.
      A warning dialog displays.

   c. Click **Continue**.
      The agent is added to the group and displays under the group in the Agents pane.

Remove Agent from a Group

Using the Actions pane, you can remove an agent from a group that contains more than one agent. The agents that are defined within the group are moved to the default agent group automatically.

1. Select the check box for one or more agents.
   The Actions pane displays.

2. Select **Remove Agent From Group** and click **OK.**
   A warning dialog displays.

3. Click **Continue.**

Copy Agent and Group Settings

You can copy the settings of an agent or a group. The settings can be applied to a specific agent or a group. Any changes to the configuration settings are applied to the agent or the group.

> **Note:** When an offline agent is restarted, the agent settings are reset to the group settings.

Follow these steps:

1. To copy settings to an agent or group, right-click the agent or group and select **Copy Settings**.

2. To apply settings from an agent or group, right-click the group or agent and select **Apply Settings.**

3. To use agent or group settings when agents are moved into a new group, click the down arrow from **Settings** and select an option:

   - **Use Group Settings:** The group settings are used.

   - **Use Agent Settings**: The agent settings are used.

Analyzing Business Transactions

The **Analyze Transactions** window in DevTest Portal displays the business transactions that have been captured. These transactions are also referred to as *paths*.

The information that you can view for the transactions includes the execution times, thread utilization, log messages, SQL statements, request and response data, and stack traces.

The top of the **Analyze Transactions** window indicates the number of transactions that occurred. The **Results** field displays the number of transactions during an initial query.

> **Note:** Frames for each transaction follow a specific execution order and are not arranged by a particular agent.

Graphical View and List View

The **Analyze Transactions** window provides a graphical view and a list view.

When you open the window, the list view is displayed. To switch to the graphical view, click .

Graphical view is a visual representation of the transactions. This view displays up to a maximum of 100 transactions. You can do the following operations:

- Search transactions by text, time, and date
- Filter paths by using refinement options
- Exclude data from capture
- View the transaction details
- Pin and unpin transactions
- Share points of interest
- Share a link to a transaction
- Export a path diagram to a PDF file

- Export and import paths
- View paths for transactions that have identical transaction paths
- Shelve transaction frames
- Generate all artifacts for an agent

To view a portion of a large transaction at a time, click **Show outline navigation view** .

In the list view, the transactions appear in a table with the name, start time, wall time, CPU time, and agent name. You can do the following operations:

- Search transactions by text, time, and date
- Filter paths by using refinement options
- View information about a path. For example, the path and agent name, start time, wall time, and CPU time
- View the transaction details
- Export and import paths
- View paths for transactions that have identical paths
- Shelve transaction frames
- Generate all artifacts for an agent

Search and Filter Transactions

The **Analyze Transactions** window contains intelligent search and filter refinements to find the transactions frames you want to analyze. The search and filter refinements are available in the search bar and **Refine By** pane in the list or graphical view.

The following graphic displays the search bar:

Screen capture of the search bar

Enter text strings and select a time period to limit the number of transactions frames that display. The default time period is **1 Day**.

Use the following procedures to search transactions:

- Intelligent Search for Transactions
- Search for Transactions by Time

Click to display the **Refine By** pane to use the filtering options. The following graphic displays the **Refine By** pane:

Screen capture of Refine By pane

The filtering options that are available are based on the frame types that display in the list or graphical view.

When a filter is applied, the following refinement icon displays:

Screen capture of refinement filter with filter applied

Filter transactions using the following procedures:

- Filter Transactions
- Filter Frames Within Transactions
- Filter by Starting and Ending Transactions

Intelligent Search for Transactions

Use natural language processing (NLP) keywords to find transactions. Enter a payload text search from the **Enter Search Text** field. Use a payload search string for example, **response has lisa_simpson and category has EJB**.

You can use other keywords in the search. For example, enter a transaction ID, a category, and a local or remote IP address.

The following graphic displays the search bar in the **Analyze Transactions** window:

The following words or parts of speech can be used when forming your search phrase:

Noun, singular (n)	Verb (v)	Conjunction (c)	Examples
agent name, category, class name, exec time, local IP, remote IP, response, request, transaction id, transaction time, point of interest, operation, tags	is, has, have, include, includes	and, or	n = JBoss_LISA_Bank n + v + n = category is Java n + v + n + c + n= point of interest has EJB and Java

Follow these steps:

1. Select **Application Insight, Analyze Transactions** in the left navigation menu.
2. Enter the text search criteria in the search text field and click **Search** .
   A list of paths display that is based on the search criteria.
3. (Optional) To refine your search, click and select options or enter fields in the **Refine by** pane.

Search for Transactions by Time

The **Analyze Transactions** window can contain hundreds of transaction paths. You can narrow down the paths by filtering the search for transactions by time and date. The results of the search are listed up to 25 transactions at a time.

The following time-range options are available:

- Minutes (1 through 30)
- Hours (1 through 12)
- Days (1 through 15)
- All Time

The following date-range options are available in **Advanced** search:

- Start date and time
- End date and time

The default date range is **1 Day**.

Follow these steps:

1. Select **Application Insight, Analyze Transactions** from the left navigation menu.
2. Click

.
3. Select a time-range option.
4. For an advance search, perform the following steps:
   a. Click **Advanced**.
   b. Enter or select from the options and click **Apply**.

Filter Transactions

Use the filter options to narrow down the number of transactions that appear in the **Analyze Transactions** window. Enter a filter option from the search text fields or select the check boxes. For example, if you know the class name to filter, select the corresponding **Class Name** check box. Multiple filter options can be selected. The **Analyze Transactions** window returns transactions containing that class name.

The following filter options are available:

- Agent
- Category
- Class Name
- Execution Time

- Local IP
- Operation
- Point of Interest
- Remote IP
- Tags
- Transaction ID

Follow these steps:

1. Select Application Insight, Analyze Transactions from the left navigation menu.
   The **Analyze Transactions** window displays in list view by default.
2. Click **Refine transaction search result by applying filters** in the list or graphical view to open the **Refine By** pane.
3. Select the appropriate check box, select **Select All**, or enter text.
   The **Analyze Transactions** window returns transactions for the specified filter option.
4. To clear selections for a filter option, click **Clear** for the corresponding filter.
5. To clear all filter options, click **Clear All.**
6. Click **X** to close the **Refine By** pane.

Filter Frames Within Transactions

The **Analyze Transactions** window lets you filter the frames that occur within transactions.

For example, the following graphic displays paths that contain EJB frames:

Screen capture of filter frames within transactions

You can remove the EJB calls from the path by clicking **Filtering options** and clearing the **EJB** check box.

The following graphic displays the filtering options:

Screen capture of filtering options

The following graphic displays the transactions without the EJB frames:

Screen capture of filtered transaction with out EJB frames

The dotted lines indicate that the EJB frames occurred before the SQL activity.

If you filter the JDBC frames for the same transactions, only the calls that occurred before the JDBC frames display. The following graphic displays the transactions without the JDBC frames:

Screen capture of transactions without JDBC frames.

Follow these steps:

1. Select Application Insight, Analyze Transactions from the left navigation menu.

2. Click **Graphical** .
   The transactions display in the graphical view.

3. Click **Filtering Options** .

4. Select or clear the appropriate category check box.
   The transaction paths display the frames according to the selected or cleared category.

Filter by Starting and Ending Transactions

The **Analyze Transactions** window lets you filter the transactions by setting a start transaction and an end transaction.

In the list view, you drag-and-drop the start and end transactions into the **Refine By** pane.

In the graphical view, use the right-click menu options.

The refinement settings are also applied when the **Shelve All Occurred Transactions** option is selected for the transactions. The start and end transactions filter is applied in the shelf when creating a baseline.

Follow these steps:

1. Select **Application Insight, Analyze Transactions** from the left navigation menu.
   The **Analyze Transactions** window displays in the list view by default.

2. Perform one of the following operations:

- In the list view, right-click a transaction and select **Set Starting Transaction** or **Set Ending Transaction**.

- Click and drag-and-drop a transaction to the **Starting Transaction** or **Ending Transaction** field in the **Refine By** pane.
  The list populates the transactions that occur between the starting and ending transaction.

- Open the graphical view, right-click a transaction, and select **Transaction Time**, **Set Starting Transaction,** or **Set Ending Transaction**.
  Transactions that occur between the starting and ending transactions display.

Exclude Data from Agent Capture

The DevTest Java Agent can capture a large amount of data, including data that is not relevant or valuable. For example, you might not be interested in the data about logging in to an application.

You can prevent the agent from capturing data by specifying a URL, transaction frame, or Java class.

Wildcards are supported for URLs. For more information, see Excluding from Interception and Virtualization.

Some transaction frames and Java classes do not support being excluded.

For merged transaction frames, you can exclude only by URL or class.

The following graphic shows an excluded URL.

Screen capture of excluded URL.

Notes:

- Be careful when excluding data. If a transaction has multiple frames, excluding one frame can cause unpredictable behavior. For example, the category of the root frame can change or the remaining frames might not stitch correctly.
- This feature is not supported for transactions that are created from the DevTest extension for Google Chrome.

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** from the left navigation menu. The **Analyze Transactions** window opens.
2. Do one of the following actions:
   - Display the graphical view.
   - Display the list view and click the transaction link from the **Name** column.
3. To exclude a URL:
   a. Right-click a frame and select **Capture**, **Disable URL Capture**.
   b. (Optional) Modify the URL to include one or more wildcards.
   c. Click **OK**.
4. To exclude a frame, right-click the frame and select **Capture**, **Disable Capture**.
5. To exclude a class, right-click a frame and select **Capture**, **Disable Class Capture**.
6. To view the items that are currently excluded, click **Manage exclude frame capture rules** .
7. To make an item available for capture again:
   a. Click **Manage exclude frame capture rules** .
   b. Select the item.
   c. Click **Include**.

View Transaction Details

The transaction detail dialog contains information about each transaction in a path.

The upper area contains the path graph.

The lower area contains the frame information.

Use the button to expand and collapse the area for the path graph and the frame information.

The following graphic shows the transaction detail dialog:

Screen capture of Transaction Details dialog

The title bar contains the name of the root transaction frame.

In the path graph, a visual path of the transaction displays an icon representing the container type. If a container type is not recognized, a Java icon is used.

The path graph also indicates which frames are not eligible for artifacts. Point to the **X** to the left of the frame to view this information.

To view the usage counts for identical and partial transaction paths, select the **Enable** check box for the **Usage Count** field at the top of the path graph. See View Usage Counts for Identical and Partial Transaction Paths for more information.

Follow these steps:

1. Select **Application Insight, Analyze Transactions** from the left navigation menu.
   The **Analyze Transactions** window displays.

2. (Default) In the list view, select the transaction link in the **Name** column.

3. In the graphical view, click a frame.
   The transaction detail dialog opens.

4. Review the path graph and the frame information.

5. (Optional) To increase or decrease the view of the path, use the **+** and **-** buttons.

6. (Optional) To pan the view of the path, select the path while holding and moving it around the path graph.

7. (Optional) To navigate to the previous and next transaction, click the arrows at the left of the title bar.

8. Click **X** to close the dialog.

Path Graph

A *path graph* contains a graphical representation of a path and its transaction frames. The path graph displays in the upper area of the transaction detail dialog.

When you select a root transaction frame in graphical or list view from the **Analyze Transactions** window, the transaction detail dialog displays with the path graph at the top.

You view the path graph from the **Analyze Transactions** window by:

- Clicking a path in the graphical view
- Clicking the transaction link from the **Name** column in the list view

The icons indicate the type of frame. Examples of the frame types are EJB, HTTP, JMS, REST, web service, DevTest, and unknown. The DevTest frame type represents a step executing inside a test case or virtual service.

- General Path Information
- Frame Name
- Execution Time
- Incoming and Outgoing Request Information

General Path Information

The path graph includes the following general information:

Start Time

The date and time when the frame began running.

Wall Time

The execution time for the frame.

Transaction ID

The unique identifier of the path.

Frame Name

The name of the frame appears below the icon.

The format depends on the type of frame.

The frame name for HTTP is the path portion of the URL. Paths that are created using the **VS Traffic to Application Insight Companion** are indicated with the **virtualized** label in the component name. An example is **POST/itkoExamples/EJB3AccountControlBean(virtualized)**. See VS Traffic to CAI Companion for more information about using this companion.

The frame name for JDBC is SQL Activity (*N*), where *N* is the number of SQL statements that were used to generate the frame.

The frame name for JMS consists of the following parts:

- A string that identifies the type of operation. If the operation involves the production of a message, the string is **send:**. If the operation involves the consumption of a message, the string is **recv:**.
- The name of the queue
- (Optional) The name of the connection factory

An example is **recv:queue/ORDERS.REQUEST@ConnectionFactory**.

The frame name for JMS can also be **DevTest**. This name is used when a test case receives a JMS message.

The frame name for RMI is the Java class and method that was executed.

The frame name for webMethods is the flow service name.

The frame name for WebSphere MQ consists of the following parts:

- The string **put** or **get**
- The WebSphere MQ host name
- The queue manager name
- The queue name

An example is **put-172.24.255.255-QueueManager-ORDERS.REQUEST**.

The frame name for a web service is the path portion of the URL.

Execution Time

The execution time appears below the frame name.

In certain scenarios, the execution time for the leftmost component is the time for the entire transaction.

If the execution time is so low that it rounds down to zero, the value **0 ms** is displayed.

Incoming and Outgoing Request Information

In the path graph, you can view the incoming and outgoing request payloads by selecting the line that connects a frame. The **Incoming Request** and **Outgoing Request** tabs display in the lower area of the transaction detail dialog.

The following graphic displays the incoming and outgoing request information in the path graph:

Screen capture of incoming and outgoing request in path graph

Frame Information

The lower area of the transaction detail dialog displays information about the frame that is selected in the path graph.

The **Frame** tab and the **Log Messages** tab appear for all frame types.

You can select and copy field information from the available tabs. To copy, select the information, right-click, and select **Copy**. Keyboard shortcut keys can be used to copy and paste frame information from the tabs.

- Frame Tab
- Request Tab
- Response Tab
- Log Messages Tab
- XML Tab
- Exception Tab
- Payload Tab
- SQL Summary Tab
- Request Headers Tab
- Response Headers Tab
- Incoming Request Tab
- Outgoing Request Tab

Frame Tab

The **Frame** tab shows the metadata values and the location values of the selected frame. The following details are listed:

Frame Meta Values

- Name: The name of the frame.
- Tags: Any frame tags that are associated with the frame.
- Category: The category of the frame.
- Start time: The time at which the frame began running.
- Wall time: How much time it took to run the frame from start to finish. Also known as root frame response time.
- ID: The unique frame ID.
- Parent ID: The parent frame ID of the current frame.

Location Values

- Host: The name of the computer on which the frame ran.
- Agent: The name of the agent which captured the frame.
- Thread: The thread name that the frame is running.
- Method: The java methods captured in the frame.
- Session: Session ID of the session this frame is associated with.

If the execution time for a frame is so low that it rounds down to zero, the value **0 ms** is displayed.

Request Tab

The **Request** tab displays the actual request that the selected frame sent. This tab appears only for non-JMS and non-JDBC frames.

If a WebSphere MQ transaction includes a binary payload, the **Request** tab displays a human-readable version of the payload. The nontext characters are removed. You can view the original binary payload in the **XML** tab.

For CICS transactions, the raw data displays in hex format. The decoded text output on the right depends on which EBCDIC code is selected.

Response Tab

The **Response** tab displays the actual response that the selected frame received. This tab appears only for non-JMS and non-JDBC frames.

If a WebSphere MQ transaction includes a binary payload, the **Response** tab displays a human-readable version of the payload. The nontext characters are removed. You can view the original binary payload in the **XML** tab.

For CICS transactions, the raw data displays in hex format. The decoded text output on the right depends on which EBCDIC code is selected.

Log Messages Tab

The **Log Messages** tab contains any log messages. The following details are listed:

- Level of information
- The logger name (if any)
- The text of the log message

By default, the minimum level of log messages is **Warning**. To change the minimum level, configure the **Java logging level** property.

You can configure this property from the Agents window of the DevTest Portal. The property appears in the **Settings** tab.

To sort the columns, click the drop-down arrow in a column heading and select **Sort Ascending** or **Sort Descending**. To show or hide columns, click the drop-down arrow in a column heading, point to **Columns**, and select or clear the check boxes.

XML Tab

The **XML** tab displays the raw XML of the instrumentation response in the frame. This tab appears for all frames other than JDBC frames.

The **XML** tab is additive. The frames to the right contain less information than the frames on the left. The first frame contains the complete response.

Exception Tab

If the frame contains an exception, the **Exception** tab displays the stack trace.

This feature has the following prerequisites:

- The capture level of the **Exception** protocol must be set to **Full Data** for the agent.
- The **Capture all the supported exception frames** property must be enabled for the agent. To access this property in the **Settings** tab, select the **Transactions** category.

Payload Tab

The **Payload** tab displays the JMS payload. This tab appears only for JMS frames.

SQL Summary Tab

The **SQL Summary** tab provides a summary of the SQL information. This tab appears only for JDBC frames.

The following details are listed:

- A sequential identifier that is assigned to each SQL statement
- SQL statement
- Number of rows that were returned
- Number of invocations

- Average time (milliseconds)
- Total time (milliseconds)

Clicking the SQL statement displays the results.

Double-clicking the SQL statement opens a dialog that displays the entire statement.

To sort the columns, click the drop-down arrow in a column heading and select **Sort Ascending** or **Sort Descending**. To show or hide columns, click the drop-down arrow in a column heading, point to **Columns**, and select or clear the check boxes.

Request Headers Tab

The **Request Headers** tab displays information about the components of the header section of request. This tab displays for only SOAP**,** REST, HTTP, HTTPS, and Web Service frames.

Response Headers Tab

The **Response Headers** tab displays information about the components of the header section of response. This tab displays for only SOAP**,** REST, HTTP, HTTPS, and Web Service frames.

Incoming Request Tab

The **Incoming Request** tab displays information about the incoming frame request. This tab displays when a line between frames is selected in the path graph.

Outgoing Request Tab

The **Outgoing Request** tab displays information about the outgoing frame request. This tab displays when a line between frames is selected in the path graph.

Pin and Unpin Transactions

You can pin transactions to identify them as points of interest (POIs). Pinned transactions display in the **Points of Interest** portlet in the **Home** page dashboard.

The following graphic displays a pinned defective transaction:

Screen capture of a pinned transaction

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** from the left navigation menu.
2. In the list view, click the transaction link in the **Name** column, right-click a transaction frame, and select **Pin Transaction**.
3. In the graphical view, right-click a transaction frame and select **Pin Transaction**.

4. Enter a description and click **OK**.
   The transaction displays a blue pin. The tooltip contains the pin description that was entered.

5. To unpin, right-click the transaction and select **Remove Pin**.

6. (Optional) Go to the **Home** page and refresh the **Points of Interest** portlet.
   The pinned transaction displays in the **Points of Interest** list.

Share Point of Interest

You can view the point of interest link to the transaction to share it. The link contains the URL where the point of interest is located within Application Insight.

> **Note:** Before you can share a point of interest, pin a transaction. For more information about how to pin a transaction, see Pin and Unpin Transactions.

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** from the left navigation menu.
   The **Analyze Transactions** window displays in list view by default.

2. Click the transaction link in the **Name** column or click **Graphical** .

3. Right-click the transaction and select **Share, Share Point of Interest** to view the URL.

4. Click **X** to close the **Share Point of Interest** dialog.

Share a Link

You can view the link to the transaction to share it. The link contains the URL where the transactions are located within Application Insight.

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** from the left navigation menu.
   The **Analyze Transactions** window displays in the list view by default.

2. Click the transaction link in the **Name** column or click **Graphical** .

3. Right-click the transaction and select **Share, Share Link** to view the URL.

4. You can go directly to the transaction by:

   - typing the URL in a web browser

   - highlighting the link, right-clicking, and selecting **Go to http://<URL>**

5. Click **X** to close the **Share Link** dialog.

Label the Links Between Transaction Frames

You can annotate specific links between transaction frames by adding a label. For example, you can add a label to recognize an issue with the communication between two frames. Labels can be added to links in the graphical view or the list view from the transaction detail dialog.

The following graphic displays a label in the path graph:

Screen capture of path graph with labeled link

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** from the left navigation menu.
2. Open the path graph from either the list view or the graphical view.
3. Point to the link until the link becomes highlighted, right-click the link, and select **Label this link**.
4. Enter the label and click **OK**.
   The label is saved and added to the specific link. The tooltip contains the label that was entered.
5. To edit or clear the label, point to the link until the link becomes highlighted, right-click and edit or clear the label, and click **OK**.

Merge Repeated Paths

A repeated path contains duplicate transactions that occur in a path. You can merge these paths to identify the same transactions and reduce the number of transactions that are listed. Merging repeated paths helps you to identify the cause of a defect. For example, you can see where all of the occurred, repeated login transactions that fail for an application.

Repeated paths have an orange badge. The badge includes the number of paths that are merged.

The following graphic displays repeated paths that have been merged in the list view:

	Name	Start Time	Wall Time	CPU Time	Agent
	/cars-app/features/	03/07/2016 8:10:33 AM 832	35 ms	46 ms	CarWeb
	/cars-app/api/validate	03/07/2016 8:10:33 AM 660	159 ms	156 ms	CarWeb
	/cars-app/api/login Merged (2)	03/07/2016 8:10:33 AM 198	433 ms	390 ms	CarWeb

Screen capture of merged transactions in list view

The following graphic displays repeated paths that have been merged in the graphical view:

Screen capture of merged transaction in graphical view

Artifacts that are created using merged transactions are named with the transaction ID.

When creating baselines from merged transactions in the shelf, merged transaction can be viewed, modified by order, and deleted.

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** from the left navigation menu.
2. In the list or graphical view, click .
   The repeated paths merge into one transaction. An orange badge displays with a count of the repeated paths.
3. To unmerge the transactions, click .

Merge Behavior for URLs

Web HTTP and REST HTTP URLs can be treated as duplicates even if the names have segments that vary.

Consider the following REST HTTP URLs:

- /cars-app/carShop/car/2T1KR32E37C639014
- /cars-app/carShop/car/2T1KR32E37C639015
- /cars-app/carShop/car/2T1KR32E37C639016

The last segment represents the vehicle identification number for a car. When evaluating the URLs for merging, the broker replaces this segment with a parameter.

By default, the broker evaluates the segments beginning at position 3. If the varying segment were in position 2, you must configure the broker to begin at position 2.

The default pattern string for evaluating segments is **[a-zA-Z]+[0-9]{5,}[a-zA-Z]***.

To change the configuration settings from the DevTest Portal, use the following broker properties:

- The URL parameter starting position
- The regular expression to match the URL parameter

The corresponding properties in the **rules.xml** file are **lisa.agent.transaction.url.param.start.position** and **lisa.agent.transaction.url.param.pattern**, respectively.

Changes to the properties affect future transactions. They do not affect existing transactions.

View Usage Counts for Identical and Partial Transaction Paths

You can view the usage counts for the partial path that connects two frames that are identical within other transaction paths. The connecting two identical frames are identified from all the transactions in the search result including the selected transaction. For JDBC transaction frames, the same category is used to determine the usage counts. For all other frame types, the usage counts are based on the class name, method, category, agent, and display name.

Displaying the usage count is enabled from the transaction detail dialog.

For example, you can see how many times a partial transaction in a path that is named **/lisabank /depositmoney.do** with identical paths occurs during a one-day period. Use the search and filter criteria to narrow the transactions that appear in the list of the **Analyze Transactions** window.

The following graphic displays the usage counts for the transaction that is named **/lisabank /depositmoney.do** in the path graph:

Screen capture of usage counts with multiple traffic levels

Use the legend that is located in the path graph to view the traffic usage for a transaction path. The following graphic displays the legend for the usage traffic:

Screen capture of legend for usage counts traffic

The legend uses color to represent the transactions with the lowest to the highest traffic. The higher the traffic is for a transaction, the thicker the line connecting the nodes become.

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** from the left navigation menu.
2. (Optional) Enter a search criteria using the search bar and refinement panel.
3. In the list or the graphical view, select a transaction path and open the transaction detail dialog:
   - (Default) In the list view, click the transaction link in the **Name** column.
   - In the graphical view, double-click the parent frame.

The transaction detail dialog opens.

4. Select the **Enable** check box for **Usage Count**.
   The path graph displays the total number of times a transaction from the selected path was used in the transactions listed.

5. (Optional) Use the zoom buttons to view the details of the usage counts.

6. (Optional) Click to expand the view of the path graph to display large paths.

7. (Optional) Click **Show legend** to view the traffic for the transaction usage.

View Producer and Consumer Relationship

You can identify and visualize the transactions of producers and consumers for an agent. The producer is also known as the common API.

To be classified as a common API, the producer must be called by consumers from at least two agents other than the agent in which the producer is running.

The following graphic shows two paths. Frame D is the producer. Frame B and frame C are the consumers.

The following graphic displays the **Common APIs** dialog, which provides a visual representation of the producer and consumer relationship. The web service frame at the right is the producer.

Screen capture of Common APIs dialog.

Transaction frames from the following protocols are supported:

- EJB
- HTTP
- HTTPS
- JMS
- MQ
- REST
- RMI
- Web Service

You can shelve the transaction frames in the **Common APIs** dialog by right-clicking a frame and selecting one of the following options:

- **Shelve Frame**
  This option is available for producer and consumer transactions.
- **Shelve All Occurred Transactions**
  This option is available for consumer transactions.

The shelved producer or consumer transactions are labeled in the shelf as **Common API Frame**. See Shelve Transactions and Using the Shelf for more information about shelving transactions.

You can generate baseline tests and virtual services that represent multiple consumers of a business service. A regression test represents the data from all the consumers and documents the reuse of a common API. This data helps you understand how changes to the API can affect multiple consumers.

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** from the left navigation menu.
2. Click **Common APIs** .
3. (Optional) Change the selected agent in the drop-down list.

4. Select the check box for one or more common APIs.

5. Click **Show Common**.
   The **Common APIs** dialog displays the producer and consumers for the common APIs for the agent.

Transaction Stitching Rules

Stitching is the process of assembling partial transactions into complete transactions. Sometimes transactions that are meant to be stitched together are displayed as partial transactions in the **Analyze Transactions** window. You can create rules to stitch transaction frames together by a specific string; for example, GUID.

You identity the string and set the parent and child key. Only the last frame in a transaction path can be set as the parent key. Only a root transaction frame can be set as the child key.

The stitching rules are saved to the **rules.xml** file and are applied to the new transactions that are captured.

> **Note:** For transactions with SQL frames, only frames before the SQL frame can be set as the parent key.

The following options are available to configure parent and child keys:

- **Use Exact String**
  This option uses this string for stitching. For example, if you want to stitch transactions by category.

- **Same as Parent Key**
  This option is available only for setting the child key. The child transaction frame inherits the parent key information.

- **Using Built-in Regular Expression**
  These options are predefined expressions that are used to identify how transactions are stitched together. The options are IPv6, IPv4, GUID, email, and URL.

- **Using Custom Regular Expression**
  This option lets you create a custom expression.

Create Stitching Rules

Create stitching rules to assemble transaction frames together that share a specific string.

Follow these steps:

1. From the **Analyze Transaction** window in the list view, click the transaction link in the **Name** column.
   The transaction details dialog displays.

2. Select the XML tab and highlight the string to use for stitching other transactions and click **Set Parent Key** .
   The **Set Parent Key** dialog displays. The path of the transactions is populated in the **Path** field.

3. Select one of the following options:
   - Use Exact String
   - Using Built-in Regular Expression
   - Using Custom Regular Expression
4. Navigate to the child frame using the navigation buttons and click **Set Child Key** .
5. Select one of the following options:
   - Use Exact String
   - Same as Parent Key
   - Using Built-in Regular Expression
   - Using Custom Regular Expression

Manage Stitching Rules

Once the rules are created, you can view, delete, and group the payload stitching rules. Grouping rules together determines how two or more transactions frames are arranged into a single transaction path. This arranged transaction path represents the actual operations that occurred within the application.

The **Stitched Payloads** tab of the **Manage Frame Rules** dialog lists the rules into two categories, ungrouped and grouped. New rules are listed in the ungrouped category by default.

1. Click **Manage Frame Rules** from the **Analyze Transactions** window.
   The **Manage Frame Rules** dialog opens.
2. Select the **Stitched Payloads** tab.
   The payload stitching rules display in a list.
3. To delete a rule, click **Delete rule** .
4. To group rules, select the rule check box and click **Group Payload Stitching Rules** .
5. To move a rule from a group, click **Remove Rule from the Group** .
6. To ungroup all of the rules in a group, click **Ungroup all items** .

Delete Transactions

Transactions with exceptions or bad data can be deleted from the database in the list view of the **Analyze Transactions** window. You cannot delete transactions in the graphical view. Delete transactions using the right-click method or using the **More Actions** delete options.

Imported transactions and transactions with points of interests or tickets can be excluded from the deletion process. You can export transactions before deleting them from the database.

Multiple transactions can take more time to delete. The **Results** transactions count at the top of the **Analyze Transactions** window decreases as transactions are deleted.

Users with the following roles can delete transactions:

- Power user
- Super user
- Administrator

> **Note:** To select a transaction, click next to the transaction link in the **Name** column. Clicking the link opens the transaction details dialog.

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** from the left navigation menu.
2. Select one or more transactions, right-click, and select **Delete Selected**.
3. For more delete options, click **More Actions** and select an option**:**
   - **Delete Selected**
     Option to delete all the manually selected transactions in the list.
   - **Delete Page**
     Option to delete the current page of transactions without manually selecting any transactions.
   - **Delete All**
     Option to delete all transactions from the database without manually selecting every transaction.

   The **Delete Transaction** dialog displays. The checkbox for **Exclude POI transactions** and **Exclude ticketed/imported transactions** is selected by default.
4. To delete transactions with POIs and tickets, clear the **Exclude POI transactions** checkbox.
5. To delete imported transactions or transactions with support tickets, clear the **Exclude ticketed/imported transactions** checkbox.
6. To export transactions before deleting, select the **Export before deleting** checkbox and enter the filename.
7. Click **Delete**.
8. Click **OK** to close the **Information** dialog.

9. Click **Refresh** to update the list of transactions.

Export and Import Paths

You can export paths and import paths and recorded transactions from the **Analyze Transactions** window. Paths can be imported into other installations. You can export paths for CA Support to debug.

More Information:

- Export Paths
- Import Paths

Export Paths

You can export transaction paths into a zip file to be shared and imported. Transactions are exported as XML files and are named using the transaction ID. Use the search refinements on the **Analyze Transactions** search bar to narrow the transactions that display.

The following export options are available:

- **Export Selected** exports the selected transactions that display in the list view.

  You can export selected transactions by clicking **More Actions** or by right-clicking the transactions and selecting **Export Selected**. To select multiple transactions, hold down the Shift key.

- **Export Page** exports the page of transactions that display in the list view. A maximum of 500 merged transactions can be exported.

  Note: When there are multiple pages of transactions, click **More Actions** and select **Export Page** for each page you want to export.

- **Export View** exports the paths that display in the graphical view. A maximum of 500 merged transactions can be exported.

- **Export All** exports all the transactions that display based on the search results. If the number of transaction exceeds 1000, the export will take longer to complete. A warning dialog displays to enable you to continue or cancel the export.

  Note: When using the Safari browser to export a transactions, by default the file is named **unknown** without an extension. The Safari browser does not allow files name to be edited when downloading. Ensure you add the .zip extension to the file name to import the transaction. This behavior applies when using the **export selected**, **export page**, and **export all** options to export transactions.

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** from the left navigation menu.
2. Use the search refinements to find the transactions you want to export.
3. Click **More Actions** and select an export option.

   The **Export Path** dialog opens.
4. Enter a name for the export zip file.
5. Click **Export and download path**.
   A zip file is created.

More Information:

- Intelligent Search for Transactions
- Search for Transactions by Time
- Filter Transactions
- Import Paths

Import Paths

You can import a zip file that contains transaction paths from various sources into the Application Insight database.

From the **Analyze Transactions** window, you can also import recorded transactions paths for use in Document Transactions. The recorded transactions paths display in the Document Transactions and Analyze Transactions window.

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** from the left navigation menu.
2. Click **More Actions** and select **Import**.
   The **Import Transactions** dialog displays.
3. Click **Choose** and select the zip file.
4. Review the paths that display in the table and click **Import**.
   The **Imported** column displays:
   - **Yes** when the path is imported into the database.
   - **Duplicate** when the path already exists in the database.
   - **Failed** when the paths were not imported.

5. Click **OK**.
   A **Confirmation** dialog displays.

6. Click **OK** and click **Refresh** from the **Analyze Transactions** window.
   The imported paths display in the list.

Export Path Diagram to PDF

You export a path diagram into a PDF document to view, save, and print the path information. You create the PDF document in the list or the graphical view. Use the search and filter refinements that are available in both views. See Search and Filter Transactions for more information about refining your search results to find transaction paths.

Notes:

- Microsoft Internet Explorer and Mozilla Firefox requires Adobe Reader to view the PDF. For Google Chrome, enable either the Chrome PDF viewer or Adobe Reader.
- PDF generation is not supported when running the DevTest Portal in HTTPS mode.

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** in the left navigation menu.

2. Use the search and filter refinements to find transactions to create a PDF document.

3. Select a path in the list or the graphical view and click .
   The path diagram displays in the preview browser.

4. Point to the bottom, right of the browser to display the task bar.
   The task bar contains options to zoom, save, and print.

5. To save the PDF file, click **Save**.

6. Enter the file name and click **Save**.

7. To print the PDF, click **Print**.
   The print setup displays.

8. Click **Print**.

Shelve Transactions

From the **Analyze Transactions** window, select transactions to generate artifacts by *shelving* them. When you shelve transactions, the frame is added to the shelf. From the shelf, you generate artifacts, create documentation, and set points of interest. The types of artifacts that can be generated are virtual services, baselines, data sets, and request/response pairs.

Transactions can be added to the shelf as individual frames, as all occurred transactions, and as merged identical transactions.

For more information about working with transactions in the shelf, see Using the Shelf.

> **Note:** When the login session expires or when the browser running DevTest Portal is closed or refreshed, the transactions in the shelf are removed.

Transactions can be shelved from the graphical view and the list view from the Transactions details dialog. Right-click a transaction and select one of the following shelving options:

- Shelve Frame
- Shelve Frames (Advanced)
- Shelve All Occurred Transactions
- Shelve (X) Merged Transactions

Shelve Frame

To add an individual frame to the shelf, right-click the frame and select **Shelve Frame**.

Shelve Frames (Advanced)

Shelving frames using the advanced options let you shelve the same types of transaction frames from selected transactions or from transactions based on a search result.

The following advanced options are available:

- **Same category of frames from this transaction**
  Allows you to shelve transactions with the same category.

- **Same category of frames from the search result**
  Allows you to shelve transactions with the same category from a search result. You can create a custom filter to refine the search results by agent and remote IP address.

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** from the left navigation menu.
   The **Analyze Transactions** window displays in the list view by default.

2. Click the transaction link in the **Name** column or click **Graphical** .

3. Right-click a transaction and select **Shelve Frames (Advanced)**.

4. To shelve frame from the selected transaction, select **Same category of frames from this transaction.**

5. To shelve frame from the search result, select **Same category of frames from the search result.**

6. (Optional) To create a custom filter to refine the search result, do the following steps:

   a. Select the **Custom Filter** checkbox.

   b. Select **Inbound frames** and select from the agent name from the list. The inbound frames list the agents that were filtered from the **Analyze Transaction** window.

c. Select **Outbound frames** and select the checkbox for the remote IP address. The Outbound frames list the remote IP address that is returned from the search in the **Analyze Transactions** window.

d. To add options to the filter, click the arrow next to **Options** and enter the fields.

e. Click **Apply Filter.**
The applicable frames are listed.

7. Select frames from the list to shelve and click **Shelve Frames**.
The frame is added to the shelf.

8. To unshelve a transaction, right-click a transaction, and select **Unshelve**.

Shelve All Occurred Transactions

You can add all the transactions and frames with the same transaction ID that occurred during a captured transaction into the shelf. Examples of all occurred transactions are button clicks and log outs. The shelf displays all the occurred transactions as one transaction. You create virtual services, create baselines, and create documentation as a single shelved item. A blue pin and the word **All** displays above the shelved frame.
The following graphic displays a shelved transaction with all the occurred transactions in the path:

Screen capture of a shelved all occurred transactions in graphical view

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** from the left navigation menu.
The **Analyze Transactions** window displays in the list view by default.

2. Click the transaction link in the **Name** column or click **Graphical** .

3. Right-click a transaction and select **Shelve All Occurred Transactions**.

Shelve (X) Merged Transactions

Shelving merged transactions lets you create artifacts and set points of interest for a group of identical paths. When merged transactions are added to the shelf, the number of transactions increases by one. See Merge Repeated Paths for more information about merging repeated transactions.
The following graphic displays shelved and merged transactions in the shelf popup:

Screen capture of merged transactions in the shelf pop-up

The stateful baselines are not supported for merged transactions. You can create consolidated and expanded baselines only.

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** from the left navigation menu.
   The **Analyze Transactions** window displays in the list view by default.

2. Click the merged transaction link in the **Name** column or click **Graphical** .

3. Right-click a transaction and select **Shelve (X) Merged Transactions**.
   (X) represents the number of merged frames for the transactions
   A blue pin icon displays above the shelved transaction frame. The frame is added to the shelf. The total number of shelved transactions in the shelf increases by one.

4. To unshelve a transaction, right-click a transaction, and select **Unshelve**.

Generate Artifacts for an Agent

You can generate virtual services with VRS support, baselines, and request/response pairs directly from an agent without shelving any transactions. From the **Analyze Transactions** window, you display a topological view of all the agents that are capturing data for a specified time period. The search and filter refinements set for transactions determine which agents display in the agent topology. For example, the agents that display as a result of searching by **1 Day** or **All Time** and filtering by **Transaction ID**.

Virtual services are created based on the root transaction frame and are consolidated by the frame type. The virtual services that are created from agents are stateless. Only transaction frames that support virtual services are included in the virtual service creation.

Baselines are created based on the root transaction frame. An expanded baseline is created for all the frame types that support Application Test Steps. Another baseline is created for frame types that only support Transaction Frame Steps. All other frame types that do not support an expanded baseline are removed from the baseline creation.

The virtual services and baselines use the naming convention of **prefix_agent_Protocol_UTCTimeDataStamp**. The default prefix is the user name and can be edited.

> **Note:** CICS agents only support virtual service creation. A VRS or a copybook bundle file must be used to create the virtual service.

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** from the left navigation menu.
2. Click **Display agent topology** .
   The **Agents** dialog displays in the graphical view.
3. Right-click an agent and select **Generate All Artifacts for Agent**.
4. Select one or more of the following artifact options:
   - **Baseline**
   - **RR Pair**
   - **Virtual Service**
     To include a VRS file, select the **Use VRS File** check box and select or enter the name of the VRS file.
     To include a copybook file, select the **Use Copybook Bundle** check box and select a file and data protocol from the drop-down list. This option is only available for CICS agents.
5. Click **Generate**.
6. Select a project, edit the prefix (optional), and click **Create**.
   A confirmation dialog displays a link to the artifacts that were successfully created.
7. (Optional) Click a link and review the information or perform more tasks that are related to the artifact type created.
   A new tab opens with the project name of the artifact.
8. Click **OK** to close the confirmation dialog.
9. Close the **Agents** dialog.

> **More Information:**
>
> - Business Transactions and Transaction Frames
> - Search for Transactions by Time

Work with Defects

The **Analyze Transactions** window lets you identify and analyze defects in transactions and detect performance bottlenecks.

From the **Analyze Transactions** window, you can:

- Annotate transaction with exceptions, log messages, response time percentage, and view any transactions that have been annotated
- Pin annotated transactions

From the shelf, you generate virtual services, baselines, and produce defect documentation for transactions with defects. For more information, see Using the Shelf.

- Annotate a Transaction to View Defects
- Generate a Defect Report

Annotate a Transaction to View Defects

Application Insight exposes the transactions that have violations and display them in the path. You can shelve the defective transaction to generate a virtual service, baseline test, and documentation.

The **Analyze Transactions** window in the graphical view contains the following annotation options:

- **Exceptions**
  Identify and view transactions with exception violations.
- **Log messages**
  Identity and view log messages for errors and warning.
- **Response times percentile**
  Set the percentile of the worst performers and view them with in the transaction path.
- **Only show annotated transactions**
  View only transactions that were annotated.

The following graphic displays an example of a transaction with annotated exceptions:

Screen capture of an example of annotated transactions in graphical view.

Transactions with log messages that contain warnings are yellow and exceptions, error log messages, and response time violations are red.

Transactions that are annotated and pinned in the **Analyze Transactions** window appear in the **Home** page, **Points of Interests** list.

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** from the left navigation menu.
   The **Analyze Transactions** window displays the transactions in a graphical view.

2. Search for a transaction using the search refinement options.

3. Click **Annotation options** and select one or more options.
   Transactions with defects display.

4. (Optional) Select a transaction to view more information about a defect.
   The transaction detail dialog displays.

5. (Optional) Shelve the transaction and generate artifacts. For more information about shelving, see Shelving Transactions.

Generate a Defect Report

You can generate a report for transactions with defects. You can save the report as a PDF document and print.

Follow these steps:

1. Select **Application Insight**, **Analyze Transactions** from the left navigation menu.
   The **Analyze Transactions** window displays.

2. Click **Graphical** .

3. Click **Annotate options** and select one or more options to annotate from the menu.
   Frames with violations appear in red.

4. Right-click the transaction that you want a report and select **Shelve** or **Shelve All Occurred** .

5. Open the shelf popup or dialog and click .

6. Enter the title and click **OK**.
   The report displays in a browser as a PDF document.

7. To save, click **Save,** enter a file name. and click **Save**.

8. To print, click **Print**.

Using the Shelf

The shelf provides a means of accumulating multiple business transaction frames to generate artifacts, documentation, and set points of interest.

For information about how to add transactions to the shelf, see Shelve Transactions.

You access the shelf from the **Analyze Transactions** window in the list and graphical view. The following graphic displays the shelf icon:

The number that is located to the right of the icon indicates the number of shelved frames. There are 500 frames that are displayed at a time. To narrow the view, use filters.

> **Note:** When the login session expires or when the browser running DevTest Portal is closed or refreshed, the transactions in the shelf are removed.

The shelf has two methods of generating artifacts:

- Shelf dialog
- Shelf popup

Shelf Dialog

The shelf dialog lets you search and manage shelved transactions and create artifacts.

To open the shelf dialog, click

or click **View Shelf (X frames)** from the popup.

The following graphic displays the shelf dialog:

Screen capture of the shelf dialog with artifact labels

If a transaction with a point of interest (POI) is shelved, the name of the POI displays after the start time in the shelf dialog.

Shelf Popup

The shelf popup provides a quick way to create artifacts without opening the shelf dialog.

To display the shelf popup, point to the shelf icon.

The following graphic displays the shelf popup:

Screen capture of shelf popup

A maximum of five transactions display at a time. When you delete a transaction by clicking **X**, the next transaction displays.

More Information:

- Work with Transactions in the Shelf
- Creating Virtual Services
- Creating Baselines
- Create Data Sets
- Create Request/Response Pairs from Transaction Frames

Work with Transactions in the Shelf

This page takes you through the steps of working with shelved transactions to generate artifacts, set points of interest, and create documentation.

The following scenario assumes that you have shelved a transaction frame.

1. Search for Frames in the Shelf
2. Delete Frames in the Shelf
3. Set Points of Interest Transactions in the Shelf
4. Generate Artifacts
5. Create Documentation from the Shelf
6. Clear All Transactions from the Shelf

Search for Frames in the Shelf

The search capability lets you search for transaction frames in the shelf by entering a full text string or payload search criteria.

Follow these steps:

1. In the **Analyze Transactions** window, click .
   The shelf displays a list of all the shelved transaction frames.

2. Enter the text string or payload search criteria in the **Search Shelf** field.
   The shelf returns a list of transaction frames that are based on the search criteria.

Delete Frames in the Shelf

You can delete transaction frames from the shelf dialog and from the shelf popup.

Follow these steps:

1. Open the shelf.

2. Locate the frame that you want to delete and click **X**.
   The transaction frame is removed from the shelf.

Set Points of Interest in the Shelf

From the shelf, you can pin transactions as a point of interest (POI). Transactions with points of interest display in the **Points of Interest** portlet from the **Home** page dashboard. If a POI is pinned from **Analyze Transactions**, the name of the POI displays in the shelf dialog when the transaction is shelved.

Follow these steps:

1. Open the shelf and click .

2. Enter a pin description and click **OK**.

3. To view the POI in the **Home** page dashboard, select the **Home** tab and click .

Generate Artifacts

You can generate the following business artifacts for transaction frames in the shelf:

- Virtual services
- Baselines
- Data sets
- Request/response pairs

The types of artifacts that can be generated are indicated with colored labels in the list of transactions.

Follow these steps:

1. Open the shelf.
2. (Optional) Click **X** to delete any frames for which you do not want to create an artifact.
3. Click an artifact option and click **Create**.
4. Select a project, and edit the prefix name (optional).
5. Select an option to keep or delete transactions in the shelf after the artifact is created.
6. Click **Create**.
   The **Jobs in Progress** dialog opens.
7. Use the job progress indicator tooltip to display information about the artifact.
   The tooltip indicates if the artifact was generated successfully or unsuccessfully and includes any error messages. The tooltip may contain a link to the location of virtual services, baselines, and request/response pairs in the portal.
8. Close the dialog.

> **Note:** When creating virtual services, baselines, and data sets, there might be extra options to configure. For example, you can select types of steps and can use magic dates when creating baselines. Refer to the topics in the following list for detailed steps.

For detailed steps in creating specific types of artifacts, see the following topics:

- Creating Virtual Services
- Creating Baselines
- Create Data Sets
- Creating R/R Pairs

Create Documentation from the Shelf

> **Note:** If the portal is running in HTTPS mode, PDF generation is not supported.

You can create reports from transactions in the shelf. Ensure pop-ups are enabled in the browser to display the report in a new browser window. You can print the report and can save it as a PDF document.

The report includes the following information about the transaction frame and merged paths:

- Agent name
- Machine name
- IP address
- Agent version
- Properties
- Summary information such as the number of folds and response times

The agent boundary for the transactions displays in the **Transactions Detail Information** section at the end of the report.

Follow these steps:

1. Open the shelf and click .
2. Enter a title for the report and click **OK**.
   The report displays in another browser window.
3. To print, point to the bottom, right of the window to display the task bar, and click the print icon.
4. To save, point to the bottom, right of the window to display the task bar, and click the save icon.

Clear All Transactions from the Shelf

You remove all the transactions that are currently in the shelf by clicking from the shelf dialog or popup.

Create and Manage Tickets

The Application Insight feature lets you capture detailed information about application behavior, including defects. This functionality can help speed up the resolution of defects by enhancing collaboration between testers and developers.

The application must be running on a Java agent-enabled computer. A ticket is created in the application. The ticket is viewed in DevTest Portal.

Tickets include such information as a title, severity level, and description. Tickets also include the corresponding paths, which expose the actions of the underlying components.

Disabling the Tickets Feature

To make this functionality work, Application Insight injects JavaScript into the user interface code of the application.

If you want to prevent Application Insight from injecting the JavaScript, clear the **Enable capture** property. Making this change prevents the ticket functionality from working.

The **Enable capture** property is an agent property that you can configure from the **Agents** window of DevTest Portal. The name of this property in the **rules.xml** file is **lisa.agent.enable.capture**.

Create Tickets in an Application

This procedure assumes that the application is running on a Java agent-enabled computer.

The following graphic shows the ticket submission dialog.

Screen capture of ticket submission dialog.

For the email functionality to work, the email settings must be configured.

If you previously created a ticket for the application and you then configure the email settings, you must restart the browser for the settings to take effect.

To enable the HP ALM - Quality Center functionality, see Send Tickets to HP ALM - Quality Center.

Before you begin, open the **Settings**, **Agents** window in DevTest Portal and ensure that the capture level for the **HTTP Server** protocol is set to **Full Data**. If the browser in which you plan to open the ticket submission dialog is running, be sure to restart the browser.

See Configure Browsers to Support Screenshots for information about supported browser and how-to steps.

Follow these steps:

1. While pressing the Alt key, click anywhere in the application window.
   The ticket submission dialog opens.

2. Click **Create a Ticket**.

3. Enter the following information:
   - **Title:** A title for the ticket.
   - **Email**: A message about the new ticket is sent to this email address. This field appears only if the email settings are configured.
   - **External Defect Tracking Number**
   - **Severity**: The options are Low, Medium, High, and Critical.
   - **Description**: This field is optional.
4. To include a screen shot of the application at the time of capture, ensure that the **Take screenshot** check box is selected.

   > **Note:** For Java 7 Update 51 and later, you must configure the Java security for the **Take screenshot** option to work.

   To configure the security for the Java 7 Update 51 and later, follow these steps:

   a. Open the **Java Control Panel** and click the **Security** tab.

   b. Click **Edit Site List...** and enter http://<machine name>:<port>.

   c. Click **Add** and **OK**.
5. If the functionality for sending tickets to HP ALM - Quality Center is enabled, select the **Raise this as a defect in Quality Center** check box.
6. Click **Submit**.
7. If the functionality for sending tickets to HP ALM - Quality Center is enabled, do the following actions:

   a. Type the user name, password, domain, and project of HP ALM - Quality Center and click **Save**.

   b. If an additional set of fields appears, enter the required information and click **Save**. A message indicates that the ticket was submitted.
8. To display the ticket in the **Manage Tickets** window, click **View Ticket**.
9. Click **Close** to close the ticket submission dialog.
10. To view the ticket in the **New Ticket Alerts** portlet of the **Home** page, click **Refresh** in the **Home** page.

Email Settings for New Tickets

When you create a ticket, Application Insight sends an email notification to a specified email address. The email includes a link to the ticket that is accessible from the **Manage Tickets** window by clicking **Link** in the **Actions** column.

Before you create a ticket, configure the following properties for the broker. To access these properties in the **Settings** tab, expand the **Tickets** category and select **Email**.

The following properties display:

- **SMTP server**
  Contains the name of the SMTP server to use for sending the email. You must use a server that allows unauthentication connection to send email.

- **From**
  Specifies the sender that appears in the email.

- **Subject**
  Specifies the subject line of the email. In the default value, the characters **%1** are replaced with the title of the ticket.

- **Body**
  Specifies the body of the email. In the default value, the characters **%1** are replaced with the link to the ticket.

Send Tickets to HP ALM - Quality Center

When you create a ticket, you can specify that the ticket is sent to HP ALM - Quality Center.

HP ALM - Quality Center 11 is supported.

To enable this functionality, you must set the configuration properties for HP ALM - Quality Center

Set the Configuration Properties for HP ALM - Quality Center

Application Insight queries the HP ALM - Quality Center instance for the required fields. These fields are added to the ticket submission dialog.

Follow these steps:

1. Open the **Agents** window.

2. In the left portion, select the target agent.

3. Click the **Settings** tab.

4. Expand the **Application Insight** category and select **Quality Center**.

5. Configure the following properties:

   - **QC host**
     The IP address or host name of the HP ALM - Quality Center REST API. For example:

     `172.24.255.255`

   - **QC port**
     The port on which the HP ALM - Quality Center REST API is listening. For example:

     `8080`

6. For https connections, select the **Use SSL** check box.

7. Save the property changes.

Keystore Configuration

For https connections, you select the **Use SSL** check box as noted in the procedure. In addition, you must configure the keystore with the server certificate.

Connect to HP ALM server and export the server certificate. One way to perform this action is by using the Certificate Export Wizard in Internet Explorer. The result is a file with the name that you specify (for example, **HPQC.cer**).

Find the JRE that the target application server is using. Then import the server certificate into the keystore that is associated with the Java Runtime Environment (JRE). The following example uses the **keytool** utility, which is included in the JRE.

```
keytool.exe -import -trustcacerts -alias ALMCERT -keystore "..\lib\security"
\cacerts -file C:\HPQC.cer
```

Restart the target application.

Manage a Ticket

The **Manage Tickets** window lets you view all the tickets that were captured and stored in the Application Insight database.

This page guides you through the steps of managing a ticket.

1. Search Tickets
2. Identify Transaction Defects
3. Obtain the Direct URL of a Ticket
4. View the Transactions for a Ticket
5. View Screen Shot for a Ticket
6. Edit Ticket Information
7. Generate Artifacts for a Ticket
8. Generate Documentation for a Ticket

Search Tickets

You search for tickets to display the defects that are found during testing from the **Manage Tickets** window. Tickets appear in descending order by date. The **Manage Tickets** window contains search criteria for finding tickets in the Application Insight database.

You can find tickets using the following search criteria:

- Reporter - the user who created the ticket
- Status - new, identified, and closed
- Duration of time - by minutes, hours, days, and all time
- Advanced - start and end date, and start and end time

- Search by entering text

The initial status of a ticket is **New**. When editing the ticket information, you can change the status to **Identified** or **Closed**. Tickets that are new or updated display in the **New Ticket Alerts** portlet in the **Home** page.

Identify Transaction Defects

You can mark frames in the transactions for a ticket to identify the origin of a defect.

Only tickets with a status of **New** can be identified.

When you identify a ticket, a red icon is placed on the frame and the background of the path graph turns red. The ticket is removed from the **New Ticket Alert** portlet of the **Home** page dashboard.

The following graphic shows an identified frame in a path graph.

Screen capture of an identified frame in a path graph.

Follow these steps:

1. Select **Application Insight**, **Manage Tickets** in the left navigation menu.
   The Manage Tickets window displays.

2. Click **Graphical** from the **Actions** column.
   The Transactions dialog opens with a visual representation of the transaction paths for the ticket.

3. To mark a frame, right-click the frame and select **Identify Problem (Update Ticket Status)**
   The status changes from New to Identified in the **Actions** column.

4. To unmark a frame, right-click the frame and select **Unidentify Problem**.
   The status changes from Identified to New in the **Actions** column.

5. (Optional) Go to the **Home** page dashboard and click **Refresh** to update the **New Ticket Alerts** portlet.

Obtain the Direct URL of a Ticket

You can obtain a URL that points directly to an existing ticket.

Follow these steps:

1. Select **Application Insight**, **Manage Tickets** in the left navigation menu.
   The Manage Tickets window displays.

2. Click **Share link** for the ticket.
   The case URL displays.

3. Copy the URL that appears in the field.

4. You can go directly to the ticket by:

   - typing the URL in a web browser

   - highlighting the link, right-clicking, and selecting **Go to http://<URL>**

View the Transactions for a Ticket

The path graph lets you view the corresponding transactions for each ticket. The paths can be displayed with the payload data to determine the cause of the defect.

Follow these steps:

1. Select **Application Insight**, **Manage Tickets** in the left navigation menu.
   The Manage Tickets window displays.

2. Click **Graphical** in the **Actions** column.
   The path graph for the transaction displays.

3. Review the transaction.

4. (Optional) Right-click a frame and select from the following options:

   - **Identify Problem (Update Ticket Status)**

   - **Generate All Artifacts**

   - **Generate Document**

   - **Unidentify Problem**

View Screen Shot for a Ticket

You can view the screen shot of the application that was taken at the time of capture.

Follow these steps:

1. Select **Application Insight**, **Manage Tickets** in the left navigation menu.
   The Manage Tickets window displays.

2. Click **View Screen shot** from the Actions column.
   The screen shot displays.

3. Click **X** to close the screen shot.

Edit Ticket Information

You can change the information for an existing ticket from the **Ticket Editor**.

The following graphic displays the **Ticket Editor** dialog.

Screen capture of Ticket Editor dialog.

Follow these steps:

1. Click **Application Insight, Manage Tickets** from the left navigation menu.
   The **Manage Tickets** window displays a list of tickets.

2. Click **Edit Ticket** in the **Actions** column for the ticket you want to edit.
   The **Ticket Editor** dialog opens.

3. Edit the appropriate fields and click **Save**.
   The top of the **Manage Tickets** window displays a message that the ticket was successfully updated.

> **Note:** If you change the **Status** from **New** to **Identified** or **Closed**, the ticket is removed from the **New Tickets Alert** portlet in the **Home** page dashboard.

Generate Artifacts for a Ticket

From the **Manage Ticket** window, you can generate the following artifacts, if applicable:

- Baseline (only expanded is supported)
- Virtual Services

The generated artifacts display for the selected project in the **Manage** menu options. The following graphic displays a baseline that was created in the Bank v5 project from the **Manage Tickets** window:

Screen capture of Manage menu options.

Follow these steps:

1. Select **Application Insight**, **Manage Tickets** in the left navigation menu.

2. Click **Graphical** , right-click a transaction, and select **Generate All Artifacts**.
   The **Generate All Artifacts** dialog opens.

3. Select the type of artifact you want to generate and click **Generate**.
   Click **Advanced** from the **Generate All Artifacts** dialog if you want to configure the options.

4. In the **Select Project** dialog, select a project from the drop-down list and click **Create**.
   A confirmation dialog displays.

5. Click **OK**.

Generate Documentation for a Ticket

You can create a PDF report for a ticket from the **Manage Tickets** window. The agent boundary for transactions display in the **Transactions Detail Information** section at the end of the report.

Follow these steps:

1. Select **Application Insight**, **Manage Tickets** in the left navigation menu.

2. Click **Graphical** .

3. Right-click a transaction and select **Generate Document**.
   The **Title for Report** dialog displays.

4. Enter a title and click **OK**.
   The report opens in a separate browser.

5. To save, click Save from the menu bar at the bottom, right of the browser, enter a title and click **OK**.

6. To print, click Print from the menu bar at the bottom, right of the browser.

Configure Browsers to Support Screenshots

Internet browsers must be configured to support screenshots when creating tickets using the Alt-click method. The latest version of JRE must be installed to avoid security issues that block browsers from other versions of JRE.

The following browsers are supported:

- Mozilla Firefox
- Google Chrome
- Internet Explorer 11

Follow these steps:

1. Install the latest version of JRE.

2. From the Windows control panel, launch the Java control panel and select **Programs**, **Java** .

3. From the **Security** tab, ensure that the website with the DevTest Java Agent is listed under the **Exception Site** list.

4. Select the **Restore Security** options.

5. When using Google Chrome, the NPAPI plug-in for Java must be enabled by performing the following steps:

   a. In Google Chrome, type chrome://plugins.

   b. For Java(TM), select **Enable** and **Always allowed to run**.

Creating Virtual Services

The Application Insight feature lets you create virtual services from transactions in the Application Insight database.

Notes:

- This feature requires a separate license.
- For detailed information about service virtualization, see Using CA Service Virtualization.

When you generate the transactions, ensure that the capture levels for the appropriate protocols are set to **Full Data**.

You can create virtual services by using the following approaches:

- **Analyze Transactions** window in DevTest Portal
- **Document Transactions** window in DevTest Portal
- Application Insight command-line tool

The procedures in this section describe how to create virtual services by using the shelf in the **Analyze Transactions** window.

You cannot create virtual services for transaction frames that have a category of **GUI**.

Consolidation of Transactions When Creating Virtual Services

When creating virtual services for transactions with the same category, Application Insight consolidates the transactions into one virtual service per category. For example, when three Web HTTP transactions are shelved, only one virtual service is created. You can view all the transactions that were consolidated into a virtual service in the shelf. Only single and merged transactions can be consolidated. All occurred transaction frames cannot be consolidated.

The consolidated virtual service name is the category name when all the transactions are consolidated into one virtual service.

When all the category frames for one agent are consolidated, the virtual service is named using the category and agent.

To display the consolidated frames for the single virtual service, click and click . You can change the order of the frames by dragging-and-dropping them up or down in the list.

Note: You have the option of deleting any of the consolidated Web HTTP transactions before creating the virtual service.

The following protocols are supported for consolidation when creating a virtual service:

- REST: one virtual service
- All SOAP: one virtual service
- All HTTP: one virtual service
- EJB: one virtual service per agent
- JDBC: one virtual service per agent

- JCA: one virtual service per agent
- TIBCO: one virtual service per agent

> **Note:** The JMS, WebSphere MQ, and SAP protocols are not supported.

Create Virtual Services from CICS Transactions

If you want to create virtual services from CICS transactions, ensure that the capture level for the CICS protocol is set to **Full Data**. See Configure Capture Levels for Agents for more information about setting up capture levels.

You create a virtual service using a VRS file or a copybook bundle file. See CICS Copybook Data Protocol and Create Copybook Bundles for more information.

Use the following procedure to create a virtual service from transactions that contain one or more CICS object frames.

The following video shows an example of creating a virtual service from CICS transactions.

Follow these steps:

1. Add a CICS transaction frame to the shelf.
2. Open the shelf and click .
3. To change the default name, select the name and make your edits, then click to save.
4. If the **Report "No Match"** and **Bypass Virtual Service** options are available, configure the response for unknown requests.
5. (Optional) When you have a custom data protocol or chain of data protocols needed to create your virtual service, you can reference an existing VRS file and have them applied when you generate your virtual service. Click **Choose** and select the file.
6. To use a copybook bundle file, perform the following steps:
   a. Select **Use Copybook Bundle** and click the down arrow to select the file.
   b. Select an encoding option from the drop-down list.
7. (Optional) To view and manage the consolidated frames, perform the following steps:
   a. Click **List frames** to view the frames.

      The frames are listed in the right pane.
   b. Drag-and-drop the frames up and down the list to change the order.

c. Click **X** and click **Continue** to remove a frame from the listing.

8. Click **Next**.

9. Select the project where the virtual service is created and (optional) edit the name prefix. The default prefix is the user name.

10. Perform one of the following steps:

    - To create the virtual service only, select **Create**.

    - To create and deploy the virtual service, select **Create and deploy** and enter the **Group Tag** and **Think Time Scale**.

11. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Finish**.
    The **Jobs in Progress** dialog opens.

12. Point to the tooltip to perform the following steps:

    - Review the information about the virtual service and error messages.

    - Select the link to navigate to the virtual service or the deployment.

Create Virtual Services from EJB Transactions

Use the following procedure to create a virtual service from a set of EJB transactions in the Application Insight database.

All methods of the same EJB are virtualized.
A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, the following options might be available for configuring the body of these responses:

- **Report "No Match"**
  Causes an exception to be raised in the virtualized application.

- **Bypass Virtual Service**
  Allows the original request to pass straight through, as if the class and method were not virtualized at all.

Follow these steps:

1. Add an EJB transaction frame to the shelf.

2. Open the shelf and click .

3. To change the default name, select the name and make your edits, then click to save.

4. If the **Report "No Match"** and **Bypass Virtual Service** options are available, configure the response for unknown requests.

6. If the **Treat all transactions as stateless** check box is editable and you want the virtual service to contain only stateless transactions, ensure that the check box is selected.

7. (Optional) When you have a custom data protocol or chain of data protocols needed to create your virtual service, you can reference an existing VRS file and have them applied when you generate your virtual service. Click **Choose** and select the file.

8. (Optional) To view and manage the consolidated frames, perform the following steps:

   a. Click **List frames** to view the frames.
   The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

   b. Drag-and-drop the frames up and down the list to change the order.

   c. Click **X** and click **Continue** to remove a frame from the listing.

9. Click **Next**.

10. Select the project where the virtual service is created and (optional) edit the name prefix. The default prefix is the user name.

11. Perform one of the following steps:

    - To create the virtual service only, select **Create**.

    - To create and deploy the virtual service, select **Create and deploy** and enter the **Group Tag** and **Think Time Scale**.

12. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Finish**.
    The **Jobs In Progress** dialog opens.

13. Point to the tooltip to perform the following steps:

    - Review the information about the virtual service and error messages.

    - Select the link to navigate to the virtual service or the deployment.

> **Note:** For more information about consolidated transactions, see Creating Virtual Services.

Example: EJB User Control Bean

The following graphic shows a path graph that includes EJB frames. The **Shelve All Occurred Transactions** action has been applied to one of the **getUser()** frames.

Screen capture of a path graph with EJB frames.

The following graphic shows the service image that was generated. The service image contains a stateless transaction. Notice that multiple methods are virtualized.

Screen capture of a generated service image.

In the virtual service model, the **Virtual Java Listener** step contains the agent name and the EJB name.

Create Virtual Services from Java Transactions

If you want to create virtual services from Java transactions, ensure that the capture level for the Java protocol is set to **Full Data**. See Configure Capture Levels for Agents for more information about setting up capture levels.

Use the following procedure to create a virtual service from transactions that contain one or more Java object frames.
A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, the following options might be available for configuring the body of these responses:

- **Report "No Match"**
  Causes an exception to be raised in the virtualized application.

- **Bypass Virtual Service**
  Allows the original request to pass straight through, as if the class and method were not virtualized at all.

Follow these steps:

1. Add a Java transaction frame to the shelf.

2. Open the shelf and click .

3. To change the default name, select the name and make your edits, then click to save.

4. If the **Report "No Match"** and **Bypass Virtual Service** options are available, configure the response for unknown requests.

5. If the **Treat all transactions as stateless** check box is editable and you want the virtual service to contain only stateless transactions, ensure that the check box is selected.

6. (Optional) When you have a custom data protocol or chain of data protocols needed to create your virtual service, you can reference an existing VRS file and have them applied when you generate your virtual service. Click **Choose** and select the file.

7. (Optional) To view and manage the consolidated frames, perform the following steps:

   a. Click **List frames** to view the frames.
   The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

   b. Drag-and-drop the frames up and down the list to change the order.

   c. Click **X** and click **Continue** to remove a frame from the listing.

8. Click **Next**

9. Select the project where the virtual service is created and (optional) edit the name prefix. The default prefix is the user name.

10. Perform one of the following steps:

    - To create the virtual service only, select **Create**.

    - To create and deploy the virtual service, select **Create and deploy** and enter the **Group Tag** and **Think Time Scale**.

11. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Finish**.
    The **Jobs in Progress** dialog opens.

12. Point to the tooltip to perform the following steps:
    - Review the information about the virtual service and error messages.
    - Select the link to navigate to the virtual service or the deployment.

Create Virtual Services from JCA Transactions

Use the following procedure to create a virtual service from a set of JCA transactions in the Application Insight database.
A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, the following options might be available for configuring the body of these responses:

- **Report "No Match"**
  Causes an exception to be raised in the virtualized application.
- **Bypass Virtual Service**
  Allows the original request to pass straight through, as if the class and method were not virtualized at all.

Follow these steps:

1. Add a JCA transaction frame to the shelf.
2. Open the shelf and click .
3. To change the default name, select the name and make your edits, then click to save.
4. If the **Report "No Match"** and **Bypass Virtual Service** options are available, configure the response for unknown requests.
5. If the **Treat all transactions as stateless** check box is editable and you want the virtual service to contain only stateless transactions, ensure that the check box is selected.
6. (Optional) When you have a custom data protocol or chain of data protocols needed to create your virtual service, you can reference an existing VRS file and have them applied when you generate your virtual service. Click **Choose** and select the file.
7. (Optional) To view and manage the consolidated frames, perform the following steps:
   a. Click **List frames** to view the frames.
   The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.
   b. Drag-and-drop the frames up and down the list to change the order.
   c. Click **X** and click **Continue** to remove a frame from the listing.

8. Click **Next**.

9. Select the project where the virtual service is created and (optional) edit the name prefix. The default prefix is the user name.

10. Perform one of the following steps:

    - To create the virtual service only, select **Create**.

    - To create and deploy the virtual service, select **Create and deploy** and enter the **Group Tag** and **Think Time Scale**.

11. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Finish**.
    The **Jobs in Progress** dialog opens.

12. Point to the tooltip to perform the following steps:

    - Review the information about the virtual service and error messages.

    - Select the link to navigate to the virtual service or the deployment.

> **Note:** For more information about consolidated transactions, see Creating Virtual Services.

Create Virtual Services from JDBC Transactions

Use the following procedure to create a virtual service from a set of JDBC transactions in the Application Insight database.
A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, the following options might be available for configuring the body of these responses:

- **Report "No Match"**
  Causes an exception to be raised in the virtualized application.

- **Bypass Virtual Service**
  Allows the original request to pass straight through, as if the class and method were not virtualized at all.

Follow these steps:

1. Add a JDBC transaction frame to the shelf.

2. Open the shelf and click .

3. To change the default name, select the name and make your edits, then click to save.

4. If the **Report "No Match"** and **Bypass Virtual Service** options are available, configure the response for unknown requests.

5. If the **Treat all transactions as stateless** check box is editable and you want the virtual service to contain only stateless transactions, ensure that the check box is selected.

6. (Optional) When you have a custom data protocol or chain of data protocols needed to create your virtual service, you can reference an existing VRS file and have them applied when you generate your virtual service. Click **Choose** and select the file.

7. (Optional) To view and manage the consolidated frames, perform the following steps:

   a. Click **List frames** to view the frames.
   The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

   b. Drag-and-drop the frames up and down the list to change the order.

   c. Click **X** and click **Continue** to remove a frame from the listing.

8. Click **Next**.

9. Select the project where the virtual service is created and (optional) edit the name prefix. The default prefix is the user name.

10. Perform one of the following steps:

    - To create the virtual service only, select **Create**.

    - To create and deploy the virtual service, select **Create and deploy** and enter the **Group Tag** and **Think Time Scale**.

11. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Finish**.
    The **Jobs in Progress** dialog opens.

12. Point to the tooltip to perform the following steps:

    - Review the information about the virtual service and error messages.

    - Select the link to navigate to the virtual service or the deployment.

> **Note:** For more information about consolidated transactions, see Creating Virtual Services.

Create Virtual Services from JMS Transactions

Use the following procedure to create a virtual service from a set of JMS transactions in the Application Insight database.

> **Note:** This feature is supported for configurations in which JNDI is used to obtain the JMS connection factory. This feature is also supported for configurations in which a direct API from IBM WebSphere MQ is used to obtain the JMS connection factory.

Follow these steps:

1. Add a JMS transaction frame to the shelf. Be sure to select a frame that corresponds to the start of the JMS transaction. Do not select a response frame.

2. Open the shelf and click .

3. To change the default name, select the name and make your edits, then click to save.

4. If the **Treat all transactions as stateless** check box is editable and you want the virtual service to contain only stateless transactions, ensure that the check box is selected.

5. (Optional) When you have a custom data protocol or chain of data protocols needed to create your virtual service, you can reference an existing VRS file and have them applied when you generate your virtual service. Click **Choose** and select the file.

6. (Optional) To view and manage the consolidated frames, perform the following steps:

   a. Click **List frames** to view the frames.
   The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

   b. Drag-and-drop the frames up and down the list to change the order.

   c. Click **X** and click **Continue** to remove a frame from the listing.

7. Click **Next**.

8. Select the project where the virtual service is created and (optional) edit the name prefix. The default prefix is the user name.

9. Perform one of the following steps:

   - To create the virtual service only, select **Create**.

   - To create and deploy the virtual service, select **Create and deploy** and enter the **Group Tag** and **Think Time Scale**.

10. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Finish**.
    The **Jobs in Progress** dialog opens.

11. Point to the tooltip to perform the following steps:

    - Review the information about the virtual service and error messages.

- Select the link to navigate to the virtual service or the deployment.

Create Virtual Services from REST Transactions

Use the following procedure to create a virtual service from a set of REST transactions in the Application Insight database.

Follow these steps:

1. Add a REST transaction frame to the shelf.
2. Open the shelf and click .
3. To change the default name, select the name and make your edits, then click to save.
4. If the **Treat all transactions as stateless** check box is editable and you want the virtual service to contain only stateless transactions, ensure that the check box is selected.
5. (Optional) When you have a custom data protocol or chain of data protocols needed to create your virtual service, you can reference an existing VRS file and have them applied when you generate your virtual service. Click **Choose** and select the file.
6. (Optional) To view and manage the consolidated frames, perform the following steps:
   a. Click **List frames** to view the frames.
   The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.
   b. Drag-and-drop the frames up and down the list to change the order.
   c. Click **X** and click **Continue** to remove a frame from the listing.
7. Click **Next**.
8. Select the project where the virtual service is created and (optional) edit the name prefix. The default prefix is the user name.
9. Perform one of the following steps:
   - To create the virtual service only, select **Create**.
   - To create and deploy the virtual service, select **Create and deploy** and enter the **Group Tag** and **Think Time Scale**.
10. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Finish**.
    The **Jobs in Progress** dialog opens.
11. Point to the tooltip to perform the following steps:

- Review the information about the virtual service and error messages.
- Select the link to navigate to the virtual service or the deployment.

> **Note:** For more information about consolidated transactions, see Creating Virtual Services.

Create Virtual Services from RMI Transactions

Use the following procedure to create a virtual service from a set of RMI transactions in the Application Insight database.

A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, the following options might be available for configuring the body of these responses:

- **Report "No Match"**
  Causes an exception to be raised in the virtualized application.
- **Bypass Virtual Service**
  Allows the original request to pass straight through, as if the class and method were not virtualized at all.

Follow these steps:

1. Add an RMI transaction frame to the shelf.
2. Open the shelf and click .
3. To change the default name, select the name and make your edits, then click to save.
4. If the **Report "No Match"** and **Bypass Virtual Service** options are available, configure the response for unknown requests.
5. If the **Treat all transactions as stateless** check box is editable and you want the virtual service to contain only stateless transactions, ensure that the check box is selected.
6. (Optional) When you have a custom data protocol or chain of data protocols needed to create your virtual service, you can reference an existing VRS file and have them applied when you generate your virtual service. Click **Choose** and select the file.
7. (Optional) To view and manage the consolidated frames, perform the following steps:
   a. Click **List frames** to view the frames.
   The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

b. Drag-and-drop the frames up and down the list to change the order.

c. Click **X** and click **Continue** to remove a frame from the listing.

8. Click **Next**.

9. Select the project where the virtual service is created and (optional) edit the name prefix. The default prefix is the user name.

10. Perform one of the following steps:

    - To create the virtual service only, select **Create**.

    - To create and deploy the virtual service, select **Create and deploy** and enter the **Group Tag** and **Think Time Scale**.

11. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Finish**.
    The **Jobs in Progress** dialog opens.

12. Point to the tooltip to perform the following steps:

    - Review the information about the virtual service and error messages.

    - Select the link to navigate to the virtual service or the deployment.

Note: For more information about consolidated transactions, see Creating Virtual Services.

Create Virtual Services from SAP ERPConnect Transactions

ERPConnect is a library that lets you interact with SAP from .NET applications.

You can create a virtual service from a set of ERPConnect transactions in the Application Insight database.

The general steps are as follows:

1. Install and configure the agent files.

2. Exercise the .NET application.

3. Generate the virtualization artifacts.

4. Deploy the virtual service.

Notes:

- ERPConnect is not included with DevTest Solutions.

- For information about deploying the virtual service, see Using CA Service Virtualization.

Install and Configure the Agent Files for SAP ERPConnect

Use the following procedure to install and configure the agent files for SAP ERPConnect.

The following versions of the .NET Framework are supported:

- .NET 2.0 Framework, v2.0.50727
- .NET 3.0 Framework, v3.0.4506
- .NET 3.5 Framework, v3.5.21022

Before you start, install Visual C++ Redistributable for Visual Studio 2012 Update 4. As of this writing, you can obtain this component from the Microsoft web site.

The procedure varies depending on which of the following .NET applications you have:

- .NET application that is running in an Internet Information Services (IIS) server
- Stand-alone .NET application

To install and configure the agent files for SAP ERPConnect when you have a .NET application that is running in an IIS server:

1. Copy the following files from the **LISA_HOME\agent** directory to a directory on the computer where the IIS server is located:
   - **LisaAgent.dll**
   - **NativeAgent32.dll**
   - **NativeAgent64.dll**
   - **LisaAgentLauncher.exe**
2. Open a command prompt from the directory where you copied the files.
3. Run the following command:

   ```
   LisaAgentLauncher.exe /i
   ```

   The agent files are registered on the computer.
4. Run the following command:

   ```
   LisaAgentLauncher.exe /iis /name <agent-name> /url tcp://<broker-host>:
   <broker-port>
   ```

   To specify a domain, add the **/domain** option.
5. When the worker process for IIS starts, the agent is enabled.

To install and configure the agent files for SAP ERPConnect when you have a stand-alone .NET application:

1. Copy the following files from the **LISA_HOME\agent** directory to the directory that contains the executable for the .NET application:

- **LisaAgent.dll**
- **NativeAgent32.dll**
- **NativeAgent64.dll**
- **LisaAgentLauncher.exe**

2. Open a command prompt from the directory where you copied the files.
3. Run the following command:

   ```
   LisaAgentLauncher.exe /i
   ```

   The agent files are registered on the computer.
4. Configure the environment variables as follows:

   ```
   set COR_PROFILER={BEB45448-91FA-4A7C-BF9A-68AA889DC873}
   set COR_ENABLE_PROFILING=1
   set COR_LISA_AGENT=name=<agent-name>
   ```
5. Restart the .NET application.

Generate the SAP ERPConnect Virtualization Artifacts

Use the following procedure to create a virtual service from a set of ERPConnect transactions in the Application Insight database.
A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, the following options might be available for configuring the body of these responses:

- **Report "No Match"**
  Causes an exception to be raised in the virtualized application.
- **Bypass Virtual Service**
  Allows the original request to pass straight through, as if the class and method were not virtualized at all.

Follow these steps:

1. Add an SAP transaction frame to the shelf.
2. Open the shelf and click .
3. To change the default name, select the name and make your edits, then click to save.
4. If the **Report "No Match"** and **Bypass Virtual Service** options are available, configure the response for unknown requests.
6. If the **Treat all transactions as stateless** check box is editable and you want the virtual service to contain only stateless transactions, ensure that the check box is selected.
7. (Optional) When you have a custom data protocol or chain of data protocols needed to create your virtual service, you can reference an existing VRS file and have them applied when you generate your virtual service. Click **Choose** and select the file.

8. Click **Next**.

9. Select the project where the virtual service is created and (optional) edit the name prefix. The default prefix is the user name.

10. Perform one of the following steps:

    - To create the virtual service only, select **Create**.

    - To create and deploy the virtual service, select **Create and deploy** and enter the **Group Tag** and **Think Time Scale**.

11. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Finish**.
    The **Jobs in Progress** dialog opens.

12. Point to the tooltip to perform the following steps:

    - Review the information about the virtual service and error messages.

    - Select the link to navigate to the virtual service or the deployment.

Create Virtual Services from SAP IDoc Transactions

You can create virtual services from SAP IDoc transactions.

The following graphic shows the architecture of the recording phase. The DevTest Java Agent is configured for a Java application. The Java application sends an IDoc message to an SAP system. The agent observes the method calls and generates a corresponding path that can be viewed in DevTest Portal.

Stateful conversations are not supported.

You do not need to install and configure the agent on the SAP system.

After the recording phase, you generate the virtualization artifacts and deploy the virtual service.

> **Note:** For information about installing and configuring the agent, see Agents. For information about deploying the virtual service, see Using CA Service Virtualization.

For information about troubleshooting SAP connections, see SAP Connectivity Troubleshooting from the DevTest Communities.

Generate the SAP IDoc Virtualization Artifacts

Use the following procedure to generate the SAP IDoc virtualization artifacts.
A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, the following options might be available for configuring the body of these responses:

- **Report "No Match"**
  Causes an exception to be raised in the virtualized application.

- **Bypass Virtual Service**
  Allows the original request to pass straight through, as if the class and method were not virtualized at all.

Follow these steps:

1. Add an SAP transaction frame to the shelf.

2. Open the shelf and click .

3. To change the default name, select the name and make your edits, then click to save.

4. If the **Report "No Match"** and **Bypass Virtual Service** options are available, configure the response for unknown requests.

5. If the **Treat all transactions as stateless** check box is editable and you want the virtual service to contain only stateless transactions, ensure that the check box is selected.

6. (Optional) When you have a custom data protocol or chain of data protocols needed to create your virtual service, you can reference an existing VRS file and have them applied when you generate your virtual service. Click **Choose** and select the file.

7. Click **Next**.

8. Select the project where the virtual service is created and (optional) edit the name prefix. The default prefix is the user name.

9. Perform one of the following steps:

   - To create the virtual service only, select **Create**.

   - To create and deploy the virtual service, select **Create and deploy** and enter the **Group Tag** and **Think Time Scale**.

10. Select an option to keep or delete the transactions in the shelf once the virtual service is created.

11. Select the project where the virtual service will be created and click **Finish**.
    The **Jobs in Progress** dialog opens.

12. Point to the tooltip to perform the following steps:

    - Review the information about the virtual service and error messages.

    - Select the link to navigate to the virtual service or the deployment.

SAP IDoc Destination Filtering

You can change which SAP IDoc destinations are virtualized.

In the virtual service model that is generated, open the **Virtual Java Listener** step and double-click **SAP** protocol in the lower right list. The **Protocol Configuration** dialog opens. This dialog contains a set of key/value pairs.

The following graphic shows the **Protocol Configuration** dialog.

Screen capture of Protocol Configuration dialog.

The key that begins with **destination.** and ends with a number specifies which destination to virtualize.

To virtualize multiple destinations, add a destination key for each one. You can use any number, as long as the number is unique in the **Protocol Configuration** dialog.

To virtualize all destinations, remove the destination key.

You can ignore the keys that resemble Java class names.

Create Virtual Services from SAP JCo Transactions

You can create virtual services from SAP JCo transactions.

The following graphic shows the architecture of the recording phase. The DevTest Java Agent is configured for a Java application. The Java application uses SAP JCo to make an RFC call to a function in an SAP system. The agent observes the call and generates a corresponding path that can be viewed in DevTest Portal.

This feature supports SAP JCo 3.0 only.

This feature supports RFC calls with input and output parameters and table data, and stateful calls.

You do not need to install and configure the agent on the SAP system.

After the recording phase, you generate the virtualization artifacts and deploy the virtual service.

Note: For information about installing and configuring the agent, see Agents. For information about deploying the virtual service, see Using CA Service Virtualization.

More Info

- Generate the SAP JCo Virtualization Artifacts
- SAP JCo Destination Filtering
- SAP JCo Connection Settings in Detail from the DevTest Communities
- SAP Connectivity Troubleshooting from the DevTest Communities

Generate the SAP JCo Virtualization Artifacts

Use the following procedure to generate the SAP JCo virtualization artifacts.
A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, the following options might be available for configuring the body of these responses:

- **Report "No Match"**
  Causes an exception to be raised in the virtualized application.

- **Bypass Virtual Service**
  Allows the original request to pass straight through, as if the class and method were not virtualized at all.

The following graphic shows a path graph that includes SAP frames. This path graph illustrates the standard SAP JCo pattern of getting a function and then executing the function.

Screen capture of SAP frames in the path graph.

When the virtualization artifacts are generated, Application Insight checks the destination name of the selected SAP component. Application Insight then searches all of the visible paths for SAP components that have this destination. These SAP components are used to create the virtual service.

To view the destination name of the selected SAP component, click the **XML** tab in the transaction details dialog.

Follow these steps:

1. Add an SAP transaction frame to the shelf.

2. Open the shelf and click .

3. To change the default name, select the name and make your edits, then click to save.

4. If the **Report "No Match"** and **Bypass Virtual Service** options are available, configure the response for unknown requests.

5. If the **Treat all transactions as stateless** check box is editable and you want the virtual service to contain only stateless transactions, ensure that the check box is selected.

6. (Optional) When you have a custom data protocol or chain of data protocols needed to create your virtual service, you can reference an existing VRS file and have them applied when you generate your virtual service. Click **Choose** and select the file.

7. Click **Next**.

8. Select the project where the virtual service is created and (optional) edit the name prefix. The default prefix is the user name.

9. Perform one of the following steps:

   - To create the virtual service only, select **Create**.

   - To create and deploy the virtual service, select **Create and deploy** and enter the **Group Tag** and **Think Time Scale**.

10. Select an option to keep or delete the transactions in the shelf once the virtual service is created.

11. Select the project where the virtual service will be created and click **Finish**.
    The **Jobs in Progress** dialog opens.

12. Point to the tooltip to perform the following steps:

    - Review the information about the virtual service and error messages.

    - Select the link to navigate to the virtual service or the deployment.

SAP JCo Destination Filtering

You can change which SAP JCo destinations are virtualized.

In the virtual service model that is generated, open the **Virtual Java Listener** step and double-click **SAP** protocol in the lower right list. The **Protocol Configuration** dialog opens. This dialog contains a set of key/value pairs.

The following graphic shows the **Protocol Configuration** dialog.

Screen capture of Protocol Configuration dialog.

The key that begins with **destination.** and ends with a number specifies which destination to virtualize.

To virtualize multiple destinations, add a destination key for each one. You can use any number, as long as the number is unique in the **Protocol Configuration** dialog.

To virtualize all destinations, remove the destination key.

You can ignore the keys that resemble Java class names.

Create Virtual Services from TIBCO ActiveMatrix BusinessWorks Transactions

You can use Application Insight to virtualize a process in TIBCO ActiveMatrix BusinessWorks. When the virtual service is deployed, it substitutes the behavior of process activities with the responses collected in the service image.

The following approaches are supported:

- Virtualize from DevTest Workstation. You use the Virtual Service Image Recorder to capture traffic and generate the virtual service. You do not need to enable Application Insight.
- Virtualize from DevTest Portal. You generate transactions that appear in DevTest Portal. At any time afterward, you can use the transactions to create the virtual service. Application Insight must be enabled. This approach provides more flexibility, but has greater overhead than the first approach.

You can generate virtual services that are stateful or stateless. Stateless is recommended. You can control this setting from the Virtual Service Image Recorder and DevTest Portal.
A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, the following options might be available for configuring the body of these responses:

- **Report "No Match"**
  Causes an exception to be raised in the virtualized application.
- **Bypass Virtual Service**
  Allows the original request to pass straight through, as if the class and method were not virtualized at all.

These procedures assume that the DevTest Java Agent was installed and configured on TIBCO ActiveMatrix BusinessWorks.

Note: For detailed information about installing and configuring the DevTest Java Agent, see Agents. For detailed information about using the Virtual Service Image Recorder and deploying virtual services, see Using CA Service Virtualization.

To virtualize TIBCO ActiveMatrix BusinessWorks from DevTest Workstation:

1. Go to DevTest Workstation and start the Virtual Service Image Recorder.
   You are prompted to provide basic information.

2. Specify the names of the service image and virtual service model.

3. Set the transport protocol to **Java**.

4. Click **Next**.
   You are prompted to select the Java classes to virtualize.

5. Select the agent and move it into the **Connected Agents** list.

6. Expand the **Protocols** arrow and move the TIBCO BW protocol into the right pane.

7. Click **Next**. If necessary, trigger the execution of the TIBCO processes that you want to record. As the processes run, the agent records the activities.

8. When the recording has completed, click **Next** and perform the remaining steps in the Virtual Service Image Recorder. You do not need to select a data protocol.
   A service image and virtual service model are created.

9. Deploy and start the virtual service model.

To virtualize TIBCO ActiveMatrix BusinessWorks from DevTest Portal:

1. Trigger the execution of the TIBCO processes.

2. Add a TIBCO transaction frame to the shelf. You can virtualize the entire conversation by selecting the leftmost frame. You can virtualize a specific activity by selecting the frame that includes the activity name.

3. Open the shelf and click .

4. To change the default name, select the name and make your edits, then click to save.

5. If the **Report "No Match"** and **Bypass Virtual Service** options are available, configure the response for unknown requests.

6. If the **Treat all transactions as stateless** check box is editable and you want the virtual service to contain only stateless transactions, ensure that the check box is selected.

7. (Optional) When you have a custom data protocol or chain of data protocols needed to create your virtual service, you can reference an existing VRS file and have them applied when you generate your virtual service. Click **Choose** and select the file.

8. (Optional) To view and manage the consolidated frames, perform the following steps:

a. Click **List frames** to view the frames.
The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

b. Drag-and-drop the frames up and down the list to change the order.

c. Click **X** and click **Continue** to remove a frame from the listing.

9. Click **Next**.

10. Select the project where the virtual service is created and (optional) edit the name prefix. The default prefix is the user name.

11. Perform one of the following steps:

    - To create the virtual service only, select **Create**.

    - To create and deploy the virtual service, select **Create and deploy** and enter the **Group Tag** and **Think Time Scale**.

12. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Finish**.
The **Jobs in Progress** dialog opens.

13. Point to the tooltip to perform the following steps:

    - Review the information about the virtual service and error messages.

    - Select the link to navigate to the virtual service or the deployment.

Note: For more information about consolidated transactions, see Creating Virtual Services.

Example: Virtualize an Add User Process

The following graphic shows a process definition in TIBCO Designer. The process contains the following activities: File Poller, JMS Queue Sender, Confirm, and End.

Screen capture of process definition example

The File Poller activity monitors a text file. When the file is changed, the activity starts the process.

The JMS Queue Sender activity sends a message to the specified queue.

The following graphic shows a service image that the Virtual Service Image Recorder generated. A conversation with four nodes appears in the **Transactions** tab. The four nodes correspond to the four activities in the process definition.

Screen capture of a generated service image

Each node in the conversation includes an operation field. The value of this field consists of the fully qualified process name and the activity name. In the preceding graphic, the value for the selected node is **Tibco:VSE/Processes/AddUser.JMS Queue Sender**.

The virtual service model is the default for the Java transport protocol.

Create Virtual Services from Web HTTP Transactions

Use the following procedure to create a virtual service from a set of Web HTTP transactions in the Application Insight database.

Follow these steps:

1. Add a Web HTTP transaction to the shelf.
2. Open the shelf click .
3. To change the default name, select the name and make your edits, then click to save.
4. If the **Treat all transactions as stateless** check box is editable and you want the virtual service to contain only stateless transactions, ensure that the check box is selected.

5. (Optional) When you have a custom data protocol or chain of data protocols needed to create your virtual service, you can reference an existing VRS file and have them applied when you generate your virtual service. Click **Choose** and select the file.

6. (Optional) To view and manage the consolidated frames, perform the following steps:

   a. Click **List frames** to view the frames.
   The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

   b. Drag-and-drop the frames up and down the list to change the order.

   c. Click **X** and click **Continue** to remove a frame from the listing.

7. Click **Next**.

8. Select the project where the virtual service is created and (optional) edit the name prefix. The default prefix is the user name.

9. Perform one of the following steps:

   - To create the virtual service only, select **Create**.

   - To create and deploy the virtual service, select **Create and deploy** and enter the **Group Tag** and **Think Time Scale**.

10. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Finish**.
    The **Jobs in Progress** dialog opens.

11. Point to the tooltip to perform the following steps:

    - Review the information about the virtual service and error messages.

    - Select the link to navigate to the virtual service or the deployment.

> **Note:** For more information about consolidated transactions, see Creating Virtual Services.

Create Virtual Services from Web Service Transactions

Use the following procedure to create a virtual service from a set of web service transactions in the Application Insight database.

Web services with attachments are not supported.

Follow these steps:

1. Add a SOAP transaction frame to the shelf.

2. Open the shelf and click .

3. To change the default name, select the name and make your edits, then click to save.

4. If the **Treat all transactions as stateless** check box is editable and you want the virtual service to contain only stateless transactions, ensure that the check box is selected.

5. (Optional) When you have a custom data protocol or chain of data protocols needed to create your virtual service, you can reference an existing VRS file and have them applied when you generate your virtual service. Click **Choose** and select the file.

6. (Optional) To view and manage the consolidated frames, perform the following steps:

   a. Click **List frames** to view the frames.
   The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

   b. Drag-and-drop the frames up and down the list to change the order.

   c. Click **X** and click **Continue** to remove a frame from the listing.

7. Click **Next**.

8. Select the project where the virtual service is created and (optional) edit the name prefix. The default prefix is the user name.

9. Perform one of the following steps:

   - To create the virtual service only, select **Create**.

   - To create and deploy the virtual service, select **Create and deploy** and enter the **Group Tag** and **Think Time Scale**.

10. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Finish**.
    The **Jobs in Progress** dialog opens.

11. Point to the tooltip to perform the following steps:

    - Review the information about the virtual service and error messages.

    - Select the link to navigate to the virtual service or the deployment.

Note: For more information about consolidated transactions, see Creating Virtual Services.

Create Virtual Services from webMethods Transactions

Use the following procedure to create a virtual service from a set of webMethods Integration Server transactions in the Application Insight database.

webMethods Integration Server includes the concepts of a flow service and a pipeline. A flow service lets you encapsulate a group of services and manage the flow of data among them. The pipeline is a data structure that contains the input and output values for a flow service. You can add steps to a flow service to perform such actions as invoking services and changing data in the pipeline.

The flow service is the unit that is virtualized.
A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, the following options might be available for configuring the body of these responses:

- **Report "No Match"**
  Causes an exception to be raised in the virtualized application.

- **Bypass Virtual Service**
  Allows the original request to pass straight through, as if the class and method were not virtualized at all.

Follow these steps:

1. Add a webMethods Integration Server transaction frame to the shelf.

2. Open the shelf and click .

3. To change the default name, select the name and make your edits, then click to save.

4. If the **Report "No Match"** and **Bypass Virtual Service** options are available, configure the response for unknown requests.

5. If the **Treat all transactions as stateless** check box is editable and you want the virtual service to contain only stateless transactions, ensure that the check box is selected.

6. (Optional) When you have a custom data protocol or chain of data protocols needed to create your virtual service, you can reference an existing VRS file and have them applied when you generate your virtual service. Click **Choose** and select the file.

7. (Optional) To view and manage the consolidated frames, perform the following steps:

   a. Click **List frames** to view the frames.
   The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

   b. Drag-and-drop the frames up and down the list to change the order.

   c. Click **X** and click **Continue** to remove a frame from the listing.

8. Click **Next**.

9. Select the project where the virtual service is created and (optional) edit the name prefix. The default prefix is the user name.

10. Perform one of the following steps:

    - To create the virtual service only, select **Create**.

    - To create and deploy the virtual service, select **Create and deploy** and enter the **Group Tag** and **Think Time Scale**.

11. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Finish.**
    The **Jobs in Progress** dialog opens.

12. Point to the tooltip to perform the following steps:

    - Review the information about the virtual service and error messages.

    - Select the link to navigate to the virtual service or the deployment.

Note: For more information about consolidated transactions, see Creating Virtual Services.

Example: Update User

The following graphic shows a path graph that includes webMethods Integration Server frames. The flow service in this example is named **updateUser**. The purpose is to send updated information for a user to a database.

Note: If you select any of the webMethods Integration Server frames, you can view the pipeline data in the transaction detail dialog.

When a path graph has multiple webMethods Integration Server frame, you must select one frame for virtualization. The decision of which frame to select depends on the system under test. This example could result in the following decisions:

- To ensure that your software interfaces with a webMethods process, select the leftmost webMethods Integration Server frame.

- To remove the database, select the webMethods Integration Server frame that directly calls the database.

Create Virtual Services from WebSphere MQ Transactions

Use the following procedure to create a virtual service from a set of WebSphere MQ transactions in the Application Insight database.

Follow these steps:

1. Add a WebSphere MQ transaction frame to the shelf. Be sure to select a frame that corresponds to the start of the MQ transaction. Do not select a response frame.
2. Open the shelf and click .
3. To change the default name, select the name and make your edits, then click to save.
4. If the **Treat all transactions as stateless** check box is editable and you want the virtual service to contain only stateless transactions, ensure that the check box is selected.
5. (Optional) When you have a custom data protocol or chain of data protocols needed to create your virtual service, you can reference an existing VRS file and have them applied when you generate your virtual service. Click **Choose** and select the file.
6. (Optional) To view and manage the consolidated frames, perform the following steps:
   a. Click **List frames** to view the frames.
   The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.
   b. Drag-and-drop the frames up and down the list to change the order.
   c. Click **X** and click **Continue** to remove a frame from the listing.
7. Click **Next**.
8. Select the project where the virtual service is created and (optional) edit the name prefix. The default prefix is the user name.
9. Perform one of the following steps:
   - To create the virtual service only, select **Create**.
   - To create and deploy the virtual service, select **Create and deploy** and enter the **Group Tag** and **Think Time Scale**.
10. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Finish**.
    The **Jobs in Progress** dialog opens.

11. Point to the tooltip to perform the following steps:
    - Review the information about the virtual service and error messages.
    - Select the link to navigate to the virtual service or the deployment.

VRS Files in Application Insight

The final step of the Virtual Service Image Recorder in DevTest Workstation lets you create a recording session file. The file extension is .vrs.

The .vrs file contains information about the recording, such as the transport protocol, the data protocols, and associated settings.

When you create a virtual service from transactions in DevTest Portal or from the Application Insight command-line tool, you can select a .vrs file. The virtual service uses the settings in the file.

For the command-line tool, the **--vrs** option lets you specify the name and location of the file. For example:

```
PFCmdLineTool.exe --virtualize=BeanAddaddress_vrs --from=2015-07-21T15:00:00 --
to=2015-07-22T15:02:00 --to-dir=C:\test --frame-name="/itkoExamples
/EJB3UserControlBean.invoke" --vrs=C:\test.vrs
```

The transport protocol in the .vrs file must be the same as the protocol of the selected frames to be virtualized.

The base path in HTTP/S recording session files is ignored. Instead, the base path is calculated based on the selected frames to be virtualized.

Creating Baselines

A *baseline* is a test case that helps you determine whether a system is still functioning the same way at a later time.

> **Note:** This feature requires a separate license.

Baseline Creation Approaches

You can create baselines by using any of the following approaches:

- Generate transactions with Application Insight and use them to create a baseline from the **Analyze Transactions** window
- Generate transactions with Application Insight and use them to create a baseline with the command-line tool
- Create a virtual service and specify that you want to create a baseline at the same time
- Create a baseline from an existing virtual service

- Create a baseline from a Swagger 2.0 document

The procedures in this section describe how to create baselines from the **Analyze Transactions** window.

For information about the virtual service approaches, see Create Baseline from Virtual Service.

For information about the Swagger approach, see Quick Start Window.

Workflow

The following graphic provides a high-level view of how baselines work for the first two approaches.

The left portion shows the initial regression cycle. Manual testing is performed on a system. Application Insight captures the activity and creates transactions, which are then used to generate baselines.

The right portion shows the subsequent automated regression tests. The baselines are run as necessary against the system under test.

Each baseline contains information about the expected response from the system. During an automated regression cycle, the baseline compares the expected response with the actual response. If the responses do not match, the differences are highlighted.

When you generate the transactions, ensure that the capture levels for the appropriate protocols are set to **Full Data**.

You cannot create baselines for transaction frames that have a category of **GUI**.

Baseline Types

You can create the following types of baselines:

- Stateful

- Data-driven
- Expanded

The naming convention for a baseline is **(prefix)_(transactionframename)-(baselinetype)-(date)-(time)**. The valid values of the baseline type portion are S (stateful), D (data-driven), and E (expanded). For example, the following name is for an expanded baseline:

```
admin_1445436734664_lisabankbuttonclick.do-E-20151020-145034
```

If you add a merged transaction frame to the shelf, you cannot create a stateful baseline. You can create data-driven and expanded baselines.

In some cases, the baseline type is automatically selected. One such case is when you generate all artifacts for an agent in the **Analyze Transactions** window.

- Stateful Baselines
- Data-driven Baselines
- Expanded Baselines

Stateful Baselines

A stateful baseline is a baseline test case that applies to an entire conversation or session, instead of one of its transactions. A conversation is a set of transactions in a session.

Stateful baselines are supported for the following categories of transaction frame:

- EJB
- REST
- Web HTTP
- Web service

The following graphic shows a stateful baseline.

Screen capture of a stateful baseline.

Data-driven Baselines

A data-driven baseline is composed of a single test case. The test case includes a test step that reads from a Large Data data set.

The following graphic shows a data-driven baseline.

Screen capture of a data-driven baseline.

Data-driven baselines are useful when you need to modify a baseline after it is generated.

For example, assume that you shelve a merged transaction frame that has a large number of repeated paths. If you create an expanded baseline, you might need to make the same change to all the test cases in the baseline. If you create a data-driven baseline, you can make the change just once.

Expanded Baselines

An expanded baseline is composed of a suite of test cases and a suite document for running the test cases.

In DevTest Workstation, the test cases are located in the **Baselines** folder of the **Projects** panel. The suite document is located in the **Suites** folder.

The following graphic shows a test case in an expanded baseline.

Screen capture of a test case in an expanded baseline.

EJB Baselines

This section contains the following pages that provide information about EJB baselines:

- Generate an EJB Baseline
- EJB Baseline Contents
- Workaround for EJB Baseline Over SSL on WebSphere App Server 8.5

Generate an EJB Baseline

Use the following procedure to generate an EJB baseline from a set of transactions in the Application Insight database.

Follow these steps:

1. Add one or more EJB transaction frames to the shelf.
2. Open the shelf and click .

3. To change the default name, click , edit the name, then click to save.

4. Click the plus sign that appears to the left of the name.

6. If the creation mode field is available, select the baseline type.

7. If the step generation field is available, select one of the following options:

   - **Use Application Test Steps**

     The baseline includes a step that corresponds to the transaction frame that you added to the shelf. For example, selecting a SOAP frame results in a baseline that includes a **Web Service Execution (XML)** step. The step makes the client calls to the system under test at runtime.

   - **Use Transaction Frame Step**

     The baseline includes an **Execute Transaction Frame** step. The system under test is not called at runtime. Instead, the agent plays back the transaction inside the JVM memory.

8. (Optional) Configure the baseline to use magic dates.

   - **Apply magic dates to test cases**

     Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.

9. (Optional) Click **Choose File** and select a test template for the baseline.

10. If the following fields are available for editing, configure them:

    - **JNDI Factory**
      The JNDI factory class to use when generating EJB baseline tests.

    - **JNDI URL**
      The JNDI URL to use when generating EJB baseline tests.

    - **Security Principal**
      The JNDI user to use when generating EJB baseline tests.

    - **Security Credentials**
      The JNDI password to use when generating EJB baseline tests.

11. (Optional) To view and manage merged frames, perform the following steps:

    a. Click **List frames** to view the frames.
       The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

b. Drag-and-drop the frames up and down the list to change the order.

c. Click **X** and click **Continue** to remove a frame from the listing.

12. Click **Next**.

13. Select the project where the baseline will be created.

14. (Optional) Edit name prefix. The default prefix is the user name.

15. Select an option to keep or delete the transactions in the shelf once the baseline is created and click **Finish**.
    The **Jobs in Progress** dialog opens.

16. Point to the tooltip and click the link to navigate to the location of the artifact or to view error messages.

Additional Steps for JBoss AS 7.3, JBoss EAP 6.2, and WildFly 8.2

If you want to generate an EJB baseline with the **Use Application Test Steps** option on the listed JBoss platforms, additional steps are required.

Before you generate the baseline, copy the following files to the **LISA_HOME\hotDeploy** directory:

- Application JAR files for the EJB under test
- **jboss-client.jar** file

Before you run the baseline, copy the **jboss-ejb-client.properties** file to the **LISA_HOME\hotDeploy** directory.

The following example shows the format of the **jboss-ejb-client.properties** file.

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false

remote.connections=default

remote.connection.default.host=10.20.30.40
remote.connection.default.port=4447
remote.connection.default.connect.options.org.xnio.Options.
SASL_POLICY_NOANONYMOUS=false

remote.connection.default.username=JBossApplicationUser
remote.connection.default.password=JBossApplicationPassword
```

The default port number for JBoss AS 7.3 and JBoss EAP 6.2 is 4447. The default port number for WildFly 8.2 is 8080.

If the JBoss server is remote from the DevTest installation from where you are testing, the username and password are mandatory. If the JBoss server and the system under test are the same, the username and password are not mandatory.

For more information about the **jboss-ejb-client.properties** file, see the JBoss documentation.

If the JBoss server is remote, ensure that the JBoss server is bound to a network interface that can be accessed from the remote system under test and not just localhost, which is the default. Typically, starting JBoss with the **-b 0.0.0.0** option will bind to all local interfaces.

EJB Baseline Contents

Stateful EJB baselines have one or more Enterprise JavaBean Execution steps. Each step includes the following assertions:

- Embedded assertion with the name **Assert on Result**
- Embedded assertion with the name **If environment error**
- Assert on Invocation Exception assertion with the name **Any Exception Then Fail**

Consolidated EJB baselines have an Execute Transaction Frame step that reads from a Large Data data set.

The data set contains the following information:

- Agent name
- Transaction frame
- Expected response

By default, the data set is local, rather than global.

The Execute Transaction Frame step includes a Graphical XML Side-by-Side Comparison assertion with the name **Assert Response Equals**. This assertion verifies that the actual response matches the expected response.

The test cases in an expanded EJB baseline have an **Enterprise JavaBean Execution** step or an **Execute Transaction Frame** step.

Workaround for EJB Baseline Over SSL on WebSphere App Server 8.5

WebSphere Application Server 8.5 was released with SSL enabled by default for RMI/IIOP and EJB connections. As a result, creating and running EJB baselines on WebSphere Application Server 8.5 will not succeed.

To work around this problem, use one of the following approaches.

Approach 1

Disable the SSL required setting on WebSphere Application Server.

Follow these steps:

1. Login to the WebSphere administrative console.
2. Go to Security, Global Security, RMI/IIOP security, CSIv2 inbound communications.
3. Set the value of the **Transport** field to **SSL-supported**.
4. Go to Security, Global Security, RMI/IIOP security, CSIv2 outbound communications.
5. Set the value of the **Transport** field to **SSL-supported**.

Approach 2

Configure your client to handle SSL-enabled connection.

The default password for the IBM truststore is **WebAS**.

On a base application server, the default key and truststores are stored in the node directory of the configuration repository. For example, the default **key.p12** and **trust.p12** stores are created with the **AppSrv01** profile name, the **myhostNode01Cell** name, and the **myhostNode01** node name.

For a standard WebSphere Application Server installation with a standalone application server profile, the key and truststores are in the following locations:

- C:
\WebSphere\AppServer\profiles\AppSrv01\config\cells\myhostNode01Cell\nodes\myhostNoc
p12

- C:
\WebSphere\AppServer\profiles\AppSrv01\config\cells\myhostNode01Cell\nodes\myhostNoc
p12

The **key.p12** file is optional and can be ignored.

Follow these steps:

1. Run the following command on the WebSphere Application Server computer to convert the truststore file from PKCS12 format to JKS format:

   - On UNIX:
   - `./jre/bin/keytool -importkeystore -srckeystore` *`truststore`* `-srcstoretype PKCS12 -deststoretype JKS -destkeystore trust.jks`
   - On Windows:
   - `.\jre\bin\keytool -importkeystore -srckeystore` *`truststore`* `-srcstoretype PKCS12 -deststoretype JKS -destkeystore trust.jks`

     Truststore specifies the name of the truststore file that is specific for your installation, for example, **trust.p12**.
     When prompted for the destination keystore password, specify something at least eight characters long. You will need this password when you edit the **ssl.client.props** file on the WebSphere Application Server computer.

2. Create a directory on the DevTest computer to store the **trust.jks** file that you created in the previous step. Ensure that the full path to the directory does not contain spaces, for example: **C:\WAS_security**.

3. Copy the **trust.jks** file to the directory that you created.

4. Copy the **ssl.client.props** and **sas.client.props** files located at **C:\WebSphere\AppServer\profiles\AppSrv01\profiles\AppSrv01\properties** to a temporary directory.

5. Open the copied **ssl.client.props** file and locate the following property:

   `com.ibm.ssl.trustStorePassword={xor}CDo9Hgw=`

6. Change the value of the property as follows:

   ```
   com.ibm.ssl.trustStorePassword=password
   ```

   Specify the password that you changed using **keytool**.

7. Run the following command from the temporary directory:

   ```
   C:\WebSphere\AppServer\bin\PropFilePasswordEncoder.bat ssl.client.props com.
   ibm.ssl.keyStorePassword,com.ibm.ssl.trustStorePassword
   ```

   The passwords in the copied **ssl.client.props** file are encoded.

8. Copy the **ssl.client.props** file from the temporary directory on the WebSphere Application Server computer to the DevTest computer.

9. Update the **ssl.client.props** file on the DevTest computer as follows:

   a. Change the value of the **user.root** property to the directory where you copied the **trust.jks** file, for example:

      ```
      user.root=C:/WAS_security
      ```

   b. Change the value of the **com.ibm.ssl.trustStore** property to your **trust.jks** file:

      ```
      com.ibm.ssl.trustStore=${user.root}/trust.jks
      ```

   c. Replace the following occurrences in the file with the indicated value:
      IbmPKIX with SunX509
      IbmX509 with SunX509
      IBMJSSE2 with SunJSSE
      SSL_TLS with SSL
      PKCS12 with JKS
      IBMJCE with SUN

   d. Change the property value for **com.ibm.ssl.enableSignerExchangePrompt** to false, instead of gui:

      ```
      com.ibm.ssl.enableSignerExchangePrompt=false
      ```

10. Copy the **sas.client.props** file from the WebSphere Application Server computer to the agent installation directory on the DevTest computer.

11. Change the property value for **com.ibm.CORBA.loginSource** in the copied **sas.client.props** file to none, instead of prompt (standard value):

    ```
    com.ibm.CORBA.loginSource=none
    ```

Follow these steps:

1. Copy the **com.ibm.ws.orb_*.jar** from the **C:\WebSphere\AppServer\profiles\AppSrv01\runtime** directory to the **LISA_HOME/DevTest/lib** directory, so it is on the client classpath.

2. Add the following vm arguments to registry and workstation, then restart DevTest registry and workstation.
   If you start the registry as a service, add these properties to **RegistryService.vmoptions**.
   If you start the registry from the command prompt, add it to **Registry.vmoptions**.
   For DevTest Workstation, add the following properties in **Workstation.vmoptions** file located under **LISA_HOME/devtest/bin**.

```
-Dcom.ibm.SSL.ConfigURL=file: C:/WAS_security/ssl.client.props
-Dcom.ibm.CORBA.ConfigURL=file: C:/WAS_security/sas.client.props
```

JMS Baselines

JMS baselines are supported for configurations in which JNDI is used to obtain the JMS connection factory.

JMS baselines are also supported for configurations in which a direct API from IBM WebSphere MQ, Java CAPS, SonicMQ, or TIBCO is used to obtain the JMS connection factory.

> **Note:** If a service or client reuses or forwards JMS message objects without changing them, the behavior of Application Insight is undefined.

This section contains the following pages:

- Generate a JMS Baseline
- JMS Baseline Transaction Frames
- JMS Baseline Contents

Generate a JMS Baseline

Use the following procedure to generate a JMS baseline from a set of transactions in the Application Insight database.

Follow these steps:

1. Add a JMS transaction frame to the shelf.
2. Open the shelf and click .
3. To change the default name, click , edit the name, then click to save.
4. Click the plus sign that appears to the left of the name.
5. If the creation mode field is available, select the baseline type.
6. If the step generation field is available, select one of the following options:
   - **Use Application Test Steps**

     The baseline includes a step that corresponds to the transaction frame that you added to the shelf. For example, selecting a SOAP frame results in a baseline that includes a **Web Service Execution (XML)** step. The step makes the client calls to the system under test at runtime.

   - **Use Transaction Frame Step**

     The baseline includes an **Execute Transaction Frame** step. The system under test is not called at runtime. Instead, the agent plays back the transaction inside the JVM memory.

7. (Optional) Configure the baseline to use magic dates:

   - **Apply magic dates to test cases**

     Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.

8. (Optional) Click **Choose File** and select a test template for the baseline.

9. (Optional) To view and manage merged frames, perform the following steps:

   a. Click **List frames** to view the frames.
      The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

   b. Drag-and-drop the frames up and down the list to change the order.

   c. Click **X** and click **Continue** to remove a frame from the listing.

10. Click **Next**.

11. Select the project where the baseline will be created.

12. (Optional) Edit name prefix. The default prefix is the user name.

13. Select an option to keep or delete the transactions in the shelf once the baseline is created and click **Finish**.
    The **Jobs in Progress** dialog opens.

14. Point to the tooltip and click the link to navigate to the location of the artifact or to view error messages.

JMS Baseline Transaction Frames

The procedure for generating a JMS baseline includes a step where you add a JMS transaction frame to the shelf.

For a scenario with one request and one response, the path graph from which you select a transaction frame to shelve contains four transaction frames. These frames represent the following activities:

- The client produces the request message.
- The service consumes the request message.
- The service produces the response message.
- The client consumes the response message.

Select the frame that represents the client producing the request message. For an exception to this rule, see the following section about duplicate nodes.

Application Insight searches for transactions that contain the same name as the frame that you select. The baseline contains all transactions that involve sending a message to or receiving a message from the same queue.

Duplicate Nodes

Under some circumstances, the path graph contains two adjacent frames that represent the same produce or consume operation. The names of the adjacent frames both begin with either **send:** or **recv:**.

If the first two frames in the path graph begin with **send:**, select the first of the frames.

If the first two frames in the path graph begin with **recv:**, select the second of the frames.

JMS Baseline Contents

Consolidated JMS baselines for most single-request, single-response scenarios have two steps:

- A do-nothing step that reads from a Large Data data set
- A messaging step that combines the request and response. The messaging step can be any of the following steps: JMS Messaging (JNDI), IBM WebSphere MQ, JCAPS Messaging (Native), SonicMQ Messaging (Native), or TIBCO Direct JMS. In the WebSphere MQ step, the client mode is set to JMS.

The baseline can also include a Unique Code Generator data set for generating a correlation ID.

The data set contains information that changes for each transaction in the baseline. This information includes the request payload, response payload, and JMS message properties. By default, the data set is local, instead of global.

The messaging step includes a Graphical XML Side-by-Side Comparison assertion with the name **Assert Response Equals**. When you run the baseline, this assertion verifies that the actual response matches the expected response. If the responses do not match, the test generates an error.

Other scenarios divide the request and response into separate steps.

Configuration properties are generated for destination information, connection information, and XML diff options. If the client and service use different JNDI connection settings, then two sets of properties are generated for the connection information.

The test cases in an expanded JMS baseline are similar to consolidated JMS baselines. However, they do not include a Large Data data set. The data is stored in the step instead.

As with consolidated JMS baselines, configuration properties are generated.

REST Baselines

This section contains the following pages that provide information about REST baselines:

- Generate a REST Baseline
- REST Baseline Contents
- HTTP Client Calls

Generate a REST Baseline

Use the following procedure to generate a REST baseline from a set of transactions in the Application Insight database.

Parameterization is supported for stateful and data-driven baselines. The purpose of parameterization is to make it easier for you to update baselines with new data.

Follow these steps:

1. Add one or more REST transaction frames to the shelf.
2. Open the shelf and click .
3. To change the default name, click , edit the name, then click to save.
4. Click the plus sign that appears to the left of the name.
5. If the creation mode field is available, select the baseline type.
6. If the step generation field is available, select one of the following options:
   - **Use Application Test Steps**

     The baseline includes a step that corresponds to the transaction frame that you added to the shelf. For example, selecting a SOAP frame results in a baseline that includes a **Web Service Execution (XML)** step. The step makes the client calls to the system under test at runtime.

   - **Use Transaction Frame Step**

     The baseline includes an **Execute Transaction Frame** step. The system under test is not called at runtime. Instead, the agent plays back the transaction inside the JVM memory.
7. If the stateful baseline fields are available, configure them as needed:
   - **Parameter Level**
     Specifies whether to enable parameterization and, if so, whether to match on value only or both key and value. The default is **Disable**. When the **Database Validation** checkbox is selected, the **Parameter Level** field is disabled and displays the **Value** option.

   - **Http User**
     Specifies the web application user name. Application Insight encrypts the value and adds the result to the Authorization header field in the baseline.

   - **Http Password**
     Specifies the web application user password. Application Insight encrypts the value and adds the result to the Authorization header field in the baseline.

   - **Assertion Option**
     Lets you add one or two assertions to the baseline.
     The first assertion that you can add is named **Assert Response Code Equals**. The second assertion that you can add is named **Assert Response Equals** or **Assert**

Response Contains.
For each assertion, you specify what happens when the assertion returns false. You can select to fail the test, generate an error, or generate a warning.

8. (Optional) Configure the baseline to use magic dates.

   - **Apply magic dates to test cases**

     Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.

9. (Optional) To configure the baseline to validate any database inserts, updates, and deletes, select the **Database Validation** check box.

10. (Optional) Click **Choose File** and select a test template for the baseline.

11. (Optional) To view and manage merged frames, perform the following steps:

    a. Click **List frames** to view the frames.
    The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

    b. Drag-and-drop the frames up and down the list to change the order.

    c. Click **X** and click **Continue** to remove a frame from the listing.

12. Click **Next**.

13. Select the project where the baseline will be created.

14. (Optional) Edit name prefix. The default prefix is the user name.

15. Select an option to keep or delete the transactions in the shelf once the baseline is created and click **Finish**.
    The **Jobs in Progress** dialog opens.

16. Point to the tooltip and click the link to navigate to the location of the artifact or to view error messages.

Stateful Baselines and Parameterization

You can add *parameterization* to REST stateful baselines.

When creating the baseline, Application Insight tries to detect common items in the request and response.

To configure this behavior, you specify one of these levels:

- Match on value only

- Match on key and value

Assume that the request has the following URL and JSON body:

```
GET http://myserver:8080/datamanagement/a/templates/4?user=admin
Request:{"name":"demo_02","applicationId":"3","new":true}
```

If you set the parameter level to value only, Application Insight examines the following items: **a**, **templates**, **4**, **admin**, **demo_02**, and **3**.

If you set the parameter level to key and value, Application Insight examines the following items: **user/admin**, **name/demo_02**, and **applicationId/3**.

In the URL, the first element after the server and port is ignored.

Boolean types are ignored.

Assume that the request has the following XML body:

```
<addUser xmlns="http://ejb3.examples.itko.com/" userID=123>
 <username xmlns="">webapp-1381707309</username>
 <password xmlns="">example-pwd</password>
</addUser>
```

If you set the parameter level to value only, Application Insight examines the following items: **123** , **webapp-1381707309**, and **example-pwd**.

If you set the parameter level to key and value, Application Insight examines the following items: **userID/123**, **username/webapp-1381707309**, and **password/example-pwd**.

When common items are found, Application Insight creates one or more parameters. Each parameter is stored in a data set or defined in a filter. These examples show the format of the parameters:

- **Step1_name_1**
- **Step1_param_2**
- **Step1_url_param_3**
- **Step1_rsp.filter_param_4**
- **Step2_param_5**

The parameters can be used in various parts of the baseline. This example shows a parameter in the URL field of a REST step:

```
http://{{WSSERVER}}:{{WSPORT}}/TestAppParameter/rest/testmethods/
{{Step1_url_param_3}}
```

The following properties let you configure the minimum length and exclusion strings:

- **lisa.pathfinder.stateful.baseline.parameter.string.min.length**
- **lisa.pathfinder.stateful.baseline.parameter.string.exclusion**

For more information about these properties, see DevTest Property File (lisa.properties).

Data-Driven Baselines and Parameterization

The XML representation of a REST transaction frame can include a set of request parameters. For example:

```
<RequestParameters>
<Parameter><Key>roleAdmin</Key><Value>on</Value></Parameter>
<Parameter><Key>password</Key><Value>test2</Value></Parameter>
<Parameter><Key>firstname</Key><Value>test2</Value></Parameter>
<Parameter><Key>phone</Key><Value>1234567</Value></Parameter>
<Parameter><Key>confirmpassword</Key><Value>test2</Value></Parameter>
<Parameter><Key>lastname</Key><Value>test2</Value></Parameter>
<Parameter><Key>useremail</Key><Value>test2@company.com</Value></Parameter>
<Parameter><Key>username</Key><Value>test2</Value></Parameter>
</RequestParameters>
```

If you create a data-driven baseline from the frame, Application Insight adds DevTest parameters to the query portion of the URL. For example:

```
http://{{SERVER}}:{{PORT}}/lisabank/createuser.do?roleAdmin={{roleAdmin}}&password=
{{password}}&firstname={{firstname}}...
```

Notice that only the values are parameterized. The keys are left intact.

To view the URL and the parameters, go to DevTest Workstation and open the Large Data data set in the baseline.

REST Baseline Contents

Stateful REST baselines have one or more REST steps.

When you create a stateful REST baseline, you can choose to add assertions and, if so, which assertions to add.

The possible assertions are:

- **Assert Response Code Equals.** This assertion verifies that the actual response code matches the expected response code.
- **Assert Response Equals.** This assertion verifies that the actual response matches the expected response. The type of assertion depends on whether the response format is XML, JSON, or raw text.
- **Assert Response Contains.** This assertion verifies that the actual response contains the expected response.

Data-driven REST baselines have a test step that reads from a Large Data data set. The test step is either a REST step or an Execute Transaction Frame step.

If the test step is a REST step, the data set contains such information as the URL, request body, expected response, and response code.

If the test step is an Execute Transaction Frame step, the data set contains the transaction frame and the expected response.

By default, the data set is local, instead of global.

The test step includes one or more of the following assertions:

- **Assert Response Code Equals.** This assertion verifies that the actual response code matches the expected response code.

- **Assert Response Equals.** This assertion verifies that the actual response matches the expected response. The type of assertion depends on whether the response format is XML, JSON, or raw text.

The test cases in an expanded REST baseline are similar to data-driven REST baselines. However, they do not include a Large Data data set. The data is stored in the test step instead.

HTTP Client Calls

When an application makes an HTTP client call using the Apache HttpClient API for POST, GET, PUT, and DELETE, the REST category is assigned to the transaction frame.

To force the POST and GET calls to be categorized as HTTP instead, add the **lisa.agent.rest.enabled** property to the **agent** element of the **rules.xml** file. Set the value to false.

```
<property key="lisa.agent.rest.enabled" value="false"/>
```

If the registry is running when you update the **rules.xml** file, restart the registry.

Forcing the PUT and DELETE calls to be categorized as HTTP is not supported.

TIBCO BusinessWorks Baselines

This section contains the following pages that provide information about TIBCO BusinessWorks baselines:

- Generate a TIBCO BusinessWorks Baseline
- TIBCO BusinessWorks Baseline Contents

Generate a TIBCO BusinessWorks Baseline

Use the following procedure to generate a TIBCO ActiveMatrix BusinessWorks baseline from a set of transactions in the Application Insight database.

The step generation field does not let you select the **Use Application Test Steps** option. The only valid option is **Use Transaction Frame Step**.

Follow these steps:

1. Add a TIBCO ActiveMatrix BusinessWorks transaction frame to the shelf.
2. Open the shelf and click .
3. To change the default name, click , edit the name, then click to save.
4. Click the plus sign that appears to the left of the name.
5. If the creation mode fields are available, select the baseline type.
6. (Optional) Configure the baseline to use magic dates.

- **Apply magic dates to test cases**

  Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.

7. (Optional) Click **Choose File** and select a test template for the baseline.

8. (Optional) To view and manage merged frames, perform the following steps:

   a. Click **List frames** to view the frames.
   The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

   b. Drag-and-drop the frames up and down the list to change the order.

   c. Click **X** and click **Continue** to remove a frame from the listing.

9. Click **Next**.

10. Select the project where the baseline will be created.

11. (Optional) Edit name prefix. The default prefix is the user name.

12. Select an option to keep or delete the transactions in the shelf once the baseline is created and click **Finish**.
    The **Jobs in Progress** dialog opens.

13. Point to the tooltip and click the link to navigate to the location of the artifact or to view error messages.

TIBCO BusinessWorks Baseline Contents

Consolidated TIBCO ActiveMatrix BusinessWorks baselines have an Execute Transaction Frame step that reads from a Large Data data set.

The data set contains the following information:

- Agent name
- Transaction frame
- Expected response

By default, the data set is local, instead of global.

The Execute Transaction Frame step includes a Graphical XML Side-by-Side Comparison assertion with the name **Assert Response Value Equals**. This assertion verifies that the actual response matches the expected response.

The test cases in an expanded TIBCO ActiveMatrix BusinessWorks baseline have an Execute Transaction Frame step.

TIBCO Enterprise Message Service Baselines

The baseline feature is supported for TIBCO Enterprise Message Service (EMS).

The procedure is the same as for JMS baselines. However, before you start, copy the **tibjms.jar** file into the **LISA_HOME\lib** directory.

Web HTTP Baselines

This section contains the following pages that provide information about Web HTTP baselines:

- Generate a Web HTTP Baseline
- Web HTTP Baseline Contents
- HTTP Client Call

Generate a Web HTTP Baseline

Use the following procedure to generate a Web HTTP baseline from a set of transactions in the Application Insight database.

Signed Web HTTP transactions are supported. However, Application Insight does not save SSL information to the baseline. If the SSL information is required, configure the following fields in the generated baseline:

- **SSL Keystore File**
- **SSL Keystore Password**
- **SSL Key Alias**
- **SSL Key Password**

These fields are located in the HTTP/HTML Request step.

Parameterization is supported for stateful and data-driven baselines. The purpose of parameterization is to make it easier for you to update baselines with new data.

Follow these steps:

1. Add one or more Web HTTP transaction frames to the shelf.
2. Open the shelf and click .
3. To change the default name, click , edit the name, then click to save.
4. Click the plus sign that appears to the left of the name.
5. If the creation mode field is available, select the baseline type.
6. If the step generation field is available, select one of the following options:

- **Use Application Test Steps**

  The baseline includes a step that corresponds to the transaction frame that you added to the shelf. For example, selecting a SOAP frame results in a baseline that includes a **Web Service Execution (XML)** step. The step makes the client calls to the system under test at runtime.

- **Use Transaction Frame Step**

  The baseline includes an **Execute Transaction Frame** step. The system under test is not called at runtime. Instead, the agent plays back the transaction inside the JVM memory.

7. If the stateful baseline fields are available, configure them as needed:

   - **Http User**
     Specifies the web application user name. Application Insight encrypts the value and adds the result to the Authorization header field in the baseline.

   - **Http Password**
     Specifies the web application user password. Application Insight encrypts the value and adds the result to the Authorization header field in the baseline.

   - **Assertion Option**
     Lets you add one or two assertions to the baseline.
     The first assertion that you can add is named **Assert Response Code Equals**. The second assertion that you can add is named **Assert Response Equals** or **Assert Response Contains**.
     For each assertion, you specify what happens when the assertion returns false. You can select to fail the test, generate an error, or generate a warning.

8. (Optional) Configure the baseline to use magic dates.

   - **Apply magic dates to test cases**

     Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.

9. (Optional) To configure the baseline to validate any database inserts, updates, and deletes, select the **Database Validation** check box.

10. (Optional) Click **Choose File** and select a test template for the baseline.

11. (Optional) Edit the header information.

12. (Optional) To view and manage merged frames, perform the following steps:

    a. Click **List frames** to view the frames.
       The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

b. Drag-and-drop the frames up and down the list to change the order.

c. Click **X** and click **Continue** to remove a frame from the listing.

13. Click **Next**.

14. Select the project where the baseline will be created.

15. (Optional) Edit name prefix. The default prefix is the user name.

16. Select an option to keep or delete the transactions in the shelf once the baseline is created and click **Finish**.
    The **Jobs in Progress** dialog opens.

17. Point to the tooltip and click the link to navigate to the location of the artifact or to view error messages.

Stateful Baselines and Parameterization

You can add *parameterization* to Web HTTP stateful baselines.

When creating the baseline, Application Insight examines the following areas for information to parameterize:

- URL query string
  An example of a URL with a query string is **http://api.openweathermap.org/data/2.5/weather?q=London**.

- Request body
  Key/value pairs are candidates for parameterization.

If Application Insight creates one or more parameters, it stores them in a data set. These examples show the format of the parameters:

- **Step1_zip_1**

- **Step1_city_2**

- **Step1_type_3**

- **Step1_balance_4**

- **Step1_name_5**

The parameters can be used in various parts of the baseline. The following graphic shows parameters in the **POST Parameters** section of the HTTP/HTML Request step.

Screen capture of parameters.

Data-Driven Baselines and Parameterization

The XML representation of a Web HTTP GET transaction frame can include a set of request parameters. For example:

```
<RequestParameters>
<Parameter><Key>cmd</Key><Value>add</Value></Parameter>
<Parameter><Key>pwd</Key><Value>example-pwd</Value></Parameter>
<Parameter><Key>user</Key><Value>webapp-41066911</Value></Parameter>
</RequestParameters>
```

If you create a data-driven baseline from the frame, Application Insight adds DevTest parameters to the query portion of the URL. For example:

```
http://{{SERVER}}:{{PORT}}/lisabank/createuser.do?cmd={{cmd}}&pwd={{pwd}}&user=
{{user}}
```

Notice that only the values are parameterized. The keys are left intact.

To view the URL and the parameters, go to DevTest Workstation and open the Large Data data set in the baseline.

The XML representation of a Web HTTP POST transaction frame with the **application/x-www-form-urlencoded** content type can have key/value pairs in the request body. For example:

```
<req>
<![CDATA
[username=test5&password=test5&confirmpassword=test5&firstname=test5&lastname=test
5&roleAdmin=on&useremail=test5%40company.com&phone=1234567]]>
</req>
```

If you create a data-driven baseline from the frame, Application Insight replaces the keys and values with DevTest parameters.

To view the parameters and their definitions, go to DevTest Workstation and open the baseline. As shown in the following graphic, the parameters appear in the **POST Parameters** section of the HTTP/HTML Request step.

POST Parameters

Key	Value
{{post_key_1}}	{{post_value_1}}
{{post_key_2}}	{{post_value_2}}
{{post_key_3}}	{{post_value_3}}
{{post_key_4}}	{{post_value_4}}
{{post_key_5}}	{{post_value_5}}
{{post_key_6}}	{{post_value_6}}
{{post_key_7}}	{{post_value_7}}
{{post_key_8}}	{{post_value_8}}

Screen capture of parameters.

To view the definitions, open the Large Data data set.

Web HTTP Baseline Contents

Stateful Web HTTP baselines have one or more HTTP/HTML Request steps.

Each step includes the **Assert Response Code Equals** assertion. This assertion verifies that the actual response code matches the expected response code.

If the response format is XML, the step also includes the **Assert Response Equals** assertion. This assertion verifies that the actual response matches the expected response.

Data-driven Web HTTP baselines have an HTTP/HTML Request step that reads from a Large Data data set.

The data set contains such information as the URL, request body, expected response, and response code. By default, the data set is local, rather than global.

The test step includes the **Assert Response Code Equals** assertion. If the response format is XML, the step also includes the **Assert Response Equals** assertion.

The test cases in an expanded Web HTTP baseline are similar to data-driven Web HTTP baselines. However, they do not include a Large Data data set. The data is stored in the test step instead.

HTTP Client Call

When an application makes an HTTP client call using the Apache HttpClient API for POST, GET, PUT, and DELETE, the REST category is assigned to the transaction frame.

To force the POST and GET calls to be categorized as HTTP instead, add the **lisa.agent.rest.enabled** property to the **agent** element of the **rules.xml** file. Set the value to false.

```
<property key="lisa.agent.rest.enabled" value="false"/>
```

If the registry is running when you update the **rules.xml** file, restart the registry.

Forcing the PUT and DELETE calls to be categorized as HTTP is not supported.

Web Service Baselines

This section contains the following pages that provide information about web service baselines:

- Generate a Web Service Baseline
- Web Services with Attachments
- Web Service Baseline Contents

Generate a Web Service Baseline

Use the following procedure to generate a web service baseline from a set of transactions in the Application Insight database.

Web services with attachments are supported.

Signed SOAP messages are supported on all platforms except IBM WebSphere Application Server 8.5. However, Application Insight does not save SSL information to the baseline. If the SSL information is required, configure the following fields in the generated baseline:

- **SSL Keystore File**
- **SSL Keystore Password**
- **SSL Key Alias**
- **SSL Key Password**

These fields are located in the Web Service Execution (XML) step.

This feature supports transport-level security, which is a point-to-point security model. This feature does not support signed message-level security, which is an end-to-end security model.

Follow these steps:

1. Add one or more web service transaction frames to the shelf.
2. Open the shelf and click .
3. To change the default name, click , then click to save.
4. Click the plus sign that appears to the left of the name.
5. If the creation mode field is available, select the baseline type.
6. If the step generation field is available, select one of the following options:
   - **Use Application Test Steps**

     The baseline includes a step that corresponds to the transaction frame that you added to the shelf. For example, selecting a SOAP frame results in a baseline that includes a **Web Service Execution (XML)** step. The step makes the client calls to the system under test at runtime.

   - **Use Transaction Frame Step**

     The baseline includes an **Execute Transaction Frame** step. The system under test is not called at runtime. Instead, the agent plays back the transaction inside the JVM memory.

7. If the stateful baseline fields are available, configure them as needed:
   - **Parameter Level**
     Specifies whether to enable parameterization and, if so, whether to match on value only or both key and value. This field is available when there are REST and Web Service transactions included. The default is **Disable**. When the **Database Validation** checkbox is selected, the **Parameter Level** field is disabled and displays the **Value** option.

   - **Http User**
     Specifies the web application user name. Application Insight encrypts the value and adds the result to the Authorization header field in the baseline.

   - **Http Password**
     Specifies the web application user password. Application Insight encrypts the value and adds the result to the Authorization header field in the baseline.

   - **Assertion Option**
     Lets you add one or two assertions to the baseline.
     The first assertion that you can add is named **Assert Response Code Equals**. The second assertion that you can add is named **Assert Response Equals** or **Assert Response Contains**.
     For each assertion, you specify what happens when the assertion returns false. You can select to fail the test, generate an error, or generate a warning.

8. (Optional) Configure the baseline to use magic dates.

   - **Apply magic dates to test cases**

     Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.

9. (Optional) To configure the baseline to validate any database inserts, updates, and deletes, select the **Database Validation** check box.

10. (Optional) Click **Choose File** and select a test template for the baseline.

11. (Optional) To view and manage merged frames, perform the following steps:

    a. Click **List frames** to view the frames.
    The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

    b. Drag-and-drop the frames up and down the list to change the order.

    c. Click **X** and click **Continue** to remove a frame from the listing.

12. Click **Next**.

13. Select the project where the baseline will be created.

14. (Optional) Edit name prefix. The default prefix is the user name.

15. Select an option to keep or delete the transactions in the shelf once the baseline is created and click **Finish**.
    The **Jobs in Progress** dialog opens.

16. Point to the tooltip and click the link to navigate to the location of the artifact or to view error messages.

Stateful Baselines and Parameterization

You can add *parameterization* to web service stateful baselines.

When creating the baseline, Application Insight tries to detect common items in the request and response.

To configure this behavior, you specify one of these levels:

- Match on value only
- Match on key and value

Assume that the request has the following XML body:

```
<addUser xmlns="http://ejb3.examples.itko.com/" userID=123>
 <username xmlns="">webapp-1381707309</username>
 <password xmlns="">example-pwd</password>
</addUser>
```

If you set the parameter level to value only, Application Insight examines the following items: **123** , **webapp-1381707309**, and **example-pwd**.

If you set the parameter level to key and value, Application Insight examines the following items: **userID/123**, **username/webapp-1381707309**, and **password/example-pwd**.

When common items are found, Application Insight creates one or more parameters. Each parameter is stored in a data set or defined in a filter. These examples show the format of the parameters:

- **Step1_name_1**
- **Step1_param_2**
- **Step1_url_param_3**
- **Step1_rsp.filter_param_4**
- **Step2_param_5**

The parameters can be used in various parts of the baseline. This example shows a parameter in the Raw XML tab of a Web Service Execution (XML) step:

```
<username xmlns="">{{Step1_param_1}}</username>
```

The following properties let you configure the minimum length and exclusion strings:

- **lisa.pathfinder.stateful.baseline.parameter.string.min.length**
- **lisa.pathfinder.stateful.baseline.parameter.string.exclusion**

For more information about these properties, see DevTest Property File (lisa.properties).

Web Services with Attachments

You can generate a baseline for web services that have the following attachment types:

- MIME
- DIME
- XOP
- MTOM

Application Insight assumes that if a web service has multiple attachments, the attachments are the same type.

The **Use Transaction Frame Step** option is not supported for web services with attachments.

Outbound web service calls with attachments are not supported.

When the baseline is created, the attachment content is saved in the frame. The type is **Base64 Encoded**.

If the frame is generated from an MTOM attachment transaction, the attachment type in the baseline is XOP.

By default, the maximum size of an attachment is 10 MB. If the attachment is larger, the transaction frame is marked as truncated and the attachment content is not saved in the frame.

To change the default maximum size, go to the Agents window. Select the agent or broker. Click the **Settings** tab and select the **Transactions** category. Configure the **Max Attachment Size** property. Set the value to the number of kilobytes, not megabytes. Then click **Save**.

Web Service Baseline Contents

Stateful web service baselines have one or more Web Service Execution (XML) steps. Each step includes the following assertions:

- **Assert Response Code Equals.** This assertion verifies that the actual response code matches the expected response code.
- **Assert Response Equals.** This assertion verifies that the actual response matches the expected response. The type of assertion depends on whether the response format is XML, JSON, or raw text.

Consolidated web service baselines have a test step that reads from a Large Data data set. The test step is either a Web Service Execution (XML) step or an Execute Transaction Frame step.

If the test step is a Web Service Execution (XML) step, the data set contains information such as the URL, request body, expected response, and response code. The data set can also include data from web service attachments.

If the test step is an Execute Transaction Frame step, the data set contains the transaction frame and the expected response.

By default, the data set is local, instead of global.

The test step includes one or both of the following assertions:

- **Assert Response Code Equals.** This assertion verifies that the actual response code matches the expected response code.
- **Assert Response Equals.** This assertion verifies that the actual response matches the expected response. The type of assertion depends on whether the response format is XML, JSON, or raw text.

The test cases in an expanded web service baseline are similar to consolidated web service baselines. However, they do not include a Large Data data set. The data is stored in the test step instead.

webMethods Baselines

This section contains the following pages that provide information about webMethods baselines:

- Generate a webMethods Baseline
- webMethods Baseline Contents

Generate a webMethods Baseline

Use the following procedure to generate a webMethods Integration Server baseline from a set of transactions in the Application Insight database.

webMethods Integration Server includes the concepts of a flow service and a pipeline. A flow service lets you encapsulate a group of services and manage the flow of data among them. The pipeline is a data structure that contains the input and output values for a flow service. You can add steps to a flow service to perform such actions as invoking services and changing data in the pipeline.

The flow service is the unit that is baselined.

Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Follow these steps:

1. Add a webMethods Integration Server transaction frame to the shelf.
2. Open the shelf and click .
3. To change the default name, click , edit the name, then click to save.
4. Click the plus sign that appears to the left of the name.
5. If the creation mode field is available, select the baseline type.
6. If the step generation field is available, select one of the following options:
   - **Use Application Test Steps**

     The baseline includes a step that corresponds to the transaction frame that you added to the shelf. For example, selecting a SOAP frame results in a baseline that includes a **Web Service Execution (XML)** step. The step makes the client calls to the system under test at runtime.

   - **Use Transaction Frame Step**

     The baseline includes an **Execute Transaction Frame** step. The system under test is not called at runtime. Instead, the agent plays back the transaction inside the JVM memory.
7. (Optional) Configure the baseline to use magic dates.
   - **Apply magic dates to test cases**

     Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.
8. (Optional) Click **Choose File** and select a test template for the baseline.
9. (Optional) To view and manage merged frames, perform the following steps:

a. Click **List frames** to view the frames.
The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

b. Drag-and-drop the frames up and down the list to change the order.

c. Click **X** and click **Continue** to remove a frame from the listing.

10. Click **Next**.

11. Select the project where the baseline will be created.

12. (Optional) Edit name prefix. The default prefix is the user name.

13. Select an option to keep or delete the transactions in the shelf once the baseline is created and click **Finish**.
The **Jobs in Progress** dialog opens.

14. Point to the tooltip and click the link to navigate to the location of the artifact or to view error messages.

Example: Update User

The following graphic shows a path graph that includes webMethods Integration Server frames. The flow service in this example is named **updateUser**. The purpose is to send updated information for a user to a database.

Note: If you select any of the webMethods Integration Server frames, you can view the pipeline data in the transaction detail dialog.

When a path graph has multiple webMethods Integration Server frames, you must select one frame for baselining. Typically, you select the entry point. In this example, you would select the leftmost webMethods Integration Server frame.

webMethods Baseline Contents

Consolidated webMethods baselines have a test step that reads from a Large Data data set. The test step is either a webMethods Integration Server Services step or an Execute Transaction Frame step.

If the test step is a webMethods Integration Server Services step, the data set contains such information as the URL, request body, expected response, and response code.

If the test step is an Execute Transaction Frame step, the data set contains the agent name, transaction frame, and expected response.

By default, the data set is local, instead of global.

The test step includes one or both of the following assertions:

- **Assert Response Code Equals**. This assertion verifies that the actual response code matches the expected response code.
- **Assert Response Equals**. This assertion verifies that the actual response value matches the expected response value.

The test cases in an expanded webMethods baseline are similar to consolidated webMethods baselines. However, they do not include a Large Data data set. The data is stored in the test step instead.

WebSphere MQ Baselines

This section contains the following pages that provide information about WebSphere MQ baselines:

- Generate a WebSphere MQ Baseline
- WebSphere MQ Baseline Scenarios
- WebSphere MQ Baseline Properties
- WebSphere MQ Baseline Contents

Generate a WebSphere MQ Baseline

Use the following procedure to generate a WebSphere MQ baseline from a set of transactions in the Application Insight database.**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Follow these steps:

1. Determine which WebSphere MQ scenario you want to use, and enable the DevTest Java Agent as appropriate.
2. (Optional) Configure the WebSphere MQ properties in the **rules.xml** file.
3. Add a WebSphere MQ transaction frame to the shelf. The path graph from which you select a transaction frame to shelve includes multiple WebSphere MQ transaction frames. Be sure to select the *first* WebSphere MQ transaction frame.
4. Open the shelf and click .
5. To change the default name, click , edit the name, then click to save.
6. Click the plus sign that appears to the left of the name.
7. If the creation mode field is available, select the baseline type.
8. If the step generation field is available, select one of the following options:
   - **Use Application Test Steps**

The baseline includes a step that corresponds to the transaction frame that you added to the shelf. For example, selecting a SOAP frame results in a baseline that includes a **Web Service Execution (XML)** step. The step makes the client calls to the system under test at runtime.

- **Use Transaction Frame Step**

  The baseline includes an **Execute Transaction Frame** step. The system under test is not called at runtime. Instead, the agent plays back the transaction inside the JVM memory.

9. (Optional) Configure the baseline to use magic dates.

   - **Apply magic dates to test cases**

     Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.

10. (Optional) Click **Choose File** and select a test template for the baseline.

11. (Optional) To view and manage merged frames, perform the following steps:

    a. Click **List frames** to view the frames.
    The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

    b. Drag-and-drop the frames up and down the list to change the order.

    c. Click **X** and click **Continue** to remove a frame from the listing.

12. Click **Next**.

13. Select the project where the baseline will be created.

14. (Optional) Edit name prefix. The default prefix is the user name.

15. Select an option to keep or delete the transactions in the shelf once the baseline is created and click **Finish**.
    The **Jobs in Progress** dialog opens.

16. Point to the tooltip and click the link to navigate to the location of the artifact or to view error messages.

WebSphere MQ Baseline Scenarios

When working with WebSphere MQ, you can enable the DevTest Java Agent on the client, the server, or both the client and the server. The contents of the path graph vary depending on the scenario.

Note: DevTest can behave as if it is an agent-enabled client. This behavior occurs when a test case uses an IBM WebSphere MQ step to invoke the service.

Scenario 1: Client Enabled, Service Not Enabled

The client is agent-enabled, and the service is not agent-enabled. This scenario simulates a Java-based client running against a Windows or mainframe-based service that cannot be agent-enabled.

When you view the transaction details in the DevTest Portal, the path graph contains two nodes.

The first node represents the request. To view the message body, select the node and then click the **Request** tab.

The second node represents the response. To view the message body, select the node and then click the **Response** tab.

Scenario 2: Client Not Enabled, Service Enabled

The client is not agent-enabled, and the service is agent-enabled. This scenario simulates a Windows or mainframe-based client that cannot be agent-enabled running against a Java-based service.

The first transaction in the DevTest Portal is a priming transaction. The agent on the service side discovers that it is running on the service side and initializes itself.

The subsequent transactions are the same as for scenario 1. The path graph contains a request node and a response node.

Scenario 3: Client Enabled, Service Enabled

Both the client and the service are agent-enabled. This scenario simulates a Java-based client running against a Java-based service.

The first transaction in the DevTest Portal is a priming transaction. The agent on the service side discovers that it is running on the service side and initializes itself.

For the subsequent transactions, the path graph contains four nodes:

- The first node represents the client PUT for the request.
- The second node represents the service GET for the request.
- The third node represents the service PUT for the response.
- The fourth node represents the client GET for the response.

WebSphere MQ Baseline Properties

The configuration file for the DevTest Java Agent is named **rules.xml**.

Note: For detailed information about this file, see DevTest Java Agent.

Before you create a WebSphere MQ baseline, you can add the **QUEUE_REQUEST_MATCHES** and **QUEUE_RESPONSE_MATCHES** properties to the **rules.xml** file. These properties indicate which queues are for requests and which queues are for responses.

The **QUEUE_REQUEST_MATCHES** and **QUEUE_RESPONSE_MATCHES** properties are optional. These properties are necessary only if the agent cannot determine on its own whether a message is a request or a response.

If you include these properties, the property names must be followed by a colon and the actual queue name. The value of each property is a period followed by an asterisk. For example:

```
<property key="QUEUE_REQUEST_MATCHES:ORDERS.REQUEST" value=".*"/>
<property key="QUEUE_RESPONSE_MATCHES:ORDERS.RESPONSE" value=".*"/>
```

WebSphere MQ Baseline Contents

Consolidated WebSphere MQ baselines have the following steps:

- A do-nothing step that reads from a Large Data data set
- A WebSphere MQ step that sends the request
- A WebSphere MQ step that receives the response

The data set contains information that changes for each transaction in the baseline. This information includes the request payload and the response payload.

In the WebSphere MQ steps, the client mode is set to native client.

The first WebSphere MQ step sends the request using data from the data set.

The second WebSphere MQ step receives the response. The step includes a Graphical XML Side-by-Side Comparison assertion with the name **Assert Response Equals**. This assertion verifies that the actual response matches the expected response. If the responses do not match, the test generates an error.

The test case can also include a Message/Correlation ID Generator data set, which generates a 24-byte code that is used as the correlation ID.

The following graphic shows an example test case.

Screen capture of WebSphere MQ baseline test case.

Binary payloads affect the structure of a consolidated WebSphere MQ baseline. The baseline has the following steps:

- A do-nothing step that reads from a Large Data data set
- A Java Script step that decodes the request payload from Base64 notation into an array of bytes

- A WebSphere MQ step that sends the request
- A WebSphere MQ step that receives the response
- A Java Script step that encodes the response payload from an array of bytes into Base64 notation

The Large Data data set includes two versions of the original request payload: a human-readable version and the Base64 representation.

The Large Data data set includes two versions of the original response payload: a human-readable version and the Base64 representation.

The second Java Script step includes an assertion with the name **Check Base64 Response**. This assertion verifies that the Base64 representation of the actual response matches the Base64 representation of the expected response. If the responses do not match, the test generates an error.

The test cases in an expanded WebSphere MQ baseline are similar to consolidated WebSphere MQ baselines. However, they do not include a Large Data data set. The data is stored in the steps instead.

For binary payloads, the test cases have the following steps:

- A Parse Text as Response step that contains a Base64 representation of the original request payload
- An Output Log Message step that writes a human-readable version of the original request payload to the log file
- A Java Script step that decodes the actual request payload from Base64 notation into an array of bytes
- A WebSphere MQ step that sends the request
- A WebSphere MQ step that receives the response
- A Parse Text as Response step that contains a Base 64 representation of the original response payload
- An Output Log Message step that writes a human-readable version of the original response payload to the log file
- A Java Script step that encodes the actual response payload from an array of bytes into Base64 notation

Generic Baselines

The term *generic baseline* refers to baselines that are generated for protocols other than the following protocols:

- EJB
- JMS
- REST

- TIBCO ActiveMatrix BusinessWorks
- TIBCO Enterprise Message Service (EMS)
- Web HTTP
- Web service
- webMethods Integration Server
- WebSphere MQ

This section contains the following pages:

- Generate a Generic Baseline
- Generic Baseline Contents

Generate a Generic Baseline

Use the following procedure to generate a generic baseline.

Follow these steps:

1. Add one or more transaction frames to the shelf for a protocol other than the protocols listed in Generic Baselines.
2. Open the shelf and click .
3. To change the default name, select the name and make your edits, then click to save.
4. Click the plus sign that appears to the left of the name.
5. If the creation mode field is available, select the baseline type.
6. If the step generation field is available, select one of the following options:
   - **Use Application Test Steps**

     The baseline includes a step that corresponds to the transaction frame that you added to the shelf. For example, selecting a SOAP frame results in a baseline that includes a **Web Service Execution (XML)** step. The step makes the client calls to the system under test at runtime.

   - **Use Transaction Frame Step**

     The baseline includes an **Execute Transaction Frame** step. The system under test is not called at runtime. Instead, the agent plays back the transaction inside the JVM memory.
7. (Optional) Configure the baseline to use magic dates.
   - **Apply magic dates to test cases**

Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.

8. (Optional) Click **Choose File** and select a test template for the baseline.

9. (Optional) To view and manage merged frames, perform the following steps:

   a. Click **List frames** to view the frames.
   The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

   b. Drag-and-drop the frames up and down the list to change the order.

   c. Click **X** and click **Continue** to remove a frame from the listing.

10. Click **Create**.

11. Select the project where the baseline will be created.

12. (Optional) Edit name prefix. The default prefix is the user name.

13. Select an option to keep or delete the transactions in the shelf once the baseline is created.

14. Click **Create**.

Generic Baseline Contents

Consolidated generic baselines have a test step that reads from a Large Data data set. The test step is one of the following:

- A step that corresponds to the node that you selected in the path graph
- An **Execute Transaction Frame** step

If the test step is a step that corresponds to the node that you selected in the path graph, the data set contains such information as the URL, request body, expected response, and response code.

If the test step is an **Execute Transaction Frame** step, the data set contains the transaction frame and the expected response.

By default, the data set is local, instead of global.

The test step includes an assertion with the name **Assert Response Value Equals**. This assertion verifies that the actual response value matches the expected response value.

The test cases in an expanded generic baseline are similar to consolidated generic baselines. However, they do not include a Large Data data set. The data is stored in the test step instead.

Database Validation

When creating a baseline from DevTest Portal or the Application Insight command-line tool, you can indicate that the baseline will perform database validation. One or more steps are added to the baseline to confirm that any inserts, updates, and deletes are done successfully.

You can use this feature with stateful and expanded baselines, but not with data-driven baselines. A data-driven baseline consolidates all transactions into one step, and uses a data set to store the data for each transaction. This behavior is incompatible with the database validation process.

You can use this feature with the following protocols:

- REST
- Web HTTP
- Web service

If you enable database validation in DevTest Portal, the **Parameter Level** field is disabled because the only valid setting is **Value**.

- Supported Databases
- Supported Data Types
- Supported SQL
- Database Validation Steps

Supported Databases

The following databases and drivers are supported:

Apache Derby

Apache Derby 4.2.3 JDBC driver to Apache Derby database

Oracle

- Oracle 11g JDBC driver to Oracle 11g database
- Oracle 11g JDBC driver to Oracle 12c database

IBM DB2

- IBM DB2 9.5 JDBC driver to IBM DB2 9.5 database
- IBM DB2 9.5 JDBC driver to IBM DB2 9.8 database
- IBM DB2 9.5 JDBC driver to IBM DB2 10.5 database

Microsoft SQL Server

- Microsoft JDBC Driver 4.0 to Microsoft SQL Server 2008 R2 database
- Microsoft JDBC Driver 4.0 to Microsoft SQL Server 2012 database
- jTDS 1.3.1 JDBC driver to Microsoft SQL Server 2008 R2 database

- jTDS 1.3.1 JDBC driver to Microsoft SQL Server 2012 database

Before you run a baseline that has database validation steps, ensure that the driver is in the **LISA_HOME\lib\shared** directory.

Supported Data Types

The following data types are supported:

- Date
- Float
- Double
- BigDecimal
- Byte
- Short
- Int
- Long
- String
- CharacterStream
- Null
- Time
- Timestamp
- Boolean

Supported SQL

The following SQL formats are supported:

- Insert into {{table}} values ({{value1}}, {{ value2}});
- Insert into {{table}} ({{column1}}, {{column2}}) values ({{value1}}, {{ value2}});
- Insert into {{table}} ({{column1}}, {{column2}}) values (?, ?);
- Insert into {{table1}} ({{column1}}, {{column2}}) (select {{ column3}}, {{ column4}} from {{table2}} where {{condition}});
- Insert into {{table1}} ({{column1}}, {{column2}}) (select * from {{table2}} where {{condition}});
- Update {{table}} set {{column1}}={{value1}}, {{column2}}={{value2}} where {{condition}}
- Update {{table}} set {{column1}}={{value1}}, {{column2}}={{value2}} where {{condition with column1}}

- Update {{table}} set {{column1}}={{column2}} where {{condition}}
- Update {{table}} set {{column1}}=?, {{column2}}=? where {{column3}}=?
- Delete from {{table}} where {{condition}}
- Delete from {{table}} where {{column1}}=?

Database Validation Steps

The database validation steps in a baseline are SQL Database Execution (JDBC) steps.

The following graphic shows a stateful baseline that has database validation steps.

Screen capture of stateful baseline with database validation steps.

The name of each step includes the type of SQL statement that is being validated and the table name. For example:

- JDBC validation - INSERT - USERS
- JDBC validation - UPDATE - ACCOUNTS

The **Documentation** area for each step displays the complete SQL statement.

If you open the editor for a step, you can view the SELECT statement that is used to perform the validation. The following example validates an update of the ACCOUNTS table:

```
SELECT * FROM ACCOUNTS WHERE (USER_ID = '{{Step1_param_14}}') AND (account_id =
'{{Step1_JDBC_rsp.filter_id_17}}')
```

Notice the use of parameters in the statement.

For stateful and expanded baselines, each step includes an Ensure Result Set Size assertion.

For expanded baselines, each test case includes a Fail Test Case companion.

> **Note:** DevTest might not have enough information to create database validation steps with an adequate degree of certainty. The steps are generated, but the names begin with the word **Suggested**. You can modify or remove these steps.

Test Templates

When creating a baseline from DevTest Portal or the Application Insight command-line tool, you can specify a test case whose steps will be included in the baseline. The test case is referred to as a *test template*.

This feature is also available when you create a functional test.

The following windows in DevTest Portal let you select a test template:

- Analyze Transactions
- Document Transactions

The following graphic shows the template chooser in the Analyze Transactions window.

Screen capture of template chooser.

To specify the template from the command-line tool, use the **--cai-template** option.

The template must include a Do-Nothing step that has the name **CAI Template Place Holder**. This name is case insensitive.

In the generated baseline, the baseline steps are inserted after the Do-Nothing step.

The companions, global filters, global assertions, and other test-level information from the template are included in the baseline.

Note: If the template is located in a different project from the baseline, any data sets in the template are *not* copied to the **Data** folder of the baseline project. You must copy the data sets from the template project to the baseline project manually.

Example: Web Application Login

A baseline can include the authentication token that was used when the agent captured the transaction. When you run the baseline, you might receive an authentication failure because the web application requires a new token. Adding a template during baseline creation can help prevent the authentication failure.

The following graphic shows a template that contains the following steps:

- REST
- Do-Nothing
- Output Log Message

Screen capture of template.

The template contains an Inject HTTP Header global filter that defines the **Authorization** header. The header value is set to **{{token_type}}{{space}}{{access_token}}**. The **Skip To Step** field is set to the Do-Nothing step, which means that the header is used in all subsequent steps of the baseline. The **{{space}}** parameter is defined in the **project.config** file as a single space character.

The REST step calls the login API. The REST step contains two JSON Path filters:

- The first JSON Path filter saves the value of the token type to the **{{token_type}}** property.
- The second JSON Path filter saves the value of the access token to the **{{access_token}}** property.

The Output Log Message step indicates that the test is complete and displays the token.

The template also contains a JSON Ignored Nodes companion that excludes the access token from the baseline comparison.

The following graphic shows a stateful baseline for which the preceding template was specified. Notice the steps that were added between the Do-Nothing step and the Output Log Message step.

Screen capture of stateful baseline with template.

Run a Baseline

The procedure for running a stateful or consolidated baseline is different than the procedure for running an expanded baseline.

To run a stateful or consolidated baseline, do one of the following actions:

- Stage a quick test
- Stage a test case

For information about how to stage a quick test or a test case, see Running Test Cases and Suites.

You can monitor the results in the Consolidated Baseline tab.

For information about how to run an expanded baseline, see Run a Test Suite.

You can monitor the results in the Baseline Results panel.

Consolidated Baseline Tab

The **Consolidated Baseline** tab in DevTest Workstation lets you monitor and update baseline test cases at run time.

When you do either of the following tasks, the **Consolidated Baseline** tab appears in the Test Monitor window:

- Perform a quick test on a baseline test case
- Stage and run a baseline test case

The following graphic shows the **Consolidated Baseline** tab for a baseline test case that contains three scenarios. In the test run, the last scenario failed. The diff viewer shows where the issue occurred in the last scenario.

Screen capture of Consolidated Baseline tab for sample test run.

The left portion of the tab contains the rows of the baseline test case's Large Data data set. Each row includes a status icon:

- The color gray indicates that the scenario has not yet been run.
- The color green plus a white arrow indicates that the scenario is running.
- The color green indicates that the scenario succeeded.
- The color red indicates that the scenario failed.
- The color orange indicates that the scenario stopped.
- The color yellow indicates that the expected response was updated with the actual response.

After the baseline test case is run, you can select a row to display the results. If the scenario succeeded or failed, the diff viewer shows the expected response and the actual response. Any differences are highlighted. If the scenario stopped, the cycle history appears instead.

You can use the **Settings** tab to change the comparison options.

> **Note:** For information about the comparison options, see Graphical XML Side-by-Side Comparison.

The toolbar at the bottom contains the following buttons:

- **Enable**
  Includes the scenario in subsequent test runs. Be sure to click **Save** afterward.
- **Disable**
  Prevents the scenario from running in subsequent test runs. Be sure to click **Save** afterward.

- **Update**
  Updates the expected response with the actual response. You can perform this action when a scenario failed not because of a functional error but because the system under test has changed. Be sure to click **Save** afterward.

- **Undo**
  Reverses the most recent change.

- **Redo**
  Performs the most recent change again.

- **Save**
  Saves any changes that were made.

Baseline Results Panel

The **Baseline Results** panel in DevTest Workstation lets you monitor and update baseline suites at run time.

When you run a baseline suite, the **Baseline Results** panel opens in the **Stage Suite Execution** tab.

The following graphic shows the **Baseline Results** panel for a baseline suite that contains three tests. The last test has failed. The diff viewer shows where the issue occurred in the last test.

Screen capture of Baseline Results panel for sample suite run.

Each test includes a status icon:

- The color green indicates that the test succeeded.

- The color red indicates that the test failed (that is, an assertion fired that failed the test).

- The color orange indicates that the test stopped. For example, a test step threw an unexpected exception.

After the baseline suite is run, you can select a test to display the results. If the test succeeded or failed, the diff viewer shows the expected response and the actual response. Any differences are highlighted. If the test stopped, the cycle history appears instead.

If a test failed, you can perform one of the following actions:

- Select the test, right-click a row in the **Diff Viewer** tab, and click **Ignore**. In subsequent runs of the suite, the row is skipped for all tests in the suite.
- Select the test and click **Ignore** at the bottom. In subsequent runs of the suite, the test does not run.
- Select the test and click **Update** at the bottom. The expected response is updated with the actual response.

The toolbar at the bottom contains the following whole suite commands:

- **Rerun Suite**
  Runs the suite again.
- **View Reports**
  Displays the run statistics in the DevTest Portal.

Dynamic Responses and Transaction Frame Steps

Assume that a baseline has an Execute Transaction Frame step.

If the response changes on every request, running the Execute Transaction Frame step in the Interactive Test Run (ITR) utility will appear to hang.

This problem occurs because of the default assertion, which tries to match the previously captured response. The assertion fails because the response keeps changing.

You can still run the Execute Transaction Frame step manually. Open the step in DevTest Workstation and click **Execute**.

To make the ITR work, use one of the following approaches to update the response settings for the Execute Transaction Frame step with XPath filters:

- Manually add the XPath filters.
- In the Execute Transaction Frame step, click the **Response** tab and display the **Select Contents** subtab. For each node that represents a dynamic response, right-click the node and select **Ignore**. The node is added to the **Ignored Nodes** list.

Create Data Sets

Large Data data sets can contain large data tables with any number of rows, columns, and cell sizes. You can create Large Data data sets that can be used when generating consolidated baseline tests from DevTest Portal. A consolidated baseline can be modified after it is generated. Creating a data set from shelved transactions allows you to modify an existing baseline test without generating a new test. For example, you can use a data set to change a consolidated baseline which was generated from merged transactions.

Follow these steps:

1. Add one or more transactions frames to the shelf.

2. Open the shelf and click .

3. To change the default name click , edit the name, then click to save.

4. Specify the type of step that is included in the baseline:

   - **Use Application Test Steps**

     The baseline includes a step that corresponds to the transaction frame that you added to the shelf. For example, selecting a SOAP frame results in a baseline that includes a **Web Service Execution (XML)** step. The step makes the client calls to the system under test at runtime.

   - **Use Transaction Frame Step**

     The baseline includes an **Execute Transaction Frame** step. The system under test is not called at runtime. Instead, the agent plays back the transaction inside the JVM memory.

5. (Optional) To view and manage merged frames, perform the following steps:

   a. Click **List frames** to view the frames.
   The frames are listed in the right pane. For REST, Web HTTP, and Web Service frames, the protocol, method, and path information displays. For all other frames, the category and operation information displays. All JDBC frames have the name **SQL Activity**. To display the SQL information, place the mouse pointer over a JDBC frame.

   b. Drag-and-drop the frames up and down the list to change the order.

   c. Click **X** and click **Continue** to remove a frame from the listing.

6. Click **Next**.

7. Select the project where the data set is created.

8. (Optional) Edit the name prefix. The default prefix is the user name.

9. Select an option to keep or delete the transactions in the shelf once the data set is created.

10. Click **Finish**.
    The **Jobs in Progress** dialog opens with the job status.

11. Point to the tooltip to display information about the data set.The LDS file is created and available in DevTest Workstation from the data folder of the specified project.

Create Request/Response Pairs from Transaction Frames

You can create a data set that contains request/response pairs for shelved transaction frames. The request/response pairs can be uploaded from Application Insight and imported into a test from DevTest Portal or from DevTest Workstation.

> **Note:** To create data sets, the Extract Data permission must be set for the Runtime User in the CAI Administration permission. For more information about this permission, see Standard Permissions.

See Create a Test by Importing R/R Pairs from a File for more information about importing request/response pairs into the **Test Editor**.

You can create request/response pairs for the following transaction types:

- CICS
- JMS
- REST
- SOAP
- WebSphere MQ

The following conditions apply to shelved transaction frames when creating request/response pairs:

- Meta data files for JMS and WebSphere MQ transaction frames are created.
- Single frames generate one request/response pair per frame.
- Merged frames generate request/response pairs for each frame in the merged path.
- All occurred frames generate request/response pairs for each frame matching the frame name.
- Frames are consolidated into one request/response pair zip file for each protocol per agent.
- All occurred transactions have a separate zip file even when other frames with the same protocol are shelved.

The request/response pair zip file uses a naming convention of **(prefix)_(transactionframename)_rrpairs_(UTCTimeDateStamp).zip**. For example, **admin_1426740114537_itkoExamplesEJB3AccountControlBean_getTransactions_rrpairs_142675 zip**.

The files in the zip file for REST and SOAP use the following naming convention:

- (prefix)_(transactionframename)_rrpairs_(frametype)-n-req.txt

- (prefix)_(transactionframename)_rrpairs_(frametype)-n-rsp.txt

The files in the zip file for JMS and WebSphere MQ use the following naming convention:

- (prefix)_(transactionframename)_rrpairs_(frametype)-n-req.txt
- (prefix)_(transactionframename)_rrpairs_(frametype)-n-req-meta.properties
- (prefix)_(transactionframename)_rrpairs_(frametype)-n-rsp.txt
- (prefix)_(transactionframename)_rrpairs_(frametype)-n-rsp-meta.properties

n represents a numeric value that is assigned to transaction frames with duplicate names.

Follow these steps:

1. Add one or more transactions frames to the shelf.
2. Open the shelf and click .
   Only transaction frames that can create data sets are listed.
3. To change the default name, click , edit the name, then click to save.
4. Click **X** to remove any transaction frames that you do not want a request/response pair created.
5. To view a list of consolidated transaction frames for a request/response pair, click .
   The right pane displays a list of transactions by name and method.
6. To remove a consolidated transaction frame from a request/response pair list, click **X** for the transaction frame in the right pane.
   The frame is removed from the list and is not included in the generated request/response pair zip file.
7. To download the request/response pairs from individual transactions frames, click .
   A zip file is created.
8. To create request/response pairs for all the transaction frames in the shelf, click **Next**.
9. Select the project where the data set is created.
10. (Optional) Edit the name prefix. The default prefix is the user name.
11. Select an option to keep or delete the transactions in the shelf once the data set is created.
12. Click **Finish.**
    The **Jobs in Progress** dialog opens with the job status.
13. Point to the tooltip to display information about the request/response pair.
    The request/response pairs are created and available in DevTest Workstation in the **Data**, **RR** pair folder of the specified project. You can use the files in the zip file to create a new VS image from the request/response pairs manually.

Document Transactions

Documenting business transactions generates visibility into the system architecture and the flow of data. This visibility exposes what is happening in the backend code and database.

Document a test by recording test transactions and analyzing the generated backend calls. After the tests are run, determine whether the expected results match the actual results. You can compare one or more tests. Transactions are marked with a blue pin and display in a graphical view to identify the exceptions that have occurred during the recording. The name of the recorded test case displays in the **Points of Interest** list in the dashboard of the **Home** page.

You document transactions for testing from the **Document Transactions** window. You can perform the following tasks:

- Record test transactions
- Search for saved transaction recordings
- View and analyze recordings
- Create documentation from recorded transactions
- Delete recordings
- Compare recordings
- Show recordings in difference view
- Export recorded transactions
- Import recorded transactions
- Create a functional test for recorded transactions

The DevTest extension for the Google Chrome browser also supports documenting transactions.

More Information:

- Document a Transaction for a Manual Test

Document a Transaction for a Manual Test

This page takes you through the steps of documenting a transaction for a manual test.

- Record Your Test Transactions
- Search for Recorded Test Transactions
- View and Analyze Recorded Test Transactions
- Delete Test Recordings
- Remove Pinned Transactions from Recording
- Compare Recordings
- Show Recording Difference View

- Create Documentation from Recorded Transactions
- Export Recorded Transactions
- Import Recorded Transactions
- Create a Functional Test for Recorded Transactions

Record Your Test Transactions

Document the exceptions that occur in business transactions for a specific IP address by recording transactions. See Configure Capture Levels for Agents for information about how to adjust capture levels.

By default, the IP address displays for the system that is using the browser to connect to the DevTest Server. You can change the IP address when running the browser that connects to the agent application from a remote system. If the client browser is interacting with the application over NAT or proxy server, select the address from the **Recording IP** list. The **Recording IP** drop-down list displays up to ten addresses. If more than ten addresses are found, you can find the address by enter the address in the search field. Click the **Recording IP** field to refresh the list of addresses.

During the recording, the captured transactions are pinned as points of interest (POI). The pinned transactions display in the **Analyze Transactions** window and in the **Points of Interest** portlet of the **Home** page dashboard.

Follow these steps:

1. Select **Application Insight**, **Document Transactions** from the left navigation menu.
2. Click **Start case recorder** to start a new case recording.
   The **Record Case** dialog opens.
3. From the **Recording IP** list, select an address for the system that is interacting with the application that you want to record.
4. Enter a name for the test case.
   Including the test identifier from your ALM in the name is recommended. This name appears in the list of the **Points of Interest** portlet from the **Home** page.
5. (Optional) Select **Auto Refresh** to display the captured transactions automatically.
6. Click **Start**.
7. Run the agent application in another browser window, either on the same system or on a remote system.
8. Perform the required work on the agent system that is being recorded.
   The recorded frames display in the **Captured Frames** pane.
   A blue pin is added to the frame of the IP address that is being recorded. The captured frames display automatically in the **Captured Frames** pane when **Auto Refresh** is selected. Otherwise, click **Refresh**.
9. (Optional) To view the recorded transactions in a graphical view, click .
10. (Optional) To view the recorded transactions in a list view, click .

11. (Optional) Click **Show Outline** and move the map to view portions of large transactions at a time.

12. Click **Stop**.

13. To save the recording, click **Save**.
    The recordings that are saved are listed in the **Recording** pane of the **Document Transactions** window.

14. To create documentation for your recorded transactions, click **Create Document**.

Search for Recorded Test Transactions

Narrow the list of recordings to analyze using the search capability in the **Document Transactions** window.

Follow these steps:

1. Select **Application Insight**, **Document Transactions** from the left navigation menu.

2. Enter the text search criteria in the **Enter Search Text** field and click **Search**.

View and Analyze Recorded Test Transactions

The **Recording** pane of the **Document Transactions** window displays a list of all the recorded test transactions that were saved. From the **Recordings** list, you select a recording to display in graphical or list view from the **Recording of:** pane. You can also share a link.

Follow these steps:

1. Select **Application Insight**, **Document Transactions** from the left navigation menu.

2. Select a saved test recording from the list in the **Recording** pane.
   The test transaction opens in the lower pane in graphical view by default.

3. View the tooltip to display the transaction details.

4. To view the transaction details dialog, click a frame in the **Recordings of:** pane.
   The transactions details dialog opens.

5. To view the transaction details in list view, click **List** above the **Recording of:** pane.

6. To share the link to a recording, click **Share Link** in the **Saved Recordings** pane.

Delete Test Recordings

The **Saved Recordings** pane of the **Document Transactions** window lists of all the recorded test transactions that were saved. You can delete one or multiple recordings that are listed.

Follow these steps:

1. Select **Application Insight**, **Document Transactions** from the left navigation menu.

2. To delete one or more recordings, select the check box to the left of the recording name and click **Clear Recordings** .

3. To delete all the recordings in the list, select the **Recording** check box and click **Clear Recordings** .

Remove Pinned Transactions from Recording

During a recording, all captured transactions are pinned as points of interest (POIs) and display in the **Recording of:** pane. The transactions are also listed and pinned in **Analyze Transactions** and appear in the **Points of Interest** portlet of the **Home** page. Remove a pinned transaction that is not required for the recording. Removing a recorded pin in the **Document Transactions** window also removes the pin from the transaction in **Analyze Transactions**. The pin is removed from the **Points of Interest** portlet.

1. Select **Application Insight**, **Document Transactions** from the left navigation menu.
2. Select a recording in the **Saved Recordings** pane and click **List** .
3. Select one or more transactions in the **Recording Of:** pane and click **Remove Pin**.

Compare Recordings

Compare two recordings side-by-side in the **Document Transactions** window. In the following graphic, recording **Jh-test1** displays in the left pane and recording **Jh-test2** displays in the right pane. The comparison of the two recordings displays in **Show graphical differences** view by default. Change the view to display the recordings in **Show text differences**.

Use the legend to view information about the recording comparison. For example, transactions that are changed appear in yellow while unchanged transactions appear in white. The panes appear in blue or pink depending on which pane has the transaction that occurred or did not occur.

The following graphic displays a comparison of two recordings in **Show graphical differences** view.

The following graphic displays two recordings in **Show text difference** view:

In this view, expand the tree to display more information about the recordings.

This procedure requires at least two saved recordings.

Follow these steps:

1. From the **Document Transactions** window, click **Compare Recordings** .
   The **Compare** dialog displays.

2. Enter **Recording name (left)** and **Recording name (right)** fields and click **OK** .
   Alternatively, you drag the recording name from the **Saved Recordings** list and drop the name into the Recording name fields.
   A recording displays in a left and right pane in graphical differences view by default.

3. (Optional) Click to display the recordings in text difference view.

4. Analyze the comparison.

5. (Optional) Point to the Transaction Comparison legend to view information about the recordings.

6. Click **X** to close.

Show Recording Difference View

When comparing two recordings, select options for viewing the differences of the recordings from **Recording Diff View**. Use the legend to view information about the comparison. For example, transactions with no changes for both recordings display in white.

The following recording difference view options are available in text difference view:

- Show changed values only
  Select this option to display only transactions with changed values.

- Show response time difference
  Select this option to display the transaction with different response times. The red, downward arrow represents transaction frames with slower response times. The green, upward arrow represents transaction frames with faster response times.

- Show all details
  Select this option to display all the transaction details. Expand the tree to view more details of the transaction.

The following recording difference view options are available in graphical differences view :

- Show response time difference (X) ms
  Enter a value for the response time difference or use the up and down arrows. Only the transactions that display have the specified response time difference.

- Show changed values only
  Select this option to view only the values for transactions that changed. Transactions that are changed appear in yellow. If there are no changes, the panes are empty.

Follow these steps:

1. Complete and save at least two recordings.

2. In the **Document Transactions** window, click **Compare Recordings** .

3. Enter the **Recording name** fields and click **OK**.
   Alternatively, you can drag the names from the **Saved Recordings** list into the **Recording name** fields.
   The recordings display in a left and right pane in **show graphical differences** view by default.

4. (Optional) Click **Show text difference** to display the differences in a tree view.

5. Click **Recording Diff View** and select an option.

6. Analyze the comparison.

7. (Optional) Point to the Transaction Comparison legend to view information about the recordings.

8. Click **X** to close.

Create Documentation from Recorded Transactions

> **Note:** PDF generation is not supported when running DevTest Portal in HTTPS mode.

Create a report of the transactions that were recorded during a test from the **Document Transactions** window. The report contains information about each frame that was recorded. The agent boundary for transactions displays in the **Transactions Detail Information** section at the end of the report. You can save and print the report.

Follow these steps:

1. From the **Document Transactions** window, record and run your test.

2. Click **Create Document**.

3. Enter the title of the report and click **OK**.
   The report is generated and displays in a browser.

   > **Note:** The report can take a few minutes to generate depending on the number of transactions.

4. To save the report, point to the bottom, right of the browser to display the taskbar, click **Save**, and enter the fields.

5. To print, click **Print** and enter the fields.

Export Recorded Transactions

From the **Document Transactions** window, export recorded transaction paths into a zip file to be shared and imported. The recorded transactions are exported as XML files and are named using the name of the record case. The exported transactions can be imported into other Application Insight installations. You can export paths for CA Support to debug.

All or multiple recordings can be exported at a time. Use the search bar to find specific recordings to be exported.

> **Note:** When using the Safari browser to export a recording, the file is named **unknown** without an extension by default. The Safari browser does not allow files name to be edited when downloading. Ensure that you add the .zip extension to the file name to import the recording.

Follow these steps:

1. Select a recorded transaction from the **Saved Recordings** pane and click **Export Recording** from the **Actions** column.

2. (Optional) To select multiple recordings, select the checkboxes next to the list of recordings or select the **Recording** check box to select all recordings.
   The **Export Recording** dialog opens.

3. Enter the file name and click **Export and download recording.**
   A zip file with the name of the recorded case is created.

Import Recorded Transactions

From the **Document Transactions** window, you import a zip file that contains recorded transaction paths from various sources. The imported transactions from the recording display in the **Saved Recordings** pane with a POI. The imported, recorded transactions can be searched by name of the recording name.

From the **Document Transactions** window, you can also import transaction paths that are not recorded for use in **Analyze Transactions**. The transactions paths display in the **Analyze Transactions** and the **Document Transactions** window.

> **Note:** Duplicate recordings cannot be imported.

You can also perform this task from the command line.

Follow these steps:

1. From the **Document Transactions** window, click **Import Recording** .
   The **Import Recording** dialog opens.
2. Click **Choose** and select the zip file.
3. Review the recordings that display in the table and click **Import**.
   The Imported column displays:
   - **Yes** when the recording is imported into the database.
   - **Duplicate** when the recording already exists in the database.
   - **Failed** when the recording were not imported.
4. Click **OK**.
   A **Confirmation** dialog opens.
5. Click **OK**.
   The imported recording displays in the **Saved Recordings** pane.

Create a Functional Test for Recorded Transactions

Testing business functionality might require multiple APIs to be called in a specific order to reach the result of a business test case. Functional tests can create a functional API test in the same order as the business transactions have occurred from the UI. The data is automatically correlated between the API test, making the tests more robust.

Functional tests from recorded transactions can only be created from the root frames of transactions.

The following protocols are supported for creating functional testing of recorded transactions:

- - REST HTTP
  - Web HTTP
  - Web Service HTTP

Magic dates can be applied to the test cases when creating a functional test. Magic dates convert the date strings in the functional test to variable definition strings. Instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.

Assertions can be added when at least one of the supported protocols is a root frame in the recorded transactions. **Assertion Response Code Equals** and **Assertion Response Content** are the names of the two assertions that can be configured**.** For each assertion, you specify what happens when the assertion returns false. You can select to fail the test, generate an error, or generate a warning. If the response is not JSON, the default without the configuration option is used.

Follow these steps:

1. From the **Document Transactions** window, select the check box for one or multiple recordings in the **Saved Recordings** pane.

2. Click **Create Functional Test** .
   The **Create in Project** dialog displays.

3. Select a project, enter a prefix name (optional, default is admin).

4. Select or enter the following fields:

   - **Parameter Level**
     Specifies whether to enable parameterization and, if so, whether to match on value only or both key and value.

     > **Note:** This field is only enabled when at least one REST root frame is included in the recording.

   - **Http User**
     Specifies the web application user name. Application Insight encrypts the value and adds the result to the Authorization header field in the functional test.

   - **Http Password**
     Specifies the web application user password. Application Insight encrypts the value and adds the result to the Authorization header field in the functional test.

5. To configure the functional test to use magic dates, select **Apply magic dates to test cases.**

6. To configure the functional test to validate the database, select Database Validation.

7. To create assertions, select **Create Assertion** and select from the options.

8. To upload a test case document, click **Choose File** and select a test template.

9. Click **Finish**.
   The Jobs in Progress dialog opens.

10. Point to the tooltip to view the job status and error messages.

Troubleshooting for Documenting Transactions

This page provides information on troubleshooting for documenting transactions.

Symptom:

When I start a recording and then exercise the application, the list of captured frames is empty no matter how long I wait. The **Analyze Transactions** window does show the frames.

Solution:

If the computer has multiple IP addresses, the agent might be selecting a different IP address than the one that you entered. Try the following steps:

1. Go to the **Analyze Transactions** window.
2. Open the transaction detail dialog and select the **XML** tab.
3. Locate the **rip** attribute. The value specifies the remote IP address.
4. Use that IP address in the **Document Transactions** window.

Native MQ Application Insight

Native MQ Application Insight is a WebSphere MQ API exit that monitors the get and put calls for the queues of a queue manager. Native MQ Application Insight copies the get and put calls to a special queue named ITKO_PATHFINDER. A separate DevTest component uses the data in the ITKO_PATHFINDER queue to assemble transactions that can be viewed in DevTest Portal. You can then use the transactions to create virtual services, baselines, and so on.

This feature is supported on WebSphere MQ 6.0 and 7.1.

The following graphic shows an example of how Native MQ Application Insight works.

The **rules.xml** file on the computer where DevTest is installed points to the computer where WebSphere MQ is installed. The graphic shows the required properties.

The queue manager has three queues:

- MyQueue1
- MyQueue2
- ITKO_PATHFINDER

The ITKO_PATHFINDER queue contains copies of the get and put calls for MyQueue1 and MyQueue2. The data is converted into transactions and sent to DevTest Portal.

Native MQ Application Insight ignores certain types of queues that are not relevant. For example, any queue that has a name beginning with **SYSTEM.** is ignored.

MQPUT and MQPUT1 calls are both supported.

WebSphere MQ User Privileges for Native MQ Application Insight

This section describes the recommended permissions for the WebSphere MQ user that accesses the ITKO_PATHFINDER queue.

Provide the user with the **+connect** and **+inq** permissions on the queue manager.

The following example uses the **setmqaut** command to grant these permissions:

```
setmqaut -m MyQueueManager -t qmgr -p user1 +connect +inq
```

Provide the user with the **+get** permission on the ITKO_PATHFINDER queue.

The following example uses the **setmqaut** command to grant this permission:

```
setmqaut -m MyQueueManager -n "ITKO_PATHFINDER" -t queue -p user1 +get
```

Create the ITKO_PATHFINDER Queue

Native MQ Application Insight places copies of get and put calls in a queue named ITKO_PATHFINDER.

If the queue fills up, the oldest message is deleted.

Follow these steps:

1. Go to WebSphere MQ Explorer.
2. Create a queue with the name ITKO_PATHFINDER. This queue must be owned by the same queue manager as the queues that you want to monitor. The persistence setting, maximum queue depth, and so on, are up to you.

Install the API Exit for Native MQ Application Insight

The LISA_HOME\agent\native_mq directory contains the API exit files.

Follow these steps:

1. Shut down WebSphere MQ.
2. Navigate to the LISA_HOME\agent\native_mq directory.
3. Copy the exit to the appropriate location, depending on operating system.
   - **Windows**
     Copy the **iTKO_MQ_Pathfinder.dll** file to the MQ_HOME\exits directory.
   - **Linux**
     Copy the **iTKO_MQ_Pathfinder_linux** and **iTKO_MQ_Pathfinder_linux_r** files to the /var/mqm/exits directory. The mqm user must own the files. The files must have the read and execute bits set (chmod a+rx).
   - **Linux 64 bit**
     Copy the **iTKO_MQ_Pathfinder_linux_x64** and **iTKO_MQ_Pathfinder_linux_x64_r** files to the /var/mqm/exits64 directory. The mqm user must own the files. The files must have the read and execute bits set (chmod a+rx).

- **SUSE Linux 64 bit**
  Copy the **iTKO_MQ_Pathfinder_sles10.3_x64** and **iTKO_MQ_Pathfinder_sles10.3_x64_r** files to the /var/mqm/exits64 directory. The mqm user must own the files. The files must have the read and execute bits set (chmod a+rx).

- **Solaris**
  Copy the **iTKO_MQ_Pathfinder_sol_sparc** file to the /var/mqm/exits directory. The mqm user must own the file. The file must have the read and execute bits set (chmod a+rx).

- **Solaris 64 bit**
  Copy the **iTKO_MQ_Pathfinder_sol_sparc.64** file to the /var/mqm/exits64 directory and remove the .64 extension. The mqm user must own the file. The file must have the read and execute bits set (chmod a+rx).

- **AIX**
  Copy the **iTKO_MQ_Pathfinder_aix** and **iTKO_MQ_Pathfinder_aix_r** files to the /var/mqm/exits directory. The mqm user must own the files. The files must have the read and execute bits set (chmod a+rx).

- **AIX 64 bit**
  Copy the **iTKO_MQ_Pathfinder_aix.64** and **iTKO_MQ_Pathfinder_aix_r.64** files to the /var/mqm/exits64 directory and remove the .64 extension. The mqm user must own the files. The files must have the read and execute bits set (chmod a+rx).

4. Restart WebSphere MQ.

Configure the API Exit for Native MQ Application Insight

After you install the API exit, provide information such as the module and the entry point.

Configure the Exit on WebSphere MQ 6.0

Follow these steps:

1. Go to WebSphere MQ Explorer.
2. If you want to register with a single queue manager, right-click the queue manager and choose Properties.
3. If you want to register with all queue managers, right-click the root node and choose Properties.
4. In the Name field, enter any descriptive name.
5. In the Function field, enter **EntryPoint**. This value is case-sensitive.
6. In the Module field, enter the path to the API exit file that you installed (for example, **/var/mqm/exits/iTKO_MQ_Pathfinder_linux**).
7. Click OK.
8. Restart WebSphere MQ.
9. Send a test message to a queue.

10. Browse the ITKO_PATHFINDER queue and verify that the test message was mirrored.

Configure the Exit on WebSphere MQ 7.1

Follow these steps:

1. Go to WebSphere MQ Explorer or open the **qm.ini** configuration file.
2. Configure the queue manager to run an API exit.
   - Set the Name attribute to any descriptive name.
   - Set the Function attribute to **EntryPoint**. This value is case-sensitive.
   - Set the Module attribute to the API exit file that you installed (for example, **/var/mqm/exits/iTKO_MQ_Pathfinder_linux**).
   - If you want Native MQ Application Insight to generate log files, set the Data attribute to the directory where the log files are written.
3. Restart the queue manager.
4. Send a test message to a queue.
5. Browse the ITKO_PATHFINDER queue and verify that the test message was mirrored.

Example: qm.ini API Exit Configuration on AIX

Add the following lines to **/var/mqm/qmgrs/[QueueManager]/qm.ini**, where **[QueueManager]** is the name of the queue manager in your environment:

```
ApiExitLocal:
   Module=/var/mqm/exits64/iTKO_MQ_Pathfinder_aix
   Name=Pathfinder
   Sequence=100
   Function=EntryPoint
```

This example applies to an AIX system that has 64-bit kernel packages installed. If the system is a pure 32-bit, change the Module attribute to **/var/mqm/exits/iTKO_MQ_Pathfinder_aix**.

You could copy **iTKO_MQ_Pathfinder_aix** and **iTKO_MQ_Pathfinder_aix_r** into both **/var/mqm/exits/** (32-bit libraries) and **/var/mqm/exits64/** (64-bit libraries). Configure the Module attribute with a relative path such as **Module=iTKO_MQ_Pathfinder_aix**. This action enables WebSphere MQ to pick up the correct version of the libraries based on the default ClientExitPath attribute path that is defined in the global WebSphere MQ configuration file (**/var/mqm/mqs.ini**). The **/var/mqm/mqs.ini** file looks like the following example:

```
   DefaultPrefix=/var/mqm
ClientExitPath:
   ExitsDefaultPath=/var/mqm/exits
   ExitsDefaultPath64=/var/mqm/exits64
```

Configure the Agent Properties for Native MQ Application Insight

The **rules.xml** file for the DevTest Java Agent includes a set of Native MQ Application Insight properties that you must configure.

Follow these steps:

1. Go to the **LISA_HOME** directory and open the **rules.xml** file.

2. Add the following required properties to the **console** element:
   MQMIRROR_HOST
   The IP address or hostname of the server where WebSphere MQ is running.
   MQMIRROR_PORT
   The port number on which WebSphere MQ is listening for connection requests.
   MQMIRROR_CHANNEL
   The WebSphere MQ channel for connecting to the queue manager.
   MQMIRROR_QMNAME
   The queue manager that owns the queues you want to monitor.

3. If you need a user and password to access WebSphere MQ, add the **MQMIRROR_USER** and **MQMIRROR_PASSWD** properties.

4. If you want to specify which queues to include, add the **MQMIRROR_INCLUDE_QUEUES** property.

5. If you want to specify which queues to exclude, add the **MQMIRROR_EXCLUDE_QUEUES** property.

6. If you want to specify the number of milliseconds to wait before attempting to reconnect if WebSphere MQ goes down, add the **MQMIRROR_CONNECT_INTERVAL** property.

7. Click **Save**.

8. Restart all services.

Example

This example includes all of the required properties and some of the optional properties.

```
<?xml version="1.0" encoding="UTF-8"?>
<rules>
   <console>
      <property key="MQMIRROR_HOST" value="localhost"/>
      <property key="MQMIRROR_PORT" value="1414"/>
      <property key="MQMIRROR_CHANNEL" value="SYSTEM.DEF.SVRCONN"/>
      <property key="MQMIRROR_QMNAME" value="QueueManager"/>
      <property key="MQMIRROR_USER" value="mqm"/>
      <property key="MQMIRROR_PASSWD" value="mqmpass"/>
      <property key="MQMIRROR_INCLUDE_QUEUES" value="APP\.QUEUE.*"/>
      <property key="MQMIRROR_EXCLUDE_QUEUES" value="CICS_GATEWAY.*|SYSTEM.*"/>
      <property key="MQMIRROR_CONNECT_INTERVAL" value="5000"/>
    </console>
</rules>
```

MQMIRROR_INCLUDE_QUEUES and MQMIRROR_EXCLUDE_QUEUES

By default, Native MQ Application Insight monitors all the queues of a queue manager. You can use the **MQMIRROR_INCLUDE_QUEUES** and **MQMIRROR_EXCLUDE_QUEUES** properties to override this behavior. The property values are regular expressions. The **MQMIRROR_EXCLUDE_QUEUES** property takes precedence over the **MQMIRROR_INCLUDE_QUEUES** property.

If you want to specify a queue name that contains a dot character, place a backslash immediately before the dot character.

If you want to use a dot character as a regular expression construct, you do not need to include a backslash.

QUEUE_REQUEST_MATCHES and QUEUE_RESPONSE_MATCHES

Each transaction frame includes an **is request** flag and an **is response** flag. The transaction assembler uses the flags to help stitch together full MQ transactions.

For an MQ transaction frame, these flags are set based on guesswork: a combination of checking message type, correlation IDs, and replyTo in the MQ message headers.

The **QUEUE_REQUEST_MATCHES** and **QUEUE_RESPONSE_MATCHES** properties let you override this behavior. For example:

```
<property key="QUEUE_REQUEST_MATCHES:queue.in" value=".*"/>
<property key="QUEUE_RESPONSE_MATCHES:queue.out" value=".*substring.*"/>
```

The property names must be followed by a colon and the queue name. The values must be a regular expression.

The regular expressions are checked against the entire MQ message body:

- If the **QUEUE_REQUEST_MATCHES** property matches the body, the request flag is set.
- If the **QUEUE_RESPONSE_MATCHES** property matches the body, the response flag is set.
- If both properties match the body, the request flag takes precedence.

Generate Transactions from the ITKO_PATHFINDER Queue

The **LisaAgent.jar** file includes an option for creating transactions that are based on the messages that have been mirrored to the **ITKO_PATHFINDER** queue.

If the broker is running on another computer, you must also specify the **-u** option with the URL to the broker.

Follow these steps:

1. (WebSphere MQ 6) Copy the following JAR files from the **MQ_HOME\java\lib** directory to the directory where the **LisaAgent.jar** file is located:
   - com.ibm.mq.jar
   - com.ibm.mq.pcf.jar
   - com.ibm.mqjms.jar
   - connector.jar
   - dhbcore.jar
2. (WebSphere MQ 7.1) Copy the following JAR files from the **MQ_HOME\java\lib** directory to the directory where the **LisaAgent.jar** file is located:
   - com.ibm.mq.commonservices.jar
   - com.ibm.mq.headers.jar
   - com.ibm.mq.jar

- com.ibm.mq.jmqi.jar
- com.ibm.mq.pcf.jar
- com.ibm.mqjms.jar
- connector.jar
- dhbcore.jar

3. Ensure that a TCP listener is alive on the WebSphere MQ server, listening at the port that was configured for the MQMIRROR_PORT property.

4. Run the following command:

   ```
   java -jar LisaAgent.jar -m
   ```

   The messages are retrieved from the **ITKO_PATHFINDER** queue and assembled into transactions. You can view the transactions in the DevTest Portal.

Stop and Resume Message Mirroring

You can stop the copying of messages to the ITKO_PATHFINDER queue without restarting the queue manager.

This feature is supported for WebSphere MQ in client mode and bindings mode.

Directory Configuration

In the next section, you create a file named **stopMirror.txt** and place it in a directory. By default, you must use one of the following directories:

- (UNIX, Linux, AIX) **/tmp**
- (Windows) **C:\temp**

To change the location of the directory where the file must be placed on UNIX, Linux, and AIX, do the following steps:

1. In a shell prompt, type the following command:

   ```
   # export LISA_HOME=/some/directory
   ```

2. In the same shell, stop and restart the queue manager or queue managers.

3. In the same shell, stop and restart the application.

To change the location of the directory where the file must be placed on Windows, do the following steps:

1. Create a global environment variable named **LISA_HOME** and set the value to your preferred directory.

2. Stop and restart the queue manager or queue managers.

Add the stopMirror.txt File

The presence of the **stopMirror.txt** file causes the messages to not be copied.

Follow these steps:

1. Create a file with the name **stopMirror.txt**. The file name is case sensitive. The file can be empty.

2. Place the file in the default directory or the new directory that you configured. Messages are no longer copied to the ITKO_PATHFINDER queue.

3. To resume the copying of messages, delete or rename the file.

Troubleshooting Native MQ Application Insight

This page provides information on troubleshooting Native MQ Application Insight issues.

AIX Problems

If you configure the API exit files and then have a problem with an application on your system, do the following tasks:

- Verify that you copied the correct exit files to the appropriate WebSphere MQ directory.

- Check the error logs in the **/var/mqm/errors** directory.

If you see errors such as the following where the exits library fails to load, try the workaround that is listed afterward.

> **Note:** If a 32-bit application is used, you might see **/var/mqm/exits /iTKO_MQ_Pathfinder_aix_r** (that is, **exits** instead of **exits64**).

Problem 1:

```
-----------------------------------------------------------------------------
06/03/15 19:14:10 - Process(336040.1) User(mqm) Program(java)
AMQ6175: The system could not dynamically load the shared library
'/var/mqm/exits64/iTKO_MQ_Pathfinder_aix_r'. The system returned error number
'8' and error message 'Could not load module
/var/mqm/exits64/iTKO_MQ_Pathfinder_aix_r.
The module has an invalid magic number.'. The queue manager will continue without
this module.

EXPLANATION:
This message applies to AIX systems. The shared library
'/var/mqm/exits64/iTKO_MQ_Pathfinder_aix_r' failed to load correctly due to a
problem with the library.
ACTION:
Check the file access permissions and that the file has not been corrupted.
----- amqxufnx.c : 1154 -------------------------------------------------------
06/03/15 19:14:10 - Process(336040.1) User(mqm) Program(java)
AMQ7214: The module for API Exit 'Pathfinder' could not be loaded.

EXPLANATION:
The module '/var/mqm/exits64/iTKO_MQ_Pathfinder_aix_r' for API Exit
'Pathfinder' could not be loaded for reason xecU_S_LOAD_FAILED.
ACTION:
Correct the problem with the API Exit module 'Pathfinder'.
----- amqzuax0.c : 634 --------------------------------------------------------
```

Problem 2:

Exception in thread "main" java.lang.UnsatisfiedLinkError: /usr/mqm/java/lib64/libmqjbnd05.so: load ENOENT on shared library(s) /usr/mqm/java/lib64/libmqjbnd05.so libc.a

Workaround:

First, determine whether the AIX operating system is a 32-bit or 64-bit architecture.

```
# getconf KERNEL_BITMODE
64
```

Then create a symbolic link of **libc.a** in the WebSphere MQ installation directory. The default directory is **/usr/mqm** on AIX. Notice the space and a dot character after **.a**.

```
# cd /usr/mqm/lib
# ln -s /usr/lib/libc_r.a .
# ln -s /usr/lib/libc.a .
```

If the architecture is 64 bit, create another symbolic link. Notice the space and a dot character after **.a**.

```
# cd /usr/mqm/lib64
# ln -s /usr/lib/libc_r.a .
# ln -s /usr/lib/libc.a .
```

Finally, verify that the symbolic link or links were created correctly.

```
# ls -l /usr/mqm/lib64/libc*.a
lrwxrwxrwx 1 root system 15 Jun 4 21:23 /usr/mqm/lib64/libc.a -> /usr/lib/libc.a
lrwxrwxrwx 1 root system 17 Jun 4 21:23 /usr/mqm/lib64/libc_r.a -> /usr/lib/libc_r.
a

# ls -l /usr/mqm/lib/libc*.a
lrwxrwxrwx 1 root system 15 Jun 5 18:05 /usr/mqm/lib/libc.a -> /usr/lib/libc.a
lrwxrwxrwx 1 root system 17 Jun 5 18:06 /usr/mqm/lib/libc_r.a -> /usr/lib/libc_r.a
```

Java Application Cannot Load MQ Libraries

Symptom:

Exception in thread "main" java.lang.UnsatisfiedLinkError: Can't find library mqjbnd05 (libmqjbnd05.a or .so) in sun.boot.library.path or java.library.path

Solution:

Set the following environment:

```
LIBPATH=/usr/mqm/lib64:/usr/mqm/java/lib64/:/var/mqm/exits64/:/usr/lib:/lib;
export LIBPATH (remove 64 if 32-bit jre is used).
LD_LIBRARY_PATH="$LIBPATH"; export LD_LIBRARY_PATH
JAVA_VM_OPTS="-Djava.library.path=$LIBPATH $JAVA_VM_OPTS" (provide these options
when starting your java process)
```

Application Insight Command-Line Tool

The **PFCmdLineTool** command lets you perform various Application Insight tasks from the command line.

The main options are **--count, --roots, --paths, --export, --import, --baseline,** and **--virtualize**.

This command has the following format:

```
PFCmdLineTool [--count|--roots|--paths|--export|--import|--baseline|--virtualize|--
help|--version] [task-specific options] [search-criteria]
```

- Search Criteria
- Querying and Display
- Export and Import
- Baselines
- Virtualization Artifacts
- Agent Condition
- Capture Levels
- Miscellaneous
- Example: Display Root Transaction Frames
- Example: Display Transaction Frame Hierarchy
- Example: Generate a Baseline Suite
- Example: Generate a Stateful Baseline
- Example: Generate a Baseline with a Template
- Example: Generate Virtualization Artifacts
- Example: Check the Agent Condition
- Example: Configure the Capture Level
- Example: Search Transaction Frame
- Example: Generate Large Data Set File
- Example: Import Recorded Transactions

Search Criteria

For all the main options except **--import**, the set of transaction frames that are acted on is defined by those matching the search criteria specified. Use any of the following options to specify the search criteria: **--from, --to, --localIP, --remoteIP, --category, --class, --method, --session, --transaction, - agent, --min-time**, **--tag**, and **--max-frames**. These options are used to narrow the set of root transaction frames that are acted on.

Alternately, you can use the **--frame** option to limit the search results to a single transaction frame.

- **--from=start-time**
  Specifies the start time of the window of transaction frames you want. Use either of the following formats: **yyyy-mm-dd** or **yyyy-mm-ddThh:mm:ss**. If this option and the **--frame** option are not specified, the start of the current day is used.

- **--to=end-time**
  Specifies the end time of the window of transaction frames you want. Use either of the following formats: **yyyy-mm-dd** or **yyyy-mm-ddThh:mm:ss**. If this option and the **--frame** option are not specified, the end of the current day is used.

- **--localIP=IP address**
  Specifies the client-side IP address that Application Insight records.

- **--remoteIP=IP address**
  Specifies the server-side IP address that Application Insight records.

- **--category=category**
  Specifies the type of transaction frame. You can search for more than one category at a time by repeating this option.
  Values: amx, client, dn_default, dn_remoting, dn_sql, ejb, gui, jca, jdbc, jms, logging, mq, rest_http, rmi, rmi_http, rmi_ssl, thread, throwable, tibco, web_http, web_https, wm, wps, ws_http, ws_https

- **--class=class-name**
  Specifies the class name. The value can be either the full class name or a string ending with '%' to perform a "starts with" type of match.

- **--method=method-name**
  Specifies the method name. The value can be either the full method name or a string ending with '%' to perform a "starts with" type of match.

- **--session=session-id**
  Specifies the session identifier.

- **--transaction=transaction-id**
  Specifies the transaction identifier.

- **--agent=agent-name**
  Specifies the name of the agent that is the source for the transaction frames you want.

- **--min-time=min-time**
  Specifies the minimum execution time (in milliseconds) of transaction frames you want to see. Frames that are executed faster than this value are filtered out.

- **--tag=name, --tag=name=value**
  Specifies a frame tag that is associated with a transaction frame. You can specify the name only, or both the name and value. You can search for more than one tag at a time by repeating this option.

- **--max-frames=max-frames**
  Specifies the maximum number of root transaction frames that are retrieved. If this option is not specified, it defaults to 10000. There can be circumstances when more frames are processed than the maximum.

- **--frame=frame-id**
  Specifies the identifier of the particular transaction frame desired.

- **--username=frame-name**
  Specifies the DevTest user name. This option is used only with the **--import-docgen** option.

- **--password=frame-name**
  Specifies the DevTest password. This option is used only with the **--import-docgen** option.

Querying and Display

The following options let you query and display transactions.

- **-c, --count**
  Displays the number of root transaction frames that match the specified search criteria.

- **-r, --roots**
  Displays the root transaction frames that match the specified search criteria.

- **-p, --paths**
  Displays the transaction frame hierarchy for a set of root transaction frames.

Export and Import

The following options let you export and import paths.

- **-e output-zip-file-name, --export=output-zip-file-name**
  Exports the selected transaction frames from the database.

- **-i input-zip-file-name, --import=input-zip-file-name**
  Imports transactions or document recordings into the database. This option ignores any search criteria that is also specified. If you are importing a document recording, you must specify a user name and password.

- **--no-persist**
  Specifies that the import operation does not persist the data, causing the import file to be validated only.

- **--import-docgen=input-zip-file-name**
  Imports document recordings into the database. This option ignores any search criteria that is also specified. You must specify a user name and password.

> **Note:** When DevTest is connected to a database other than the default Derby, you need to specify the database to the broker element of the **rules.xml** file.
>
> <database driver="yourdatabasedriver" url="yourdatabaseurl" user="yourdatabaseuser" password="yourdatabasepassword"/>
>
> This setting is used for the lifetime of the agent.

Baselines

The following options let you create baseline test cases and suites. The **--to-dir** option or the **--to-project** option is required. You cannot use both **--baseline-template** and **--cai-template**.

- **-b name, --baseline=name**
  Generates baseline tests for the selected transaction frames.

- **--refer=frame-id**
  Specifies the identifier of the particular transaction frame that is designated as the key transaction frame. If this option is not specified, the first transaction frame in the query result is designated as the key transaction frame.

- **--stateful**
  Specifies that a stateful baseline is generated.

- **--consolidated**
  Specifies that a consolidated baseline is generated.

- **--force-tf-steps**
  Specifies that baseline tests are generated using the Execute Transaction Frame step only. If this option is not specified, any available CA Application Test test steps are used whenever possible. This option is not supported for web services with attachments.

- **--magic-dates**
  Specifies that baseline tests have magic date processing applied.

- **--baseline-template=baseline-template**
  Specifies the path to a test case to use when generating baseline tests. This option supports consolidated and expanded baselines only.

- **--cai-template=cai-template**
  Specifies the path to a test template to use when generating baseline tests.

- **--http-user=http-user**
  Specifies the web application user name. Application Insight encrypts the value and adds the result to the Authorization header field in the baseline. This option is applicable to stateful baselines for the following protocols: REST, Web HTTP, and Web Service.

- **--http-passwd=http-passwd**
  Specifies the web application user password. Application Insight encrypts the value and adds the result to the Authorization header field in the baseline. This option is applicable to stateful baselines for the following protocols: REST, Web HTTP, and Web Service.

- **--param-level=param-level**
  Specifies the parameterization level for REST and web service stateful baselines. The valid values are 1 (match on value only) and 2 (match on key and value).

- **--assertion=assertion-type**
  Specifies the stateful baseline assertion type.
  Values: 1, 2, 3. If the value is set to 1 and the assertion returns false, the test fails. If the value is set to 2 and the assertion returns false, the test generates a warning. If the value is set to 3 and the assertion returns false, the test generates an error. The default value is 1.

- **--jndi-factory=factory-class-name**
  Specifies the JNDI factory class to use when generating EJB baseline tests.

- **--jndi-url=url**
  Specifies the JNDI URL to use when generating EJB baseline tests.

- **--jndi-user=user-id**
  Specifies the JNDI user to use when generating EJB baseline tests.

- **--jndi-user-cred=user-credentials**
  Specifies the JNDI user credentials to use when generating EJB baseline tests.

- **--to-dir=output-pfi-file-name**
  Specifies the name of the directory where the generated PFI file is written.

- **--to-project=lisa-project-dir**
  Specifies the root directory of a project into which the generated artifacts are written, without the requirement of an intermediate import file. You can specify this option instead of, or in addition to, the **--to-dir** option.

- **--database-validation**
  Performs database validation for HTTP, web service, or REST transactions.

Virtualization Artifacts

The following options let you generate virtualization artifacts. The **--to-dir** option or the **--to-project** option is required.

- **-v name, --virtualize=name**
  Generates virtualization artifacts for the selected transaction frames.

- **--refer=frame-id**
  Specifies the identifier of the particular transaction frame that is designated as the key transaction frame. If this option is not specified, the first transaction frame in the query result is designated as the key transaction frame.

- **--force-stateless**
  Specifies that VSE conversational processing is applied when generating a service image.

- **--unknown-request-action=no_match|no_hijack**
  Specifies the action that is taken for unknown requests. This option applies to generated virtualization artifacts based around Java VSE. The valid values are **no_match** (return a "no match") and **no_hijack** (bypass the virtual service).

- **--to-dir=output-pfi-file-name**
  Specifies the name of the directory where the generated PFI file is written.

- **--to-project=lisa-project-dir**
  Specifies the root directory of a project into which the generated artifacts are written, without the requirement of an intermediate import file. You can specify this option instead of, or in addition to, the **--to-dir** option.

- **--consolidate-by-name**
  Specifies that only frames with the same name as the selected frame are included.

- **--vrs=file-location**
  Specifies the name and location of the recording session file that contains the configuration information for the recording or import operation. If the **--force-stateless** option is also specified, it is ignored because the recording session file has this information.

Agent Condition

The following options let you check for a specified condition in an agent. The only supported condition is whether the agent is currently dispatching. The term *dispatching* refers to the action of capturing transactions.

- **--check**
  Checks for a specified condition in an agent.

- **--agent=agent-name**
  Specifies the name of the agent.

- **--condition=condition**
  Specifies the condition to check for.
  Values: Dispatching

- **--check-timeout=number-of-seconds**
  Specifies the number of seconds to wait for the condition before timing out.

Capture Levels

The following options let you modify the capture level for each protocol that the Java agent can capture.

> **Note:** Setting different capture levels is not supported for queue-based client and server communication, for example, WebSphere MQ and JMS.

- **--set-weights**
  Modifies the capture level for one or more protocols.

- **--agent=agent-name**
  Specifies the name of the agent.

- **--protocols="protocol[,protocol]"**
  Specifies the protocol names. If you include more than one value, separate the values by commas.
  Values: ALL, HTTP Client, HTTP Server, Logging, Category, Exception, GUI, EJB, JMS, MQ, JCA, RCP, RMI, SAP, Tibco, WPS, WebMethods, JDBC, JAVA

- **--weights="weight[,weight]"**
  Specifies the protocol weights. If you include more than one value, separate the values by commas.
  Values: 0, 4, 8. The value 0 corresponds to the **Counts** level. The value 4 corresponds to the **Counts and Paths** level. The value 8 corresponds to the **Full Data** level.

Miscellaneous

The following options are also available.

- **--broker=broker-url**
  Specifies the broker connection string. The default value is **tcp://localhost:2009**.

- **--search-frame**
  Searches for the transaction frame that matches the specified search criteria. You can use any of the following arguments: **--frame-name**, **--method**, **--parent-frame-name**, **--child-frame-name**, **--category**, **--localIP**, **--remoteIP**, and **--class**.

- **--frame-name=frame-name**
  Specifies the name of a transaction frame to search for.

- **--parent-frame-name**
  Specifies the name of a parent transaction frame to search for.

- **--child-frame-name**
  Specifies the name of a child transaction frame to search for.

- **-h, --help**
  Displays help text.

- **--lds=large-dataset-file-name**
  Generates a large data set file for the selected transaction frames, without creating a consolidated baseline. You must also specify the following options: **--frame-name**, **--from**, **--to** , and **--to-dir**.

- **--version**
  Displays the version number.

Example: Display Root Transaction Frames

This example shows how to display the root transaction frames for a specified agent and start time.

```
PFCmdLineTool --roots --agent=JBoss_LISABank --from=2014-03-14T12:28:00
```

```
Frame ID                                Time                      Category  Local IP
      Remote IP      Name     Exec Time
------------------------------------  -----------------------  --------  ---------
----  -------------  -------  ---------
c3a9c830-abae-11e3-b937-0024d6ab5ce2  2014-03-14 12:28:
04 660  web_http  10.132.92.143  10.132.92.143  Unknown  984 ms
```

Example: Display Transaction Frame Hierarchy

This example shows how to display the transaction frame hierarchy for a set of root transaction frames.

```
PFCmdLineTool --paths --agent=JBoss_LISABank --from=2014-03-14T12:28:00
```

```
984 ms /lisabank/buttonclick.do (c3a9c830-abae-11e3-b937-0024d6ab5ce2)
   50 ms $Proxy93.getUser (c3c73b40-abae-11e3-b937-0024d6ab5ce2)
      37 ms EJB3UserControlBean.getUser (c3c8c1e0-abae-11e3-b937-0024d6ab5ce2)
         3 ms SQL Activity (1) (c3c8c1e0-abae-11e3-b937-0024d6ab5ce2-SQL)
```

Example: Generate a Baseline Suite

This example shows how to generate a baseline suite.

```
PFCmdLineTool --baseline=BaselineName --refer=c3c8c1e0-abae-11e3-b937-
0024d6ab5ce2 --to-dir=C:\DevTest
```

Example: Generate a Stateful Baseline

This example shows how to generate a stateful baseline.

```
PFCmdLineTool --baseline=BaselineName --stateful --from=2014-03-14T12:28:00 --
refer=c3c8c1e0-abae-11e3-b937-0024d6ab5ce2 --to-dir=C:\DevTest
```

Example: Generate a Baseline with a Template

This example shows how to specify a test template when generating a baseline.

```
PFCmdLineTool --baseline=BaselineName --consolidated --cai-template=C:
\DevTest\Projects\MyProject\Tests\template.tst --to-project=C:
\DevTest\Projects\MyProject --from=2015-06-04T04:17:35 --to=2015-06-04T05:24:50
```

Example: Generate Virtualization Artifacts

This example shows how to generate virtualization artifacts.

```
PFCmdLineTool --virtualize=VSEServiceName --from=2014-03-14T12:28:00 --
refer=c3c8c1e0-abae-11e3-b937-0024d6ab5ce2 --category=ejb --to-dir=C:\DevTest
```

Example: Check the Agent Condition

This example shows how to check whether an agent is capturing transactions. The output indicates that the agent is not capturing transactions.

```
PFCmdLineTool --check --agent=JBoss_LISABank --condition=Dispatching --check-
timeout=10
Agent has not yet started dispatching.
```

Example: Configure the Capture Level

This example shows how to modify the capture level for one protocol.

```
PFCmdLineTool --set-weights --agent=JBoss_LISABank --protocols=EJB --weights=8
```

This example shows how to modify the capture level for multiple protocols. Notice the use of quotation marks.

```
PFCmdLineTool --set-weights --agent=JBoss_LISABank --protocols="EJB,JMS" --
weights="8,4"
```

Example: Search Transaction Frame

This example shows how to verify whether Application Insight captured a particular transaction frame by the frame name. If the search finds more than one matching frame, the newest frame is displayed. The output consists of the name, duration, and unique identifier.

```
PFCmdLineTool --search-frame --frame-name=EJB3AccountControlBean.addAccount --
from=2014-03-20T10:20:00
EJB3AccountControlBean.addAccount, 141 ms, 27ca1bd0-b03c-11e3-a8f1-005056ba138a
```

Example: Generate Large Data Set File

This example shows how to generate a large data set file for a transaction frame.

```
PFCmdLineTool --lds=test1 --frame-name=/jmx-console/ --from=2015-05-20T11:00:00 --
to=2015-05-20T11:01:00 --to-dir=c:\test
```

Example: Import Recorded Transactions

This example shows how to import a zip file that contains recorded transactions.

```
PFCmdLineTool --import=c:/docgen.zip --username=admin --password=admin
```

VS Traffic to CA Application Insight Companion

The VS Traffic to CA Application Insight Companion pushes data to the Application Insight database to generate virtual services and visibility into the application. When you add this companion to a virtual service model, it performs the following actions:

- Captures the requests that the model processes and their corresponding responses.
- Saves the requests in the Application Insight database as transaction frames.

This companion supports HTTP, JMS, and WebSphere MQ models.

Prerequisites

- Messages from an application are captured by the VSE Recorder.

- A virtual service model file was created. See Using CA Service Virtualization for more information about creating a virtual service model.

Follow these steps:

1. Open the virtual service model.
2. In the right panel, select **Companions** and click **Add** .
   The **Companions** menu opens.
3. Select **Virtual Service Environment**, **VS Traffic to CA Application Insight Companion**.
4. Save the virtual service model.
5. Right-click the virtual service and select **Deploy/Redeploy to VSE@default**. See Using CA Service Virtualization for more information about deploying a virtual service.
   A message indicates that the virtual service was deployed to the **VSE@default** server.
6. Invoke the virtual service.
   The transaction frame is created.

   Note: The transaction frames that are created by this companion display in DevTest Portal with a virtualized label at the end of the name. An example of a component name for an HTTP transaction is **POST/itkoExamples /EJB3AccountControlBean(virtualized).** See Path Graph for more information about component names.

7. Go to DevTest Portal and create the baseline test. See Creating Baselines for more information about creating baselines for HTTP, JMS, and WebSphere protocols.
8. Create a DevTest test from the baseline.
9. Verify the test in DevTest Workstation.

DevTest Extension for Google Chrome

The DevTest extension for the Google Chrome browser lets you perform the following tasks:

- Capture and analyze HTTP traffic
- Create tickets
- Create functional tests
- Document transactions

This feature is also referred to as the Chrome extension or the Chrome plug-in.

You access the extension from the Chrome Developer Tools (DevTools) window.

The extension requires the following components to be running:

- Enterprise Dashboard Server
- Registry
- Broker
- Portal

The registry, broker, and portal components must be running on the same system. This requirement is specific to the extension. For other parts of DevTest, the portal can run on another system.

The extension does not require the use of the DevTest Java Agent.

More Information:

- Installing and Configuring the DevTest Extension for Google Chrome
- Using the DevTest Extension for Google Chrome

Installing and Configuring the DevTest Extension for Google Chrome

You install the extension from the Chrome Web Store. You then perform various configuration tasks as needed.

The procedures in this page are based on version 50 of the Chrome browser.

- Install the Extension
- Configure the Server Settings
- Include and Exclude URLs
- Configure the SSL Settings

Install the Extension

The extension is available from the Chrome Web Store. You can go directly to the Chrome Web Store, or you can access it from DevTest Portal.

The name of the extension in the Chrome Web Store is **CA SV/App Test Extension for Chrome**.

For Chrome versions 65 and later, you must use the DevTest Chrome extension version 1.0.5.14 or later.

The following procedure describes how to install the extension from DevTest Portal.

Follow these steps:

1. Open DevTest Portal.
2. Click the down arrow to the right of your user name and select **The Sandbox**.

3. Click the **Download Extension** link.

4. Click **ADD TO CHROME**.
   The browser prompts you to confirm the new extension.

5. Click **Add extension**.

Configure the Server Settings

To communicate with DevTest Server, the extension uses a group of server settings. The following procedure describes how to change the values.

If DevTest Server is installed on a different computer than the extension, be sure to change the **Host Name** setting.

Follow these steps:

1. Click the Chrome menu icon and select **More tools**, **Developer tools**.

2. Click **CA CAI**.

3. Click the **Server settings** icon.

4. Change one or more of the following settings:

   a. **Host Name**
      The name of the computer where DevTest is installed. The default value is localhost.

      You can set the server name with the prefix "http://" or "https://". The default behavior if neither is present is "http://". When SSL is enabled between DevTest components, you must use "https://".

   b. **Console Port**
      The port number of the embedded web server in DevTest. The default value is 1505.

   c. **Portal Port**
      The port number of DevTest Portal. The default value is 1507.

   d. **User Name**
      The user name that is used to access DevTest Solutions. The default value is admin.

   e. **Password**
      The password that is used to access DevTest Solutions. The default value is admin.

5. Click **Update**.

Include and Exclude URLs

The extension gives you control over which URLs to include or exclude.

> **Note:** The include settings and the exclude settings are mutually exclusive. You cannot enable both the include settings and the exclude settings at the same time.

By default, the extension is configured to exclude certain types of request and response URLs. For example:

- Request URLs that end in .js
- Request URLs that end in .png or .jpg
- Responses that have content type of **image/png** or **image/jpeg**
- Responses that have content type of **application/javascript**

If you configure URLs to include, the extension captures *only* those URLs.

Follow these steps:

1. Click the Chrome menu icon and select **More tools**, **Developer tools**.
2. Click **CA CAI**.
3. Click the **URL filters** icon.
4. To configure the include settings:
    a. Select the **Include URL** option.
    b. Enter the URL to include as a string or regular expression.
    c. Click **Add**.
    d. To include any paths that are selected in the left pane of the extension interface, click **Add selected paths**.
5. To configure the exclude settings:
    a. Select the **Exclude URL** option.
    b. Add and remove the request filters as needed. You can also disable all the request filters.
    c. Add and remove the response filters as needed. You can also disable all the response filters.
    d. To exclude any paths that are selected in the left pane of the extension interface, click **Add selected paths**.
6. To save your changes and apply them to the current list of paths, click **Save & Apply**.
7. To save your changes without applying them to the current list of paths, click **Save**.

Configure the SSL Settings

You can configure the client certificate information for two-way SSL.

The following graphic shows the **SSL Settings** dialog after the configuration settings are saved.

Screen capture of SSL Settings dialog.

Follow these steps:

1. Click the Chrome menu icon and select **More tools**, **Developer tools**.
2. Click **CA CAI**.
3. Click the **SSL settings** icon.
4. Select the **Enable Client Certification** check box.
5. Click **Choose File** and select the keystore file.
6. Enter the values for the keystore password, key alias, and key password.
7. Click **Save**.
   The keystore file is uploaded to DevTest Server. The **File Path On Server** field displays the location.

Using the DevTest Extension for Google Chrome

You open the extension from within the web page that you want to capture. You can then capture and analyze HTTP traffic, create tickets, create functional tests, and document transactions.

The procedures in this page are based on version 50 of the Chrome browser.

- Capture and Analyze HTTP Traffic
- Add Transaction Frames to the Shelf
- Create Tickets
- Create Functional Tests
- Document Transactions

Capture and Analyze HTTP Traffic

You can capture HTTP traffic from any web page.

The following graphic shows the left and center panes of the extension interface. These panes let you view information about the captured traffic.

Screen capture of left and center panes.

The initial status of a transaction is a red circle with a diagonal line inside . This status indicates that the transaction has been captured but not saved in DevTest Server.

When the transaction is saved, the status changes to a green circle with a white checkmark inside .

For each transaction, you can view the following details:

- Request that was sent to the endpoint
- Response that was received from the endpoint
- Transaction frame data in JSON format

The transaction frame data includes the agent ID and agent name. These items refer to the extension, not the DevTest Java Agent.

If you configured the SSL settings for two-way SSL, the transaction frame data includes the SSL information.

The extension cannot capture attachment content in the HTTP traffic.

You can also view the transactions in the **Analyze Transactions** window of DevTest Portal. The **Agent** column for these transactions contains the value **ChromeDevTools-Agent**.

Follow these steps:

1. Open the web page.
2. Click the Chrome menu icon and select **More tools**, **Developer tools**.
3. Click **CA CAI**.
4. (Optional) Select the **Capture screenshots** check box. This feature is supported for the following content types: HTML, JSON, and XML. The screenshot does not include any portion of the page that is hidden behind the Chrome DevTools window. You can increase the visible portion of the page by undocking the Chrome DevTools window into a separate window.

5. (Optional) Select the **Enable merging (inconsistent)** check box. If the DevTest Java Agent is configured in the system under test, the extension attempts to merge the transactions that it creates with the more detailed transactions that the DevTest Java Agent creates. Because of a timing issue with the broker, the merge might not succeed.

6. Perform operations in the web page.
   One or more transactions appear in the left pane.

7. Click a transaction.
   The path graph appears in the center pane.

8. To view the details for a node in the path graph, click the node. If you selected the **Capture screenshots** check box, the details include the screenshot.

9. To remove the transactions in the left pane, click the **Clear all transaction panels** icon.

Add Transaction Frames to the Shelf

The extension includes a shelf that is similar to the shelf in the **Analyze Transactions** window of DevTest Portal.

> **Note:** Before you can create functional tests or document transactions, you must add one or more transaction frames to the shelf.

The shelf is located in the right pane.

The following graphic shows a pair of transaction frames in the shelf.

Screen capture of frames in shelf.

The following procedure describes an option that is named **Shelve relevant (current session)**. If you make the same HTTP call more than once in the current session, the duplicate calls are treated as relevant calls. Selecting the option adds the frames for these duplicate calls to the shelf.

Follow these steps:

1. To shelve one or more frames in the left pane, select the check box for each frame and click the **Shelve selected paths** icon.

2. To shelve only the REST frames in the left pane, click the **Shelve RESTful web service API paths** icon.

3. To shelve all frames in the left pane, click the **Shelve all paths** icon.

4. To shelve a frame in the path graph, right-click the node and select **Shelve**.

5. To shelve the relevant frames of a frame in the path graph, right-click the node and select **Shelve relevant (current session)**.

6. To remove all frames from the shelf, click the **Clear shelf** icon.

7. To unshelve a frame, right-click the frame and select **Unshelve**.

Create Tickets

You can use the extension to create tickets from a web page. You can then view the tickets in DevTest Portal.

Follow these steps:

1. Click the **Create ticket** icon.

2. Enter the title, external defect tracking number, severity, and description.

3. To include a screenshot of the application at the time of capture, ensure that the **Capture screenshot** check box is selected.

4. Click **Submit.**

5. (Optional) If you included a screenshot, add a notation to the screenshot.

Create Functional Tests

You can create functional tests from transactions that have been captured.

The default name of a functional test includes the prefix **GtoolsXtn_**. When creating a functional test, be sure to assign a new name to avoid over-writing a previous test. Click the test name and type a new name.

If you configured the SSL settings for two-way SSL, the following properties are added to the **project.config** file:

- ssl-keystore-file
- ssl-keystore-password
- ssl-alias
- ssl-key-password

You can view, edit, and run the functional tests in DevTest Workstation. You can view and run the functional tests in DevTest Portal.

> **Note:** If the method is POST and the request content type is **application/x-www-form-urlencoded**, the category of the transaction frame is changed from REST to Web Http. This change makes it easier to modify the POST parameters in the generated functional test.

Follow these steps:

1. Add one or more transaction frames to the shelf.
2. Click the **Create functional test from shelved frames** icon.
3. (Optional) Change the default name.
4. Specify whether to create a stateful functional test or a data-driven functional test.
5. Click **Create**.
6. Select the project in which to create the functional test.
7. Specify whether to keep or delete the transaction frames in the shelf once the functional test is created.
8. Click **Create**.

> When you create a functional test with the Chrome extension, a config file with a name starting with "GtoolsXtn_" is also created. Use this config file when running the test.

Document Transactions

DevTest Portal includes a feature that lets you document business transactions.

The extension also includes this feature. One difference is that a functional test is automatically created, rather than being optional.

If you open the functional test in DevTest Workstation, the **Documentation** area contains a URL. You can use this URL to open the documented transaction in DevTest Portal.

The generated report can contain a maximum of 50 screenshots. You can increase the maximum number by adding the **docgen.pdf.report.screenshots.max** property to the **phoenix.properties** file. However, increasing the number might cause the JVM to run out of memory and crash the portal services.

Follow these steps:

1. Add one or more transaction frames to the shelf.
2. Click the **Create document transaction and functional test from shelved frames** icon.
3. Enter a name for the document.
4. Click **Create**.
5. Select the project in which to create the functional test.
6. Specify whether to keep or delete the transaction frames in the shelf once the functional test is created.
7. Click **Create**.
   The transaction is documented, and the functional test is created.

8. (Optional) Click **Generate report**.
   The report displays in the browser. You can save or print the report.

CAI Top N and Metrics Reports

CAI Top N and Metrics reports contain information about the data that is captured during business transactions in DevTest Portal. The reports can be searched, viewed, managed, shared, and exported. These reports are divided into the following categories:

- CAI Inventory Reports
- CAI Top N Reports

For more information about viewing and manipulating reports see View and Navigate Reports.

CAI Inventory Reports

The following CAI Inventory reports are available:

- Exception Count by Type
- Exception History
- Frame Type Run List
- Log Count by Severity
- Log Message History
- Ticket History

Exception Count by Type

The Exception Count by Type report displays exception occurrences by frequency in a bar chart.

Exception History

The Exception History report displays the exceptions that occurred over a specified time. The report displays information in a line chart.

Frame Type Run List

The Frame Type Run List report displays information by frame type in a table.

Log Count by Severity

The Log Count by Severity report displays the log count for agents by severity. The severity levels can be information logging, warning logging, and error logging. The report contains a legend that represents the severity type counts. You can click the legend to hide or show the severity count type.

Log Message History

The Log Message History report displays a historical count of log messages. The report contains a legend that represents the log message types. You can click the legend to hide or show the type of log message count.

Ticket History

The Ticket History report displays a historic count of tickets that were generated.

CAI Top N Reports

The following CAI Top N reports are available:

- Top N Response Calls
- Top N Most Invoked Method Functions
- Top N Slowest Calls by Category
- Top N Slowest Calls by Standard Deviation

Top N Response Calls

The Top N Response Times report displays which responses took the longest time to occur during a period. The bar graph displays the longest and average response time for each transaction. The response time is in milliseconds. The method and name of the transaction is included on the graph, for example, GET/lisabank/viewtransaction.

Top N Most Invoked Method Functions

The Top N Most Invoked Method Functions report displays which methods and frames were invoked the most during a period.

Top N Slowest Calls by Category

The Top N Slowest Calls by Category report displays the top slowest calls by category. You can filter the report by the categories that pertain to the calls.

Top N Slowest Calls by Standard Deviation

The Top N Slowest Calls by Standard Deviation report displays the slowest call and the standard deviation.

Application Insight for CICS Walkthrough

This page walks you through using the DevTest CICS Agent with Application Insight.

The following prerequisites apply to this walkthrough:

- One or more DevTest LPAR Agents have been installed and configured.
- For each DevTest LPAR Agent, one or more DevTest CICS Agents have been installed and configured.

This walkthrough includes the following steps:

- Configure the Mainframe Bridge Properties
- Start the DevTest Components
- Enable Transaction Capture for CICS Agents
- Verify Enablement
- Run the CICS Transaction
- View Transaction Data in DevTest Portal
- Create a Virtual Service

Configure the Mainframe Bridge Properties

The mainframe bridge is a component that enables DevTest clients to communicate with DevTest mainframe agents.

Open the **local.properties** file.

Add the mainframe bridge properties to define the connection to the DevTest LPAR Agent.

In the following example, the bridge is configured to run in client mode:

```
lisa.mainframe.bridge.enabled=true
lisa.mainframe.bridge.mode=client
lisa.mainframe.bridge.server.host=172.31.255.255
lisa.mainframe.bridge.server.port=3997
```

Start the DevTest Components

Start the following components:

- Enterprise Dashboard Server
- Broker
- Registry
- Portal

Enable Transaction Capture for CICS Agents

Open the DevTest Portal.

Select **Settings**, **Agents** in the left navigation menu. In the **Agents** window, the CICS agents are displayed.

Screen capture of agents.

For each CICS Agent on which you want to enable Application Insight, do the following steps:

1. Click the agent to select it.
2. On the right, there is a pane to select the capture level:

Screen capture of Adjust Captures for Protocol pane.

3. The capture level may be set differently for Programs (CICS LINK, XCTL, etc.) and Distributed Transaction Processing commands (DTP). Set the capture level to **Counts** if only counts are required; **Counts and Paths** if paths with no payloads (data) are required; or **Full Data** if paths with data are required. **Full Data** is required to inspect the COMMAREAs and Containers or if virtual service generation is required.

4. Back on the selected agent in the agent list, click **Show more** «.

5. Click to start capturing data.
   The color of the icon changes to blue.

Remember to stop capture when you are done.

Verify Enablement

If you want to confirm that the CICS Agents are enabled, log on to each CICS region and run TKOM.

The text **PF ON** indicates that the agent is enabled.

```
                              ITKO AGENT MONITOR                CICS 040200
XPCREQ Started   XPCREQC Started   XEIIN  Started   XEIOUT  Started
GWA:00167534  Length:00F0  EYE:ITKOEXITGWA V081000C040200 Agent ECB:807CF148
0000 C9E3D2D6 C5E7C9E3 C7E6C140 E5F0F8F1 F0F0F0C3 F0F4F0F2 F0F00000 000000F0
0020 030D0000 001F0400 00000000 00000000 807CF148 00000008 00000000 00000000
0040 00000000 00000001 00000000 00000000 00000000 00000000 00000000 00000000
0060 00000000 00000000 00000000 1B7A7000 00000000 00000000 00000000 00000000
0080 00000000 00000000 00000004 00000000 00000000 00000000 1B778000 1B7B1810
00A0 1B779F50 E2F6F7F0 C3C9C3E2 E3E2F140 C3C9C3E2 F1404040 E2F0E6F1 40404040
00C0 E2E5E2C3 D7D3C5E7 40404040 40404040 40404040 40404040 00000000 00000000
00E0 00000000 00000000 00000000 00000000
STATUS 1: EXITS INITALIZED,RUNNING
STATUS 2: AGENT RUNNING,CONNECTED,PF ON
RecQ Seq: 00000008 Head: 00000000 Tail: 00000000 Count: 00000000 Max: 00000001
PLbQ Seq: 00000000 Head: 00000000 Tail: 00000000 Count: 00000000 Max: 00000000
LPAQ               Head: 00000000 Tail: 00000000 Count: 00000000 Max: 00000000
TstQ Seq: 00000000 Head: 00000000 Tail: 00000000 Count: 00000000 Max: 00000000
CTBL:01F4 Token: 00000000 Used:0000 XTBL:01F4 Used:0000  FMHT:01F4 Used:0000
Prog List @ 1B7A7000 U Use: 00000000 Rec Ct: 00000000 Plb Ct: 00000000
Record Count: 00000004  Playback Count: 00000000  Event Count: 00000000
Agent ENQ: TKOA  Task 00000961
Press CLEAR to exit
 PF1     PF2     PF3     PF4       PF5      PF6       PF7       PF8      PF9
Unused Refresh Pgm List Unused  Strt Agnt Stop Agnt Strt Exit Stop Exit   PF
```

Screen capture of output with PF ON text.

Run the CICS Transaction

Run the CICS transaction.

The examples in this walkthrough use the CICS transaction DEM3. This transaction runs on CICS1 and performs DPL LINKs to CICS2.

View Transaction Data in DevTest Portal

Return to DevTest Portal.

Select **Application Insight**, **Analyze Transactions** in the left navigation menu.

The **Analyze Transactions** window displays the generated transactions. If the transactions do not appear, click the refresh icon or adjust the time interval until they do appear.

The following graphic shows the transaction for the example scenario. All of the CICS activity on CICS regions CICS1, CICS2, and CICS3 is stitched together into a single transaction that originated with CICS transaction id DEM3 on CICS region CICSTS1.

	Name	Start Time	Wall Time	CPU Time	Agent
	DEM3-CICSTS1 -Root-DEMOAPP3	07/27/2016 8:43:28 AM 103	1611261 ms	0 ms	S0W1-CICSTS1

Screen capture of transaction for example scenario.

Click the transaction. The transaction detail panel opens. At the top is the path showing the flow of programs.

Screen capture of path graph.

Click a node to see its details. In this example, click the DEMOINQ node (the second node in the path from the left). Selecting the Response tab and selecting codepage IBM037 shows the response COMMAREA in EBCDIC.

Frame | Request | Response | Log Messages | XML | Exception

Raw Values — Encoding: IBM037

```
        00010203 04050607 08090A0B 0C0D0E0F
0000    F0F0F0F0 F0F0F0F1 D5E8C340 40404040    0 0 0 0 0 0 0 1 N Y C
0010    40404040 40404040 4040C299 96A69540                      B r o w n
0020    40404040 40404040 4040F1F6 F0F440C3                      1 6 0 4   C
0030    9996A2A2 40C28595 8440D984 40404040    r o s s   B e n d   R d
0040    40404040 40404040 40404040 40404040
0050    40404040 40404040 4040D793 81959640                      P l a n o
0060    40404040 4040E3E7 F7F5F0F2 F3404040                T X 7 5 0 2 3
0070    40F9F7F2 4BF5F5F5 4BF0F0F0 F15BF16B      9 7 2 . 5 5 5 . 0 0 0 1 $ 1 ,
0080    F0F0F04B F0F04040 4040405B F9F6F24B    0 0 0 . 0 0           $ 9 6 2 .
0090    F1F64040 40404040 40D58194 85408195    1 6               N a m e   a n
00A0    8440C184 849985A2 A2409985 83969984    d   A d d r e s s   r e c o r d
00B0    40998581 84404040 40404040 40404040      r e a d
00C0    40404040 40404040 40404040 40404040
```

Screen capture of Response tab.

Create a Virtual Service

Open the transaction detail dialog for the transaction.

Right-click a node in the path and choose **Shelve Frame**. The node is added to the shelf.

Close the transaction detail dialog and click the shelf icon.

Click the **Create a virtual service from the shelved frames** icon . The **Shelf - Create Virtual Service** dialog appears.

This dialog lets you associate copybooks with the request and response of the virtual service. An existing vrs file may be used or a Copybook Bundle. If choosing a Copybook Bundle, also specify the codepage to be used.

Screen capture of Create Virtual Service dialog.

Click Next. The following dialog appears to select a project and a prefix for the name of the virtual service.

Screen capture of Create in Project dialog.

Click Finish. The following dialog appears to indicate success or failure. Once the virtual service is created, you can manage it like any other virtual service in the portal.

Screen capture of Jobs in Progress dialog.

Best Practices for Application Insight for CICS Agents

Application Insight for CICS lets you virtualize services so that you do not have to test on a production system or necessarily have a lab that is equal to your production environment. CICS agents are not intended for use in live production environments during production up time. Application Insight is not intended to scale to production levels. In an environment for development testing, that level of scaling is not typically necessary.

> **Note:** This page describes the load capacities, limitations, and tuning capabilities for the Application Insight Broker regarding CICS agent volume. This analysis does not necessarily extend to Java Agents.

Capacity

A server with a broker running with a heap allocation of 4GB can accommodate between 16 to 20 active CICS agents. When you exceed this number, “Max broker memory usage” messages are likely to appear, and the broker ceases to function properly. This situation can occur even if the system is idle.

The behavior above describes the edge of capacity. Once the active agent threshold is exceeded, memory errors are no longer defects. There is some tuning that can potentially extend the capability. However, these procedures tune to parameters that are outside of the original design intent, and they are not fully supported.

Broker Scalability

Application Insight is designed to have a single broker per registry. This is an architectural constraint. Do not attempt to have more than one broker on a registry. To scale beyond a single broker, install multiple instances of the DevTest environment.

CICS Agent Granularity

LPAR agents are message and management hubs for multiple CICS agents. if the broker mainframe bridge is in server mode, it is possible to have multiple LPAR agents per broker, .

If the number of CICS agents exceeds the broker threshold, it is possible organize each region onto its own LPAR agent. Then you can disable unused groups to perform recording or playback tasks. This approach works as long as the selected subgroup does not exceed the 16 to 20 agent threshold. This approach allows a larger number of available agents, while only having a number of agents active that allows the broker to run properly.

Memory and Message Tuning

In the event of memory issues related to messaging (agent capacity), the following actions might mitigate the problem.

- **Increase JVM memory.** The problem might be solved by increasing the Broker's JVM Heap size.

  1. Find **Broker.vmoptions** in the **bin** folder of your installation directory.
  2. Edit this file and add the following line:

     ```
     \-Xmx4g
     ```

     Note: The **4g** indicates 4GB. You can choose to set this value higher, but you must ensure that your machine has the necessary capacity. This is a request of the operating system that is constrained by the physical memory available and other applications that are using memory.

- **Decrease the number of active agents**, as described in **CICS Agent Granularity** above.

- **Retune messaging components.**

  - Add one or both of the following configuration statements to the broker **rules.xml** file, located in the installation directory on the computer where the broker is running.

    ```
    <property comment="Polling interval for agent Info" key="lisa.agent.agent.
    update.int" value="5000"/>
    <property comment="Polling interval for agent Stats" key="lisa.agent.stats.
    sampling.interval" value="5000"/>
    ```

    - Add these statements inside the **<broker></broker>** block.
    - These statements can be on one line each.
    - The values are in milliseconds, so that 5000 = 5 seconds. You can modify this value up or down, but the default value is 1000.

    Note: Modifying these values too far can cause timeouts and negatively impact performance in Portal for agent related activities. These settings are outside of the default settings and were not intended as tuning parameters. You might be able to exceed the standard agent capacity by altering these settings, but this practice is not fully supported.

Integrate with CA API Management

DevTest Solutions lets you capture the request and response data for APIs that are managed by CA API Gateway. You can then use Application Insight to create baselines and virtual services for the APIs.

> **Note:** CA API Gateway is a component of CA API Management.

You can link only one installation of DevTest Solutions to one gateway.

Configure and Deploy Policy

DevTest Solutions includes a policy that you must deploy to CA API Gateway. Policies define restrictions for the consumption of a published service that is protected by CA API Gateway.

You use the **APIM Integration** dialog to specify the deployment information. The following graphic shows the **APIM Integration** dialog with all the fields filled out. The regular expression and the mode are optional fields.

Screen capture of APIM Integration dialog.

You can update the policy after it has been deployed. For example, you can disable capture by changing the mode from **Record** to **Do Nothing**.

This procedure requires the following DevTest components to be running:

- Enterprise Dashboard Server
- Registry
- Broker
- Portal

This procedure assumes that the REST Management service has been published for the gateway.

Follow these steps:

1. Open DevTest Portal.
2. Click the down arrow to the right of your user name and select **APIM Integration**.
   The **APIM Integration** dialog appears.
3. In the **URL** field, enter the URL to the gateway. For example, enter **https://gateway.mycompany.com:8443**.
4. In the **Gateway Name** field, enter a unique name for the gateway.
5. In the **User Name** and **Password** fields, enter the credentials for logging in to the gateway.
6. In the **User Name** and **Password** fields that appear under **DevTest Login**, enter the credentials for logging in to DevTest Solutions.
7. (Optional) By default, every API in the gateway is recorded. You can enter a regular expression to filter the APIs.
8. (Optional) In the **Mode** field, select one of the following options:
   - **Record**
     Capture the data.
   - **Do Nothing**
     Let the data pass through without being captured.
9. Click **Deploy**.
   The policy is added to the **Deployed Policy List** tab.
10. To update the regular expression or mode for a deployed policy, make the change in the **Deployed Policy List** tab and click **Commit**.
11. To remove a deployed policy, click the button in the **Delete** column of the **Deployed Policy List** tab.

Capture Request and Response Data

Once the policy is deployed, you can invoke the APIs and capture request and response data. Application Insight converts the captured data to transactions.

In the **Analyze Transactions** window, the agent name has the following format:

```
APIM-<gateway-ip-address>
```

Before you invoke the APIs, ensure that the following DevTest components are running:

- Enterprise Dashboard Server
- Registry
- Broker

Follow these steps:

1. Invoke one or more APIs.

2. Go to the **Analyze Transactions** window in DevTest Portal.
3. View the transactions that Application Insight generated for the captured data.
4. You can now shelve transactions and create baselines and virtual services.

Using CA Application Trace Kit

CA Application Trace Kit (ATK) provides visibility into Java-based applications in which one or more DevTest Java Agents have been embedded.

CA Application Trace Kit also lets you manage the main components of the DevTest Java Agent: the agent itself, the broker, and the consoles.

> **Note:** CA Application Trace Kit is a preview feature in this release.

- Start the Application Trace Kit
- Application Trace Kit Layout
- Live Paths
- Saved Paths
- Configure the Amount of Data Captured
- Configure Agent Properties
- View and Configure Logging
- View and Manage Classes
- Manage Files
- Execute Remote Commands from the Terminal
- CPU Profiling
- Reinvoke a Transaction

Start the Application Trace Kit

Before you start the Application Trace Kit, ensure that the broker is running.

You can start the Application Trace Kit from the command line. The **bin** directory of a DevTest installation contains the executable file. Add the **-u** option to specify the broker. For example:

```
C:\Program Files\CA\DevTest\bin>ATK.exe -u tcp://localhost:2009
```

You can add the **-console** option to display console output.

```
C:\Program Files\CA\DevTest\bin>ATK.exe -u tcp://localhost:2009 -console
```

If you are running the Application Trace Kit on Windows and the Start menu includes an entry for DevTest Solutions, you can start the Application Trace Kit from the Start menu instead of from the command line.

Application Trace Kit Layout

The following graphic shows the layout of the Application Trace Kit.

Screen capture of Application Trace Kit layout.

The left area of the user interface has the following components:

- **Broker URL**
  Lets you specify the URL to the broker.
- **Pathfinder**
  Provides access to the following windows:
  - **Live Paths**
    Displays a snapshot of the live traffic captured by all the agents that are connected to the broker.
  - **Saved Paths**
    Displays the traffic that has been persisted to the DevTest database.
  - **Database**
    Displays the structure of the DevTest database and lets you make queries.
  - **Tickets**
    Displays any tickets generated in Application Insight.
- **Manage Agents**
  Displays the agents that are connected to the broker.
- **Manage Consoles**
  Displays the broker and the consoles that are connected to the broker.

> **Note:** If the **Manage Consoles** pane is not wide enough to display the broker and console names, the names are replaced by white space. Widen the pane until the names appear. Alternately, you can place the mouse pointer over the white space to display the names as tooltips.

The main area of the interface includes a toolbar. The toolbar changes depending on what you select in the left area:

- If you select an item in the **Pathfinder** pane, the toolbar provides access to the associated windows.
- If you select an item in the **Manage Agents** or **Manage Consoles** pane, the toolbar provides access to the associated management tools.

Live Paths

The **Live Paths** tab displays a snapshot of the live traffic captured by all the agents that are connected to the broker.

When you first view the traffic for an application in the **Live Paths** tab, you might find that the volume or granularity is not ideal. You can address these issues by modifying the agent configuration. For example, you can remove protocols from being captured or add protocols that are not currently being captured.

The following graphic shows multiple transactions in the **Live Paths** tab. One transaction is selected.

Screen capture of transactions in Live Paths tab.

The toolbar contains two refresh icons:

- **Show transactions currently in memory**
  Displays the transactions that have been sent from one or more agents and are currently in memory. If the icon is gray, no new transactions are available to display. If the icon is blue, you can click the icon to display the latest transactions.

- **Time out partial transactions, flush them to the database and show transactions currently in memory**
  Stops the agents from trying to stitch any transaction fragments into complete transactions.

Each node in the **Component** column represents a transaction frame.

The horizontal bars in the **Timing** column provide a visual indicator of the duration of each frame. For frames other than the root frame, the horizontal bars also indicate when the frame started and ended in relation to the root frame. A horizontal bar can include a vertical divider, which indicates the proportion of time that is spent in CPU compared to the wall clock time. You can view the time, duration, and CPU information by placing the mouse pointer over a horizontal bar.

To configure the amount of data that appears, click **Filters**. You can set each protocol to one of the following levels:

- **Hide**
  Remove all frames for this protocol.

- **Collapse**
  Merge multiple frames for this protocol at the same level.

- **View**
  Display all frames for this protocol.

To view the agent and thread for a frame, place the mouse pointer in the first column.

When a transaction spans multiple agents, the first column uses different shading for each agent. The following graphic shows an example of this behavior for two agents.

Screen capture of shading in first column.

To view more information about a frame in the **Component** column, select the frame. One or more of the following tabs appear:

- **Details**
  Displays basic information about the frame.

- **Request**
  Displays the payload that was sent. The format varies depending on the protocol.

- **Request (Out)**
  Displays the value of the arguments at the end of method execution.

- **Response**
  Displays the payload that was received. The format varies depending on the protocol.

- **Screenshot**
  Displays the image that is associated with a GUI event.

- **Debug**
  Displays information that can be used for debugging.
  The initial rows contain the frame ID, the ID of the parent frame, the transaction ID, and the session ID. If the frame does not have a session ID, the value **null** appears.

If you enable profiling for an agent and then record some transactions, you can display the full call tree for any frame. Right-click the frame and select **Show call tree**.

You can filter the call tree by absolute duration and relative duration.

Saved Paths

The **Saved Paths** tab displays the transactions that have been persisted to the DevTest database.

The following graphic shows the **Saved Paths** tab.

Screen capture of Saved Paths tab.

To view the full details of a transaction, double-click the transaction in the **Call** column. The window that appears is similar to the **Live Paths** tab.

To run the transaction again, do the following steps:

1. Click the green arrow in the full details window.
   The **Invocation** window appears.

2. (Optional) Change the parameters in the **State**, **Input**, or **Output** tabs.

3. (Optional) Change the selected agent.

4. Click the green arrow.

Configure the Amount of Data Captured

You can configure the amount of data that a Java agent captures for each protocol. The following capture levels are available:

- **Discover**
  Capture only the number of method invocations.

- **View**
  Capture basic method information and timing.

- **Payload**
  Capture all method information, including the request and response.

Follow these steps:

1. Select the agent in the **Manage Agents** pane.
2. Click **Overview**.
3. Expand the **Settings** pane.
4. Change the capture level for one or more protocols.

Configure Agent Properties

You can view and update the properties for a Java agent.

Follow these steps:

1. Select the agent in the **Manage Agents** pane.
2. Click **Overview**.
3. Expand the **Settings** pane.
4. Change the value of one or more properties.
5. To revert to the persisted settings, click **Resync**.
6. To save the changes for the current session only, click **Apply**.
7. To save the changes for the current session and future sessions, click **Save**.

View and Configure Logging

You can view the log messages for a Java agent.

The following example shows a partial set of log messages.

```
[DevTest AGENT:A][INFO][7336][166][ActiveMQ Session Task][04/28 14:04:39 (167)]
Stopping Local CPU Profiling...
[DevTest AGENT:A][INFO][7336][167][ActiveMQ Session Task][04/28 14:05:39 (376)]
Starting Local CPU Profiling...
[DevTest AGENT:A][INFO][7336][168][ActiveMQ Session Task][04/28 14:06:05 (204)]
Setting weights: [com.itko.lisa.remote.transactions.interceptors.JDBCInterceptor:
4]
```

Follow these steps:

1. Select the agent in the **Manage Agents** pane.
2. Click **Log**.

3. To filter out the messages below a certain log level, select the log level from the drop-down list.

View and Manage Classes

You can view and manage the full class hierarchy that is loaded in the Java virtual machine.

The following graphic shows an example of the class hierarchy. The **javax.ejb.EJBMetaData** interface is selected in the left pane.

Screen capture of classes.

The **Metadata** tab contains the following information for the selected class or interface:

- Method signatures
- Superclasses or implemented interfaces
- Subclasses or implementing classes

The **Source** tab lets you decompile the selected class or interface for debugging purposes only. When you click the **Source** tab, you must confirm that you have sufficient rights to view the decompiled code.

The **Live Instances** tab lets you view objects that are currently live in your server.

Follow these steps:

1. Select the agent in the **Manage Agents** pane.
2. Click **Classes**.
3. Select a class or interface in the left pane.

Manage Files

The Application Trace Kit includes a window that displays two sets of files:

- Files on the local computer
- Files on the computer where the selected agent is installed

You can perform file management tasks from this window.

Follow these steps:

1. Select the agent in the **Manage Agents** pane.
2. Click **Explorer**.
3. To move a file from one computer to another, drag and drop the file.
4. To open a file, double-click the file.
5. To create new directories, delete files, and rename files, use the right-click menu.

Execute Remote Commands from the Terminal

The Application Trace Kit lets you display a command prompt or shell for the computer where the selected agent is installed.

You can perform any command that the operating system supports. For example, you can restart a process or check memory usage.

Follow these steps:

1. Select the agent in the **Manage Agents** pane.
2. Click **Terminal**.

CPU Profiling

By default, the DevTest Java Agent captures only a subset of the traffic for an application. If you enable CPU profiling, the agent captures most or all of the traffic.

This feature is useful when you think that the agent is not capturing enough data or the right type of data for your purposes.

CPU profiling can impair the performance of the application. Therefore, we recommend that you enable this feature only on a temporary basis.

The following graphic shows the CPU profiling tab.

Screen capture of CPU profiling tab.

As in the **Live Paths** tab, the first column uses different shading for each agent.

The data that is also captured when CPU profiling is disabled appears in blue.

Follow these steps:

1. Click the CPU profiling icon for one or more agents in the **Manage Agents** pane.
2. Generate traffic and display the corresponding transactions in the **Live Paths** tab.
3. Select a node in the tree.
4. Click the **Show profiling tree** icon .
   The CPU profiling tab appears.
5. To view more information about a row, select the row. For most rows, only a limited amount of information is available.

Reinvoke a Transaction

You can create baselines from transactions in DevTest Portal. The Application Trace Kit lets you reinvoke a transaction, which is similar to creating baselines. The advantage is that the process is quicker than creating baselines.

The following graphic shows the results of reinvoking a transaction. The expected response is the output that was originally recorded. The actual response is the output from reinvoking the transaction. The green square to the left of the **Actual Response** label indicates that the expected response and the actual response are identical.

Screen capture of reinvoking a transaction.

Follow these steps:

1. Select a transaction in the **Live Paths** tab.
2. Click the **Invoke this transaction again** icon .
3. Select the **Output** tab.
4. Click the green arrow.
5. If the square to the left of the **Actual Response** label is red, place the mouse pointer over the square to display information about the differences.

CA Application Trace Kit Database Tools

CA Application Trace Kit (ATK) includes tools for two types of databases.

> **Note:** CA Application Trace Kit is a preview feature in this release.

- Explore the DevTest Database
- Explore Databases Connected to Agents

Explore the DevTest Database

You can perform the following tasks on the DevTest database, which the Java agent uses to persist data:

- View the database schema
- Display the agent DDL statements
- Run SQL queries

Follow these steps:

1. Click **Database**.
2. View the database schema in the left pane.
3. To display the DDL statements, click the database icon in the upper left corner.
4. To run a SQL query, enter the query in the script tab and click **Evaluate**.

Explore Databases Connected to Agents

You can perform the following tasks on a database that is connected to a Java agent-enabled application:

- View the database schema
- Re-evaluate any transaction frame query
- Run arbitrary queries

The following graphic shows the **SQL Explorer** window. A query has been run against a database table.

Screen capture of SQL Explorer.

Follow these steps:

1. Open the **Live Paths** tab.
2. Locate a transaction that has one or more JDBC frames.
3. Right-click a JDBC frame and select **Show SQL Explorer**.
4. View the database schema in the left pane.
5. To run a SQL query, enter the query in the script tab and click **Evaluate**.

Creating Java Agent Extensions

CA Application Trace Kit (ATK) lets you create the following types of extensions for the DevTest Java Agent:

- Application Insight
- Server VSE

- Broker

> **Note:** CA Application Trace Kit is a preview feature in this release.

Create, Build, and Upload the Extension

The window in which you create the extension is not a full-featured IDE, but it does include such features as syntax highlighting.

For detailed information about each type of extension, see Java Agent Extensions.

Follow these steps:

1. Select the agent in the **Manage Agents** pane.
2. Click **Extensions**.
3. Click **Create** and select the extension type.
4. Change the default class name.
5. When you finish modifying the code, click **Build**.
6. Click **Upload**.

Using Extensions to Reduce Noise

You might not be interested in a particular method, class, or package that the agent is capturing. You can use the Application Trace Kit to exclude them from being captured.

Start from the **Live Paths** tab. Right-click a frame and select **Extensions**, **Exclude**. The window for creating extensions appears.

The **block()** method includes an if statement that is based on the frame you selected.

The result of the default if statement is to exclude all frames whose class and method names match exactly. For example:

```
if (clazz.equals("com.itko.examples.ejb3.EJB3UserControlBean") && method.equals
("getUser")) {
     return true;
}
```

You can change the if statement to exclude all frames whose class, method, and signature match exactly. For example:

```
if (clazz.equals("com.itko.examples.ejb3.EJB3UserControlBean") && method.equals
("getUser") && signature.equals("(Ljava/lang/String;)) {
     return true;
}
```

You can change the if statement to exclude all frames from a class. For example:

```
if (clazz.equals("com.itko.examples.ejb3.EJB3UserControlBean")) {
     return true;
}
```

You can change the if statement to exclude all frames from a package. For example:

```
if (clazz.startsWith("com.itko.examples.ejb3.")) {
     return true;
}
```

Live Instances and OQL Queries

These features of CA Application Trace Kit (ATK) require the native agent. In the agent parameters string, include the option **heap=true**.

The general syntax of the Object Query Language (OQL) is:

```
SELECT <result-expression> FROM <class-name> <alias> [<where> <filter-expression>]
```

Note: CA Application Trace Kit is a preview feature in this release.

Live Instances

The **Live Instances** tab in the **Classes** window lets you view objects that are currently live in your server.

The text area at the top contains the OQL query that was used to perform the search.

The panel below the text area contains a list of objects. When you click an object, the **Object Details** subtab provides more information.

The following graphic shows the **Live Instances** tab.

Screen capture of Live Instances tab.

Query the Heap with OQL

You can use OQL to query for object instances or values in the Java heap using arbitrary filters.

The general syntax of the Object Query Language (OQL) is:

```
SELECT <result-expression> FROM <class-name> <alias> [<where> <filter-expression>]
```

This feature useful if you have a class with hundreds or thousands of instances and you want to inspect a specific instance.

The following graphic shows the window where you run the queries.

Screen capture of Remote Script window.

Follow these steps:

1. Select the agent in the **Manage Agents** pane.
2. Click **Scripting**.
3. Enter the query in the upper pane.
4. Click **Evaluate**.

Using Virtual Service Catalog

Virtual Service Catalog provides a central repository for viewing and sharing virtual services. From a single dashboard, you can quickly locate the available services that are deployed to a VSE and accessible in any Enterprise Dashboard that is connected to Virtual Service Catalog.

This section contains the following pages that describe the functionality of Virtual Service Catalog:

- View Virtual Services
- Configure Virtual Service Catalog

For more information about installing Virtual Service Catalog, see Installing Virtual Service Catalog .

Start Virtual Service Catalog

Follow these steps:

1. Start Virtual Service Catalog by doing one of the following:
   - Double-click the **Virtual Service Catalog** desktop icon.

- Launch the **vscatalog** executable from the ***Install Directory*/bin** folder.

2. Start the Virtual Service Catalog UI by doing one of the following:

   - Double-click the **Virtual Service Catalog UI** desktop icon.

   - Enter **https://localhost:51110/** in a browser.
     If Virtual Service Catalog is not installed locally, substitute the host name for *localhost* in your browser.

     > **Note:** The browser will display a warning that the connection is not secure or that the certificate for the website is invalid. This connection is flagged because it is using a self-signed certificate. Proceed to the site.

   The Virtual Service Catalog login page opens.

3. Enter your user name and password and click **Login**.
   The Home tab opens.

4. From the Home tab:

   - Click **View All Virtual Services** to open the Virtual Services tab and see a list of all available virtual services.
     For more information about viewing virtual services, see View Virtual Services.

   - Click **Manage Enterprise Dashboard Connections** to open the Connection tab and configure the Enterprise Dashboard(s) that you want to access.
     For more information, see Configure Virtual Service Catalog.

     > **Note:** Access to the Connection tab requires administrator privileges.

> **More Information:**
>
> - View Virtual Services
> - Configure Virtual Service Catalog

View Virtual Services

The Virtual Services tab lists all of the available services that are deployed to a VSE and accessible in an Enterprise Dashboard that is connected to Virtual Service Catalog. The ability to view and search your existing assets allows teams across your organization to leverage existing virtual services. Reuse of assets reduces the time a team spends creating and maintaining new virtual services.

Additionally, you can drill-down into specific arguments, matching options, and responses within a service to determine whether an existing service meets a given set of needs and/or specific data scenarios.

> **Note:** A custom DPH is not listed as part of the virtual service in Virtual Service Catalog.

To view a list of virtual services:

1. Click the **Home** tab.
2. Click **View All Virtual Services** to see a list of all virtual services that are deployed to a VSE.
3. The list of virtual services opens in a new tab.
   - Click **Show/Hide Columns** to select the columns that you want to display in the table.
     - Select the check box for each column that you want to display.
     - Clear the check box for each column that you want to hide.
   - Click any column header to sort the list by that column.
     - The header of the sorting column is highlighted and displays an up or down arrow to indicate an ascending or descending sort.
     - Click the column header again to toggle between an ascending and descending sort.
     - You can only sort the list of services by one column at a time.
   - Enter a keyword in the Search field of any column header and press **Enter** to search that column for a specific keyword.

     - If you enter search criteria in multiple column headers, the list of services is limited to services that match all entered criteria. For example, if you enter search criteria in the Name column and the Transport Protocol column, the services that display match the criteria in the Name column AND the Transport Protocol column.
     - To remove search criteria, you can manually remove the criteria or click the **X** in the search field.
   - Click a link in the **Name** column to view additional details for each operation in the virtual service.
   - Hover over the **Operations/Path** column to see the complete list of operations. Long lists of operations are truncated to display in the table.

- Click the link in this column to view additional details for each operation in the virtual service.

- Click **Refresh** to manually refresh the list of virtual services. There is no automatic refresh in Virtual Service Catalog.

View Virtual Service Details

From the Virtual Services tab, click a link in either the **Name** or **Operations/Path** columns to see additional details for a specific virtual service.
The tab for the selected virtual service opens.

Screen capture of the virtual services detail tab

The individual operations for the virtual service are listed in the left pane.

- Click an operation in the left pane to view the request and response data for that operation in the Specifics section of the page.

- If there are multiple request and response data sets for an operation, you can use the arrows at the top of the Specifics section to move through the data.

- The Response Data section is divided into two tabs, one for the response content and one for the response headers.

- Click **Show Raw Request** to view the actual syntax of the request body. This button only displays if there is content in the request body.

Screen capture of the raw request

Configure Virtual Service Catalog

This page includes instructions for the following configuration tasks for Virtual Service Catalog:

- Configure the Identity and Access Manager Instance
- Configure Connection Settings
- Configure Custom Ports
- Configure Logging Output to a File

Note: Before you can use Virtual Service Catalog, you must create users with the appropriate permissions in Identity and Access Manager. Identity and Access Manager must installed as part of the DevTest installation prior to installing Virtual Service Catalog. Identity and Access Manager must also be running before users can access Virtual Service Catalog. For more information, see Installing and Configuring DevTest Server and Identity and Access Manager.

Note: Virtual Service Catalog only supports Enterprise Dashboards and Registries that are on DevTest 10.3 or later.

Configure the Identity and Access Manager Instance

Identity and Access Manager lets you manage user IDs and permissions for accessing Virtual Service Catalog. During the installation process, you define the instance of Identity and Access Manager to use for accessing Virtual Service Catalog. You can change the instance of IAM that is used through the application.yml file.

To update the instance of Identity and Access Manager:

1. Navigate to the Virtual Service Catalog installation directory.
2. Open the **application.yml** file.
3. Add the following property to define the host and URL of the Identity and Access Manager instance that you want to use.

```
svcatalog:
  auth:
    host:
      url: https://{{host}}:{{port}}/auth
```

4. Save your changes and close the file.

> **Note:** After making changes to application.yml, you must restart Virtual Service Catalog for those changes to take effect.

Configure Connection Settings

The Connection tab lets you configure each Enterprise Dashboard that you want to access from the catalog. When you add configuration information for an Enterprise Dashboard, you can then view all of the virtual services that are deployed to a VSE for the registries that connect to that dashboard.

> **Note:** Configuring Virtual Service Catalog connections requires administrator privileges. You must create a user with administrator privileges for Virtual Service Catalog (Virtual Service Catalog Administrator role) in Identity and Access Manager before you configure your connection settings.

To configure connection settings:

1. Click the **Home** tab in the Virtual Service Catalog UI.
2. Click **Manage Enterprise Dashboard Connections** to configure the Enterprise Dashboard(s) that are connected to Virtual Service Catalog.

   The Connection tab opens. This tab shows each Enterprise Dashboard that is configured in Virtual Service Catalog and the current connection status.
3. To add another Enterprise Dashboard, enter the URL for the dashboard and click **Connect**.

- To remove an Enterprise Dashboard, click **Delete** in the **Actions** column.
- To refresh the list of connected Enterprise Dashboards, click **Refresh** .

4. To view the status of each registry that is connected to a specific Enterprise Dashboard, click **Expand** next to the dashboard URL.
5. To view the virtual services for the connected Enterprise Dashboards:
   a. Click the **Home** tab.
   b. Click **View All Virtual Services**.
   The Virtual Services tab opens.

> **Note:** If the Virtual Services tab is open when you add or remove an Enterprise Dashboard, you must manually refresh the tab to reflect the services for the updated connections.

Configure Custom Ports

> **Important!** If your components are using non-default (custom) port numbers, you must specify the port in the **application.yml** file in your Virtual Service Catalog installation directory. After making changes to application.yml, you must restart Virtual Service Catalog for those changes to take effect.

If you include multiple properties in the application.yml file, you must add a separator (---) between the properties. For example:

```
svcatalog:
  auth:
    host:
      url: https://{{host}}:{{port}}/auth
---
lisa:
  webserver:
    port: YOUR_PORT_NUMBER_IN_DEVTEST
```

Custom Port for Embedded Web Server

If you are not using the default port for your embedded web server (1505), you must specify the port in the application.yml file in your Virtual Service Catalog installation directory.

```
lisa:
  webserver:
    port: PORT_NUMBER
```

Custom Port for Virtual Service Catalog

If you want to use a port other than the default port (51110) for Virtual Service Catalog, you must specify the port in the application.yml file in your Virtual Service Catalog installation directory.

```
server:
  port: PORT_NUMBER
```

Configure Logging Output to a File

To configure Virtual Service Catalog to write logging output to a file:

1. Create the following .yml file, where **mylogfile.log** is the path and name of the log file that you want to use.

   ```
   logging:
     level:
       com:
         ca:
           ad:
             sv: DEBUG
     file: mylogfile.log
   ```

2. Save this file as **application.yml**.

3. Place this file in the installation directory for Virtual Service Catalog.

More Information:

- Using Virtual Service Catalog
- Installing Virtual Service Catalog
- Identity and Access Manager

Using the APIs

DevTest Solutions provides the following public APIs:

- Virtual Service Invoke API
  This API lets you create and manage virtual services. You can create virtual services and add transactions by recording, R/R pairs, or a specification document. You can also export a virtual service as a MAR file.

- Test Invoke API
  This API lets you create and manage your test cases and suites. You can execute test cases and test suites as well as retrieve run data from your tests and test suites.

- REST Invoke API
  This API lets you manage and monitor coordinator servers, simulator servers, and VSEs. You can also create a virtual service from R/R pairs.

- LISA Invoke API
  This is the original DevTest API that lets you run test cases, test suites, and model archives (MARs).

LISA Invoke API

LISA Invoke is a REST-like web application that lets you perform the following tasks with a URL:

- Run test cases
- Run suites
- Run model archives (MARs)

You can perform these tasks synchronously or asynchronously.

The response consists of an XML document.

The **lisa.properties** file includes the following configuration properties for LISA Invoke:

```
lisa.portal.invoke.base.url=/lisa-invoke
lisa.portal.invoke.report.url=/reports
lisa.portal.invoke.server.report.directory={{lisa.tmpdir}}{{lisa.portal.invoke.
report.url}}
lisa.portal.invoke.test.root={{LISA_HOME}}
```

You can override these properties in the **local.properties** file.

To view the LISA Invoke home page, go to http://*hostname*:1505/lisa-invoke/, where *hostname* is the computer where the registry is running.

- Run Test Cases with LISA Invoke
- Run Test Suites with LISA Invoke
- Run Model Archives with LISA Invoke
- Invoke the Callback Service with LISA Invoke
- LISA Invoke Responses

Run Test Cases with LISA Invoke

The syntax for running test cases with LISA Invoke is:

```
/lisa-invoke/runTest?testCasePath=testCasePath&stagingDocPath=stagingDocPath&
[configPath=configPath]&[async=true]&[coordName=csName]&[-D<CustomTag>=x]
```

The parameters are:

- **testCasePath**
  The path to the test case to invoke.
- **stagingDocPath**
  The path to the staging document. If not provided, a default staging document is created.
- **configPath**
  The path to the configuration. If not provided, the project.config file for the project is used.
- **async**
  If true, then the response includes a callback key. If not provided, the parameter is set to false.

- **coordName**
  The path to the coordinator. If not provided, the default coordinator name is used.

- **-D<*CustomTag*>**
  Defines any custom tag to any value. For example:

  ```
  &[-DVersion=1.5.3]
  ```

> To run the following examples, you must uncomment the **lisa.portal.invoke.test.root** property in **lisa.properties** or set it in **local.properties**. The value should be the base path needed to find the project or projects where the tests or suites are located. For the example, it wants the **Lisa_Home** directory (where DevTest is installed) as it goes to the examples directory for the tests and suites in that project.

Example: Synchronous Invocation

The following URL performs a synchronous invocation of the **AccountControlMDB** test case in the **examples** project.

```
http://localhost:1505/lisa-invoke/runTest?testCasePath=examples/Tests
/AccountControlMDB.tst&stagingDocPath=examples/StagingDocs/1user1cycle0think.stg
```

The following XML response indicates that the test case passed.

```
<?xml version="1.0" encoding="UTF-8"?>
<invokeResult>
  <method name="RunTest">
    <params>
      <param name="stagingDocPath" value="examples/StagingDocs/1user1cycle0think.
stg" />
      <param name="coordName" value="Coordinator" />
      <param name="configPath" value="" />
      <param name="testCasePath" value="examples/Tests/AccountControlMDB.tst" />
      <param name="callbackKey" value="64343533653737312D343765312D3439" />
    </params>
  </method>
  <status>OK</status>
  <result>
    <status>ENDED</status>
    <reportUrl><![CDATA[http://localhost:1505/index.html?lisaPortal=reporting
/printPreview_functional.html#Idstr=61653261643936342D613636392D3435&curtstr=T]]><
/reportUrl>
    <runId>61653261643936342D613636392D3435</runId>
    <pass count="1" />
    <fail count="0" />
    <warning count="0" />
    <error count="0" />
    <message>AccountControlMDB,Run1User1Cycle0Think</message>
  </result>
</invokeResult>
```

Example: Asynchronous Invocation

The following URL performs an asynchronous invocation of the **AccountControlMDB** test case in the **examples** project.

```
http://localhost:1505/lisa-invoke/runTest?testCasePath=examples/Tests
/AccountControlMDB.tst&stagingDocPath=examples/StagingDocs/1user1cycle0think.
stg&async=true
```

The following XML response shows the callback key in the result element.

```
<?xml version="1.0" encoding="UTF-8"?>
<invokeResult>
  <method name="RunTest">
    <params>
      <param name="stagingDocPath" value="examples/StagingDocs/1user1cycle0think.
stg" />
      <param name="coordName" value="" />
      <param name="configPath" value="" />
      <param name="testCasePath" value="examples/Tests/AccountControlMDB.tst" />
      <param name="callbackKey" value="61663038653562382D663566372D3432" />
      <param name="async" value="true" />
    </params>
  </method>
  <status>OK</status>
  <result>
    <callbackKey>61663038653562382D663566372D3432</callbackKey>
    <message>The LISA test 'examples/Tests/AccountControlMDB.
tst' was launched asynchronously at Mon Mar 26 16:05:39 PDT 2012.</message>
  </result>
</invokeResult>
```

Run Test Suites with LISA Invoke

The syntax for running test suites with LISA Invoke is:

`/lisa-invoke/runSuite?suitePath=suitePath&[configPath=configPath]&[async=true]&[-D<CustomTag>=x]`

The parameters are:

- **suitePath**
  The path to the suite that you want to invoke.
- **configPath**
  The path to the configuration.
- **async**
  If true, the response includes a callback key. If not provided, the parameter is set to false.
- **-D<*CustomTag*>**
  Defines any custom tag to any value. For example:

  `&[-DVersion=1.5.3]`

Run Model Archives with LISA Invoke

The syntax for running model archives (MARs) with LISA Invoke is:

`/lisa-invoke/runMar?marOrMariPath=[marOrMariPath]&[async=true]&[-D<CustomTag>=x]`

The parameters are:

- **marOrMariPath**
  The path to the MAR or MAR info file to invoke.
- **async**
  If true, then the response includes a callback key. If not provided, the parameter is set to false.
- **-D<*CustomTag*>**
  Defines any custom tag to any value. For example:

  `&[-DVersion=1.5.3]`

Invoke the Callback Service with LISA Invoke

To use the callback service in LISA Invoke, you must have performed an asynchronous invocation of a test case or suite. The XML response contains the callback key.

The syntax for invoking the callback service is:

```
/lisa-invoke/callback?testOrSuitePath=[testOrSuitePath]&callbackKey=[callbackKey]
&command=[status|kill|stop]
```

The parameters are:

- **testOrSuitePath**
  The path to the test case or suite.
- **callbackKey**
  The callback key that was included in the response to the asynchronous invocation.
- **command**
  The action that you want to perform: status, kill, or stop.

LISA Invoke Responses

This section describes the elements that can appear in the XML document that LISA Invoke returns.

The **method** element indicates what type of run was performed.

The **status** element contains the status of the run: OK or ERROR.

The **result** element contains one or more of the following child elements:

- **status**
  The status of a test case: RUNNING or ENDED.
- **reportURL**
  The URL to the report in the Reporting Console.
- **runId**
  The unique identifier of the run.
- **pass**
  The number of tests that passed.
- **fail**
  The number of tests that failed.
- **warning**
  The number of tests that had warnings.
- **error**
  The number of tests that had errors.
- **message**
  Information that is specific to the type of run.
- **tc**
  The name of a test case included in a suite.

- **callbackKey**
  A string that you can use to perform additional actions on the test case or suite.

REST Invoke API

The REST Invoke API lets you use the REST architectural style to perform DevTest tasks.

The APIs are divided into the following categories:

- Coordinator servers
- Simulator servers
- VSE servers

The full API documentation is available here. The API documentation provides details on how to interact with the various RESTful services.

> **Note:** You must use Google Chrome or Mozilla Firefox to access the API documentation.

- Explore the REST Invoke API from the Web Interface
- Change the Execution Mode Using the REST Invoke API
- Create a Virtual Service from R/R Pairs Using the REST Invoke API

Explore the REST Invoke API from the Web Interface

DevTest Solutions includes a simple web interface that you can use to explore the REST Invoke API.

> **Note:** You must use Google Chrome, Internet Explorer, or Mozilla Firefox to access the web interface.

By default, the web interface uses **localhost** in the URL. To access the web interface from other than **localhost**, set the following property in the **local.properties** file:

```
lisa.invoke2.swagger.basepath=http://<public_hostname_or_ip_address>:1505/api/Dcm
```

Follow these steps:

1. Ensure that the registry is running.
2. Enter http://localhost:1505/api/swagger/ in a supported browser. If the registry is on a remote computer, replace **localhost** with the name or IP address of the computer.
   The web interface appears.
3. Click **Show/Hide** for a category.

4. Click one of the API paths.

5. Use the **Response Content Type** field to specify whether the response is in XML or JSON format.

6. If the **Parameters** section appears, enter the required value for each parameter.

7. Click **Try it out!**.
   The request URL and the response are displayed.

Change the Execution Mode Using the REST Invoke API

The REST Invoke API lets you change the execution mode of a virtual service.

The following instructions assume that you are using the Postman REST client.

First, use the **GET /VSEs/{serviceName}/{virtualServiceName}** API to retrieve information about the virtual service that you want to change. This API does not require a body.

In the following example, the service name is **VSE** and the virtual service name is **proxy**.

```
GET http://localhost:1505/api/Dcm/VSEs/VSE/proxy
```

The response includes the current value of the execution mode.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<VirtualService
    xmlns="http://www.ca.com/lisa/invoke/v2.0" name="proxy"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http
://www.ca.com/lisa/invoke/v2.0 VirtualService.xsd" href="http://localhost:1505/api
/Dcm/VSEs/VSE/magical" type="application/vnd.ca.lisaInvoke.virtualService+xml">
    ...
    <ExecutionMode>Most Efficient</ExecutionMode>
    ...
</VirtualService>
```

Change the HTTP method to **PUT**.

Ensure that the request has the **Content-Type** header with the following value:

```
application/vnd.ca.lisaInvoke.virtualService+xml
```

For the body of the request, use the XML headers from the preceding response. Add the execution mode element with the new value.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<VirtualService
    xmlns="http://www.ca.com/lisa/invoke/v2.0" name="magical"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http
://www.ca.com/lisa/invoke/v2.0 VirtualService.xsd" href="http://localhost:1505/api
/Dcm/VSEs/VSE/magical" type="application/vnd.ca.lisaInvoke.virtualService+xml">
    <ExecutionMode>Transaction Tracking</ExecutionMode>
</VirtualService>
```

When you submit the request, the response includes the new value of the execution mode.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<VirtualService
    xmlns="http://www.ca.com/lisa/invoke/v2.0" name="proxy"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http
://www.ca.com/lisa/invoke/v2.0 VirtualService.xsd" href="http://localhost:1505/api
/Dcm/VSEs/VSE/magical" type="application/vnd.ca.lisaInvoke.virtualService+xml">
    ...
    <ExecutionMode>Transaction Tracking</ExecutionMode>
    ...
</VirtualService>
```

Create a Virtual Service from R/R Pairs Using the REST Invoke API

The REST Invoke API lets you create a virtual service that is based on an input ZIP file of request /response pairs.

Here is the endpoint for the **createService** call:

```
POST http://[host]:[port]/api/Dcm/VSEs/[serviceName]/actions/createService
```

Refer to the documentation on the createService call to learn about the path and form parameters that are expected.

Request/Response Pairs

The ZIP file that is used to create the virtual service must contain one or more request/response pairs (also known as R/R pairs). These R/R pairs can be located either located in the root of the ZIP file or in a subdirectory.

Files for request/response pairs must be named with a unique identifier, followed by a "-req" on the request side and a "-rsp" on the response side, with an **.xml** or **.txt** extension. For example, **addUserObject-req.xml** and **addUserObject-rsp.xml**.

Note: A request can have multiple responses. The following set of files would create one request with three responses.

- addUserObject-req.xml
- addUserObject-rsp1.xml
- addUserObject-rsp2.xml
- addUserObject-rsp3.xml

Configuring How the Virtual Service Is Created and Deployed

When creating a virtual service from request/response pairs, the following assumptions are made:

- The name for the virtual service is a generated name with the "AutoGenService-" prefix, followed by a GUID.
- The virtual service is created using the HTTP protocol handler to import the request/response pairs
- Each request/response pair is treated as text.
- The virtual service listens on a randomly available port that the VSE server assigns
- The base path of the service is set to "/"

Using a serviceProperties.xml File

In order to configure how the virtual service is created from the request/response pairs, a **serviceProperties.xml** file can be used to define the name of the virtual service, and configure the protocol handler that will be importing the request/response pairs. The **serviceProperties.xml** file must be located in the root of the ZIP file.

Here is an example of a **serviceProperties.xml** file that will create an HTTP virtual service named "TestServiceFromRRPairs" that listens on port 10999 with its base path set to "/".

```
<?xml version="1.0" ?>
<recording>
    <name>TestServiceFromRRPairs</name>
    <binary>false</binary>
    <group></group>
    <protocol>
        <class>com.itko.lisa.vse.stateful.protocol.http.HttpProtocolHandler</class>
        <config>
            <listenPort>10999</listenPort>
            <ssl toClient="false"></ssl>
            <keystoreFile></keystoreFile>
            <keystorePW></keystorePW>
            <basePath>/</basePath>
            <needSeparateBind>false</needSeparateBind>
        </config>
    </protocol>
</recording>
```

Structure of serviceProperties.xml

- **recording**
  The root element of the **serviceProperties.xml** file

- **name**
  The name of the virtual service to create. Defaults to a generated name with the "AutoGenService-" prefix, followed by a GUID. If there is already a virtual service with the same name, the newly created service replaces it.

- **binary**
  Whether the R/R pair import should treat the transactions as containing binary data.
  Default: false

  - **group**
    The group tag that the created service is labeled with.
    Default: no group tag

  - **protocol**
    Defines what protocol is used

    - **class**
      The fully qualified class to use as the transport protocol handler when importing request/response pairs.
      Default: com.itko.lisa.vse.stateful.protocol.http.HttpProtocolHandler

    - **config**
      Defines how the transport protocol handler should be configured.

      - **listenPort**
        The port that the virtual service listens on.
        Default: a randomly available port that the VSEserver assigns

- **ssl**
  Whether or not the virtual service should listen using SSL. To use SSL, set the **toClient** attribute on this element to **true**.

- **keystoreFile**
  The keystore file the virtual service uses to provide its certificate when SSL is enabled
  If SSL is not enabled, this value is ignored.
  If SSL is enabled and this element is not specified, the VSE uses the **ssl.server.cert.path** property to locate the keystore file to use.

- **keystorePW**
  The password of the keystore file the virtual service uses when SSL is enabled
  If SSL is not enabled, this value is ignored
  If SSL is enabled and this element is not specified, the VSEuses the **ssl.server.cert.pass** property to locate the password to use with keystore file.

- **basePath**
  The base path to deploy the created service to.
  Default: /

- **needSeparateBind**
  Whether not a separate binding phase is needed.
  Default: false

Sample serviceProperties.xml File for Creating HTTP Virtual Service (Without SSL)

The following is an example of a **serviceProperties.xml** file that creates an HTTP virtual service named "TestServiceFromRRPairs" that listens on port 10999 with its base path set to "/". Notice how other elements have been omitted in favor of using their defaults.

```
<?xml version="1.0" ?>
<recording>
    <name>TestServiceFromRRPairs</name>
    <protocol>
        <config>
            <listenPort>10999</listenPort>
            <basePath>/</basePath>
        </config>
    </protocol>
</recording>
```

Virtual Recording Session (VRS) File

In addition to the serviceProperties.xml configuration, a Virtual Recording Session (VRS) file can be included to configure the protocol handler to use when importing request/response pairs. This file must have a .vrs extension. If you create a zip file that contains a VRS file and other information, such as RR pairs, the VRS file must be in the root folder of the zip file.

If the **serviceProperties.xml** file contains a *<protocol>* element, it is used and the VRS file is ignored. If it does not contain a *<protocol>* element, then the VRS file is used to configure the protocol handler.

Test Invoke API

This document describes the functionality to create and manage tests and test suites using the **Test Invoke** RESTful API.

- Overview

- Execute a Test
- Retrieve an Indicated Test Run by Providing "Run Unique Id"
- Delete a Test Run by Providing "Run Unique Id"
- Retrieve all Test Runs
- Execute a Suite
- Retrieve an Indicated Suite Run by Providing "Run Unique Id"
- Delete a Run Suite by providing "Run Unique Id"
- Retrieve all Suite Runs
- Retrieve Test Reports
- Retrieve a Test Report for a Given "Test Run Unique Id"
- Retrieve all the Cycles for a Test Report for a Given "Test Run Unique Id"
- Retrieve a Cycle for a Test Report for a Given "Test Run Unique Id" and "Cycle Unique Id"
- Retrieve Test Reports
- Retrieve Suite Reports
- Retrieve a Suite Report for a Given "Suite Run Unique Id"
- Retrieve all Tests for a Suite Report Given "Suite Run Unique Id"
- Retrieve a Test Given "Test Run Unique Id" and for Suite Report Given "Suite Run Unique Id"
- Retrieve all Cycles for a Test Given "Test Run Unique Id" and for a Suite Report Given "Suite Run Unique Id"
- Retrieve a Given Cycle for "Cycle Unique Id" , for a Test Given "Test Run Unique Id" , and for a Suite Report Given "Suite Run Unique Id"

Overview

Requirements

- The DevTest registry must be running.
- API Authentication must use the DevTest user credentials.

Endpoint

The root URL for this API is:

`{{LISA_TEST_INVOKE_HOST}}:{{LISA_TEST_INVOKE_PORT}}/lisa-test-invoke/api/v1/`

Authentication Credentials

The API requires a "Basic" HTTP authentication scheme. The "Basic" HTTP authentication scheme is defined in RFC 7617, which transmits credentials as user ID/password pairs, encoded using base64. The ID/password pairs should identify a DevTest user.

Swagger Documentation

You can access the Swagger documentation at http://{{REGISTRY_HOST}}:1505/lisa-test-invoke/api/v1/swagger-ui.

Execute a Test

POST http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api/v1/tests/run
The curl command is:

```
curl -X POST -H "authorization: Basic {Encrypted Authorization}="
"{{LISA_TEST_INVOKE_HOST}}:{{LISA_TEST_INVOKE_PORT}}/lisa-test-invoke/api/v1/tests
/run " -H "content-type: multipart/form-data" -F file=@{Mar file}
```

Payload: MAR file with a test. Use 'Multipart Form Data' and the 'key' argument name as 'file'.

To create a MAR file for a test:

1. Create a MAR info file.
2. Build the MAR.

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-test-invoke/api/v1/tests/run
/MARFile"
        }
    },
    "id": "MARFile",
    "name": "test-data-driven-http-rest (Run1User1Cycle)",
    "runType": "TEST",
    "runBy": "admin",
    "testStatus": "INITIATED",
    "manuallyTerminated": false
}
```

Retrieve an Indicated Test Run by Providing "Run Unique Id"

Notice the "Run Unique Id" is "id" on the previous response .

If you automate this call, we recommend that you check to see that the test status is not "Running."

GET: http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api /v1/tests/run/RunUniqueID

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-test-invoke/api/v1/tests/run
/RunUniqueID"
        }
    },
    "id": "RunUniqueID",
    "name": "test-data-driven-http-rest (Run1User1Cycle)",
    "runType": "TEST",
```

```
    "runBy": "admin",
    "testStatus": "PASSED",
    "manuallyTerminated": false
}
```

Delete a Test Run by Providing "Run Unique Id"

Notice the "Run Unique Id" is "id" on the previous response .

DELETE: http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke /api/v1/tests/run/ RunUniqueID ?force=true

The query parameter: "?force=true" is optional; it will try to terminate the test immediately /forceful. Be careful about using this option because it can make the coordinator server unstable.

Response: Status Code = 204

Retrieve all Test Runs

GET http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api /v1/tests/run?startTime=2017-12-20T00:00:00.000-05:00&endTime=2017-12-20T16:33:00.000-05:00

The startTime and endTime values are optional. If present, they must comply to ISO 8601 format; for example, *2014-10-22T16:33:00.000-05:00.*
If not present, all tests that have been executed are returned.

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-test-invoke/api/v1/tests/run"
        }
    },
    "count": 2,
    "_embedded": {
        "runs": [
            {
                "_links": {
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/tests/run/RunUniqueID"
                    }
                },
                "id": "RunUniqueID",
                "name": "test-data-driven-http-rest (Run1User1Cycle)",
                "runType": "TEST",
                "runBy": "admin",
                "testStatus": "PASSED",
                "manuallyTerminated": false
            },
            {
                "_links": {
                    "self": {
```

```
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/tests/run/RunUniqueID"
                    }
                },
                "id": "RunUniqueID",
                "name": "test-data-driven-http-rest (Run1User1Cycle)",
                "runType": "TEST",
                "runBy": "admin",
                "testStatus": "PASSED",
                "manuallyTerminated": false
            }
        ]
    }
}
```


Execute a Suite

POST http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api /v1/suites/run

The curl command is:

```
curl -X POST -H "authorization: Basic {Encrypted Authorization}="
"{{LISA_TEST_INVOKE_HOST}}:{{LISA_TEST_INVOKE_PORT}}/lisa-test-invoke/api/v1/suites
/run " -H "content-type: multipart/form-data" -F file=@{Mar file}
```

Payload: MAR file with a suite. Use 'Multipart Form Data' and the 'key' argument name as 'file'.

> To create a MAR file for a suite:
>
> 1. Create a MAR info file.
> 2. Build the MAR.

Response:


```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-test-invoke/api/v1/suites/run
/MARFile"
        }
    },
    "id": "MARFile",
    "name": "DataDrivenMultiCycle",
    "runType": "SUITE",
    "runBy": "Unknown",
    "testStatus": "NOT_INITIATED",
    "manuallyTerminated": false
}
```

Retrieve an Indicated Suite Run by Providing "Run Unique Id"

Notice the "Run Unique Id" is "id" on the previous response .

If you automate this call, we recommend that you check to see that the test suite status is not "Running."

GET: http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api /v1/suites/run/MARFile

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-test-invoke/api/v1/suites/run
/MARFile"
        }
    },
    "id": "MARFile",
    "name": "DataDrivenMultiCycle",
    "runType": "SUITE",
    "runBy": "Unknown",
    "testStatus": "PASSED",
    "manuallyTerminated": false
}
```

Delete a Run Suite by providing "Run Unique Id"

Notice the "Run Unique Id" is "id" on the previous response .

DELETE: http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke /api/v1/suites/run/RunUniqueID ?force=true

The query parameter "?force=true" is optional; it will try to terminate the suite immediately /forceful. Be careful about using this option because it can make the coordinator server unstable.

Response: Status Code = 204

Retrieve all Suite Runs

GET http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api /v1/suites/run ?startTime=2017-12-20T00:00:00.000-05:00&endTime=2017-12-20T16:33: 00.000-05:00

The startTime and endTime values are optional. If present, they must comply to ISO 8601 format; for example, *2014-10-22T16:33:00.000-05:00.* If not present, all tests that have been executed are returned.

Response:

```
{
```

```
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-test-invoke/api/v1/suites/run"
        }
    },
    "count": 2,
    "_embedded": {
        "runs": [
            {
                "_links": {
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/run/RunUniqueID"
                    }
                },
                "id": "RunUniqueID",
                "name": "DataDrivenMultiCycle",
                "runType": "SUITE",
                "runBy": "Unknown",
                "testStatus": "ABORTED",
                "manuallyTerminated": false
            },
            {
                "_links": {
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/run/RunUniqueID"
                    }
                },
                "id": "RunUniqueID",
                "name": "DataDrivenMultiCycle",
                "runType": "SUITE",
                "runBy": "Unknown",
                "testStatus": "PASSED",
                "manuallyTerminated": false
            }
        ]
    }
}
```


Retrieve Test Reports

GET http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api /v1/tests/reports ?startTime=2017-12-20T00:00:00.000-05:00&endTime=2017-12-20T16:33: 00.000-05:00

The startTime and endTime values are optional. If present, they must comply to ISO 8601 format; for example, *2014-10-22T16:33:00.000-05:00.*

Response:


```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-test-invoke/api/v1/tests/reports"
        }
```

```
    },
    "count": 2,
    "_embedded": {
        "Tests": [
            {
                "_links": {
                    "next": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/tests/reports/RunUniqueID/cycles"
                    },
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/tests/reports/RunUniqueID"
                    }
                },
                "testRunUniqueId": "RunUniqueID",
                "testRunName": "Run1User1Cycle",
                "testcaseId": "TestCaseID",
                "testcaseName": "test-data-driven-http-rest",
                "testCaseDoc": "Put documentation of the Test Case here.",
                "startTime": "2017-12-19T12:26:03-0500",
                "endTime": "2017-12-19T12:26:10-0500",
                "elapsedTimeInMillSec": 6535,
                "user": "admin",
                "endedState": "PASSED",
                "done": true,
                "manuallyTerminated": false,
                "passCount": 1,
                "failCount": 0,
                "warningCount": 0,
                "abortCount": 0,
                "passPercent": 100,
                "failPercent": 0,
                "abortPercent": 0,
                "modelName": "test-data-driven-http-rest",
                "loadPattern": "Run N Times",
                "host": "localhost",
                "simulatorName": "tcp://localhost:2014/Simulator",
                "coordinatorId": "em2ApF_ybUIQTNajqcZsP_r7rh-6qHabQ-UkOHfSBn0W",
                "coordinatorName": "tcp://localhost:2011/Coordinator",
                "config": "/root/lisatmp_10.2.4/lads/ConfigDirectory/Bank v5
/Configs/project.config",
                "ssvUsers": "1",
                "testcase": {
                    "id": "TestCaseID",
                    "name": "test-data-driven-http-rest",
                    "documentation": "Put documentation of the Test Case here.",
                    "resultState": "",
                    "steps": [
                        {
                            "uid": "UserID",
                            "name": "http GET /3/movie/49047/",
                            "documentation": null,
                            "type": "com.itko.lisa.ws.rest.RESTNode",
                            "quiet": false,
                            "next": "http GET /3/movie/49163"
                        },
```

```
        {
            "uid": "UserID",
            "name": "http GET /3/movie/49163",
            "documentation": null,
            "type": "com.itko.lisa.ws.rest.RESTNode",
            "quiet": false,
            "next": "http GET /3/movie/49381"
        },
        {
            "uid": "UserID",
            "name": "http GET /3/movie/49381",
            "documentation": null,
            "type": "com.itko.lisa.ws.rest.RESTNode",
            "quiet": false,
            "next": "GET empty column value"
        },
        {
            "uid": "UserID",
            "name": "GET empty column value",
            "documentation": null,
            "type": "com.itko.lisa.ws.rest.RESTNode",
            "quiet": false,
            "next": "http GET /3/movie/notfound"
        },
        {
            "uid": "UserID",
            "name": "http GET /3/movie/notfound",
            "documentation": null,
            "type": "com.itko.lisa.ws.rest.RESTNode",
            "quiet": false,
            "next": "end"
        },
        {
            "uid": "UserID",
            "name": "abort",
            "documentation": null,
            "type": "com.itko.lisa.test.AbortStep",
            "quiet": true,
            "next": ""
        },
        {
            "uid": "UserID",
            "name": "fail",
            "documentation": null,
            "type": "com.itko.lisa.test.Abend",
            "quiet": true,
            "next": "abort"
        },
        {
            "uid": "UserID",
            "name": "end",
            "documentation": null,
            "type": "com.itko.lisa.test.NormalEnd",
            "quiet": true,
            "next": "fail"
        }
    ]
```

```
                },
                "stagingDocument": {
                    "name": "Run1User1Cycle",
                    "documentation": "",
                    "distributionType": "com.itko.lisa.coordinator.runpatterns.
PercentDistribution",
                    "loadPatternType": "com.itko.lisa.coordinator.runpatterns.
RunNTimesPattern",
                    "think": "100"
                }
            },
            {
                "_links": {
                    "next": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/tests/reports/ReportID/cycles"
                    },
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/tests/reports/ReportID"
                    }
                },
                "testRunUniqueId": "TestRunUniqueID",
                "testRunName": "Run1User1Cycle",
                "testcaseId": "TestCaseID",
                "testcaseName": "test-data-driven-http-rest",
                "testCaseDoc": "Put documentation of the Test Case here.",
                "startTime": "2017-12-20T13:02:20-0500",
                "endTime": "2017-12-20T13:02:25-0500",
                "elapsedTimeInMillSec": 5505,
                "user": "admin",
                "endedState": "PASSED",
                "done": true,
                "manuallyTerminated": false,
                "passCount": 1,
                "failCount": 0,
                "warningCount": 0,
                "abortCount": 0,
                "passPercent": 100,
                "failPercent": 0,
                "abortPercent": 0,
                "modelName": "test-data-driven-http-rest",
                "loadPattern": "Run N Times",
                "host": "localhost",
                "simulatorName": "tcp://localhost:2014/Simulator",
                "coordinatorId": "em2ApF_ybUIQTNajqcZsP_r7rh-6qHabQ-UkOHfSBn0W",
                "coordinatorName": "tcp://localhost:2011/Coordinator",
                "config": "/root/lisatmp_10.2.5/lads
/EAC6BDD9E5AF11E7BADA020027E349EE/Bank v5/Configs/project.config",
                "ssvUsers": "1",
                "testcase": {
                    "id": "TestCaseID",
                    "name": "test-data-driven-http-rest",
                    "documentation": "Put documentation of the Test Case here.",
                    "resultState": "",
                    "steps": [
                        {
```

```
        "uid": "UserID",
        "name": "http GET /3/movie/49047/",
        "documentation": null,
        "type": "com.itko.lisa.ws.rest.RESTNode",
        "quiet": false,
        "next": "http GET /3/movie/49163"
    },
    {
        "uid": "UserID",
        "name": "http GET /3/movie/49163",
        "documentation": null,
        "type": "com.itko.lisa.ws.rest.RESTNode",
        "quiet": false,
        "next": "http GET /3/movie/49381"
    },
    {
        "uid": "UserID",
        "name": "http GET /3/movie/49381",
        "documentation": null,
        "type": "com.itko.lisa.ws.rest.RESTNode",
        "quiet": false,
        "next": "GET empty column value"
    },
    {
        "uid": "UserID",
        "name": "GET empty column value",
        "documentation": null,
        "type": "com.itko.lisa.ws.rest.RESTNode",
        "quiet": false,
        "next": "http GET /3/movie/notfound"
    },
    {
        "uid": "UserID",
        "name": "http GET /3/movie/notfound",
        "documentation": null,
        "type": "com.itko.lisa.ws.rest.RESTNode",
        "quiet": false,
        "next": "end"
    },
    {
        "uid": "UserID",
        "name": "abort",
        "documentation": null,
        "type": "com.itko.lisa.test.AbortStep",
        "quiet": true,
        "next": ""
    },
    {
        "uid": "UserID",
        "name": "fail",
        "documentation": null,
        "type": "com.itko.lisa.test.Abend",
        "quiet": true,
        "next": "abort"
    },
    {
        "uid": "UserID",
```

```
                            "name": "end",
                            "documentation": null,
                            "type": "com.itko.lisa.test.NormalEnd",
                            "quiet": true,
                            "next": "fail"
                        }
                    ]
                },
                "stagingDocument": {
                    "name": "Run1User1Cycle",
                    "documentation": "",
                    "distributionType": "com.itko.lisa.coordinator.runpatterns.
PercentDistribution",
                    "loadPatternType": "com.itko.lisa.coordinator.runpatterns.
RunNTimesPattern",
                    "think": "100"
                }
            }
        ]
    }
}
```


Retrieve a Test Report for a Given "Test Run Unique Id"

GET http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api/v1/tests/reports/TestRunUniqueID

Response:


```
{
    "_links": {
        "next": {
            "href": "http://localhost:1505/lisa-test-invoke/api/v1/tests/reports
/TestRunUniqueID/cycles"
        },
        "self": {
            "href": "http://localhost:1505/lisa-test-invoke/api/v1/tests/reports
/TestRunUniqueID"
        }
    },
    "testRunUniqueId": "TestRunUniqueID",
    "testRunName": "Run1User1Cycle",
    "testcaseId": "TestCaseID",
    "testcaseName": "test-data-driven-http-rest",
    "testCaseDoc": "Put documentation of the Test Case here.",
    "startTime": "2017-12-20T13:02:20-0500",
    "endTime": "2017-12-20T13:02:25-0500",
    "elapsedTimeInMillSec": 5505,
    "user": "admin",
    "endedState": "PASSED",
    "done": true,
    "manuallyTerminated": false,
    "passCount": 1,
    "failCount": 0,
    "warningCount": 0,
```

```
    "abortCount": 0,
    "passPercent": 100,
    "failPercent": 0,
    "abortPercent": 0,
    "modelName": "test-data-driven-http-rest",
    "loadPattern": "Run N Times",
    "host": "localhost",
    "simulatorName": "tcp://localhost:2014/Simulator",
    "coordinatorId": "em2ApF_ybUIQTNajqcZsP_r7rh-6qHabQ-UkOHfSBn0W",
    "coordinatorName": "tcp://localhost:2011/Coordinator",
    "config": "/root/lisatmp_10.2.5/lads/ConfigDirectory/Bank v5/Configs/project.
config",
    "ssvUsers": "1",
    "testcase": {
        "id": "TestCaseID",
        "name": "test-data-driven-http-rest",
        "documentation": "Put documentation of the Test Case here.",
        "resultState": "",
        "steps": [
            {
                "uid": "UserID",
                "name": "http GET /3/movie/49047/",
                "documentation": null,
                "type": "com.itko.lisa.ws.rest.RESTNode",
                "quiet": false,
                "next": "http GET /3/movie/49163"
            },
            {
                "uid": "UserID",
                "name": "http GET /3/movie/49163",
                "documentation": null,
                "type": "com.itko.lisa.ws.rest.RESTNode",
                "quiet": false,
                "next": "http GET /3/movie/49381"
            },
            {
                "uid": "UserID",
                "name": "http GET /3/movie/49381",
                "documentation": null,
                "type": "com.itko.lisa.ws.rest.RESTNode",
                "quiet": false,
                "next": "GET empty column value"
            },
            {
                "uid": "UserID",
                "name": "GET empty column value",
                "documentation": null,
                "type": "com.itko.lisa.ws.rest.RESTNode",
                "quiet": false,
                "next": "http GET /3/movie/notfound"
            },
            {
                "uid": "UserID",
                "name": "http GET /3/movie/notfound",
                "documentation": null,
                "type": "com.itko.lisa.ws.rest.RESTNode",
                "quiet": false,
```

```
                "next": "end"
            },
            {
                "uid": "UserID",
                "name": "abort",
                "documentation": null,
                "type": "com.itko.lisa.test.AbortStep",
                "quiet": true,
                "next": ""
            },
            {
                "uid": "UserID",
                "name": "fail",
                "documentation": null,
                "type": "com.itko.lisa.test.Abend",
                "quiet": true,
                "next": "abort"
            },
            {
                "uid": "UserID",
                "name": "end",
                "documentation": null,
                "type": "com.itko.lisa.test.NormalEnd",
                "quiet": true,
                "next": "fail"
            }
        ]
    },
    "stagingDocument": {
        "name": "Run1User1Cycle",
        "documentation": "",
        "distributionType": "com.itko.lisa.coordinator.runpatterns.
PercentDistribution",
        "loadPatternType": "com.itko.lisa.coordinator.runpatterns.
RunNTimesPattern",
        "think": "100"
    }
}
```


Retrieve all the Cycles for a Test Report for a Given "Test Run Unique Id"

GET http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api /v1/tests/reports/Test/cycles


```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-test-invoke/api/v1/tests/reports
/TestRunUniqueID/cycles"
        }
    },
    "count": 1,
    "_embedded": {
        "CycleHistory": [
            {
                "_links": {
```

```
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/tests/reports/TestRunUniqueID/cycles/CycleUniqueID"
                    }
                },
                "cycleUniqueId": "CycleUniqueID",
                "cycle": "0",
                "run": "test-data-driven-http-rest [Run1User1Cycle]",
                "simulator": "tcp://localhost:2014/Simulator",
                "startTime": "2017-12-20T13:02:20-0500",
                "endTime": "2017-12-20T13:02:25-0500",
                "elapsedTimeInMillSec": 5166,
                "endedState": "PASSED",
                "_embedded": {
                    "StepExecHistory": [],
                    "preCompanions": [],
                    "postCompanions": [],
                    "messages": []
                }
            }
        ]
    }
}
```


Retrieve a Cycle for a Test Report for a Given "Test Run Unique Id" and "Cycle Unique Id"

GET http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api /v1/tests/reports/TestRunUniqueID/cycles/CycleUniqueID


```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-test-invoke/api/v1/tests/reports
/TestRunUniqueID/cycles/CycleUniqueID"
        }
    },
    "cycleUniqueId": "CycleUniqueID",
    "cycle": "0",
    "run": "test-data-driven-http-rest [Run1User1Cycle]",
    "simulator": "tcp://localhost:2014/Simulator",
    "startTime": "2017-12-20T13:02:20-0500",
    "endTime": "2017-12-20T13:02:25-0500",
    "elapsedTimeInMillSec": 5166,
    "endedState": "PASSED",
    "_embedded": {
        "StepExecHistory": [
            {
                "stepGUID": "TestStepID",
                "stepName": "http GET /3/movie/49047/",
                "type": "STEP",
                "errorCount": "0",
                "warningCount": "0",
                "quite": "false",
```

```
                "startTime": "2017-12-20T13:02:20-0500",
                "endedState": "PASSED",
                "stacktrace": null,
                "stepCommand": {
                    "request": "GET /3/movie/49047?
api_key=cd1d7f222b868aec8b843a48baa16df3 HTTP/1.1\nPragma: no-cache\nCache-
Control: no-cache\nAccept: application/json\nlisaFrameRemoteIP: 10.162.27.216
\nlisaFrameRoot: true\nlisaFrameID: ec953b50-e5af-11e7-937d-020027e349ee\n\n",
                    "response": "PG1vdmllPgogICAgPGFkdWx0IHR5cGU9ImJvb2xlYW4iPkZBT
FNFPC9hZHVsdD4KICAgIDxiYWNrZHJvcF9fcGF0aCB0eXBlPSJzdHJpbmciPi9oejNKZkFpa1lYdGFOV0l
KaFdNNHA1c3k1T1ouanBnPC9iYWNrZHJvcF9fcGF0aD4KICAgIDxiZWxvbmdzX190b19fY29sbGVjdGlvb
iBjbGFzcz0ib2JqZWN0IiBudWxsPSJ0cnVlIiAvPgogICAgPGJ1ZGdldCB0eXBlPSJudW1iZXIiPjEwNTA
wMDAwMDwvYnVkZ2V0PgogICAgPGdlbnJlPkRSQU1BPC9nZW5yZT4KICAgIDxob21lcGFnZSB0eXBlPSJzd
HJpbmciPmh0dHA6Ly9ncmF2aXR5bW92aWUud2FybmVyYnJvcy5jb20vPC9ob21lcGFnZT4KICAgIDxpZCB
0eXBlPSJudW1iZXIiPjQ5MDQ3PC9pZD4gICAgCiAgICA8dGl0bGUgdHlwZT0ic3RyaW5nIj5HcmF2aXR5P
C90aXRsZT4KICAgIDxvdmVydmlldyB0eXBlPSJzdHJpbmciPkRyLiBSeWFuIFN0b25lIChTYW5kcmEgQnV
sbG9jayksIGEgYnJpbGxpYW50IG1lZGljYWwgZW5naW5lZXIgb24gaGVyIGZpcnN0IFNodXR0bGUgbWlzc
2lvbiwgd2l0aCB2ZXRlcmFuIGFzdHJvbmF1dCBNYXR0IEtvd2Fsc2t5IChHZW9yZ2UgQ2xvb25leSkgaW4
gY29tbWFuZCBvZiBoaXMgbGFzdCBmbGlnaHQgYmVmb3JlIHJldGlyaW5nLiBCdXQgb24gYSBzZWVtaW5nb
Hkgcm91dGluZSBzcGFjZXdhbGssIGRpc2FzdGVyIHN0cmlrZXMuPC9vdmVydmlldz4KICAgIDxwb3B1bGF
yaXR5IHR5cGU9Im51bWJlciI+NDcuOTA0NTMzPC9wb3B1bGFyaXR5PgogICAgPHBvc3Rlcl9fcGF0aCB0e
XBlPSJzdHJpbmciPi8yZ1BqTFdJeXJXbEFuMkRnS01PS1RCblpZeU8uanBnPC9wb3N0ZXJfX3BhdGg+ICA
gICAgICAgICAgICAgICA8cmV2ZW51ZSB0eXBlPSJudW1iZXIiPjUzNjAwMDAwMDwvcmV2ZW51ZT4KICAgI
DxydW50aW1lIHR5cGU9Im51bWJlciI+OTE8L3J1bnRpbWU+ICAgIAogICAgPHN0YXR1cyB0eXBlPSJzdHJ
pbmciPlJlbGVhc2VkPC9zdGF0dXM+CiAgICA8dGFnbGluZSB0eXBlPSJzdHJpbmciPkRvbid0IExldCBHb
zwvdGFnbGluZT4gICAgCiAgICA8dm90ZV9fYXZlcmFnZSB0eXBlPSJudW1iZXIiPjg8L3ZvdGVfX2F2ZXJ
hZ2U+CiAgICA8dm90ZV9fY291bnQgdHlwZT0ibnVtYmVyIj43MDk8L3ZvdGVfX2NvdW50Pgo8L21vdmllP
g==",
                    "responseBase64Encoded": "true"
                },
                "_embedded": {
                    "messages": [],
                    "asserts": [
                        {
                            "name": "Ensure Result Contains \"Gravity\"",
                            "actualValue": "<movie>\n    <adult type=\"boolean\"
>FALSE</adult>\n    <backdrop__path type=\"string\">/hz3JfAikYXtaNWIJhWM4p5sy5OZ.
jpg</backdrop__path>\n    <belongs__to__collection class=\"object\" null=\"true\"
/>\n    <budget type=\"number\">105000000</budget>\n    <genre>DRAMA</genre>\n
<homepage type=\"string\">http://gravitymovie.warnerbros.com/</homepage>\n    <id
type=\"number\">49047</id>    \n    <title type=\"string\">Gravity</title>\n
<overview type=\"string\">Dr. Ryan Stone (Sandra Bullock), a brilliant medical
engineer on her first Shuttle mission, with veteran astronaut Matt Kowalsky
(George Clooney) in command of his last flight before retiring. But on a
seemingly routine spacewalk, disaster strikes.</overview>\n    <popularity type=\"
number\">47.904533</popularity>\n    <poster__path type=\"string\">
/2gPjLWIyrWlAn2DgKMOKTBnZYyO.jpg</poster__path>            \n    <revenue type=\"
number\">536000000</revenue>\n    <runtime type=\"number\">91</runtime>    \n
<status type=\"string\">Released</status>\n    <tagline type=\"string\">Don't Let
Go</tagline>    \n    <vote__average type=\"number\">8</vote__average>\n
<vote__count type=\"number\">709</vote__count>\n</movie>",
                            "expectation": "Looked for \"Gravity\" in result.",
                            "fired": "false",
                            "isGlobal": "false",
                            "message": "The assertion of type \"Result as String
Contains Given String\" passed evaluation.",
                            "result": "true"
```

```
                        }
                    ],
                    "preFilters": [],
                    "postFilters": [],
                    "datasets": [],
                    "events": [],
                    "properties": [],
                    "httpDetails": []
                }
            },
            {
                "stepGUID": "TestStepID",
                "stepName": "http GET /3/movie/49163",
                "type": "STEP",
                "errorCount": "0",
                "warningCount": "0",
                "quite": "false",
                "startTime": "2017-12-20T13:02:21-0500",
                "endedState": "PASSED",
                "stacktrace": null,
                "stepCommand": {
                    "request": "GET /3/movie/49163?
api_key=cd1d7f222b868aec8b843a48baa16df119 HTTP/1.1\nPragma: no-cache\nCache-
Control: no-cache\nAccept: application/json\nlisaFrameRemoteIP: 10.162.27.216
\nlisaFrameRoot: true\nlisaFrameID: ed24aa10-e5af-11e7-937d-020027e349ee\n\n",
                    "response": "PG1vdmllPgogICAgPGFkdWx0IHR5cGU9ImJvb2xlYW4iPkZBT
FNFPC9hZHVsdD4KICAgIDxiYWNrZHJvcF9fcGF0aCB0eXBlPSJzdHJpbmciPi9oejNKZkFpa1lYdGFOV0l
KaFdNNHA1c3k1T1ouanBnPC9iYWNrZHJvcF9fcGF0aD4KICAgIDxiZWxvbmdzX190b19fY29sbGVjdGlvb
iBjbGFzcz0ib2JqZWN0IiBudWxsPSJ0cnVlIiAvPgogICAgPGJ1ZGdldCB0eXBlPSJudW1iZXIiPjEwNTA
wMDAwMDwvYnVkZ2V0PgogICAgPGdlbnJlPkFDVElPTjwvZ2VucmU+CiAgICA8aG9tZXBhZ2UgdHlwZT0ic
3RyaW5nIj5tYXRyaXhtb3ZpZS5jb208L2hvbWVwYWdlPgogICAgPGlkIHR5cGU9Im51bWJlciI+NDkxNjM
8L2lkPiAgICAKICAgIDx0aXRsZSB0eXBlPSJzdHJpbmciPlRoZSBNYXRyaXg8L3RpdGxlPgogICAgPG92Z
XJ2aWV3IHR5cGU9InN0cmluZyI+QSBjb21wdXRlciBoYWNrZXIgbGVhcm5zIGZyb20gbXlzdGVyaW91cyB
yZWJlbHMgYWJvdXQgdGhlIHRydWUgbmF0dXJlIG9mIGhpcyByZWFsaXR5IGFuZCBoaXMgcm9sZSBpbiB0a
GUgd2FyIGFnYWluc3QgaXRzIGNvbnRyb2xsZXJzLjwvb3ZlcnZpZXc+CiAgICA8cG9wdWxhcml0eSB0eXB
lPSJudW1iZXIiPjQ0PC9wb3B1bGFyaXR5PgogICAgPHBvc3Rlcl9fcGF0aCB0eXBlPSJzdHJpbmciPi8yZ
1BqTFdJeXJXbEFuMkRnS01PS1RCblpZeU8uanBnPC9wb3N0ZXJfX3BhdGg+ICAgICAgICAgICAgICAgICA
8cmV2ZW51ZSB0eXBlPSJudW1iZXIiPjQ0MDAwMDAwMDA8L3JldmVudWU+CiAgICA8cnVudGltZSB0eXBlP
SJudW1iZXIiPjEyMDwvcnVudGltZT4gICAgCiAgICA8c3RhdHVzIHR5cGU9InN0cmluZyI+UmVsZWFzZWQ
8L3N0YXR1cz4KICAgIDx0YWdsaW5lIHR5cGU9InN0cmluZyI+VGhlIE1hdHJpeDwvdGFnbGluZT4gICAgC
iAgICA8dm90ZV9fYXZlcmFnZSB0eXBlPSJudW1iZXIiPjEwPC92b3RlX19hdmVyYWdlPgogICAgPHZvdGV
fX2NvdW50IHR5cGU9Im51bWJlciI+MTAwMDwvdm90ZV9fY291bnQ+CjwvbW92aWU+",
                    "responseBase64Encoded": "true"
                },
                "_embedded": {
                    "messages": [],
                    "asserts": [
                        {
                            "name": "Ensure Result Contains \"The Matrix\"",
                            "actualValue": "<movie>\n    <adult type=\"boolean\"
>FALSE</adult>\n    <backdrop__path type=\"string\">/hz3JfAikYXtaNWIJhWM4p5sy5OZ.
jpg</backdrop__path>\n    <belongs__to__collection class=\"object\" null=\"true\"
/>\n    <budget type=\"number\">105000000</budget>\n    <genre>ACTION<
/genre>\n    <homepage type=\"string\">matrixmovie.com</homepage>\n    <id type=\"
number\">49163</id>    \n    <title type=\"string\">The Matrix</title>\n
<overview type=\"string\">A computer hacker learns from mysterious rebels about
the true nature of his reality and his role in the war against its controllers.<
```

```
/overview>\n    <popularity type=\"number\">44</popularity>\n    <poster__path
type=\"string\">/2gPjLWIyrWlAn2DgKMOKTBnZYyO.jpg</poster__path>            \n
<revenue type=\"number\">4400000000</revenue>\n    <runtime type=\"number\">120<
/runtime>    \n    <status type=\"string\">Released</status>\n    <tagline type=\"
string\">The Matrix</tagline>    \n    <vote__average type=\"number\">10<
/vote__average>\n    <vote__count type=\"number\">1000</vote__count>\n</movie>",
                            "expectation": "Looked for \"The Matrix\" in result.",
                            "fired": "false",
                            "isGlobal": "false",
                            "message": "The assertion of type \"Result as String
Contains Given String\" passed evaluation.",
                            "result": "true"
                        }
                    ],
                    "preFilters": [],
                    "postFilters": [],
                    "datasets": [],
                    "events": [],
                    "properties": [],
                    "httpDetails": []
                }
            },
            {
                "stepGUID": "TestStepID",
                "stepName": "http GET /3/movie/49381",
                "type": "STEP",
                "errorCount": "0",
                "warningCount": "0",
                "quite": "false",
                "startTime": "2017-12-20T13:02:22-0500",
                "endedState": "PASSED",
                "stacktrace": null,
                "stepCommand": {
                    "request": "GET /3/movie/49381?
api_key=cd1d7f222b868aec8b843a48baa16df337 HTTP/1.1\nPragma: no-cache\nCache-
Control: no-cache\nAccept: application/json\nlisaFrameRemoteIP: 10.162.27.216
\nlisaFrameRoot: true\nlisaFrameID: edd387b0-e5af-11e7-937d-020027e349ee\n\n",
                    "response": "PG1vdmllPgogICAgPGFkdWx0IHR5cGU9ImJvb2xlYW4iPkZBT
FNFPC9hZHVsdD4KICAgIDxiYWNrZHJvcF9fcGF0aCB0eXBlPSJzdHJpbmciPi9oejNKZkFpa1lYdGFOV0l
KaFdNNHA1c3k1T1ouanBnPC9iYWNrZHJvcF9fcGF0aD4KICAgIDxiZWxvbmdzX190b19fY29sbGVjdGlvb
iBjbGFzcz0ib2JqZWN0IiBudWxsPSJ0cnVlIiAvPgogICAgPGJ1ZGdldCB0eXBlPSJudW1iZXIiPjEwNTA
wMDAwMDwvYnVkZ2V0PgogICAgPGdlbnJlPkZPUkVJR048L2dlbnJlPgogICAgPGhvbWVwYWdlIHR5cGU9I
nN0cmluZyI+c2V2ZW5zYW11cmFpbW92aWUuY29tPC9ob21lcGFnZT4KICAgIDxpZCB0eXBlPSJudW1iZXI
iPjQ5MzgxPC9pZD4gICAgCiAgICA8dGl0bGUgdHlwZT0ic3RyaW5nIj5TZXZlbiBTYW11cmFpPC90aXRsZ
T4KICAgIDxvdmVydmlldyB0eXBlPSJzdHJpbmciPkEgcG9vciB2aWxsYWdlIHVuZGVyIGF0dGFjayBieSB
iYW5kaXRzIHJlY3J1aXRzIHNldmVuIHVuZW1wbG95ZWQgc2FtdXJhaSB0byBoZWxwIHRoZW0gZGVmZW5kI
HRoZW1zZWx2ZXMuPC9vdmVydmlldz4KICAgIDxwb3B1bGFyaXR5IHR5cGU9Im51bWJlciI+NTA8L3BvcHV
sYXJpdHk+CiAgICA8cG9zdGVyX19wYXRoIHR5cGU9InN0cmluZyI+LzJnUGpMV0l5cldsQW4yRGdLTU9LV
EJuWll5Ty5qcGc8L3Bvc3Rlcl9fcGF0aD4gICAgICAgICAgICAgICAgICAKICAgIDxyZXZlbnVlIHR5cGU9Im51bWJ
lciI+MzM2MDAwMDwvcmV2ZW51ZT4KICAgIDxydW50aW1lIHR5cGU9Im51bWJlciI+OTk8L3J1bnRpbWU+I
CAgIAogICAgPHN0YXR1cyB0eXBlPSJzdHJpbmciPlJlbGVhc2VkPC9zdGF0dXM+CiAgICA8dGFnbGluZSB
0eXBlPSJzdHJpbmciPlNldmVuIFNhbXVyYWk8L3RhZ2xpbmU+ICAgIAogICAgPHZvdGVfX2F2ZXJhZ2Ugd
HlwZT0ibnVtYmVyIj45PC92b3RlX19hdmVyYWdlPgogICAgPHZvdGVfX2NvdW50IHR5cGU9Im51bWJlciI
+ODAwPC92b3RlX19jb3VudD4KPC9tb3ZpZT4=",
                    "responseBase64Encoded": "true"
                },
                "_embedded": {
```

```
                    "messages": [],
                    "asserts": [
                        {
                            "name": "Ensure Result Contains \"Seven Samurai\"",
                            "actualValue": "<movie>\n    <adult type=\"boolean\"
>FALSE</adult>\n    <backdrop__path type=\"string\">/hz3JfAikYXtaNWIJhWM4p5sy5OZ.
jpg</backdrop__path>\n    <belongs__to__collection class=\"object\" null=\"true\"
/>\n    <budget type=\"number\">105000000</budget>\n    <genre>FOREIGN<
/genre>\n    <homepage type=\"string\">sevensamuraimovie.com</homepage>\n    <id
type=\"number\">49381</id>    \n    <title type=\"string\">Seven Samurai<
/title>\n    <overview type=\"string\">A poor village under attack by bandits
recruits seven unemployed samurai to help them defend themselves.</overview>\n
<popularity type=\"number\">50</popularity>\n    <poster__path type=\"string\">
/2gPjLWIyrWlAn2DgKMOKTBnZYyO.jpg</poster__path>            \n    <revenue type=\"
number\">3360000</revenue>\n    <runtime type=\"number\">99</runtime>    \n
<status type=\"string\">Released</status>\n    <tagline type=\"string\">Seven
Samurai</tagline>    \n    <vote__average type=\"number\">9</vote__average>\n
<vote__count type=\"number\">800</vote__count>\n</movie>",
                            "expectation": "Looked for \"Seven Samurai\" in
result.",
                            "fired": "false",
                            "isGlobal": "false",
                            "message": "The assertion of type \"Result as String
Contains Given String\" passed evaluation.",
                            "result": "true"
                        }
                    ],
                    "preFilters": [],
                    "postFilters": [],
                    "datasets": [],
                    "events": [],
                    "properties": [],
                    "httpDetails": []
                }
            },
            {
                "stepGUID": "TestStepID",
                "stepName": "GET empty column value",
                "type": "STEP",
                "errorCount": "0",
                "warningCount": "0",
                "quite": "false",
                "startTime": "2017-12-20T13:02:23-0500",
                "endedState": "PASSED",
                "stacktrace": null,
                "stepCommand": {
                    "request": "GET /3/movie/49382?
api_key=cd1d7f222b868aec8b843a48baa16df337 HTTP/1.1\nPragma: no-cache\nCache-
Control: no-cache\nAccept: application/json\nlisaFrameRemoteIP: 10.162.27.216
\nlisaFrameRoot: true\nlisaFrameID: ee6121b0-e5af-11e7-937d-020027e349ee\n\n",
                    "response": "PG1vdmllPgogICAgPGFkdWx0IHR5cGU9ImJvb2xlYW4iPkZBT
FNFPC9hZHVsdD4KICAgIDxiYWNrZHJvcF9fcGF0aCB0eXBlPSJzdHJpbmciPi9oejNKZkFpa1lYdGFOV0l
KaFdNNHA1c3k1T1ouanBnPC9iYWNrZHJvcF9fcGF0aD4KICAgIDxiZWxvbmdzX190b19fY29sbGVjdGlvb
iBjbGFzcz0ib2JqZWN0IiBudWxsPSJ0cnVlIiAvPgogICAgPGJ1ZGdldCB0eXBlPSJudW1iZXIiPjEwNTA
wMDAwMDwvYnVkZ2V0PgogICAgPGdlbnJlPjwvZ2VucmU+CiAgICA8aG9tZXBhZ2UgdHlwZT0ic3RyaW5nI
j5Vc3VhbCBTdXNwZWN0czwvaG9tZXBhZ2U+CiAgICA8aWQgdHlwZT0ibnVtYmVyIj40OTM4MjwvaWQ+ICA
gIAogICAgPHRpdGxlIHR5cGU9InN0cmluZyI+VXN1YWwgU3VzcGVjdHM8L3RpdGxlPgogICAgPG92ZXJ2a
```

```
WV3IHR5cGU9InN0cmluZyI+Q3JpbWUgbW92aWU8L292ZXJ2aWV3PgogICAgPHBvcHVsYXJpdHkgdHlwZT0
ibnVtYmVyIj41MDwvcG9wdWxhcml0eT4KICAgIDxwb3N0ZXJfX3BhdGggdHlwZT0ic3RyaW5nIj4vMmdQa
kxXSXlyV2xBbjJEZ0tNT0tUQm5aWXlPLmpwZzwvcG9zdGVyX19wYXRoPiAgICAgICAgICAgIAogICAgPHJ
ldmVudWUgdHlwZT0ibnVtYmVyIj4zMzYwMDAwPC9yZXZlbnVlPgogICAgPHJ1bnRpbWUgdHlwZT0ibnVtY
mVyIj45OTwvcnVudGltZT4gICAgCiAgICA8c3RhdHVzIHR5cGU9InN0cmluZyI+UmVsZWFzZWQ8L3N0YXR
1cz4KICAgIDx0YWdsaW5lIHR5cGU9InN0cmluZyI+VXN1YWwgU3VzcGVjdHM8L3RhZ2xpbmU+ICAgIAogI
CAgPHZvdGVfX2F2ZXJhZ2UgdHlwZT0ibnVtYmVyIj4xMDwvdm90ZV9fYXZlcmFnZT4KICAgIDx2b3RlX19
jb3VudCB0eXBlPSJudW1iZXIiPjEwMDA8L3ZvdGVfX2NvdW50Pgo8L21vdmllPg==",
                    "responseBase64Encoded": "true"
                },
                "_embedded": {
                    "messages": [],
                    "asserts": [
                        {
                            "name": "Ensure Result Contains empty genre tag",
                            "actualValue": "<movie>\n    <adult type=\"boolean\"
>FALSE</adult>\n    <backdrop__path type=\"string\">/hz3JfAikYXtaNWIJhWM4p5sy5OZ.
jpg</backdrop__path>\n    <belongs__to__collection class=\"object\" null=\"true\"
/>\n    <budget type=\"number\">105000000</budget>\n    <genre></genre>\n
<homepage type=\"string\">Usual Suspects</homepage>\n    <id type=\"number\"
>49382</id>    \n    <title type=\"string\">Usual Suspects</title>\n    <overview
type=\"string\">Crime movie</overview>\n    <popularity type=\"number\">50<
/popularity>\n    <poster__path type=\"string\">/2gPjLWIyrWlAn2DgKMOKTBnZYyO.jpg<
/poster__path>            \n    <revenue type=\"number\">3360000</revenue>\n
<runtime type=\"number\">99</runtime>    \n    <status type=\"string\">Released<
/status>\n    <tagline type=\"string\">Usual Suspects</tagline>    \n
<vote__average type=\"number\">10</vote__average>\n    <vote__count type=\"
number\">1000</vote__count>\n</movie>",
                            "expectation": "Looked for \"<genre></genre>\" in
result.",
                            "fired": "false",
                            "isGlobal": "false",
                            "message": "The assertion of type \"Result as String
Contains Given String\" passed evaluation.",
                            "result": "true"
                        }
                    ],
                    "preFilters": [],
                    "postFilters": [],
                    "datasets": [],
                    "events": [],
                    "properties": [],
                    "httpDetails": []
                }
            },
            {
                "stepGUID": "TestStepID",
                "stepName": "http GET /3/movie/notfound",
                "type": "STEP",
                "errorCount": "0",
                "warningCount": "0",
                "quite": "false",
                "startTime": "2017-12-20T13:02:24-0500",
                "endedState": "PASSED",
                "stacktrace": null,
                "stepCommand": {
```

```
                    "request": "GET /3/movie/notfound?
api_key=cd1d7f222b868aec8b843a48baa16df337 HTTP/1.1\nPragma: no-cache\nCache-
Control: no-cache\nAccept: application/json\nlisaFrameRemoteIP: 10.162.27.216
\nlisaFrameRoot: true\nlisaFrameID: eefd88c0-e5af-11e7-937d-020027e349ee\n\n",
                    "response": "PGh0bWw+CiAgICA8aGVhZD48dGl0bGU+NDA0IE5vdCBGb3VuZ
DwvdGl0bGU+PC9oZWFkPgogICAgPGJvZHk+CiAgICAgICAgPGgxPk5vdCBGb3VuZDwvaDE+CiAgICAgICA
gPHA+VGhlIHJlcXVlc3RlZCBVUkwgd2FzIG5vdCBmb3VuZCBvbiB0aGlzIHNlcnZlci48L3A+CiAgICAgI
CAgPGhyLz4KICAgICAgICA8cD48aT5UaGUgTElTQSBWU0Ugc2VydmljZSBjb3VsZCBub3QgbWF0Y2ggeW9
1ciByZXF1ZXN0IHRvIGEgcmVjb3JkZWQgcmVxdWVzdC4mbmJzcDsgQ29uc2lkZXIgZXhwYW5kaW5nIHlvd
XIgc2VydmljZSBpbWFnZS48L2k+PC9wPgogICAgICAgIDxici8+PGZvbnQgc2l6ZT0iLTIiPlByb2R1Y2V
kIGJ5IGEgTElTQSB2aXJ0dWFsaXplZCB3ZWIgc2VydmVyLjwvZm9udD4KICAgIDwvYm9keT4KPC9odG1sP
go=",
                    "responseBase64Encoded": "true"
                },
                "_embedded": {
                    "messages": [],
                    "asserts": [
                        {
                            "name": "Ensure Result Contains \"404 Not Found\"",
                            "actualValue": "<html>\n    <head><title>404 Not
Found</title></head>\n    <body>\n        <h1>Not Found</h1>\n        <p>The
requested URL was not found on this server.</p>\n        <hr/>\n        <p><i>The
LISA VSE service could not match your request to a recorded request. 
Consider expanding your service image.</i></p>\n        <br/><font size=\"-2\"
>Produced by a LISA virtualized web server.</font>\n    </body>\n</html>\n",
                            "expectation": "Looked for \"404 Not Found\" in
result.",
                            "fired": "false",
                            "isGlobal": "false",
                            "message": "The assertion of type \"Result as String
Contains Given String\" passed evaluation.",
                            "result": "true"
                        }
                    ],
                    "preFilters": [],
                    "postFilters": [],
                    "datasets": [],
                    "events": [],
                    "properties": [],
                    "httpDetails": []
                }
            }
        ],
        "preCompanions": [],
        "postCompanions": [],
        "messages": []
    }
}
```


Retrieve Test Reports

GET http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api /v1/tests/reports/summary ?startTime=2017-12-20T00:00:00.000-05:00&endTime=2017-12-20T16:33:00.000-05:00

The startTime and endTime values are optional. If present, they must comply to ISO 8601 format; for example, *2014-10-22T16:33:00.000-05:00.*

Response:

```
{
  "_links": {
    "self": {
      "href": "http://localhost:1505/lisa-test-invoke/api/v1/suites/reports
/summary"
    }
  },
  "passCount": 5,
  "failCount": 0,
  "warningCount": 0,
  "abortCount": 4,
  "errorCount": 40,
  "runningCount": 0
}
```

Retrieve Suite Reports

GET http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api /v1/suites/reports ?startTime=2017-12-20T00:00:00.000-05:00&endTime=2017-12-20T16:33: 00.000-05:00

The startTime and endTime values are optional. If present, they must comply to ISO 8601 format; for example, *2014-10-22T16:33:00.000-05:00.*

Response

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-test-invoke/api/v1/suites/reports"
        }
    },
    "count": 5,
    "_embedded": {
        "Suites": [
            {
                "_links": {
                    "next": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests"
                    },
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID"
                    }
                },
                "suiteRunId": "SuiteRunUniqueID",
                "suiteName": "DataDrivenMultiCycle",
                "documentation": "Put documentation of the suite here.",
                "startTime": "2017-12-20T12:06:20-0500",
```

```
                "endTime": "2017-12-20T12:06:42-0500",
                "elapsedTimeInMillSec": 22727,
                "user": "admin",
                "endedState": "ABORTED",
                "done": true,
                "manuallyTerminated": false,
                "passCount": 0,
                "failCount": 0,
                "warningCount": 0,
                "abortCount": 2,
                "errorCount": 20,
                "totalTestsExecuted": 2,
                "totalTests": 2,
                "host": "localhost"
            },
            {
                "_links": {
                    "next": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests"
                    },
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID"
                    }
                },
                "suiteRunId": "SuiteRunUniqueID",
                "suiteName": "DataDrivenMultiCycle",
                "documentation": "Put documentation of the suite here.",
                "startTime": "2017-12-20T13:19:06-0500",
                "endTime": "2017-12-20T13:20:42-0500",
                "elapsedTimeInMillSec": 95525,
                "user": "admin",
                "endedState": "PASSED",
                "done": true,
                "manuallyTerminated": false,
                "passCount": 2,
                "failCount": 0,
                "warningCount": 0,
                "abortCount": 0,
                "errorCount": 0,
                "totalTestsExecuted": 2,
                "totalTests": 2,
                "host": "localhost"
            },
            {
                "_links": {
                    "next": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests"
                    },
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID"
                    }
                },
                "suiteRunId": "SuiteRunUniqueID",
```

```
                "suiteName": "DataDrivenMultiCycle",
                "documentation": "Put documentation of the suite here.",
                "startTime": "2017-12-20T12:22:52-0500",
                "endTime": "2017-12-20T12:23:25-0500",
                "elapsedTimeInMillSec": 33368,
                "user": "admin",
                "endedState": "PASSED",
                "done": true,
                "manuallyTerminated": false,
                "passCount": 1,
                "failCount": 0,
                "warningCount": 0,
                "abortCount": 0,
                "errorCount": 0,
                "totalTestsExecuted": 1,
                "totalTests": 2,
                "host": "localhost"
            },
            {
                "_links": {
                    "next": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests"
                    },
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID"
                    }
                },
                "suiteRunId": "SuiteRunUniqueID",
                "suiteName": "DataDrivenMultiCycle",
                "documentation": "Put documentation of the suite here.",
                "startTime": "2017-12-20T12:10:37-0500",
                "endTime": "2017-12-20T12:12:18-0500",
                "elapsedTimeInMillSec": 100847,
                "user": "admin",
                "endedState": "PASSED",
                "done": true,
                "manuallyTerminated": false,
                "passCount": 2,
                "failCount": 0,
                "warningCount": 0,
                "abortCount": 0,
                "errorCount": 0,
                "totalTestsExecuted": 2,
                "totalTests": 2,
                "host": "localhost"
            },
            {
                "_links": {
                    "next": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests"
                    },
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID"
```

```
                    }
                },
                "suiteRunId": "SuiteRunUniqueID",
                "suiteName": "DataDrivenMultiCycle",
                "documentation": "Put documentation of the suite here.",
                "startTime": "2017-12-20T12:05:05-0500",
                "endTime": "2017-12-20T12:05:30-0500",
                "elapsedTimeInMillSec": 25266,
                "user": "admin",
                "endedState": "ABORTED",
                "done": true,
                "manuallyTerminated": false,
                "passCount": 0,
                "failCount": 0,
                "warningCount": 0,
                "abortCount": 2,
                "errorCount": 20,
                "totalTestsExecuted": 2,
                "totalTests": 2,
                "host": "localhost"
            }
        ]
    }
}
```


Retrieve a Suite Report for a Given "Suite Run Unique Id"

GET http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api /v1/suites/reports/SuiteRunUniqueID


```
{
  "_links": {
    "self": {
      "href": "http://localhost:1505/lisa-test-invoke/api/v1/suites/reports/
{SuiteRunId}"
    },
    "tests": {
      "href": "http://localhost:1505/lisa-test-invoke/api/v1/suites/reports/
{SuiteRunId}/tests"
    }
  },
  "suiteRunId": "SuiteRunUniqueID",
  "suiteName": "AllTestsSuite",
  "documentation": "Put documentation of the suite here.",
  "startTime": "2018-05-22T10:05:25-0400",
  "endTime": "2018-05-22T10:05:35-0400",
  "elapsedTimeInMillSec": 9887,
  "user": "admin",
  "endedState": "PASSED",
  "endedStateValue": "0",
  "done": true,
  "manuallyTerminated": false,
  "passCount": 0,
  "failCount": 0,
```

```
  "warningCount": 0,
  "abortCount": 1,
  "errorCount": 1,
  "totalTestsExecuted": 10,
  "totalTests": 0,
  "host": "localhost.ca.com",
  "config": "/root/lisatmp_10.4.0/lads/ConfigDirectory/examples/Configs/project.
config",
  "path": "/root/lisatmp_10.4.0/lads/ConfigDirectory/examples/Suites/testsuite.
ste"
}
```


Retrieve all Tests for a Suite Report Given "Suite Run Unique Id"

GET http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api /v1/suites/reports/SuiteRunUniqueID /tests


```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-test-invoke/api/v1/suites/reports
/SuiteRunUniqueID/tests"
        }
    },
    "count": 2,
    "_embedded": {
        "Tests": [
            {
                "_links": {
                    "next": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests/TestRunUniqueID/cycles"
                    },
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests/TestRunUniqueID"
                    }
                },
                "testRunUniqueId": "TestRunUniqueID",
                "testRunName": "Run1User10Cycle",
                "testcaseId": "TestCaseID",
                "testcaseName": "test-data-driven-http-rest",
                "testCaseDoc": "Put documentation of the Test Case here.",
                "startTime": "2017-12-20T13:19:06-0500",
                "endTime": "2017-12-20T13:19:53-0500",
                "elapsedTimeInMillSec": 46514,
                "user": "",
                "endedState": "PASSED",
                "done": true,
                "manuallyTerminated": false,
                "passCount": 10,
                "failCount": 0,
                "warningCount": 0,
                "abortCount": 0,
                "passPercent": 100,
                "failPercent": 0,
```

```
                "abortPercent": 0,
                "modelName": "test-data-driven-http-rest",
                "loadPattern": "Run N Times",
                "host": "localhost",
                "simulatorName": null,
                "coordinatorId": "em2ApF_ybUIQTNajqcZsP_r7rh-6qHabQ-UkOHfSBn0W",
                "coordinatorName": "tcp://localhost:2011/Coordinator",
                "config": "/root/lisatmp_10.2.5/lads/ConfigDirectory/Bank v5
/Configs/project.config",
                "ssvUsers": "1",
                "testcase": {
                    "id": "TestCaseID",
                    "name": "test-data-driven-http-rest",
                    "documentation": "Put documentation of the Test Case here.",
                    "resultState": "",
                    "steps": [
                        {
                            "uid": "UserID",
                            "name": "http GET /3/movie/49047/",
                            "documentation": null,
                            "type": "com.itko.lisa.ws.rest.RESTNode",
                            "quiet": false,
                            "next": "http GET /3/movie/49163"
                        },
                        {
                            "uid": "UserID",
                            "name": "http GET /3/movie/49163",
                            "documentation": null,
                            "type": "com.itko.lisa.ws.rest.RESTNode",
                            "quiet": false,
                            "next": "http GET /3/movie/49381"
                        },
                        {
                            "uid": "UserID",
                            "name": "http GET /3/movie/49381",
                            "documentation": null,
                            "type": "com.itko.lisa.ws.rest.RESTNode",
                            "quiet": false,
                            "next": "GET empty column value"
                        },
                        {
                            "uid": "UserID",
                            "name": "GET empty column value",
                            "documentation": null,
                            "type": "com.itko.lisa.ws.rest.RESTNode",
                            "quiet": false,
                            "next": "http GET /3/movie/notfound"
                        },
                        {
                            "uid": "UserID",
                            "name": "http GET /3/movie/notfound",
                            "documentation": null,
                            "type": "com.itko.lisa.ws.rest.RESTNode",
                            "quiet": false,
                            "next": "end"
                        },
                        {
```

```
                            "uid": "UserID",
                            "name": "abort",
                            "documentation": null,
                            "type": "com.itko.lisa.test.AbortStep",
                            "quiet": true,
                            "next": ""
                        },
                        {
                            "uid": "UserID",
                            "name": "fail",
                            "documentation": null,
                            "type": "com.itko.lisa.test.Abend",
                            "quiet": true,
                            "next": "abort"
                        },
                        {
                            "uid": "UserID",
                            "name": "end",
                            "documentation": null,
                            "type": "com.itko.lisa.test.NormalEnd",
                            "quiet": true,
                            "next": "fail"
                        }
                    ]
                },
                "stagingDocument": {
                    "name": "Run1User10Cycle",
                    "documentation": "",
                    "distributionType": "com.itko.lisa.coordinator.runpatterns.
PercentDistribution",
                    "loadPatternType": "com.itko.lisa.coordinator.runpatterns.
RunNTimesPattern",
                    "think": "100"
                }
            },
            {
                "_links": {
                    "next": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests/TestRunUniqueID/cycles"
                    },
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueIDE/tests/TestRunUniqueID"
                    }
                },
                "testRunUniqueId": "TestRunUniqueID",
                "testRunName": "Run1User10Cycle",
                "testcaseId": "TestCaseID",
                "testcaseName": "test-data-driven-http-rest_01",
                "testCaseDoc": "Put documentation of the Test Case here.",
                "startTime": "2017-12-20T13:19:54-0500",
                "endTime": "2017-12-20T13:20:41-0500",
                "elapsedTimeInMillSec": 47526,
                "user": "",
                "endedState": "PASSED",
                "done": true,
```

```
                "manuallyTerminated": false,
                "passCount": 10,
                "failCount": 0,
                "warningCount": 0,
                "abortCount": 0,
                "passPercent": 100,
                "failPercent": 0,
                "abortPercent": 0,
                "modelName": "test-data-driven-http-rest_01",
                "loadPattern": "Run N Times",
                "host": "localhost",
                "simulatorName": null,
                "coordinatorId": "em2ApF_ybUIQTNajqcZsP_r7rh-6qHabQ-UkOHfSBn0W",
                "coordinatorName": "tcp://localhost:2011/Coordinator",
                "config": "/root/lisatmp_10.2.5/lads/ConfigDirectory/Bank v5
/Configs/project.config",
                "ssvUsers": "1",
                "testcase": {
                    "id": "TestCaseID",
                    "name": "test-data-driven-http-rest_01",
                    "documentation": "Put documentation of the Test Case here.",
                    "resultState": "",
                    "steps": [
                        {
                            "uid": "UserID",
                            "name": "http GET /3/movie/49047/",
                            "documentation": null,
                            "type": "com.itko.lisa.ws.rest.RESTNode",
                            "quiet": false,
                            "next": "http GET /3/movie/49163"
                        },
                        {
                            "uid": "UserID",
                            "name": "http GET /3/movie/49163",
                            "documentation": null,
                            "type": "com.itko.lisa.ws.rest.RESTNode",
                            "quiet": false,
                            "next": "http GET /3/movie/49381"
                        },
                        {
                            "uid": "UserID",
                            "name": "http GET /3/movie/49381",
                            "documentation": null,
                            "type": "com.itko.lisa.ws.rest.RESTNode",
                            "quiet": false,
                            "next": "GET empty column value"
                        },
                        {
                            "uid": "UserID",
                            "name": "GET empty column value",
                            "documentation": null,
                            "type": "com.itko.lisa.ws.rest.RESTNode",
                            "quiet": false,
                            "next": "http GET /3/movie/notfound"
                        },
                        {
                            "uid": "UserID",
```

```
                                "name": "http GET /3/movie/notfound",
                                "documentation": null,
                                "type": "com.itko.lisa.ws.rest.RESTNode",
                                "quiet": false,
                                "next": "end"
                            },
                            {
                                "uid": "UserID",
                                "name": "abort",
                                "documentation": null,
                                "type": "com.itko.lisa.test.AbortStep",
                                "quiet": true,
                                "next": ""
                            },
                            {
                                "uid": "UserID",
                                "name": "fail",
                                "documentation": null,
                                "type": "com.itko.lisa.test.Abend",
                                "quiet": true,
                                "next": "abort"
                            },
                            {
                                "uid": "UserID",
                                "name": "end",
                                "documentation": null,
                                "type": "com.itko.lisa.test.NormalEnd",
                                "quiet": true,
                                "next": "fail"
                            }
                        ]
                    },
                    "stagingDocument": {
                        "name": "Run1User10Cycle",
                        "documentation": "",
                        "distributionType": "com.itko.lisa.coordinator.runpatterns.
PercentDistribution",
                        "loadPatternType": "com.itko.lisa.coordinator.runpatterns.
RunNTimesPattern",
                        "think": "100"
                    }
                }
            ]
        }
    }
```


Retrieve a Test Given "Test Run Unique Id" and for Suite Report Given "Suite Run Unique Id"

GET http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api /v1/suites/reports/SuiteRunUniqueID/tests/SuiteRunUniqueID


```
{
    "_links": {
        "next": {
```

```
            "href": "http://localhost:1505/lisa-test-invoke/api/v1/suites/reports
/SuiteRunUniqueID/tests/SuiteRunUniqueID/cycles"
        },
        "self": {
            "href": "http://localhost:1505/lisa-test-invoke/api/v1/suites/reports
/SuiteRunUniqueID/tests/SuiteRunUniqueID"
        }
    },
    "testRunUniqueId": "SuiteRunUniqueID",
    "testRunName": "Run1User10Cycle",
    "testcaseId": "TestCaseID",
    "testcaseName": "test-data-driven-http-rest",
    "testCaseDoc": "Put documentation of the Test Case here.",
    "startTime": "2017-12-20T13:19:06-0500",
    "endTime": "2017-12-20T13:19:53-0500",
    "elapsedTimeInMillSec": 46514,
    "user": "admin",
    "endedState": "PASSED",
    "done": true,
    "manuallyTerminated": false,
    "passCount": 10,
    "failCount": 0,
    "warningCount": 0,
    "abortCount": 0,
    "passPercent": 100,
    "failPercent": 0,
    "abortPercent": 0,
    "modelName": "test-data-driven-http-rest",
    "loadPattern": "Run N Times",
    "host": "localhost",
    "simulatorName": "tcp://localhost:2014/Simulator",
    "coordinatorId": "em2ApF_ybUIQTNajqcZsP_r7rh-6qHabQ-UkOHfSBn0W",
    "coordinatorName": "tcp://localhost:2011/Coordinator",
    "config": "/root/lisatmp_10.2.5/lads/SuiteRunUniqueID/Bank v5/Configs/project.
config",
    "ssvUsers": "1",
    "testcase": {
        "id": "TestCaseID",
        "name": "test-data-driven-http-rest",
        "documentation": "Put documentation of the Test Case here.",
        "resultState": "",
        "steps": [
            {
                "uid": "UserID",
                "name": "http GET /3/movie/49047/",
                "documentation": null,
                "type": "com.itko.lisa.ws.rest.RESTNode",
                "quiet": false,
                "next": "http GET /3/movie/49163"
            },
            {
                "uid": "UserID",
                "name": "http GET /3/movie/49163",
                "documentation": null,
                "type": "com.itko.lisa.ws.rest.RESTNode",
                "quiet": false,
                "next": "http GET /3/movie/49381"
```

```
                },
                {
                    "uid": "UserID",
                    "name": "http GET /3/movie/49381",
                    "documentation": null,
                    "type": "com.itko.lisa.ws.rest.RESTNode",
                    "quiet": false,
                    "next": "GET empty column value"
                },
                {
                    "uid": "UserID",
                    "name": "GET empty column value",
                    "documentation": null,
                    "type": "com.itko.lisa.ws.rest.RESTNode",
                    "quiet": false,
                    "next": "http GET /3/movie/notfound"
                },
                {
                    "uid": "UserID",
                    "name": "http GET /3/movie/notfound",
                    "documentation": null,
                    "type": "com.itko.lisa.ws.rest.RESTNode",
                    "quiet": false,
                    "next": "end"
                },
                {
                    "uid": "UserID",
                    "name": "abort",
                    "documentation": null,
                    "type": "com.itko.lisa.test.AbortStep",
                    "quiet": true,
                    "next": ""
                },
                {
                    "uid": "UserID",
                    "name": "fail",
                    "documentation": null,
                    "type": "com.itko.lisa.test.Abend",
                    "quiet": true,
                    "next": "abort"
                },
                {
                    "uid": "UserID",
                    "name": "end",
                    "documentation": null,
                    "type": "com.itko.lisa.test.NormalEnd",
                    "quiet": true,
                    "next": "fail"
                }
            ]
        },
        "stagingDocument": {
            "name": "Run1User10Cycle",
            "documentation": "",
            "distributionType": "com.itko.lisa.coordinator.runpatterns.
PercentDistribution",
```

```
        "loadPatternType": "com.itko.lisa.coordinator.runpatterns.
RunNTimesPattern",
        "think": "100"
    }
}
```


Retrieve all Cycles for a Test Given "Test Run Unique Id" and for a Suite Report Given "Suite Run Unique Id"

GET http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api /v1/suites/reports/SuiteRunUniqueID/tests/SuiteRunUniqueID/cycles


```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-test-invoke/api/v1/suites/reports
/SuiteRunUniqueID/tests/SuiteRunUniqueID/cycles"
        }
    },
    "count": 10,
    "_embedded": {
        "CycleHistory": [
            {
                "_links": {
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests/SuiteRunUniqueID/cycles/CycleUniqueID"
                    }
                },
                "cycleUniqueId": "CycleUniqueID",
                "cycle": "0",
                "run": "test-data-driven-http-rest [Run1User10Cycle]",
                "simulator": "tcp://localhost:2014/Simulator",
                "startTime": "2017-12-20T13:19:06-0500",
                "endTime": "2017-12-20T13:19:11-0500",
                "elapsedTimeInMillSec": 4443,
                "endedState": "PASSED",
                "_embedded": {
                    "StepExecHistory": [],
                    "preCompanions": [],
                    "postCompanions": [],
                    "messages": []
                }
            },
            {
                "_links": {
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests/SuiteRunUniqueID/cycles/CycleUniqueID"
                    }
                },
                "cycleUniqueId": "CycleUniqueID",
                "cycle": "1",
                "run": "test-data-driven-http-rest [Run1User10Cycle]",
                "simulator": "tcp://localhost:2014/Simulator",
                "startTime": "2017-12-20T13:19:11-0500",
```

```
                "endTime": "2017-12-20T13:19:16-0500",
                "elapsedTimeInMillSec": 4839,
                "endedState": "PASSED",
                "_embedded": {
                    "StepExecHistory": [],
                    "preCompanions": [],
                    "postCompanions": [],
                    "messages": []
                }
            },
            {
                "_links": {
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests/SuiteRunUniqueID/cycles/CycleUniqueID"
                    }
                },
                "cycleUniqueId": "CycleUniqueID",
                "cycle": "2",
                "run": "test-data-driven-http-rest [Run1User10Cycle]",
                "simulator": "tcp://localhost:2014/Simulator",
                "startTime": "2017-12-20T13:19:16-0500",
                "endTime": "2017-12-20T13:19:21-0500",
                "elapsedTimeInMillSec": 4964,
                "endedState": "PASSED",
                "_embedded": {
                    "StepExecHistory": [],
                    "preCompanions": [],
                    "postCompanions": [],
                    "messages": []
                }
            },
            {
                "_links": {
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests/SuiteRunUniqueID/cycles/CycleUniqueID"
                    }
                },
                "cycleUniqueId": "CycleUniqueID",
                "cycle": "3",
                "run": "test-data-driven-http-rest [Run1User10Cycle]",
                "simulator": "tcp://localhost:2014/Simulator",
                "startTime": "2017-12-20T13:19:21-0500",
                "endTime": "2017-12-20T13:19:25-0500",
                "elapsedTimeInMillSec": 4772,
                "endedState": "PASSED",
                "_embedded": {
                    "StepExecHistory": [],
                    "preCompanions": [],
                    "postCompanions": [],
                    "messages": []
                }
            },
            {
                "_links": {
                    "self": {
```

```
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests/SuiteRunUniqueID/cycles/CycleUniqueID"
                    }
                },
                "cycleUniqueId": "CycleUniqueID",
                "cycle": "4",
                "run": "test-data-driven-http-rest [Run1User10Cycle]",
                "simulator": "tcp://localhost:2014/Simulator",
                "startTime": "2017-12-20T13:19:25-0500",
                "endTime": "2017-12-20T13:19:30-0500",
                "elapsedTimeInMillSec": 4679,
                "endedState": "PASSED",
                "_embedded": {
                    "StepExecHistory": [],
                    "preCompanions": [],
                    "postCompanions": [],
                    "messages": []
                }
            },
            {
                "_links": {
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests/SuiteRunUniqueID/cycles/CycleUniqueID"
                    }
                },
                "cycleUniqueId": "CycleUniqueID",
                "cycle": "5",
                "run": "test-data-driven-http-rest [Run1User10Cycle]",
                "simulator": "tcp://localhost:2014/Simulator",
                "startTime": "2017-12-20T13:19:30-0500",
                "endTime": "2017-12-20T13:19:34-0500",
                "elapsedTimeInMillSec": 4124,
                "endedState": "PASSED",
                "_embedded": {
                    "StepExecHistory": [],
                    "preCompanions": [],
                    "postCompanions": [],
                    "messages": []
                }
            },
            {
                "_links": {
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests/SuiteRunUniqueID/cycles/CycleUniqueID"
                    }
                },
                "cycleUniqueId": "CycleUniqueID",
                "cycle": "6",
                "run": "test-data-driven-http-rest [Run1User10Cycle]",
                "simulator": "tcp://localhost:2014/Simulator",
                "startTime": "2017-12-20T13:19:34-0500",
                "endTime": "2017-12-20T13:19:39-0500",
                "elapsedTimeInMillSec": 5105,
                "endedState": "PASSED",
                "_embedded": {
```

```
                    "StepExecHistory": [],
                    "preCompanions": [],
                    "postCompanions": [],
                    "messages": []
                }
            },
            {
                "_links": {
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests/SuiteRunUniqueID/cycles/CycleUniqueID"
                    }
                },
                "cycleUniqueId": "CycleUniqueID",
                "cycle": "7",
                "run": "test-data-driven-http-rest [Run1User10Cycle]",
                "simulator": "tcp://localhost:2014/Simulator",
                "startTime": "2017-12-20T13:19:39-0500",
                "endTime": "2017-12-20T13:19:44-0500",
                "elapsedTimeInMillSec": 4426,
                "endedState": "PASSED",
                "_embedded": {
                    "StepExecHistory": [],
                    "preCompanions": [],
                    "postCompanions": [],
                    "messages": []
                }
            },
            {
                "_links": {
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests/SuiteRunUniqueID/cycles/CycleUniqueID"
                    }
                },
                "cycleUniqueId": "CycleUniqueID",
                "cycle": "8",
                "run": "test-data-driven-http-rest [Run1User10Cycle]",
                "simulator": "tcp://localhost:2014/Simulator",
                "startTime": "2017-12-20T13:19:44-0500",
                "endTime": "2017-12-20T13:19:48-0500",
                "elapsedTimeInMillSec": 4630,
                "endedState": "PASSED",
                "_embedded": {
                    "StepExecHistory": [],
                    "preCompanions": [],
                    "postCompanions": [],
                    "messages": []
                }
            },
            {
                "_links": {
                    "self": {
                        "href": "http://localhost:1505/lisa-test-invoke/api/v1
/suites/reports/SuiteRunUniqueID/tests/SuiteRunUniqueID/cycles/CycleUniqueID"
                    }
                },
```

```
                "cycleUniqueId": "CycleUniqueID",
                "cycle": "9",
                "run": "test-data-driven-http-rest [Run1User10Cycle]",
                "simulator": "tcp://localhost:2014/Simulator",
                "startTime": "2017-12-20T13:19:48-0500",
                "endTime": "2017-12-20T13:19:53-0500",
                "elapsedTimeInMillSec": 4470,
                "endedState": "PASSED",
                "_embedded": {
                    "StepExecHistory": [],
                    "preCompanions": [],
                    "postCompanions": [],
                    "messages": []
                }
            }
        ]
    }
}
```


Retrieve a Given Cycle for "Cycle Unique Id" , for a Test Given "Test Run Unique Id" , and for a Suite Report Given "Suite Run Unique Id"

GET http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api /v1/suites/reports/SuiteRunUniqueID/tests/SuiteRunUniqueID/cycles/CycleUniqueID


```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-test-invoke/api/v1/suites/reports
/SuiteRunUniqueID/tests/SuiteRunUniqueID/cycles/CycleUniqueID"
        }
    },
    "cycleUniqueId": "CycleUniqueID",
    "cycle": "0",
    "run": "test-data-driven-http-rest [Run1User10Cycle]",
    "simulator": "tcp://localhost:2014/Simulator",
    "startTime": "2017-12-20T13:19:06-0500",
    "endTime": "2017-12-20T13:19:11-0500",
    "elapsedTimeInMillSec": 4443,
    "endedState": "PASSED",
    "_embedded": {
        "StepExecHistory": [
            {
                "stepGUID": "TestStepID",
                "stepName": "http GET /3/movie/49047/",
                "type": "STEP",
                "errorCount": "0",
                "warningCount": "0",
                "quite": "false",
                "startTime": "2017-12-20T13:19:06-0500",
                "endedState": "PASSED",
                "stacktrace": null,
                "stepCommand": {
                    "request": "GET /3/movie/49047?
api_key=cd1d7f222b868aec8b843a48baa16df3 HTTP/1.1\nPragma: no-cache\nCache-
Control: no-cache\nAccept: application/json\nlisaFrameRemoteIP: 10.162.27.216
\nlisaFrameRoot: true\nlisaFrameID: 4485ac80-e5b2-11e7-937d-020027e349ee\n\n",
```

```
                    "response": "PG1vdmllPgogICAgPGFkdWx0IHR5cGU9ImJvb2xlYW4iPkZBT
FNFPC9hZHVsdD4KICAgIDxiYWNrZHJvcF9fcGF0aCB0eXBlPSJzdHJpbmciPi9oejNKZkFpa1lYdGFOV0l
KaFdNNHA1c3k1T1ouanBnPC9iYWNrZHJvcF9fcGF0aD4KICAgIDxiZWxvbmdzX190b19fY29sbGVjdGlvb
iBjbGFzcz0ib2JqZWN0IiBudWxsPSJ0cnVlIiAvPgogICAgPGJ1ZGdldCB0eXBlPSJudW1iZXIiPjEwNTA
wMDAwMDwvYnVkZ2V0PgogICAgPGdlbnJlPkRSQU1BPC9nZW5yZT4KICAgIDxob21lcGFnZSB0eXBlPSJzd
HJpbmciPmh0dHA6Ly9ncmF2aXR5bW92aWUud2FybmVyYnJvcy5jb20vPC9ob21lcGFnZT4KICAgIDxpZCB
0eXBlPSJudW1iZXIiPjQ5MDQ3PC9pZD4gICAgCiAgICA8dGl0bGUgdHlwZT0ic3RyaW5nIj5HcmF2aXR5P
C90aXRsZT4KICAgIDxvdmVydmlldyB0eXBlPSJzdHJpbmciPkRyLiBSeWFuIFN0b25lIChTYW5kcmEgQnV
sbG9jayksIGEgYnJpbGxpYW50IG1lZGljYWwgZW5naW5lZXIgb24gaGVyIGZpcnN0IFNodXR0bGUgbWlzc
2lvbiwgd2l0aCB2ZXRlcmFuIGFzdHJvbmF1dCBNYXR0IEtvd2Fsc2t5IChHZW9yZ2UgQ2xvb25leSkgaW4
gY29tbWFuZCBvZiBoaXMgbGFzdCBmbGlnaHQgYmVmb3JlIHJldGlyaW5nLiBCdXQgb24gYSBzZWVtaW5nb
Hkgcm91dGluZSBzcGFjZXdhbGssIGRpc2FzdGVyIHN0cmlrZXMuPC9vdmVydmlldz4KICAgIDxwb3B1bGF
yaXR5IHR5cGU9Im51bWJlciI+NDcuOTA0NTMzPC9wb3B1bGFyaXR5PgogICAgPHBvc3Rlcl9fcGF0aCB0e
XBlPSJzdHJpbmciPi8yZ1BqTFdJeXJXbEFuMkRnS01PS1RCblpZeU8uanBnPC9wb3N0ZXJfX3BhdGg+ICA
gICAgICAgICAgICAgICA8cmV2ZW51ZSB0eXBlPSJudW1iZXIiPjUzNjAwMDAwMDwvcmV2ZW51ZT4KICAgI
DxydW50aW1lIHR5cGU9Im51bWJlciI+OTE8L3J1bnRpbWU+ICAgIAogICAgPHN0YXR1cyB0eXBlPSJzdHJ
pbmciPlJlbGVhc2VkPC9zdGF0dXM+CiAgICA8dGFnbGluZSB0eXBlPSJzdHJpbmciPkRvbid0IExldCBHb
zwvdGFnbGluZT4gICAgCiAgICA8dm90ZV9fYXZlcmFnZSB0eXBlPSJudW1iZXIiPjg8L3ZvdGVfX2F2ZXJ
hZ2U+CiAgICA8dm90ZV9fY291bnQgdHlwZT0ibnVtYmVyIj43MDk8L3ZvdGVfX2NvdW50Pgo8L21vdmllP
g==",
                    "responseBase64Encoded": "true"
                },
                "_embedded": {
                    "messages": [],
                    "asserts": [
                        {
                            "name": "Ensure Result Contains \"Gravity\"",
                            "actualValue": "<movie>\n    <adult type=\"boolean\"
>FALSE</adult>\n    <backdrop__path type=\"string\">/hz3JfAikYXtaNWIJhWM4p5sy5OZ.
jpg</backdrop__path>\n    <belongs__to__collection class=\"object\" null=\"true\"
/>\n    <budget type=\"number\">105000000</budget>\n    <genre>DRAMA</genre>\n
<homepage type=\"string\">http://gravitymovie.warnerbros.com/</homepage>\n    <id
type=\"number\">49047</id>    \n    <title type=\"string\">Gravity</title>\n
<overview type=\"string\">Dr. Ryan Stone (Sandra Bullock), a brilliant medical
engineer on her first Shuttle mission, with veteran astronaut Matt Kowalsky
(George Clooney) in command of his last flight before retiring. But on a
seemingly routine spacewalk, disaster strikes.</overview>\n    <popularity type=\"
number\">47.904533</popularity>\n    <poster__path type=\"string\">
/2gPjLWIyrWlAn2DgKMOKTBnZYyO.jpg</poster__path>            \n    <revenue type=\"
number\">536000000</revenue>\n    <runtime type=\"number\">91</runtime>    \n
<status type=\"string\">Released</status>\n    <tagline type=\"string\">Don't Let
Go</tagline>    \n    <vote__average type=\"number\">8</vote__average>\n
<vote__count type=\"number\">709</vote__count>\n</movie>",
                            "expectation": "Looked for \"Gravity\" in result.",
                            "fired": "false",
                            "isGlobal": "false",
                            "message": "The assertion of type \"Result as String
Contains Given String\" passed evaluation.",
                            "result": "true"
                        }
                    ],
                    "preFilters": [],
                    "postFilters": [],
                    "datasets": [],
                    "events": [],
                    "properties": [],
                    "httpDetails": []
```

```
                }
            },
            {
                "stepGUID": "TestStepID",
                "stepName": "http GET /3/movie/49163",
                "type": "STEP",
                "errorCount": "0",
                "warningCount": "0",
                "quite": "false",
                "startTime": "2017-12-20T13:19:07-0500",
                "endedState": "PASSED",
                "stacktrace": null,
                "stepCommand": {
                    "request": "GET /3/movie/49163?
api_key=cd1d7f222b868aec8b843a48baa16df119 HTTP/1.1\nPragma: no-cache\nCache-
Control: no-cache\nAccept: application/json\nlisaFrameRemoteIP: 10.162.27.216
\nlisaFrameRoot: true\nlisaFrameID: 450e8b90-e5b2-11e7-937d-020027e349ee\n\n",
                    "response": "PG1vdmllPgogICAgPGFkdWx0IHR5cGU9ImJvb2xlYW4iPkZBT
FNFPC9hZHVsdD4KICAgIDxiYWNrZHJvcF9fcGF0aCB0eXBlPSJzdHJpbmciPi9oejNKZkFpa1lYdGFOV0l
KaFdNNHA1c3k1T1ouanBnPC9iYWNrZHJvcF9fcGF0aD4KICAgIDxiZWxvbmdzX190b19fY29sbGVjdGlvb
iBjbGFzcz0ib2JqZWN0IiBudWxsPSJ0cnVlIiAvPgogICAgPGJ1ZGdldCB0eXBlPSJudW1iZXIiPjEwNTA
wMDAwMDwvYnVkZ2V0PgogICAgPGdlbnJlPkFDVElPTjwvZ2VucmU+CiAgICA8aG9tZXBhZ2UgdHlwZT0ic
3RyaW5nIj5tYXRyaXhtb3ZpZS5jb208L2hvbWVwYWdlPgogICAgPGlkIHR5cGU9Im51bWJlciI+NDkxNjM
8L2lkPiAgICAKICAgIDx0aXRsZSB0eXBlPSJzdHJpbmciPlRoZSBNYXRyaXg8L3RpdGxlPgogICAgPG92Z
XJ2aWV3IHR5cGU9InN0cmluZyI+QSBjb21wdXRlciBoYWNrZXIgbGVhcm5zIGZyb20gbXlzdGVyaW91cyB
yZWJlbHMgYWJvdXQgdGhlIHRydWUgbmF0dXJlIG9mIGhpcyByZWFsaXR5IGFuZCBoaXMgcm9sZSBpbiB0a
GUgd2FyIGFnYWluc3QgaXRzIGNvbnRyb2xsZXJzLjwvb3ZlcnZpZXc+CiAgICA8cG9wdWxhcml0eSB0eXB
lPSJudW1iZXIiPjQ0PC9wb3B1bGFyaXR5PgogICAgPHBvc3Rlcl9fcGF0aCB0eXBlPSJzdHJpbmciPi8yZ
1BqTFdJeXJXbEFuMkRnS01PS1RCblpZeU8uanBnPC9wb3N0ZXJfX3BhdGg+ICAgICAgICAgICAgICAgICA
8cmV2ZW51ZSB0eXBlPSJudW1iZXIiPjQ0MDAwMDAwMDA8L3JldmVudWU+CiAgICA8cnVudGltZSB0eXBlP
SJudW1iZXIiPjEyMDwvcnVudGltZT4gICAgCiAgICA8c3RhdHVzIHR5cGU9InN0cmluZyI+UmVsZWFzZWQ
8L3N0YXR1cz4KICAgIDx0YWdsaW5lIHR5cGU9InN0cmluZyI+VGhlIE1hdHJpeDwvdGFnbGluZT4gICAgC
iAgICA8dm90ZV9fYXZlcmFnZSB0eXBlPSJudW1iZXIiPjEwPC92b3RlX19hdmVyYWdlPgogICAgPHZvdGV
fX2NvdW50IHR5cGU9Im51bWJlciI+MTAwMDwvdm90ZV9fY291bnQ+CjwvbW92aWU+",
                    "responseBase64Encoded": "true"
                },
                "_embedded": {
                    "messages": [],
                    "asserts": [
                        {
                            "name": "Ensure Result Contains \"The Matrix\"",
                            "actualValue": "<movie>\n    <adult type=\"boolean\"
>FALSE</adult>\n    <backdrop__path type=\"string\">/hz3JfAikYXtaNWIJhWM4p5sy5OZ.
jpg</backdrop__path>\n    <belongs__to__collection class=\"object\" null=\"true\"
/>\n    <budget type=\"number\">105000000</budget>\n    <genre>ACTION<
/genre>\n    <homepage type=\"string\">matrixmovie.com</homepage>\n    <id type=\"
number\">49163</id>    \n    <title type=\"string\">The Matrix</title>\n
<overview type=\"string\">A computer hacker learns from mysterious rebels about
the true nature of his reality and his role in the war against its controllers.<
/overview>\n    <popularity type=\"number\">44</popularity>\n    <poster__path
type=\"string\">/2gPjLWIyrWlAn2DgKMOKTBnZYyO.jpg</poster__path>            \n
<revenue type=\"number\">4400000000</revenue>\n    <runtime type=\"number\">120<
/runtime>    \n    <status type=\"string\">Released</status>\n    <tagline type=\"
string\">The Matrix</tagline>    \n    <vote__average type=\"number\">10<
/vote__average>\n    <vote__count type=\"number\">1000</vote__count>\n</movie>",
                            "expectation": "Looked for \"The Matrix\" in result.",
                            "fired": "false",
```

```
                            "isGlobal": "false",
                            "message": "The assertion of type \"Result as String
Contains Given String\" passed evaluation.",
                            "result": "true"
                        }
                    ],
                    "preFilters": [],
                    "postFilters": [],
                    "datasets": [],
                    "events": [],
                    "properties": [],
                    "httpDetails": []
                }
            },
            {
                "stepGUID": "TestStepID",
                "stepName": "http GET /3/movie/49381",
                "type": "STEP",
                "errorCount": "0",
                "warningCount": "0",
                "quite": "false",
                "startTime": "2017-12-20T13:19:08-0500",
                "endedState": "PASSED",
                "stacktrace": null,
                "stepCommand": {
                    "request": "GET /3/movie/49381?
api_key=cd1d7f222b868aec8b843a48baa16df337 HTTP/1.1\nPragma: no-cache\nCache-
Control: no-cache\nAccept: application/json\nlisaFrameRemoteIP: 10.162.27.216
\nlisaFrameRoot: true\nlisaFrameID: 45b7eaf0-e5b2-11e7-937d-020027e349ee\n\n",
                    "response": "PG1vdmllPgogICAgPGFkdWx0IHR5cGU9ImJvb2xlYW4iPkZBT
FNFPC9hZHVsdD4KICAgIDxiYWNrZHJvcF9fcGF0aCB0eXBlPSJzdHJpbmciPi9oejNKZkFpa1lYdGFOV0l
KaFdNNHA1c3k1T1ouanBnPC9iYWNrZHJvcF9fcGF0aD4KICAgIDxiZWxvbmdzX190b19fY29sbGVjdGlvb
iBjbGFzcz0ib2JqZWN0IiBudWxsPSJ0cnVlIiAvPgogICAgPGJ1ZGdldCB0eXBlPSJudW1iZXIiPjEwNTA
wMDAwMDwvYnVkZ2V0PgogICAgPGdlbnJlPkZPUkVJR048L2dlbnJlPgogICAgPGhvbWVwYWdlIHR5cGU9I
nN0cmluZyI+c2V2ZW5zYW11cmFpbW92aWUuY29tPC9ob21lcGFnZT4KICAgIDxpZCB0eXBlPSJudW1iZXI
iPjQ5MzgxPC9pZD4gICAgCiAgICA8dGl0bGUgdHlwZT0ic3RyaW5nIj5TZXZlbiBTYW11cmFpPC90aXRsZ
T4KICAgIDxvdmVydmlldyB0eXBlPSJzdHJpbmciPkEgcG9vciB2aWxsYWdlIHVuZGVyIGF0dGFjayBieSB
iYW5kaXRzIHJlY3J1aXRzIHNldmVuIHVuZW1wbG95ZWQgc2FtdXJhaSB0byBoZWxwIHRoZW0gZGVmZW5kI
HRoZW1zZWx2ZXMuPC9vdmVydmlldz4KICAgIDxwb3B1bGFyaXR5IHR5cGU9Im51bWJlciI+NTA8L3BvcHV
sYXJpdHk+CiAgICA8cG9zdGVyX19wYXRoIHR5cGU9InN0cmluZyI+LzJnUGpMV0l5cldsQW4yRGdLTU9LV
EJuWll5Ty5qcGc8L3Bvc3Rlcl9fcGF0aD4gICAgICAgICAgICAgICAgICAKICAgIDxyZXZlbnVlIHR5cGU9Im51bWJ
lciI+MzM2MDAwMDwvcmV2ZW51ZT4KICAgIDxydW50aW1lIHR5cGU9Im51bWJlciI+OTk8L3J1bnRpbWU+I
CAgIAogICAgPHN0YXR1cyB0eXBlPSJzdHJpbmciPlJlbGVhc2VkPC9zdGF0dXM+CiAgICA8dGFnbGluZSB
0eXBlPSJzdHJpbmciPlNldmVuIFNhbXVyYWk8L3RhZ2xpbmU+ICAgIAogICAgPHZvdGVfX2F2ZXJhZ2Ugd
HlwZT0ibnVtYmVyIj45PC92b3RlX19hdmVyYWdlPgogICAgPHZvdGVfX2NvdW50IHR5cGU9Im51bWJlciI
+ODAwPC92b3RlX19jb3VudD4KPC9tb3ZpZT4=",
                    "responseBase64Encoded": "true"
                },
                "_embedded": {
                    "messages": [],
                    "asserts": [
                        {
                            "name": "Ensure Result Contains \"Seven Samurai\"",
                            "actualValue": "<movie>\n    <adult type=\"boolean\"
>FALSE</adult>\n    <backdrop__path type=\"string\">/hz3JfAikYXtaNWIJhWM4p5sy5OZ.
jpg</backdrop__path>\n    <belongs__to__collection class=\"object\" null=\"true\"
/>\n    <budget type=\"number\">105000000</budget>\n    <genre>FOREIGN<
```

/genre>\n   <homepage type=\"string\">sevensamuraimovie.com</homepage>\n   <id type=\"number\">49381</id>   \n   <title type=\"string\">Seven Samurai</title>\n   <overview type=\"string\">A poor village under attack by bandits recruits seven unemployed samurai to help them defend themselves.</overview>\n <popularity type=\"number\">50</popularity>\n   <poster__path type=\"string\">/2gPjLWIyrWlAn2DgKMOKTBnZYyO.jpg</poster__path>   \n   <revenue type=\"number\">3360000</revenue>\n   <runtime type=\"number\">99</runtime>   \n <status type=\"string\">Released</status>\n   <tagline type=\"string\">Seven Samurai</tagline>   \n   <vote__average type=\"number\">9</vote__average>\n <vote__count type=\"number\">800</vote__count>\n</movie>",
                            "expectation": "Looked for \"Seven Samurai\" in result.",
                            "fired": "false",
                            "isGlobal": "false",
                            "message": "The assertion of type \"Result as String Contains Given String\" passed evaluation.",
                            "result": "true"
                        }
                    ],
                    "preFilters": [],
                    "postFilters": [],
                    "datasets": [],
                    "events": [],
                    "properties": [],
                    "httpDetails": []
                }
            },
            {
                "stepGUID": "TestStepID",
                "stepName": "GET empty column value",
                "type": "STEP",
                "errorCount": "0",
                "warningCount": "0",
                "quite": "false",
                "startTime": "2017-12-20T13:19:09-0500",
                "endedState": "PASSED",
                "stacktrace": null,
                "stepCommand": {
                    "request": "GET /3/movie/49382?api_key=cd1d7f222b868aec8b843a48baa16df337 HTTP/1.1\nPragma: no-cache\nCache-Control: no-cache\nAccept: application/json\nlisaFrameRemoteIP: 10.162.27.216\nlisaFrameRoot: true\nlisaFrameID: 46444c70-e5b2-11e7-937d-020027e349ee\n\n",
                    "response": "PG1vdmllPgogICAgPGFkdWx0IHR5cGU9ImJvb2xlYW4iPkZBTFNFPC9hZHVsdD4KICAgIDxiYWNrZHJvcF9fcGF0aCB0eXBlPSJzdHJpbmciPi9oejNKZkFpa1lYdGFOV0lKaFdNNHA1c3k1T1ouanBnPC9iYWNrZHJvcF9fcGF0aD4KICAgIDxiZWxvbmdzX190b19fY29sbGVjdGlvbiBjbGFzcz0ib2JqZWN0IiBudWxsPSJ0cnVlIiAvPgogICAgPGJ1ZGdldCB0eXBlPSJudW1iZXIiPjEwNTAwMDAwMDwvYnVkZ2V0PgogICAgPGdlbnJlPjwvZ2VucmU+CiAgICA8aG9tZXBhZ2UgdHlwZT0ic3RyaW5nIj5Vc3VhbCBTdXNwZWN0czwvaG9tZXBhZ2U+CiAgICA8aWQgdHlwZT0ibnVtYmVyIj40OTM4MjwvaWQ+ICAgIAogICAgPHRpdGxlIHR5cGU9InN0cmluZyI+VXN1YWwgU3VzcGVjdHM8L3RpdGxlPgogICAgPG92ZXJ2aWV3IHR5cGU9InN0cmluZyI+Q3JpbWUgbW92aWU8L292ZXJ2aWV3PgogICAgPHBvcHVsYXJpdHkgdHlwZT0ibnVtYmVyIj41MDwvcG9wdWxhcml0eT4KICAgIDxwb3N0ZXJfX3BhdGggdHlwZT0ic3RyaW5nIj4vMmdQakxXSXlyV2xBbjJEZ0tNT0tUQm5aWXlPLmpwZzwvcG9zdGVyX19wYXRoPiAgICAgICAgICAgICAgIAogICAgPHJldmVudWUgdHlwZT0ibnVtYmVyIj4zMzYwMDAwPC9yZXZlbnVlPgogICAgPHJ1bnRpbWUgdHlwZT0ibnVtYmVyIj45OTwvcnVudGltZT4gICAgCiAgICA8c3RhdHVzIHR5cGU9InN0cmluZyI+UmVsZWFzZWQ8L3N0YXR1cz4KICAgIDx0YWdsaW5lIHR5cGU9InN0cmluZyI+VXN1YWwgU3VzcGVjdHM8L3RhZ2xpbmU+ICAgIAogICAgPHZvdGVfX2F2ZXJhZ2UgdHlwZT0ibnVtYmVyIj4xMDwvdm90ZV9fYXZlcmFnZT4KICAgIDx2b3RlX19jb3VudCB0eXBlPSJudW1iZXIiPjEwMDA8L3ZvdGVfX2NvdW50Pgo8L21vdmllPg==",

```
                    "responseBase64Encoded": "true"
                },
                "_embedded": {
                    "messages": [],
                    "asserts": [
                        {
                            "name": "Ensure Result Contains empty genre tag",
                            "actualValue": "<movie>\n    <adult type=\"boolean\"
>FALSE</adult>\n    <backdrop__path type=\"string\">/hz3JfAikYXtaNWIJhWM4p5sy5OZ.
jpg</backdrop__path>\n    <belongs__to__collection class=\"object\" null=\"true\"
/>\n    <budget type=\"number\">105000000</budget>\n    <genre></genre>\n
<homepage type=\"string\">Usual Suspects</homepage>\n    <id type=\"number\"
>49382</id>    \n    <title type=\"string\">Usual Suspects</title>\n    <overview
type=\"string\">Crime movie</overview>\n    <popularity type=\"number\">50<
/popularity>\n    <poster__path type=\"string\">/2gPjLWIyrWlAn2DgKMOKTBnZYyO.jpg<
/poster__path>            \n    <revenue type=\"number\">3360000</revenue>\n
<runtime type=\"number\">99</runtime>    \n    <status type=\"string\">Released<
/status>\n    <tagline type=\"string\">Usual Suspects</tagline>    \n
<vote__average type=\"number\">10</vote__average>\n    <vote__count type=\"
number\">1000</vote__count>\n</movie>",
                            "expectation": "Looked for \"<genre></genre>\" in
result.",
                            "fired": "false",
                            "isGlobal": "false",
                            "message": "The assertion of type \"Result as String
Contains Given String\" passed evaluation.",
                            "result": "true"
                        }
                    ],
                    "preFilters": [],
                    "postFilters": [],
                    "datasets": [],
                    "events": [],
                    "properties": [],
                    "httpDetails": []
                }
            },
            {
                "stepGUID": "TestStepID",
                "stepName": "http GET /3/movie/notfound",
                "type": "STEP",
                "errorCount": "0",
                "warningCount": "0",
                "quite": "false",
                "startTime": "2017-12-20T13:19:10-0500",
                "endedState": "PASSED",
                "stacktrace": null,
                "stepCommand": {
                    "request": "GET /3/movie/notfound?
api_key=cd1d7f222b868aec8b843a48baa16df337 HTTP/1.1\nPragma: no-cache\nCache-
Control: no-cache\nAccept: application/json\nlisaFrameRemoteIP: 10.162.27.216
\nlisaFrameRoot: true\nlisaFrameID: 46ad4770-e5b2-11e7-937d-020027e349ee\n\n",
                    "response": "PGh0bWw+CiAgICA8aGVhZD48dGl0bGU+NDA0IE5vdCBGb3VuZ
DwvdGl0bGU+PC9oZWFkPgogICAgPGJvZHk+CiAgICAgICAgPGgxPk5vdCBGb3VuZDwvaDE+CiAgICAgICA
gPHA+VGhlIHJlcXVlc3RlZCBVUkwgd2FzIG5vdCBmb3VuZCBvbiB0aGlzIHNlcnZlci48L3A+CiAgICAgI
CAgPGhyLz4KICAgICAgICA8cD48aT5UaGUgTEllTQSBWU0Ugc2VydmljZSBjb3VsZCBub3QgbWF0Y2ggeW9
1ciByZXF1ZXN0IHRvIGEgcmVjb3JkZWQgcmVxdWVzdC4mbmJzcDsgQ29uc2lkZXIgZXhwYW5kaW5nIHlvd
```

```
XIgc2VydmljZSBpbWFnZS48L2k+PC9wPgogICAgICAgIDxici8+PGZvbnQgc2l6ZT0iLTIiPlByb2R1Y2V
kIGJ5IGEgTElTQSB2aXJ0dWFsaXplZCB3ZWIgc2VydmVyLjwvZm9udD4KICAgIDwvYm9keT4KPC9odG1sP
go=",
                    "responseBase64Encoded": "true"
                },
                "_embedded": {
                    "messages": [],
                    "asserts": [
                        {
                            "name": "Ensure Result Contains \"404 Not Found\"",
                            "actualValue": "<html>\n    <head><title>404 Not
Found</title></head>\n    <body>\n        <h1>Not Found</h1>\n        <p>The
requested URL was not found on this server.</p>\n        <hr/>\n        <p><i>The
LISA VSE service could not match your request to a recorded request. 
Consider expanding your service image.</i></p>\n        <br/><font size=\"-2\"
>Produced by a LISA virtualized web server.</font>\n    </body>\n</html>\n",
                            "expectation": "Looked for \"404 Not Found\" in
result.",
                            "fired": "false",
                            "isGlobal": "false",
                            "message": "The assertion of type \"Result as String
Contains Given String\" passed evaluation.",
                            "result": "true"
                        }
                    ],
                    "preFilters": [],
                    "postFilters": [],
                    "datasets": [],
                    "events": [],
                    "properties": [],
                    "httpDetails": []
                }
            }
        ],
        "preCompanions": [],
        "postCompanions": [],
        "messages": []
    }
}
```


GET http:// {{LISA_TEST_INVOKE_HOST}} : {{LISA_TEST_INVOKE_PORT}} /lisa-test-invoke/api /v1/suites/reports/summary ?startTime=2017-12-20T00:00:00.000-05:00&endTime=2017-12-20T16:33:00.000-05:00

The startTime and endTime values are optional. If present, they must comply to ISO 8601 format; for example, *2014-10-22T16:33:00.000-05:00.*

Response:


```
{
  "_links": {
    "self": {
      "href": "http://localhost:1505/lisa-test-invoke/api/v1/suites/reports
/summary"
    }
  },
  "passCount": 5,
```

```
  "failCount": 0,
  "warningCount": 0,
  "abortCount": 4,
  "errorCount": 40,
  "runningCount": 0
}
```

Virtual Service Invoke API

This document describes the functionality to create and manage a virtual service using the **Virtual Service Invoke** RESTful API.

- Overview
  - Requirements
  - Endpoint
  - Authentication
  - Reference
  - Parameters
  - Glossary
  - Response Codes
- Create a Virtual Service By Recording
  - Get the VSE Id
  - Create a Builder Session
  - Create a Transport Protocol
  - Add a Data Protocol (Optional)
  - Add Transactions
  - Manage Transactions
  - Create the Virtual Service
- Add Transactions to an Existing Virtual Service Through Recording
  - Get the VSE Id
  - Create a Builder Session from a MAR (Existing VS)
  - View the Transport Protocol or Transport Protocols
  - View the Data Protocol or Data Protocols (Optional)
  - Add Transactions
  - Manage Transactions
  - Create the Virtual Service
- Create a Virtual Service from RR Pairs or a Specification Document (WSDL)
  - Get the VSE Id
  - Create a Builder Session
  - Create a Transport Protocol
  - Add a Data Protocol (Optional)
  - Add transactions
  - Manage Transactions
  - Create the virtual service
- Add Transactions to an Existing Virtual Service from RR Pairs or a Specification Document (WSDL)
  - Get the VSE Id
  - Create a Builder Session from a MAR (Existing VS)

    - View the Transport Protocol or Transport Protocols
    - View the Data Protocol or Data Protocols (Optional)
    - Add Transactions
    - Manage Transactions
    - Create the Virtual Service
  - Export a Virtual Service as MAR File
    - Get the VSE Id
    - Get the Service Id
    - Download MAR file

Overview

Requirements

- The DevTest registry must be running.
- A Virtual Service Environment (VSE) must be running.
- API Authentication must use the DevTest user credentials.

Endpoint

The root URL for this API is:

```
http://localhost:1505/lisa-virtualize-invoke/api/v2/
```

Authentication

This API requires an authentication using the DevTest credentials. Use the Authorization request header for authentication. The following is an example using curl:

```
curl -X GET
-H "Authorization: Basic XXXXXXXXXXXXX"
-H "Cache-Control: no-cache"
"http://localhost:1505/lisa-virtualize-invoke/api/v2/vses"
```

Reference

For more reference, a SWAGGER framework is set up in the following URLs for documentation and testing purposes:

- Swagger UI: http://localhost:1505/lisa-virtualize-invoke/api/v2/swagger-ui
- Swagger JSON: http://localhost:1505/lisa-virtualize-invoke/api/v2/swagger-json

Parameters

The following table is a list of parameters that are used in this document:

Parameter	Description
{vseId}	The VSE server id
{sessionId}	A builder session id
{tpId}	A transport protocol id
{dpId}	A data protocol id

Parameter	Description
{recorderId}	The recorder id
{bundleId}	A transaction bundle id
{inputContainerId}	An input container id
{contentId}	A content id
{processorId}	An input processor id
{transactionId}	A transaction id
{procId}	A bundle processor id
{serviceId}	A virtual service id

Glossary

VS
Virtual Service

VSE

Virtual Service Environment

Builder Session

Contains all the information necessary for creating or editing a VS, such as its configuration (transport protocol) and content (transactions).

Transport Protocol

Holds all the configuration for a Builder Session, most importantly the type of protocol, the recording, and target endpoints.

Input Container

A server-side container that holds uploaded artifacts (RR pairs, spec documents, for example) until they can be processed to produce transactions.

Input Processor

Responsible for processing the contents of the Input Container to produce transactions out of the provided artifacts.

Response Codes

The following table is a list of response codes that this API returns:

Code	Description
200	Successful
201	Created (success)
202	Accepted (success)
204	No Content (success)
400	Bad request (invalid input data)
404	Not Found

Code	Description
500	Server Error

Create a Virtual Service By Recording

Get the VSE Id

Request:

```
GET /vses
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m"
        }
    }
}
```

A successful response returns an array of VSE Server JSON objects. Choose the VSE to use and note the VSE id for it, which is found in the 'self' link. Using the previous example, the **{vseId}** is **j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m**.

Create a Builder Session

Request:

```
POST /vses/{vseId}/vsBuilderSessions

{
    "name": "test001",
    "description": "Optional description"
}
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/vsBuilderSessions
/efrsjq_mqVTNwXYsmQ9vxE1dYTwWNy07FD8ioKuZmVNugsgWHAU"
        }
    },
    "id": "efrsjq_mqVTNwXYsmQ9vxE1dYTwWNy07FD8ioKuZmVNugsgWHAU",
    "modifyDate": "2016-12-20T14:34:21-06:00",
    "defaultNonLeafTolerance": "WIDE",
    "name": "test001",
    "description": "Optional description",
    "defaultLeafTolerance": "LOOSE",
    "version": "2",
    "createDate": "2016-12-20T14:34:21-06:00"
```

```
}
```

The request body should be a JSON object containing a new VS builder session. A successful response returns a JSON object that represents the newly created builder session object. Using the previous example, the **{sessionId}** is **efrsjq_mqVTNwXYsmQ9vxE1dYTwWNy07FD8ioKuZmVNugsgWHAU**.

Create a Transport Protocol

Request:

```
POST /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols

{
    "typeId": "HTTP",
    "basePath": "/",
    "useGateway": true,
    "hostHeaderPassThrough": false,
    "recordingEndpoint": {
        "host": "localhost",
        "port": "8001"
    },
    "targetEndpoint": {
        "host": "localhost",
        "port": "8080"
    }
}
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/vsBuilderSessions
/efrsjq_mqVTNwXYsmQ9vxE1dYTwWNy07FD8ioKuZmVNugsgWHAU/transportProtocols
/bcQqa1IHnc0b_c2g7JkqO-bsAX_6h5sSnYn0VTbLbu8fAggefp"
        }
    },
    "id": "bcQqa1IHnc0b_c2g7JkqO-bsAX_6h5sSnYn0VTbLbu8fAggefp",
    "basePath": "/",
    "asObject": false,
    "useGateway": true,
    "hostHeaderPassThrough": false,
    "allAreStateless": false,
    "targetEndpoint": {
        "port": "8080",
        "host": "localhost"
    },
    "typeId": "HTTP",
    "desensitize": false,
    "duptxns": false,
    "recordingEndpoint": {
        "port": "8001",
        "host": "localhost"
    }
```

```
}
```

A successful response returns a JSON containing the new transport protocol. As of Release 10.0, multiple transport protocols can be created but only the first one is used by the recorder. Using the previous example, the **{tpId}** is **bcQqa1IHnc0b_c2g7JkqO-bsAX_6h5sSnYn0VTbLbu8fAggefp**.

Add a Data Protocol (Optional)

Request:

```
POST /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/dataProtocols

{
    "forRequest": true,
    "typeId": "SOAPDPH"
}
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/vsBuilderSessions
/efrsjq_mqVTNwXYsmQ9vxE1dYTwWNy07FD8ioKuZmVNugsgWHAU/transportProtocols
/bcQqa1IHnc0b_c2g7JkqO-bsAX_6h5sSnYn0VTbLbu8fAggefp/dataProtocols
/efiLyVgPDlffNNXvPXbpdKE6yM1dCuU5qJJPtxrJKCmLAUbYZJn"
        }
    },
    "id": "efiLyVgPDlffNNXvPXbpdKE6yM1dCuU5qJJPtxrJKCmLAUbYZJn",
    "forRequest": true,
    "typeId": "SOAPDPH"
}
```

This creates a data protocol for a specific builder session and transport protocol.

- Use SOAPDPH for a Web Services (SOAP) data protocol.
- Use JSONDPH for a JSON 2.0 data protocol.
- Use XMLDPH for an XML data protocol.

Using the previous example, the **{dpId}** is **efiLyVgPDlffNNXvPXbpdKE6yM1dCuU5qJJPtxrJKCmLAUbYZJn**.

Null Data Protocol:

```
POST /vses/{vseId}/vsBuilderSessions/{sessionId}
/transportProtocols/{tpId}/dataProtocols { "typeId": "NULLDPH"}
```

The null data protocol unctions the same as the data protocol.
Adding this to a specified transport protocol prevents content detection (see Content Detection).

Add Transactions

1. Start the recorder
   Request:

```
POST /vses/{vseId}/recorders

{
  "sessionId": "{sessionId}"
}
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/recorders
/bgwfXptUuOfsyVfJCRZ67db-5JkJGtaDuAnqDIrXjdLU80dWgk"
        }
    },
    "id": "bgwfXptUuOfsyVfJCRZ67db-5JkJGtaDuAnqDIrXjdLU80dWgk",
    "transactionCount": 0,
    "sessionCount": 0
}
```

The recorder is now recording.
Using the example in Create a Transport Protocol, the recorder is listening on http://localhost:8001 (the recording endpoint host and port) and forwarding the requests to http://localhost:8080 (the target endpoint host and port). Using the previous example, the **{recorderId}** is **bgwfXptUuOfsyVfJCRZ67db-5JkJGtaDuAnqDIrXjdLU80dWgk**.

2. Open the recording endpoint to record transactions

3, Stop the recorder

Request:

```
DELETE /vses/{vseId}/recorders/{recorderId}
```

4. Content Detection

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}
/transportProtocols/{tpId}/dataProtocols
```

If you did not manually add a data protocol to the transport protocol, when the recording is stopped, any DPH detected by content detection during recording is added to the transport protocol. The API specified above can be used to view any data protocols that were detected and added. To disable content detection, see Null Data Protocol.

Manage Transactions

Manage Transactions (Without Data Protocols Applied)

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/transactions
```

The response returns the first 100 transactions by default. You can add offset and limit parameters to the URL to get more transactions at a time or to do paging. For example: **? offset=0&limit=15**0. To manage a specific transaction, note the unique transaction id for each transaction.

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/transactions/{transactionId}
```

Request:

```
DELETE /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/transactions/{transactionId}
```

Manage Transactions (With Data Protocols Applied)

The following APIs return the same transactions as before (the raw versions) but they have been parsed by the data protocols applied up to and including the data protocol specified by the **{dpId}** . If you had three data protocols and you specified the id for the second one, the transactions appear with only the first and second data protocols applied.

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/dataProtocols/{dpId}/transactions
```

The response returns the first 100 transactions by default. You can add offset and limit parameters to the URL to get more transactions at a time or to do paging. For example: **? offset=0&limit=150.** To manage a specific transaction, note the unique transaction id for each transaction.

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/dataProtocols/{dpId}/transactions/{transactionId}
```

Create the Virtual Service

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}
Accept: application/zip
```

Response: The response is a ZIP file that contains the virtual service files.

Add Transactions to an Existing Virtual Service Through Recording

Get the VSE Id

Request:

```
GET /vses
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m"
        }
    }
}
```

A successful response returns an array of VSE Server JSON objects. Choose the VSE to use and note the VSE id for it, which is found in the 'self' link. Using the previous example, the **{vseId}** is **j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m**.

Create a Builder Session from a MAR (Existing VS)

Request:

```
POST /vses/{vseId}/vsBuilderSessions
Content-Type: application/zip

<Body: MAR file you want to upload>
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/vsBuilderSessions
/efrsjq_mqVTNwXYsmQ9vxE1dYTwWNy07FD8ioKuZmVNugsgWHAU"
        }
    },
    "id": "efrsjq_mqVTNwXYsmQ9vxE1dYTwWNy07FD8ioKuZmVNugsgWHAU",
    "modifyDate": "2016-12-20T14:34:21-06:00",
    "defaultNonLeafTolerance": "WIDE",
    "name": "test001",
    "description": "Optional description",
    "defaultLeafTolerance": "LOOSE",
    "version": "2",
    "createDate": "2016-12-20T14:34:21-06:00"
}
```

The request should have a VS MAR file with content type set to application/zip. A successful response returns a JSON object that represents the newly created builder session object. Using the previous example, the **{sessionId}** is **efrsjq_mqVTNwXYsmQ9vxE1dYTwWNy07FD8ioKuZmVNugsgWHAU**.

View the Transport Protocol or Transport Protocols

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols
```

A successful response returns a JSON containing a list of transport protocols. Using the id property from one of the transport protocols, as the {tpId} for the following API calls.
As of Release 10.0, multiple transport protocols can be created but only the first one is used by the recorder. To add a new data protocol, see Add a Data Protocol (Optional).

View the Data Protocol or Data Protocols (Optional)

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/dataProtocols
```

A successful response returns a JSON containing a list of data protocols. Using the id property from one of the transport protocols, as the { dpId } for the following API calls.
To add a new data protocol, see Add a Data Protocol (Optional).

Add Transactions

Start the Recorder

Request:

```
POST /vses/{vseId}/recorders

{
  "sessionId": "{sessionId}"
}
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/recorders
/bgwfXptUuOfsyVfJCRZ67db-5JkJGtaDuAnqDIrXjdLU80dWgk"
        }
    },
    "id": "bgwfXptUuOfsyVfJCRZ67db-5JkJGtaDuAnqDIrXjdLU80dWgk",
    "transactionCount": 0,
    "sessionCount": 0
}
```

The recorder is now recording. Using the example in Create a Transport Protocol, the recorder is listening on http://localhost:8001 (the recording endpoint host and port) and forwarding the requests to http://localhost:8080 (the target endpoint host and port).
Using the example above, the **{recorderId}** is **bgwfXptUuOfsyVfJCRZ67db-5JkJGtaDuAnqDIrXjdLU80dWgk**.

Open the Recording Endpoint to Record Transactions

Stop the Recorder

Request:

```
DELETE /vses/{vseId}/recorders/{recorderId}
```

Manage Transactions

Manage Transactions (Without Data Protocols Applied)

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/transactions
```

The response returns the first 100 transactions by default. You can add offset and limit parameters to the URL to get more transactions at a time or to do paging. For example: **? offset=0&limit=150** To manage a specific transaction, note the unique transaction id for each transaction.

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/transactions/{transactionId}
```

Request:

```
DELETE /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/transactions/{transactionId}
```

Manage Transactions (With Data Protocols Applied)

The following APIs return the same transactions as before (the raw versions) but they have been parsed by the data protocols applied up to and including the data protocol specified by the **{dpId}** . If you had three data protocols and you specified the id for the second one, the transactions appear with only the first and second data protocols applied.

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/dataProtocols/{dpId}/transactions
```

The response returns the first 100 transactions by default. You can add offset and limit parameters to the URL to get more transactions at a time or to do paging. For example: **? offset=0&limit=150.** To manage a specific transaction, note the unique transaction id for each transaction.

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/dataProtocols/{dpId}/transactions/{transactionId}
```

Create the Virtual Service

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}
```

```
Accept: application/zip
```

Response: The response is a ZIP file that contains the virtual service files.

Create a Virtual Service from RR Pairs or a Specification Document (WSDL)

Get the VSE Id

Request:

```
GET /vses
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m"
        }
    }
}
```

A successful response returns an array of VSE Server JSON objects. Choose the VSE to use and note the VSE id for it, which is found in the 'self' link. Using the previous example, the **{vseId}** is **j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m**.

Create a Builder Session

Request:

```
POST /vses/{vseId}/vsBuilderSessions

{
    "name": "test001",
    "description": "Optional description"
}
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/vsBuilderSessions
/efrsjq_mqVTNwXYsmQ9vxE1dYTwWNy07FD8ioKuZmVNugsgWHAU"
        }
    },
    "id": "efrsjq_mqVTNwXYsmQ9vxE1dYTwWNy07FD8ioKuZmVNugsgWHAU",
    "modifyDate": "2016-12-20T14:34:21-06:00",
    "defaultNonLeafTolerance": "WIDE",
    "name": "test001",
    "description": "Optional description",
    "defaultLeafTolerance": "LOOSE",
    "version": "2",
    "createDate": "2016-12-20T14:34:21-06:00"
```

```
}
```

The request body should be a JSON object containing a new VS builder session. A successful response returns a JSON object that represents the newly created builder session object. Using the previous example, the **{sessionId}** is **efrsjq_mqVTNwXYsmQ9vxE1dYTwWNy07FD8ioKuZmVNugsgWHAU**.

Create a Transport Protocol

Request:

```
POST /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols

{
    "typeId": "HTTP",
    "basePath": "/",
    "useGateway": true,
    "hostHeaderPassThrough": false,
    "recordingEndpoint": {
        "host": "localhost",
        "port": "8001"
    },
    "targetEndpoint": {
        "host": "localhost",
        "port": "8080"
    }
}
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/vsBuilderSessions
/efrsjq_mqVTNwXYsmQ9vxE1dYTwWNy07FD8ioKuZmVNugsgWHAU/transportProtocols
/bcQqa1IHnc0b_c2g7JkqO-bsAX_6h5sSnYn0VTbLbu8fAggefp"
        }
    },
    "id": "bcQqa1IHnc0b_c2g7JkqO-bsAX_6h5sSnYn0VTbLbu8fAggefp",
    "basePath": "/",
    "asObject": false,
    "useGateway": true,
    "hostHeaderPassThrough": false,
    "allAreStateless": false,
    "targetEndpoint": {
        "port": "8080",
        "host": "localhost"
    },
    "typeId": "HTTP",
    "desensitize": false,
    "duptxns": false,
    "recordingEndpoint": {
        "port": "8001",
        "host": "localhost"
    }
}
```

A successful response returns a JSON containing the new transport protocol. As of Release 10.0, multiple transport protocols can be created but only the first one is used by the recorder. Using the previous example, the **{tpId}** is **bcQqa1IHnc0b_c2g7JkqO-bsAX_6h5sSnYn0VTbLbu8fAggefp**.

Add a Data Protocol (Optional)

Request:

```
POST /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/dataProtocols

{
    "forRequest": true,
    "typeId": "SOAPDPH"
}
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/vsBuilderSessions
/efrsjq_mqVTNwXYsmQ9vxE1dYTwWNy07FD8ioKuZmVNugsgWHAU/transportProtocols
/bcQqa1IHnc0b_c2g7JkqO-bsAX_6h5sSnYn0VTbLbu8fAggefp/dataProtocols
/efiLyVgPDlffNNXvPXbpdKE6yM1dCuU5qJJPtxrJKCmLAUbYZJn"
        }
    },
    "id": "efiLyVgPDlffNNXvPXbpdKE6yM1dCuU5qJJPtxrJKCmLAUbYZJn",
    "forRequest": true,
    "typeId": "SOAPDPH"
}
```

Creates a data protocol for a specific builder session and transport protocol.

- Use SOAPDPH for a Web Services (SOAP) data protocol.
- Use JSONDPH for a JSON 2.0 data protocol.
- Use XMLDPH for a XML data protocol.

Using the previous example, the **{dpId}** is **efiLyVgPDlffNNXvPXbpdKE6yM1dCuU5qJJPtxrJKCmLAUbYZJn**.

Add transactions

Create a Transaction Bundle

Request:

```
POST /vses/{vseId}/bundles

{}
```

Response:

```
{
```

```
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/bundles
/1RKLr1NXz758V7NFvddEBB"
        }
    },
    "id": "1RKLr1NXz758V7NFvddEBB",
    "_embedded": {
        "transactions": []
    }
}
```

Using the previous example, the **{bundleId}** is **1RKLr1NXz758V7NFvddEBB.**

Create an Input Container

An input container is a server-side container that holds uploaded artifacts (RR pairs and spec documents, for example) until they can be processed to produce transactions.

Request:

```
POST /vses/{vseId}/bundles/{bundleId}/inputContainers

{}
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/bundles
/1RKLr1NXz758V7NFvddEBB/inputContainers/IC1Za52IBxr39PIwEUIlBt8m"
        }
    },
    "id": "IC1Za52IBxr39PIwEUIlBt8m",
    "_embedded": {
        "transSources": []
    }
}
```

Using the previous example, the **{inputContainerId}** is **IC1Za52IBxr39PIwEUIlBt8m**.

Upload Data to an Input Container

- Upload RR pair files to an input container.

Request:

```
POST /vses/{vseId}/bundles/{bundleId}/inputContainers/{inputContainerId}/contents
Content-Type: multipart/form-data

file1=<the file you want to upload>
```

```
file2=<the file you want to upload>
```

Request example (using cURL):

```
curl -X POST
-H "Authorization: Basic XXXXXXXXXXXXX"
-H "Content-Type: multipart/form-data;"
-F "file1=@/Applications/CA/DevTest/Projects/Bank v5/Data/rrpair/PayAction-req.
txt"
-F "file2=@/Applications/CA/DevTest/Projects/Bank v5/Data/rrpair/PayAction-rsp.
txt"
"http://localhost:1505/lisa-virtualize-invoke/api/v2/vses/j4xHDBZf8x3-
9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/bundles/1RKLr1NXz758V7NFvddEBB
/inputContainers/IC1Za52IBxr39PIwEUIlBt8m/contents"
```

The previous example uses RR pairs defined in the sample Bank v5 Project. The new **{contentId}** is the "id" field in the response.

- Upload a specification document (WSDL) to an input container.

Request:

```
POST /vses/{vseId}/bundles/{bundleId}/inputContainers/{inputContainerId}/contents
Accept: application/zip

{
    "url":"http://localhost:8080/itkoExamples/EJB3UserControlBean?wsdl"
}
```

The previous example requires the DevTest DemoServer (LISA Bank) to be running. Note the {contentId} in the response.

Process Data Using the Input Processor

An “Input Processor” is responsible for processing the contents of the "Input Container" to produce transactions out of the provided artifacts.

- Start an input processor.

Request:

```
POST /vses/{vseId}/bundles/{bundleId}/inputProcessors

{
    "container": "{inputContainerId}"
}
```

Response:

```
{
    "_links": {
        "self": {
```

```
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/bundles
/1RKLr1NXz758V7NFvddEBB/inputProcessors/IPrp57Jc9v2n9P7yimix5I2"
        }
    },
    "id": "IPrp57Jc9v2n9P7yimix5I2"
}
```


Using the previous example, the **{processorId}** is **IPrp57Jc9v2n9P7yimix5I2**.

- Check status of an input processor.

Request:

```
GET /vses/{vseId}/bundles/{bundleId}/inputProcessors/{processorId}
```

- Stop an input processor (when done).

Request:

```
DELETE /vses/{vseId}/bundles/{bundleId}/inputProcessors/{processorId}
```

Manage transactions in the Transaction Bundle

- Manually add a transaction to a bundle

Request: (body properties are base64 encoded!)

```
POST /vses/{vseId}/bundles/{bundleId}/transactions
```


```
{
    "request": {
        "body": "R0VUIC9pdGtvLWV4YW1wbGVzL3RleHQtcmVzcG9uc2UudHh0IEhUVFAvMS4xDQpBY
2NlcHQ6IHRleHQvaHRtbA0KSG9zdDogbG9jYWxob3N0OjgwODANCg=="
    },
    "responses": [
        {
            "body": "SFRUUC8xLjEgMjAwIE9LDQpTZXJ2ZXI6IEFwYWNoZS1Db3lvdGUvMS4xDQpFV
GFnOiBXLyIxNzUtMTQ4MzQ2NzYyMjAwMCINClgtUkJULU9wdGltaXplZC1CeTogVVNQTFJCMDEgKFJpT1M
gOS4xLjIpIFNDDQpMYXN0LU1vZGlmaWVkOiBUdWUsIDAzIEphbiAyMDE3IDE4OjIwOjIyIEdNVA0KQ29ud
GVudC1MZW5ndGg6IDE3NQ0KRGF0ZTogVHVlLCAwMyBKYW4gMjAxNyAwMDowMDowMCBHTVQNClgtUG93ZXJ
lZC1CeTogU2VydmxldCAyLjQ7IEpCb3NzLTQuMi4zLkdBIChidWlsZDogU1ZOVGFnPUpCb3NzXzRfMl8zX
0dBIGRhdGU9MjAwODA3MTgxNDM5KS9KQm9zc1dlYi0yLjANCkNvbnRlbnQtVHlwZTogdGV4dC9wbGFpbg0
KDQpUaGlzIGZpbGUgaXMgYW4gZXhhbXBsZSBvZiBhIG5vbi1odG1sIHJlc3BvbnNlIGZpbGUgdG8gdGVzd
CBvbi1vZmYgZmVhdHVyZXMgb2YgdGhlIHJlY29yZGVyLg0KDQpUZXN0aW5nIGlzIGEgbWFuZGF0ZSBhbmQ
geW91IG5lZWQgdG8gZG8gaXQuICBFdmVyeW9uZSBzaG91bGQgb3duIHF1YWxpdHkuDQoNCkNoZWVycw==="

        }
    ]
}
```


Transaction Request Example (Base64 decoded):

```
GET /itko-examples/text-response.txt HTTP/1.1
Accept: text/html
Host: localhost:8080
```

Transaction Response Example (Base64 decoded):

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
ETag: W/"175-1483467622000"
X-RBT-Optimized-By: USPLRB01 (RiOS 9.1.2) SC
Last-Modified: Tue, 03 Jan 2017 18:20:22 GMT
Content-Length: 175
Date: Tue, 03 Jan 2017 00:00:00 GMT
X-Powered-By: Servlet 2.4; JBoss-4.2.3.GA (build: SVNTag=JBoss_4_2_3_GA date=20080
7181439)/JBossWeb-2.0
Content-Type: text/plain

This file is an example of a non-html response file to test on-off features of
the recorder.

Testing is a mandate and you need to do it.  Everyone should own quality.

Cheers
```

Response:


```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/bundles
/1RKLr1NXz758V7NFvddEBB/transactions/1"
        }
    },
    "id": "1",
    "_embedded": {
        "request": {
            "_links": {
                "self": {
                    "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2
/vses/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/bundles
/1RKLr1NXz758V7NFvddEBB/transactions/1/request"
                }
            }
        },
        "responses": [
            {
                "_links": {
                    "self": {
                        "href": "http://localhost:1505/lisa-virtualize-invoke/api
/v2/vses/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/bundles
/1RKLr1NXz758V7NFvddEBB/transactions/1/responses/2"
                    }
                },
                "id": "2"
            }
        ]
    }
}
```

The "body" fields in the request are required to be **Base64** encoded. The response is a JSON that represents the new transaction. Using the previous response, the **{transactionId}** is **1** in the response.

- Manage transactions in a bundle.

The following APIs are used to view/delete the transactions that got added in the specific **{bundleId}**.

Request:

```
GET /vses/{vseId}/bundles/{bundleId}/transactions
```

Request:

```
GET /vses/{vseId}/bundles/{bundleId}/transactions/{transactionId}
```

Request:

```
DELETE /vses/{vseId}/bundles/{bundleId}/transactions/{transactionId}
```

Process Transactions Using the Bundle Processor

- Create a bundle processor.

Request :

```
POST /vses/{vseId}/bundleProcessors

{
    "sessionId": "{sessionId}",
    "bundleId": "{bundleId}"
}
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/bundleProcessors
/efruRJNtipG52v3f9kcg3opLePFUSDwDk5TQNSn3ediVbiwfjl_"
        }
    },
    "id": "efruRJNtipG52v3f9kcg3opLePFUSDwDk5TQNSn3ediVbiwfjl_",
    "status": "IN_PROCESS",
    "transactionCount": 0,
    "sessionCount": 0
}
```

Using the previous example, the **{procId}** is **efruRJNtipG52v3f9kcg3opLePFUSDwDk5TQNSn3ediVbiwfjl_**.

- Check the status of the bundle processor.

Request :

```
GET /vses/{vseId}/bundleProcessors/{procId}

{
    "sessionId": "{sessionId}",
    "bundleId": "{bundleId}"
}
```

- Delete the bundle processor (when done).

Request :

```
DELETE /vses/{vseId}/bundleProcessors/{procId}
```

Manage Transactions

Manage Transactions (Without Data Protocols Applied)

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/transactions
```

The response will return the first 100 transactions by default. You can add offset and limit parameters to the URL to get more transactions at a time or to do paging. For example: **? offset=0&limit=150.** To manage a specific transaction, note the unique transaction id for each transaction.

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/transactions/{transactionId}
```

Request:

```
DELETE /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/transactions/{transactionId}
```

Manage Transactions (With Data Protocols Applied)

The following APIs return the same transactions as before (the raw versions) but they have been parsed by the data protocols applied up to and including the data protocol specified by the **{dpId}** . If you had three data protocols and you specified the id for the second one, the transactions will appear with only the first and second data protocols applied.

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/dataProtocols/{dpId}/transactions
```

The response returns the first 100 transactions by default. You can add offset and limit parameters to the URL to get more transactions at a time or to do paging. For example: **? offset=0&limit=150.** To manage a specific transaction, note the unique transaction id for each transaction.

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/dataProtocols/{dpId}/transactions/{transactionId}
```

Create the virtual service

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}
Accept: application/zip
```

Response: The response is a ZIP file that contains the virtual service files. You can deploy the ZIP file directly using the POST method of DCM LISA Invoke REST API.

Add Transactions to an Existing Virtual Service from RR Pairs or a Specification Document (WSDL)

Get the VSE Id

Request:

```
GET /vses
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m"
        }
    }
}
```

A successful response returns an array of VSE Server JSON objects. Choose the VSE to use and note the VSE id for it, which is found in the 'self' link. Using the previous example, the **{vseId}** is **j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m**.

Create a Builder Session from a MAR (Existing VS)

Request:

```
POST /vses/{vseId}/vsBuilderSessions
Content-Type: application/zip

<Body: MAR file you want to upload>
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/vsBuilderSessions
/efrsjq_mqVTNwXYsmQ9vxE1dYTwWNy07FD8ioKuZmVNugsgWHAU"
        }
    },
    "id": "efrsjq_mqVTNwXYsmQ9vxE1dYTwWNy07FD8ioKuZmVNugsgWHAU",
    "modifyDate": "2016-12-20T14:34:21-06:00",
```

```
    "defaultNonLeafTolerance": "WIDE",
    "name": "test001",
    "description": "Optional description",
    "defaultLeafTolerance": "LOOSE",
    "version": "2",
    "createDate": "2016-12-20T14:34:21-06:00"
}
```

The request should have a VS MAR file with content type set to application/zip. A successful response returns a JSON object that represents the newly created builder session object. Using the previous example, the **{sessionId}** is **efrsjq_mqVTNwXYsmQ9vxE1dYTwWNy07FD8ioKuZmVNugsgWHAU**.

View the Transport Protocol or Transport Protocols

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols
```

A successful response returns a JSON containing a list of transport protocols. Using the id property from one of the transport protocols, as the {tpId} for the following API calls.
As of Release 10.0, multiple transport protocols can be created but only the first one is used by the recorder. To add a new data protocol, see Add a Data Protocol.

View the Data Protocol or Data Protocols (Optional)

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/dataProtocols
```

A successful response returns a JSON containing a list of data protocols. Using the id property from one of the transport protocols, as the { dpId } for the following API calls.
To add a new data protocol, see Add a Data Protocol.

Add Transactions

Create a Transaction Bundle

Request:

```
POST /vses/{vseId}/bundles

{}
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/bundles
/1RKLr1NXz758V7NFvddEBB"
        }
    },
    "id": "1RKLr1NXz758V7NFvddEBB",
    "_embedded": {
        "transactions": []
```

```
    }
}
```

Using the previous example, the **{bundleId}** is **1RKLr1NXz758V7NFvddEBB**.

Create an Input Container

An "Input Container" is a server-side container that holds uploaded artifacts (RR pairs and spec documents, for example) until they can be processed to produce transactions.

Request:

```
POST /vses/{vseId}/bundles/{bundleId}/inputContainers

{}
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/bundles
/1RKLr1NXz758V7NFvddEBB/inputContainers/IC1Za52IBxr39PIwEUIlBt8m"
        }
    },
    "id": "IC1Za52IBxr39PIwEUIlBt8m",
    "_embedded": {
        "transSources": []
    }
}
```

Using the previous example, the **{inputContainerId}** is **IC1Za52IBxr39PIwEUIlBt8m.**

Upload Data to an Input Container

- Upload RR Pair files to an input container.

Request:

```
POST /vses/{vseId}/bundles/{bundleId}/inputContainers/{inputContainerId}/contents
Content-Type: multipart/form-data

file1=<the file you want to upload>
file2=<the file you want to upload>
```

Request example (using cURL):

```
curl -X POST
-H "Authorization: Basic XXXXXXXXXXXXX"
-H "Content-Type: multipart/form-data;"
-F "file1=@/Applications/CA/DevTest/Projects/Bank v5/Data/rrpair/PayAction-req.
txt"
```

```
-F "file2=@/Applications/CA/DevTest/Projects/Bank v5/Data/rrpair/PayAction-rsp.
txt"
"http://localhost:1505/lisa-virtualize-invoke/api/v2/vses/j4xHDBZf8x3-
9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/bundles/1RKLr1NXz758V7NFvddEBB
/inputContainers/IC1Za52IBxr39PIwEUIlBt8m/contents"
```

The previous example uses RR pairs defined in the sample Bank v5 Project. The new **{contentId}** is the "id" field in the response.

- Upload a specification document (WSDL) to an input container.

Request:

```
POST /vses/{vseId}/bundles/{bundleId}/inputContainers/{inputContainerId}/contents
Accept: application/zip

{
    "url":"http://localhost:8080/itkoExamples/EJB3UserControlBean?wsdl"
}
```

The previous example requires the DevTest DemoServer (LISA Bank) to be running. Note the {contentId} in the response.

Process Data Using the Input Processor

> An “Input Processor” is responsible for processing the contents of the "Input Container" to produce transactions out of the provided artifacts.

1. Start an input processor.

Request:

```
POST /vses/{vseId}/bundles/{bundleId}/inputProcessors

{
    "container": "{inputContainerId}"
}
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/bundles
/1RKLr1NXz758V7NFvddEBB/inputProcessors/IPrp57Jc9v2n9P7yimix5I2"
        }
    },
    "id": "IPrp57Jc9v2n9P7yimix5I2"
}
```

Using the previous example, the **{processorId}** is **IPrp57Jc9v2n9P7yimix5I2**.

2. Check the status of an input processor.

Request:

```
GET /vses/{vseId}/bundles/{bundleId}/inputProcessors/{processorId}
```

3. Stop an input processor (when done).

Request:

```
DELETE /vses/{vseId}/bundles/{bundleId}/inputProcessors/{processorId}
```

Manage Transactions in the Transaction Bundle

- Manually add a transaction to a bundle

Request: (body properties are base64 encoded!)

```
POST /vses/{vseId}/bundles/{bundleId}/transactions

{
    "request": {
        "body": "R0VUIC9pdGtvLWV4YW1wbGVzL3RleHQtcmVzcG9uc2UudHh0IEhUVFAvMS4xDQpBY
2NlcHQ6IHRleHQvaHRtbA0KSG9zdDogbG9jYWxob3N0OjgwODANCg=="
    },
    "responses": [
        {
            "body": "SFRUUC8xLjEgMjAwIE9LDQpTZXJ2ZXI6IEFwYWNoZS1Db3lvdGUvMS4xDQpFV
GFnOiBXLyIxNzUtMTQ4MzQ2NzYyMjAwMCINClgtUkJULU9wdGltaXplZC1CeTogVVNQTFJCMDEgKFJpT1M
gOS4xLjIpIFNDDQpMYXN0LU1vZGlmaWVkOiBUdWUsIDAzIEphbiAyMDE3IDE4OjIwOjIyIEdNVA0KQ29ud
GVudC1MZW5ndGg6IDE3NQ0KRGF0ZTogVHVlLCAwMyBKYW4gMjAxNyAwMDowMDowMCBHTVQNClgtUG93ZXJ
lZC1CeTogU2VydmxldCAyLjQ7IEpCb3NzLTQuMi4zLkdBIChidWlsZDogU1ZOVGFnPUpCb3NzXzRfMl8zX
0dBIGRhdGU9MjAwODA3MTgxNDM5KS9KQm9zc1dlYi0yLjANCkNvbnRlbnQtVHlwZTogdGV4dC9wbGFpbg0
KDQpUaGlzIGZpbGUgaXMgYW4gZXhhbXBsZSBvZiBhIG5vbi1odG1sIHJlc3BvbnNlIGZpbGUgdG8gdGVzd
CBvbi1vZmYgZmVhdHVyZXMgb2YgdGhlIHJlY29yZGVyLg0KDQpUZXN0aW5nIGlzIGEgbWFuZGF0ZSBhbmQ
geW91IG5lZWQgdG8gZG8gaXQuICBFdmVyeW9uZSBzaG91bGQgb3duIHF1YWxpdHkuDQoNCkNoZWVycw=="

        }
    ]
}
```

Transaction Request Example (Base64 decoded):

```
GET /itko-examples/text-response.txt HTTP/1.1
Accept: text/html
Host: localhost:8080
```

Transaction Response Example (Base64 decoded):

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
ETag: W/"175-1483467622000"
X-RBT-Optimized-By: USPLRB01 (RiOS 9.1.2) SC
Last-Modified: Tue, 03 Jan 2017 18:20:22 GMT
Content-Length: 175
Date: Tue, 03 Jan 2017 00:00:00 GMT
X-Powered-By: Servlet 2.4; JBoss-4.2.3.GA (build: SVNTag=JBoss_4_2_3_GA date=20080
7181439)/JBossWeb-2.0
```

```
Content-Type: text/plain

This file is an example of a non-html response file to test on-off features of
the recorder.

Testing is a mandate and you need to do it.  Everyone should own quality.

Cheers
```

Response:


```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/bundles
/1RKLr1NXz758V7NFvddEBB/transactions/3"
        }
    },
    "id": "3",
    "_embedded": {
        "request": {
            "_links": {
                "self": {
                    "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2
/vses/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/bundles
/1RKLr1NXz758V7NFvddEBB/transactions/3/request"
                }
            }
        },
        "responses": [
            {
                "_links": {
                    "self": {
                        "href": "http://localhost:1505/lisa-virtualize-invoke/api
/v2/vses/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/bundles
/1RKLr1NXz758V7NFvddEBB/transactions/3/responses/4"
                    }
                },
                "id": "4"
            }
        ]
    }
}
```


The "body" fields in the request are required to be **Base64** encoded. The response is a JSON that represents the new transaction. Using the previous response, the **{transactionId}** is **3** in the response.

- Manage transactions in a bundle

The following APIs are used to view/delete the transactions that got added in the specific **{bundleId}**.

Request:

```
GET /vses/{vseId}/bundles/{bundleId}/transactions
```

Request:

```
GET /vses/{vseId}/bundles/{bundleId}/transactions/{transactionId}
```

Request:

```
DELETE /vses/{vseId}/bundles/{bundleId}/transactions/{transactionId}
```

Process Transactions Using the Bundle Processor

1. Create a bundle processor.

Request :

```
POST /vses/{vseId}/bundleProcessors

{
    "sessionId": "{sessionId}",
    "bundleId": "{bundleId}"
}
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZf8x3-9Hmsv25utXt2HZjEoNlliAx3LVkNoBDN4VxvYo-m/bundleProcessors
/efruRJNtipG52v3f9kcg3opLePFUSDwDk5TQNSn3ediVbiwfjl_"
        }
    },
    "id": "efruRJNtipG52v3f9kcg3opLePFUSDwDk5TQNSn3ediVbiwfjl_",
    "status": "IN_PROCESS",
    "transactionCount": 0,
    "sessionCount": 0
}
```

Using the previous example, the **{procId}** is **efruRJNtipG52v3f9kcg3opLePFUSDwDk5TQNSn3ediVbiwfjl_**.

2. Check the status of the bundle processor.

Request :

```
GET /vses/{vseId}/bundleProcessors/{procId}

{
    "sessionId": "{sessionId}",
    "bundleId": "{bundleId}"
}
```

3. Delete the bundle processor (when done).

Request :

```
DELETE /vses/{vseId}/bundleProcessors/{procId}
```

Manage Transactions

Manage Transactions (Without Data Protocols Applied)

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/transactions
```

The response returns the first 100 transactions by default. You can add offset and limit parameters to the URL to get more transactions at a time or to do paging. For example: **? offset=0&limit=150** To manage a specific transaction, note the unique transaction id for each transaction.

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/transactions/{transactionId}
```

Request:

```
DELETE /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/transactions/{transactionId}
```

Manage Transactions (With Data Protocols Applied)

The following APIs return the same transactions as before (the raw versions) but they have been parsed by the data protocols applied up to and including the data protocol specified by the **{dpId}** . If you had three data protocols and you specified the id for the second one, the transactions appear with only the first and second data protocols applied.

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/dataProtocols/{dpId}/transactions
```

The response returns the first 100 transactions by default. You can add offset and limit parameters to the URL to get more transactions at a time or to do paging. For example: ? offset=0&limit=150 To manage a specific transaction, note the unique transaction id for each transaction.

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}/transportProtocols/{tpId}
/dataProtocols/{dpId}/transactions/{transactionId}
```

Create the Virtual Service

Request:

```
GET /vses/{vseId}/vsBuilderSessions/{sessionId}
Accept: application/zip
```

Response: The response is a ZIP file that contains the virtual service files.

Export a Virtual Service as MAR File

Get the VSE Id

Request:

```
GET /vses
```

Response:

```
{
    "_links": {
        "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZgbtMPZDxayNeJ8GFwswOqP1FnQHQlJcHypYULzoitnnFP"
        }
    }
}
```

A successful response returns an array of VSE Server JSON objects. Choose the VSE to use and note the VSE id for it, which is found in the 'self' link.

Using the previous example, the **{vseId}** is:
j4xHDBZgbtMPZDxayNeJ8GFwswOqP1FnQHQlJcHypYULzoitnnFP

Get the Service Id

Request:

```
GET /vses/{vseId}/services
```

Response:

```
{
  "_links": {
    "self": {
      "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZgbtMPZDxayNeJ8GFwswOqP1FnQHQlJcHypYULzoitnnFP/services?offset=0&limit=100"
    }
  },
  "count": 2,
  "total": 2,
  "_embedded": {
    "services": [
      {
        "_links": {
          "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZgbtMPZDxayNeJ8GFwswOqP1FnQHQlJcHypYULzoitnnFP/services/hM0SMHGOzxuf"
          }
        },
        "serviceId": "hM0SMHGOzxuf",
        "serviceName": "kioskV6"
      },
      {
        "_links": {
          "self": {
            "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZgbtMPZDxayNeJ8GFwswOqP1FnQHQlJcHypYULzoitnnFP/services/hM0SMHGOzxue"
          }
        },
        "serviceId": "hM0SMHGOzxue",
        "serviceName": "kioskV5"
      }
    ]
  }
}
```

A successful response returns an array of Virtual Service JSON objects. Choose the Virtual Service to use and note the service id for it, which is found in the 'services' link.
Using the previous example, the **{serviceId}** is: **hM0SMHGOzxuf and hM0SMHGOzxue**

Download MAR file

The following instructions assume that you are using the Postman REST client.

```
GET /vses/{vseId}/services/{serviceId}
```

Response:


```
{
  "_links": {
    "self": {
      "href": "http://localhost:1505/lisa-virtualize-invoke/api/v2/vses
/j4xHDBZgbtMPZDxayNeJ8GFwswOqP1FnQHQlJcHypYULzoitnnFP/services/hM0SMHGOzxue"
    }
  },
  "serviceId": "hM0SMHGOzxue",
  "serviceName": "kioskV5",
  "modelName": "kioskV5",
  "capacity": 1,
  "txnPerSecond": 0,
  "txnCount": 0,
  "peakTxnPerSecond": 0,
  "thinkScale": 100,
  "errorCount": 0,
  "autoRestartEnabled": true,
  "status": 2,
  "statusDescription": "running",
  "startTime": "8/24/17 3:00:46 PM",
  "startTimeLong": 1503579646141,
  "endTime": "",
  "endTimeLong": 0,
  "upTime": 4016773,
  "executionMode": "Most Efficient",
  "executionModeValue": "EFFICIENT",
  "configurationName": "project.config",
  "resourceName": "8001 : http :  : /itkoExamples",
  "groupTag": ""
}
```


A successful response returns a Virtual Service JSON object.

To download the MAR file follow these steps:

1. Copy the Request URL into Postman.

2. Configure 'Basic Authorization' and add the 'Accept' header with the value: **application /zip**.
   See the screenshot below:

3. Click the **Send** and **Download** button.

4. Manually add the .mar extension after the file name.

5. Save the MAR file.

Important: The link provided for the MAR file download in 'Service Virtualization API v2' UI does not work due to Swagger limitations. The corrupted download link in Swagger UI is documented here: https://github.com/swagger-api/swagger-ui/issues/374

Using the SDK

DevTest Solutions allows you to customize many of the application functions. This section describes how to extend DevTest functionality through the Software Development Kit (SDK).

Examples and API Documentation

The following file contains the examples that are referenced in this guide: examples.zip

The SDK JavaDocs file contains the API documentation. The SDK JavaDocs are available in the **docs** folder of your installation directory.

The Integration API

DevTest provides an integration Application Programmer Interface (API) that lets you manage test execution within Java-based server-side components. This section details the basic concepts central to the integration API.

Integration API Concepts

DevTest Solutions provides a powerful framework for testing server-side components. However, if a test fails, the framework provides little information about why the test failed. Although the component under test often calculates information that is of interest to the tester, this information is difficult to retrieve.

The integration API provides several elements for getting information out of server-side components. These elements include:

- **Integrators:** The application-specific Java classes that coordinate communication with the running test. For example, an EJB Integrator informs an EJB component whether integration is turned on. For more information about integrators, see Integrating Components.

- **TransInfo Class:** Abbreviation for Transaction Information. The integrated component uses this class to indicate testing events, such as a test failure or success. For more information about using the TransInfo class, see Integrating Components.

- **HasLisaIntegrator Interface:** Implemented by object types that are returned by methods that are integrated using Java-based integrators. These object types provide the running test with the information it needs about how the test has changed while testing the component. For more information about implementing the HasLisaIntegrator interface, see Constructing a Response Object.

- **Integration Filters:** Application-specific DevTest filters that coordinate communication with the integrated component. For example, the Servlet Filter turns on integration support for a servlet component and specifies a node to execute if the servlet returns specific values. For more information about integration filters, see Integration Filters.

- **Integration Assertions:** Special assertion elements that take advantage of the additional information that integrated server-side components provide. For example, a Check Integrator Response assertion can set the next test step based on the build status reported by the integrated component. For more information about integration assertions, see Integration Assertions.

Integration Flow

The integration flow includes the following steps.

1. The test case developer adds an application-specific integration filter to a test case. When the test is run, the filter turns on integration support for the type of application indicated. For example, the test case developer adds a Servlet Filter to test an integrated servlet.

2. When the server is invoked, test-enabled server components use the application-specific integrator class and the TransInfo class to establish the communication with the running test case. For example, if a servlet fails in some way, it can call **TransInfo.setBuildStatus()** to note the failure to DevTest. For more information about the TransInfo class, see the JavaDocs in the **doc** folder of your installation directory.

3. The filter automatically processes responses from the system-under-test. The filter logs important information and processes any commands from the tested component.

The Integration Process

To integrate to the DevTest integration API:

1. Make sure **lisa-int-*release#*.jar** is in your classpath.
   You can find **lisa-int-*release#*.jar** in the **LISA_HOME\lib\core** directory.

2. Add DevTest integration to the method.
   For more information about integrating into a server-side component, see Integrate Server-Side Components.

3. Handle the output from the integrated method.
   For more information about handling integrated output, see Handle Integrated Output.

4. Use integration filters in the test case that tests the server-side component.
   For more information about integration filters, see Integration Filters.

5. Use integration assertions in the test case that tests the server-side component.
   For more information about integration Assertions, see Integration Assertions.

Integrating Components

This section explains how to integrate server-side components using the integration API.

This section contains the following pages:

- Integrate Server-Side Components

- Collect Transaction Information
- Integrators
- Handle Integrated Output

Integrate Server-Side Components

This procedure describes how to DevTest-enable a server-side component.

Follow these steps:

1. Import the necessary classes.
   The component must at least import the appropriate integrator and the **TransInfo** class. For example, to import the classes necessary to integrate a servlet, add the following import statements:

   ```
   import com.itko.lisaint.servlet.ServletIntegrator;
   import com.itko.lisaint.TransInfo;
   ```

   For more information about the **TransInfo** class, see the JavaDocs in the **doc** folder of your installation directory.

2. Declare local variables to hold state.
   The integrated component needs at least an integrator and a **TransInfo** variable, as seen in the following code:

   ```
   ServletIntegrator si = null;
   TransInfo ti = null;
   ```

3. Start a transaction.
   The integration API lets you control when the integrated component begins to communicate with the DevTest software. All the communication must happen in context of a transaction. Therefore, before beginning to communicate, start a transaction using the application-specific integrator. You can ensure that DevTest is on before that by checking with the application-specific integrator class. The following code checks if DevTest is on, then retrieves the servlet integrator from the integrator class and stores it in the variable **si**. The code then starts the transaction using the servlet integrator and stores the result in the **TransInfo** object.

   ```
   if( ServletIntegrator.isLisaOn( request ) ) {
       si = ServletIntegrator.getServletIntegrator( request );
       ti = si.startTransaction( "Hello World" );
   }
   ```

4. Interact with the test case.
   Report component status. The **TransInfo** class provides a method, **setBuildStatus()**, that allows you to specify information that the system can use to determine how to run the test. For example, to tell DevTest that the component has failed, set the build status to STATUS_FAILED, as seen in the following code:

   ```
   if( /* component failed */ ) {
       // ...
       ti.setBuildStatus( TransInfo.STATUS_FAILED, failMsg );
   }
   ```

   For more information about valid build status codes, see Build Status under Collect Transaction Information.

   - Set properties. The **TransInfo** class provides a method, **setLISAProp()**, that allows you to set an arbitrary property. This method is a useful mechanism for getting information out of an integrated component, especially when used with the Ensure Property Matches Expression assertion.

For example, if your application calculates the Dow average, you could store that value in a property that can be checked while testing. The following code creates a property named DOW_AVERAGE and assigns the value CalculatedDowAverage.

```
ti.setLISAProp( "DOW_AVERAGE", CalculatedDowAverage );
```

This mechanism can be used to send data cache data from the system under test to the Test Manager as property values. These property values can be any serializable object, including strings, any of the Java object wrappers for primitive types, or even your own classes as long as the class path contains the proper classes to deserialize the data.

- Make assertions. The **TransInfo** class provides several methods that make assertions, and then take some action if the assertion fails. For example, the **assertFailTest()** method takes a Boolean value and makes an assertion that the value is true. If the value is false, the method instructs DevTest to fail the test and passes the string message that is logged.

  ```
  ti.assertFailTest( boolean_assertion, "ASSERT FAILED" );
  ```

  For more information about TransInfo assertion methods, see Assertion Methods under Collect Transaction Information.

- Force the next node. On rare occasions, it is difficult to get information from the server-side component back to the test case. In this situation, it is helpful to be able to force the test case to visit a specific test step if the condition arises. The **TransInfo** class provides a method, **setForcedNode()**, that overrides the next test step as determined by the Integrating Components test case. In the following code, the developer forces DevTest to make **specialStep** the next test step if the **special_condition** is true.

  ```
  if (special_condition)
  ti.setForcedNode("specialStep");
  ```

5. Send the response back to DevTest.
   For more information about sending responses back to DevTest, see Handle Integrated Output.

JSP Tag Library

In addition to the Java API shown previously, DevTest provides a JSP tag library for Java web development. This API makes using the Integration API easy for JSP developers.

Pure XML Integration

DevTest makes available a pure XML form of the Integration for web applications on an as-needed basis. If you require this form of integration, contact your sales or support representative.

Test Components

Sometimes a server-side application has a number of checks and it is not clear where the test step failed. You can separate out parts of the transaction using subcomponents and can report their individual statuses. The **com.itko.lisaint.CompInfo** class represents a subcomponent.

For example, the following code creates a subcomponent of the transaction and sets its build status to STATUS_SUCCESS:

```
CompInfo ci = ti.newChildComp( "SUBCOMP" );
ci.setBuildStatus( CompInfo.STATUS_SUCCESS, "" );
ci.finished();
```

Subcomponents are treated as part of the overall transaction. If one subcomponent fails, the entire transaction fails. The following code creates a subcomponent and sets the build status to STATUS_FAILED. In this case, the entire transaction fails.

```
CompInfo ci = ti.newChildComp( "SUBCOMP" );
ci.setBuildStatus( CompInfo.STATUS_FAILED, "" );
```

For more information about the CompInfo class, see the JavaDocs in the **doc** folder of your installation directory.

Collect Transaction Information

The TransInfo class encapsulates all the information that is gathered about the execution of a specific "transaction" of the system under test. This class informs the running test case of the status of this transaction. The DevTest instance requesting this transaction works with this object. A transaction is a round trip from a test instance to the system under test and back. Transactions can be broken up into subcomponents and reported at that level with the **CompInfo** object.

To construct a TransInfo object, call the **startTransaction** method on the appropriate integrator. For example, the following code creates a TransInfo in a servlet:

```
TransInfo ti = si.startTransaction( "Hello World" );
```

For more information about the TransInfo class, see the JavaDocs in the **doc** folder of your installation directory.

Build Status

The TransInfo class provides a method, **setBuildStatus**, that lets you specify information that the system can use to determine how to run the test. The **setBuildStatus** method takes a string constant whose possible values are defined on the TransInfo class. Status constants include:

- **S - STATUS_SUCCESS:** The component executed as expected.
- **U - STATUS_UNKNOWN:** The status is not known.
- **R - STATUS_REDIRECT:** The component issued a redirect (only valid for some systems).
- **I - STATUS_INPUTERROR:** The component failed due to bad inputs.
- **E - STATUS_EXTERNALERROR:** The component failed due to external resource errors.
- **F - STATUS_FAILED:** The component failed, presumably not because of inputs or external resources.

For more information about the TransInfo class, see the JavaDocs in the **doc** folder of your installation directory.

Assertion Methods

The TransInfo class provides several methods that make assertions, and then take some action if the assertion fails. The Assertion methods include:

- **assertLog(boolean expr, String msg):** If the assertion expr is false, the message is written to the log.

- **assertEndTest(boolean expr, String msg):** If the assertion expr is false, tell DevTest to end the test normally, and write the message to the log.
- **assertFailTest(boolean expr, String msg):** If the assertion expr is false, tell DevTest to fail the test, and write the message to the log.
- **assertFailTrans(boolean expr, String msg):** If the assertion expr is false, tell DevTest to consider the transaction as failed, and write the message to the log.
- **assertGotoNode(boolean expr, String stepName):** If the assertion expr is false, tell DevTest to execute the stepName test step next. This can also be a property name in a double curly brace {{x}} notation.

For more information about the TransInfo class, see the JavaDocs in the **doc** folder of your installation directory.

Integrators

An integrator is an application-specific Java class that coordinates the communication with the running test. For example, an EJB Integrator informs an EJB component whether DevTest integration is turned on.

The **com.itko.lisaint.Integrator abstract** class provides the base set of functionality for the following integrators:

- Java Integrator
- EJB Integrator
- Servlet Integrator

Each integrator coordinates with the running test on whether DevTest and DevTest integration is turned on. After this is determined, the primary responsibility of the integrator is to start a transaction and provide access to the TransInfo object.

For more information about creating a TransInfo object, see Collect Transaction Information.

For more information about the Integrator class, see the JavaDocs in the **doc** folder of your installation directory.

Handle Integrated Output

In addition to the transaction information, the integrated component must return the results of the component to DevTest Solutions. For servlets, this involves wrapping servlet responses. For Java-based components, this involves constructing a response object.

Wrapping Servlet Responses

DevTest integrates with web-based applications by embedding a streamed version of the integrator into the HTML output of the web server. The ServletIntegrator object is converted to ASCII text and wrapped in an HTML comment. The ServletIntegrator class provides a method, **report**, that takes an output stream and performs the described wrapping before sending it back to DevTest.

The following code wraps the servlet response and sends it back toDevTest if the ServletIntegrator indicates that DevTest is on:

```
if( ServletIntegrator.isLisaOn( request ) )
si.report( out );
```

For more information about the ServletIntegrator class, see the JavaDocs in the **doc** folder of your installation directory.

Constructing a Response Object

DevTest integrates with Java-based applications by enforcing that either the method returns value types or the class you are executing itself implements the **com.itko.lisaint.java.HasLisaIntegrator** interface.

For example, assume you have implemented an object with a LoginInfo object as a return value to the login (String uid, String pwd). The LoginInfo object maintains the information about whether the test succeeded or failed. To test that method, a test case author executes the **login** method, then queries the returned LoginInfo object to determine the success or failed state.

To implement the **com.itko.lisaint.java.HasLisaIntegrator** interface, implement the **getLisaIntegrator()** method to provide an XML representation of your integration object to DevTest.

Most implementations also provide a **setLisaIntegrator()** method and store the state in a member variable, as in the following code:

```
public class LoginInfo implements HasLisaIntegrator {

    private JavaIntegrator lisa;

    public String getLisaIntegrator() {
        return lisa.toXML();
    }

    public void setLisaIntegrator(JavaIntegrator obj) {
        lisa = obj;
    }

}
```

For more information about the HasLisaIntegrator and LoginInfo classes, see the JavaDocs in the **doc** folder of your installation directory.

Testing Integrated Components

This section explains how to test integrated server-side components using integration filters and integration assertions.

Integration Filters

An integration filter is an application-specific DevTest filter that coordinates the communication with the integrated component. The inclusion of an integration filter indicates two things:

- DevTest must turn on support for integration with that type of server-side component.
- When an integrated server-side component returns results that match the attributes of the filter, execute a specific test step as the next test step.

For example, the Servlet Filter turns on integration support for a servlet component and specifies a test step to execute if the servlet returns specific values. A test case developer typically defines multiple integration filters of a single type. Each filter checks a specific condition and sets the next test step accordingly.

The model editor provides built-in support for three integration filter types:

- Integration Filter for Java Applications
- Integration Filter for Servlet Applications
- Integration Filter for EJB Applications

Defining an integration filter in a test case indicates that DevTest wants to turn on integration for that type and to listen for responses of that type. To define an integration filter, create the filter, select the type, and set the attributes.

- **Trans Build Status:** The server-side component has set the build status on the TransInfo to a specific value. This field takes one or more build status code letters. For example, to define a filter that fires if the build status is set to any terminating status, enter the value **IEF**.
- **Trans Build Message Expression:** The server-side component has supplied a string as the message parameter to the setBuildStatus method of the TransInfo that matches the specified regular expression.
- **Component Name Match:** The server-side component created a subcomponent with a name that matches the value of this field. Possible values for the field include:
  - empty: Do not evaluate, meaning that there is never a match.
  - *: Match anything except a null.
  - anything else: Evaluated as a regular expression.
- **Component Build Status:** The server-side component has set the build status on the CompInfo to a specific value. This field takes one or more build status code letters. For example, to define a filter that fires if the build status is set to any terminating status, enter the value **IEF**.
- **Component Build Message Expression:** The server-side component has supplied a string as the message parameter to the setBuildStatus method of the TransInfo that matches the specified regular expression.
- **Exception Type Match:** The server-side component threw an exception with a type that matches the value of this field. Possible values for the field include:
  - empty: Do not evaluate, meaning that there is never a match.
  - *: Match anything except a null.
  - anything else: Evaluated as a regular expression.
- **Max Build Time (millis):** The time to execute the server-side component took less than the number of milliseconds specified.
- **On Transaction Error Node:** If the server-side component matches the specified attributes, then execute the node that was specified.
- **Report Component Content:** If this box is selected, the servlet application tested includes the actual content of the components in the output.

> **Note:** Only select the Report Component Content box if necessary, as it can be very bandwidth intensive. Also, the check box is only a request. Servlet applications can ignore the request and not provide the component content.

For more information about filters, see Using CA Application Test.

Integration Assertions

An integration assertion is an application-specific DevTest assertion that executes a specific test case as the next test case if an integrated server-side component returns results that match the attributes of the assertion.

For example, the Check Integrator Response assertion specifies a node to execute if the server-side component returns specific values. A test case developer typically defines multiple integration assertions of a single type. Each assertion checks a specific condition and sets the next node accordingly.

The model editor provides built-in support for three integration assertion types:

- Check Integrator Response
- Check Integrator Component Content Response
- Check Integrator Reporting Missing Data

To define an integration assertion, create the assertion and set the attributes based on the type of assertion, as outlined in the referenced assertion types.

Check Integrator Response

The Check Integrator Response assertion specifies a test case to execute if the server-side component returns specific values. Attributes include:

- **Trans Build Status:** The server-side component has set the build status on the TransInfo to a specific value. This field takes one or more build status code letters. For example, to define an assertion that executes if the build status is set to any terminating status, enter the value **IEF**.
- **Trans Build Message Expression:** The server-side component has supplied a string as the message parameter to the setBuildStatus method of the TransInfo that matches the specified regular expression.
- **Component Name Match:**The server-side component created a subcomponent with a name that matches the value of this field. Possible values for the field include:
  - empty: Do not evaluate, meaning that there is never a match.
  - *: Match anything except a null.
  - anything else: Evaluated as a regular expression.
- **Component Build Status:** the server-side component has set the build status on the CompInfo to a specific value. This field takes one or more build status code letters. For example, to define an assertion that executes if the build status is set to any terminating status, enter the value **IEF**.

- **Component Build Message Expression:** the server-side component has supplied a String as the message parameter to the setBuildStatus method of the TransInfo that matches the specified regular expression.
- **Exception Type Match:**The server-side component threw and exception with a type that matches the value of this field. Possible values for the field include:
  - empty: Do not evaluate, meaning that there is never a match.
  - *: Match anything except a null.
  - anything else: Evaluated as a regular expression.
- **Max Build Time (millis):** The time to execute the server-side component took less than the number of milliseconds specified.

For more information about assertions, see Using CA Application Test.

Check Integrator Component Content Response

The Check Integrator Component Content Response assertion specifies a test case to execute if the server-side component returns content that matches a regular expression. Not every server-side component can return component-level content. HTTP-based applications generally return the HTTP response.

Attributes include:

- **Component Name Spec:** The expression that selects the component whose content to search.
- **Expression:** The expression to search the component content. For both fields, the expressions are evaluated in the following way:
  - A null component value is not a hit, regardless of the expression.
  - An empty expression is never a hit.
  - A * matches any not null value.
  - Any other expression is considered a regular expression.

> **Note:** The entire transaction is itself a component, so its name and content are evaluated as part of this execution.

Check Integrator Reporting Missing Data

The Check Integrator Reporting Missing Data assertion specifies a test case to execute if the server-side component does not return expected content. Not every server-side component can return component-level content. HTTP-based applications generally return the HTTP response.

Attributes include:

- **Component Name Spec:** The expression that selects the component whose content to search. The expression is evaluated in the following way:
  - An empty expression is never a hit.
  - A * matches any not null value.
  - Any other expression is considered a regular expression.

> **Note:** The entire transaction is itself a component, so its name and content are evaluated as part of this execution.

For more information about assertions, see Using CA Application Test.

Extending DevTest Solutions

This section explains the main concepts that are involved in extending DevTest Solutions.

DevTest Solutions provides most of the elements that are required to test enterprise software components. However, you can create your own elements to solve specific problems. For example, DevTest does not provide built-in support for testing FTP clients. If you write an FTP client, you could create the following:

- Custom node to test the transfer of files
- Custom node to verify the contents of transferred files
- Custom filter to store portions of the file in DevTest properties

You can create custom types of other DevTest elements, such as data sets and companions. For more information about extending other elements, see the JavaDocs in the **doc** folder of your installation directory.

The following concepts are common to all customization scenarios:

- Extension Concepts

Extension Concepts

The following concepts are common to all customization scenarios:

- lisaextensions File
- wizards.xml File
- The NamedType Interface
- Parameters and Parameter Lists
- The Test Exec Class
- Test Exceptions

lisaextensions File

The **lisaextensions** file is where you define the extension points for DevTest.

One or more custom extensions (filters, assertions, test steps, and reports, for example) are declared in a file whose extension is .lisaextensions. The name must be unique relative to any other lisaextensions files in the same environment. For example, there could be two lisaextensions files with the names **library-A.lisaextensions** and **library-B.lisaextensions**.

The lisaextensions file must be included in a JAR with the classes that do the custom extension implementation. The JAR must be placed in the DevTest classpath. At the time of startup, DevTest looks for all files with the extension **.lisaextensions** and tries to read the custom extension points from them.

Note: Using the **typemap.properties** file to define extensions is supported only for backward compatibility.

All custom extensions in must provide a controller, viewer (frequently called an editor), and an implementation. These are often three different classes. Often, DevTest provides defaults for the controller and the viewer that you can use to avoid having to write your own. You can create custom versions if these defaults are not suitable for your needs.

Use of the lisaextensions File

The custom element is registered in the lisaextensions file. This subsection shows how that is accomplished. For DevTest to connect a given implementation to its authoring-time controller and editor, the lisaextensions file is used.

Implementation Objects

Here is an example of the portion of the contents of a lisaextensions file for custom assertions:

```
asserts=com.mycompany.lisa.AssertFileStartsWith
```

The class name on the right side of the equal sign represents the location of the classes that implement the appropriate node logic. If more than one assertion is defined, all corresponding classes are mentioned in the same line, separated by commas, as in:

```
asserts=com.mycompany.lisa.AssertFileStartsWith,com.mycompany.lisa.AnotherAssert
```

Controller and Editor Objects

DevTest performs a lookup on each element that is declared as an implementation object to get the controller and the editor to associate with that implementation. For example, the assertion in the previous example has the following extra entry in the lisaextensions file:

```
com.mycompany.lisa.AssertFileStartsWith=com.itko.lisa.editor.
DefaultAssertController,com.itko.lisa.editor.DefaultAssertEditor
```

The format of this properties file entry is

```
implementation_class_name=controller_class_name,editor_class_name
```

In this example, the default controller and editor are used. The defaults are adequate for most cases. However, you can encounter situations in which custom controllers or, more frequently, custom editors must be provided. A lisaextensions file is the place to declare the classes corresponding to them.

wizards.xml File

The **LISA_HOME** home directory contains a file named **wizards.xml**. This file drives many of the wizards that are rendered in DevTest. Any wizard that prompts the user with frequently accessed test elements (assertions, test steps, filters, and so on) consults this file for the information to display in the tree view. You can add your custom extensions to this file to make them easily available to users. You can also remove built-in elements that are not used frequently. The **wizards.xml** file itself provides some useful information about how to read and modify its contents, but the concepts are here.

Wizard Tag

The wizard tag is used to provide a list of elements for a given type of test element and a given type of use. For example:

```
<Wizard element="assert" type="http">
```

DevTest predefines the element values (in this case "assert") and the types (in this case "http"). New element values or types are ignored. The type is the short name for the kind of elements that is described in the contained Entry tags. In this example, the Entry tags are documenting assertions. The type attribute drives when DevTest renders the given list. For common testing needs (like web testing, XML testing, and Java testing), DevTest allows the wizards.xml file to contain different assertions and filters that are appropriate.

Entry Tag

Here is a sample Entry tag:

```
<Entry>
   <Name>Make Diff-like Comparisons on the HTML Result</Name>
   <Help>This is the right assertion to use when you have a HTML result and you wi
sh to perform diff-
like operations on it. It lets you define simply what portions of the text may cha
nge, what may not change, and what portions must match current property values.<
/Help>
   <Type>com.itko.lisa.web.WebHTMLComparisonAssert</Type>
</Entry>
```

The following child tags are required to define a wizard entry:

- **Name:** The label that is used in the wizard itself, shown as an option button in the current version of DevTest.
- **Help:** At the bottom of the wizard panel, there is an area for more text. This text is shown when this entry has been selected.
- **Type:** The actual test case element implementation class name.

The NamedType Interface

The **com.itko.lisa.test.NamedType** interface specifies that an implementing class is an object that is associated with a DevTest object viewer. Every class that you create to extend the DevTest Solutions implements this interface if it can be edited in the Test Manager. It exposes one method, **getTypeName**, which provides the text that is associated with the element in the model editor. For example, notice the text "File First Line Filter" in the following image.

SDK Filter Wizard screen

A method in the class that defines the filter provides the text. That class must implement the **com.itko.lisa.test.NamedType** interface, and must implement the **getTypeName** method:

```
public String getTypeName()
{
    return "File First Line Filter";
}
```

Parameters and Parameter Lists

Parameters (represented by an object of type **com.itko.util.Parameter** in lisa-util-*release#*.jar) are the attributes that define and support DevTest elements.

A ParameterList (represented by an object of type **com.itko.util.ParameterList** in lisa-util-*release#*.jar) is a collection of such parameters. In a way, ParameterList can be treated as an extension of **java.util.HashMap**.

> **Note:** The **com.itko.util.ParameterList** class was deprecated in release 9.5. If you are using a DevTest API that requires this class, you can continue to use it without causing functional issues. However, you might receive warning messages when compiling your code. When the ParameterList class is ready to be retired, we intend to communicate an official upgrade process.

The class that defines a DevTest element defines the parameters that the element exposes in a callback method that returns a **com.itko.lisa.test.ParameterList**. For example, the **getParameters()** method below defines the parameters for the Parse Property Value As Argument String filter:

```
public ParameterList getParameters()
{
    ParameterList p = new ParameterList();
```

```
    p.addParameter( new Parameter
( "Existing Property", PROPKEY_PARAM, propkey, TestExec.PROPERTY_PARAM_TYPE ) );
    p.addParameter( new Parameter( "IsURL", ISURL_PARAM, new Boolean(isurl).
toString(), Boolean.class ) );
    p.addParameter( new Parameter( "Attribute", ATTRIB_PARAM, attrib, TestExec.
ATTRIB_PARAM_TYPE ) );
    p.addParameter( new Parameter( "New Property", PROP_PARAM, prop, TestExec.
PROPERTY_PARAM_TYPE ) );
    return p;
}
```

For each parameter exposed, the method creates a **com.itko.util.Parameter**. The previous example defined the following parameters:

- A property value for Existing Property
- A Boolean value for IsURL
- An attribute value for Attribute
- A property for New Property

The parameters to the constructor of Parameter are:

- The string that provides the label for the parameter in the model editor
- A string key that is used to store the value of the parameter in a Java Map
- A variable that stores the value of the parameter
- The fully qualified type of the parameter

The model editor uses the last parameter to determine the user interface element that is associated with the parameter. For example, passing **Boolean.class** renders the parameter as a check box. Passing **TestExec.PROPERTY_PARAM_TYPE** renders the parameter as a drop-down list containing the keys of all existing properties.

For more information about the **com.itko.util.Parameter** and **com.itko.util.ParameterList** classes, see the JavaDocs in the **doc** folder of your installation directory.

The Test Exec Class

The **com.itko.lisa.test.TestExec** class provides access to the state of the test and convenience functions for performing common tasks within DevTest elements. A TestExec parameter is passed to most DevTest element methods. Some of the most useful methods that the TestExec class provides include:

- **log:** Fires a EVENT_LOGMSG TestEvent including the string parameter.
- **getStateObject:** Takes a string property key and returns the value of that property.
- **setStateValue:** Takes a string property key and an Object and sets the value of that property to the specified Object value.
- **setNextNode:** Takes the string name of a node and sets the next test case to fire to that test case.
- **raiseEvent:** Allows you to fire an arbitrary DevTest event.

For more information about the **com.itko.lisa.test.TestExec** class, see the JavaDocs in the **doc** folder of your installation directory.

Test Exceptions

DevTest Solutions provides two main exception classes for handling errors in test case components:

- **com.itko.lisa.test.TestRunException (in lisa-core-*release#*.jar)**
  Indicates a problem in how the test was run. For example, trying to access a parameter that does not have an expected value would throw this type of exception.

- **com.itko.lisa.test.TestDefException (in lisa-core-*release#*.jar)**
  Indicates a problem in how the test was defined. For example, trying to reference a data set that was not defined would throw this type of exception.

For more information about exception classes, see the JavaDocs in the **doc** folder of your installation directory.

Extending Test Steps

The test steps that DevTest provides include most of the logic that is required to test enterprise software. However, you can create your own test step to accommodate a specific situation.

Each existing test step provides a step-specific Swing user interface to help users develop that type of test step. You can provide this same support by creating a Java class that extends **com.itko.lisa.test.TestNode** and providing a Swing user interface. However, there is a much simpler way to provide a custom test step.

A set of name-value pairs can adequately represent many testing situations. For example, assume that you want to test a File Transfer Protocol (FTP) client package you have written. You do not need a complex wizard to collect the information that is required to test. You need only:

- The FTP host

- The full path and name of the file

- The username and password that is used to access the FTP host

A name-value pair can represent each of these values.

DevTest provides built-in support for custom test steps that fit this profile. To create a custom test step, you create a Java class that implements **com.itko.lisa.test.CustJavaNodeInterface**. This class specifies the name-value pairs that are associated with the test step and the logic to run when the test step executes. At runtime, the model editor searches the classpath for classes that implement **CustJavaNodeInterface.** The model editor makes these classes available under the Custom Test Step Execution test step type.

The DevTest SDK provides two ways to enable you to augment the functionality of DevTest with new test cases:

- **Custom Java Test Steps:** This type of test step is faster and easier to develop, so it is often the preferred method that developers use.

- **Native Test Steps:** These test steps are created in precisely the way that CA Technologies develops test steps within DevTest and can appear in their own categories.

This section contains the following pages:

- Custom Java Test Steps
- Native Test Steps

Custom Java Test Steps

This section explains how to create a custom Java test step and use it in the model editor.

This type of test step is created by extending an abstract base class that that is provided in **Lisa.jar** . A custom Java test step is faster and easier than the native test step because it does not require you to build a user interface for the Test Manager. Instead, one is auto-generated for you by invoking the calls on this class. This efficiency comes at the cost of control over how the user interacts with your test step at the time the test is being authored. Most parameters are rendered in an appropriate manner, but some parameters could require a customized editor. For those needs, consider creating a native test step.

Create a Custom Java Test Step

Follow these steps:

1. Create a Java class that implements **com.itko.lisa.test.CustJavaNodeInterface** (in lisa-core-*release#*.jar).
   This tells DevTest Solutions that your class is a custom Java test step.

   ```
   public class FTPCustJavaNode implements CustJavaNodeInterface
   {
   }
   ```

2. Implement the required **initialize** method.
   This method is called when DevTest first loads a custom test step during testing.
   In this example, the test step needs no initialization, so it simply logs the fact that the **initialize** method was called.

   ```
   static protected Log cat = LogFactory.getLog( "com.mycompany.lisa.ext.node.
   FTPCustJavaNode" );
   public void initialize
   ( TestCase test, Element node ) throws TestDefException
   {
       cat.debug( "called initialize" );
   }
   ```

3. Implement the required **getParameters** method.
   This method specifies the name-value pairs that define the parameters to the custom test step. The simplest way to define the parameter list is to pass one long string of all parameters, with each parameter separated by an ampersand (&). Values that are specified here are used as the default values when the node is defined in the model editor.

   ```
   public ParameterList getParameters()
   {
       ParameterList pl = new ParameterList( "username=&password=&host=ftp.
   suse.com&path=/pub&file=INDEX" );
       return pl;
   }
   ```

   The previous example uses **ftp.suse.com/pub/INDEX** as the file to retrieve. This is a publicly available FTP host that allows an anonymous login. Limit unnecessary hits on the SuSE FTP site.

4. Implement the required **executeNodeLogic** method.
   This method defines the logic that runs when the test step executes. Typically, this method is used to instantiate and validate components of the system under test.
   The **TestExec** parameter provides access to the test environment, such as logs and events.
   The **Map** parameter provides access to the current value of the test step parameters. The **Object** return type allows you to pass data back to the running test, so that you can run assertions and filters on it.

   ```
   public Object executeNodeLogic
   ( TestExec ts, Map params ) throws TestRunException
   {
       ts.log( "We got called with: " + params.toString() );
       String host = (String)params.get("host");
       String username = (String)params.get("username");
       String password = (String)params.get("password");
       String path = (String)params.get("path");
       String file = (String)params.get("file");
       String storedFile = runFTP(ts,host,username,password,path,file);
       FileDataObject fdo = new FileDataObject(storedFile);
       return fdo;
   }
   ```

   Using a custom data object for the return value is common. This strategy has two benefits:

   - It encourages proper resource handling. Do not pass common resources, such as files and JDBC connections, as results. Doing so prevents the node performing a cleanup by calling a close method. If you construct your own data object, you can store and pass only the information you need from the resource. Then close the resource before returning the data object.

   - It allows you to perform conditional filtering and result-checking based on the type of the data object. For more information about using data objects to perform conditional filtering, see Create a New Filter.

   The previous code uses the **FileDataObject** custom data object to store only the path to the stored file, rather than passing an open **File** or **FileInputStream**. Any filter can perform conditional processing by checking that the type of the result object is **FileDataObject**.

Deploy a Custom Java Test Step

You must make a custom Java test step available to the model editor before you can use it in a test case.

Follow these steps:

1. Create a lisaextensions file or use an existing one if one exists.
   In the lisaextensions file, mention this class against the simpleNodes element.

   ```
   simpleNodes=com.mycompany.lisa.FTPCustJavaNode
   ```

2. Copy the JAR file that contains your custom Java test step and the lisaextensions file to the **LISA_HOME\hotDeploy** directory.
   If your custom Java test step depends on any third-party libraries, copy those libraries to the **LISA_HOME\hotDeploy** directory.
   For this example, the FTPCustJavaNode described previously has been packaged for you in **lisaint-examples.jar**. This custom Java test step depends on the FTP client that is packaged in **ftp.jar**. You can download an **examples.zip** file that contains these files from Using the SDK.
   Copy both of these files to the **LISA_HOME\hotDeploy** directory.

> **Note:** The FTP client that is used in the sample code is provided by www.amoebacode.com/.

This client puts the given class name in a convenient drop-down list for users when authoring a test case. This is optional because DevTest can be given the class name at authoring time by typing or using the Class Path Navigator.
If you are already running DevTest, exit and restart the program for this new setting to take effect.

Define a Custom Java Test Step

This procedure describes defining a custom Java test step in the model editor.

Follow these steps:

1. Change the **Type** of the node to **Custom Test Step Execution**.

2. Specify the custom test step to test.
   Enter or select the fully qualified name of the Java class in the **Class Name** field.
   To specify a custom Java test step class easily, click the ellipsis (...) next to the **Class Name** field to launch the **Class Path Navigator**. The Path Navigator only looks for custom test steps. Select the test step class that you want to test and click **OK**.

Custom Test Step Execution - Java Step - Class Path Navigator

3. Set the parameters to the custom test step.
   For each parameter in the **Parameters** list, supply a value.

Custom Test Step Execution - Java step - Parameters

Native Test Steps

This section explains how to create a native test step and use it in the model editor. The native test step is the way to provide 100 percent customized test step authoring and execution for your test steps. In this test step, you are responsible for providing the editing environment and the complete execution engine for your test step. This process is precisely how the provided native test steps are developed.

Create a Native Test Step

This procedure describes creating a native test step.

Follow these steps:

1. Create a Java class that extends **com.itko.lisa.test.TestNode**.
   This class implements the runtime logic of the test step. The following calls are important for the construction of a test step:

   ```
   public void initialize
   ( TestCase test, Element node ) throws TestDefException
   ```

   DevTest invokes this call for you to read the parameters you require for the operation from the test case XML DOM. As the test case is constructed, it is passed with the DOM Element of the step that represents this node in the test case XML.

   ```
   public void execute( TestExec ts ) throws TestRunException
   ```

   This is called when your test step logic is invoked. The workflow and state engines manage most of the control flow and data requirements for you. You can access the **TestExec** given as a parameter to perform various tasks. See the JavaDocs **TestExec** and the description of this class in Integrating Components.

2. Create a Java class that extends **com.itko.lisa.editor.TestNodeInfo**.
   To create and edit your node, provide a controller and viewer in the MVC pattern. The **TestNodeInfo** class is the base class for all test steps that are developed for DevTest. The authoring framework executes this class to interact with your test step data and logic.

3. Create a Java class that extends **com.itko.lisa.editor.CustomEditor**.
   This class provides DevTest with the actual user interface for viewing and editing your node data. The authoring framework constructs and calls this class to display your parameters, verify their validity, and save your changes back into the **TestNodeInfo** extension. This **CustomEditor** extends **JPanel** in the Java Swing API. See the JavaDocs for **CustomEditor** and view the sample code for help writing your own editors.

4. Perform the following method overrides if you want to control how the default name of the step is generated.

   - Override the **generateName** method in the class that extends **TestNode**. In this method, add the logic to create the default name.

     ```
     public String generateName()
     {
         return "default name"
     }
     ```

   - Override the **generateName** method in the class that extends **TestNodeInfo**. In this method, add a call to the **generateName** method in the class that extends **TestNode**.

     ```
     public String generateName()
     {
         return "defined name"
     }
     ```

Deploy a Native Test Step

Native test steps must be explicitly declared to DevTest at startup of the Test Manager. This declaration allows the authoring framework to associate the three classes that were defined previously. This declaration is done in the **lisaextensions** file against the nodes element. The classes themselves do not refer to one another. They are connected in the **lisaextensions** file.

For more information about the lisaextensions file, see Extending DevTest Solutions.

Define a Native Test Step

Your native test step can be accessed the same way that existing, built-in steps are made available in DevTest. See Using CA Application Test for information about how to access a test step.

Extending Assertions

This section explains how to extend the DevTest Solutions with a new assertion.

Create an Assertion

The assertions that the DevTest Solutions provides contain most of the logic that is required to test enterprise software. However, you can create your own assertion to handle a specific situation.

DevTest provides built-in support for custom assertions. To create a custom assertion, you create a Java class that extends **com.itko.lisa.test.Assertion** (in lisa-core-*release#*.jar). This class handles most of the behind-the-scenes logic that is required to execute the assertion, and provides a nice user interface in the model editor. If you have an existing Java class that extends **com.itko.lisa.test.CheckResult**, it is still supported, but this class is deprecated and not recommended.

Follow these steps:

1. Create a Java class that extends **com.itko.lisa.test.Assertion**.
   This class tells the DevTest that your class is a custom assertion and provides you with the needed assertion functionality.

   ```
   public class AssertionFileStartsWith extends Assertion
   {
   }
   ```

2. Implement the required **getTypeName** method.
   This method provides the name that is used to identify the custom assertion in the model editor.

   ```
   public String getTypeName()
   {
       return "Results File Starts With Given String";
   }
   ```

   The string that the **getTypeName** method returns is the default name of a new assertion.

3. Handle custom parameters to the assertion.
   Some assertions require more information than is provided in the node execution result. In this case, provide a parameter that allows the user to specify that information in the **Assertions** tab of the model editor.
   Implement the abstract **getCustomParameters** method. In this method, you create a **ParameterList** and add a **Parameter** for each parameter to the assertion. The constructor for the Parameter takes the following:

   - A string that provides the label for the parameter in the user interface.
   - A string that provides the key to store the value of the parameter in a Map.
   - A string representing the value of the parameter.
   - The type of the parameter.

   In this example, the custom assertion takes one parameter, the string to find in the file. The rest of the code provides support for the parameter.

   ```
   public static final String PARAM = "param";
   private String param = "";
   ...
   public String getParam()
   {
       return param;
   }
   public ParameterList getCustomParameters()
   {
       ParameterList pl = new ParameterList();
       pl.addParameter( new Parameter
   ( "Starts With String", PARAM, param, String.class ) );
       return pl;
   }
   ```

4. Initialize the custom assertion object with the value of the DOM Element.
   When DevTest tries to execute an assertion, it first creates an instance of the custom class. It then calls the **initialize** method, passing the XML element that defined the assertion. You must store the values of the parameter child elements in the new instance. For example, the test case XML can include an assertion that looks like the following example:

   ```
   <CheckResult name="CheckThatFile"
       log="Check if the node FTP'd a file that starts with 'All Visitors'"
       type="com.mycompany.lisa.AssertFileStartsWith" >
   <then>finalNode</then>
   <param>All Visitors</param>
   </Assertion>
   ```

   In the **initialize** method, you must get the text "All Visitors" into the **param** member variable. The following code grabs the text from the child element named **param** and checks to ensure that it is not null.

   ```
   public void initialize( Element rNode ) throws TestDefException
   {
       param = XMLUtils.getChildText( XMLUtils.findChildElement
   ( rNode, PARAM ) );
       if( param == null )
       throw new TestDefException( rNode.getAttribute("name"),
       "File Starts With results must have a Starts With String parameter speci
   fied.", null );
   }
   ```

   You are free to use whatever means that you prefer to read from the DOM Element. You can use a convenience class that is named **XMLUtils**, as seen in the previous example.

5. Implement the checking logic with the **evaluate** method.
   The **TestExec** parameter provides access to the test environment, such as logs and events. The **Object** parameter provides access to results returned from executing the node. The Boolean return type returns **true** if the assertion is true. Otherwise it returns **false**.
   The following code verifies that the first line of the file that is passed from the node contains the string that is stored in the **param** parameter. This code returns **true** only if the file starts with that string.

   ```
   public boolean evaluate( TestExec ts, Object oresult )
   {
       if( oresult == null )
           return false;
       FileDataObject fdo = (FileDataObject)oresult;
       String firstline = fdo.getFileFirstLine();
       return firstline.startsWith(param);
   }
   ```

   As an author of an assertion, simply declare whether the assertion as defined is true or false. Do not worry about the steps to execute after the state is returned. The workflow engine evaluates whether that is considered a fired assertion.

6. (Optional) Implement the **setValueDetail** method to integrate into Portal Test Monitoring facility.
   You can provide a detailed message for the assertion about the actual value and the expected value.

   ```
   public void setValueDetail(Object actual, Object expectation)
   ```

Deploy an Assertion

You must make a custom assertion available to the model editor before you can use it in a test case.

Follow these steps:

1. Tell DevTest to look for a new custom assertion in the **.lisaextensions** file, as:

```
asserts=com.mycompany.lisa.AssertFileStartsWith
 com.mycompany.lisa.AssertFileStartsWith=com.itko.lisa.editor.
DefaultAssertController,com.itko.lisa.editor.DefaultAssertEditor
```

2. (Optional) You can also add this assertion to the wizards in the **wizards.xml** file.

```
        <Entry>
            <Type>com.itko.lisa.lek.training.assertion.
StringPropertyComparisonAssertion</Type>
        </Entry>
</Wizard>
```

3. Copy the JAR file that contains your custom assertion and **.lisaextensions** file to the **LISA_HOME\hotDeploy** directory.
   If your custom assertion depends on any third-party libraries, copy those libraries to the **LISA_HOME\hotDeploy** directory.
   In this example, the AssertionFileStartsWith described previously has already been packaged for you in **lisaint-examples.jar**. You can download an **examples.zip** file that contains this file from Using the SDK. This custom assertion does not depend on any third-party libraries.

4. If you are already running DevTest, exit and restart the program for this new setting to take effect.

Define and Test an Assertion

This procedure describes defining a custom assertion.

Follow these steps:

1. Change the **Type** of the assertion, selecting the text that you specified in the **getTypeName** method.

2. Set the parameters to the custom assertion.
   For each parameter in the **Assertion Param Attributes** section, supply a value.

3. Use a custom assertion like you would any built-in assertion.

Extending Filters

This section explains how to extend DevTest Solutions with a new filter.

Create a Filter

The filters that the DevTest Solutions provides includes most of the logic that is required to test enterprise software. However, you can create your own filter to handle a specific situation. DevTest provides built-in support for custom filters.

Follow these steps:

1. Create a Java class that extends **com.itko.lisa.test.FilterBaseImpl**.
   This class tells DevTest that your class is a custom filter.

```
public class FilterFileFirstLine extends FilterBaseImpl
{
}
```

2. Implement the required **getTypeName** method.
   This method provides the name that is used to identify the custom filter in the model editor.

```
public String getTypeName()
{
    return "File First Line Filter";
}
```

3. Define the parameters to the filter.
   For each item in the **Filter Attributes** section of the **Filters** tab in the model editor, add a **Parameter** to the **ParameterList** for the filter. In this example, the custom filter takes two parameters:

   - A check box to identify whether the text in the first line of the file represents an FTP host
   - A text box to identify the parameter in which the line is stored

   The following code creates the two parameters:

```
public ParameterList getParameters()
{
    ParameterList p = new ParameterList();
    p.addParameter( new Parameter( "Is FTP", ISFTP_PARAM, new Boolean
(isftp).toString(), Boolean.class ) );
    p.addParameter( new Parameter
( "New Property", PROP_PARAM, prop, TestExec.PROPERTY_PARAM_TYPE));
    return p;
}
```

4. Initialize the custom filter object with the value of the DOM Element.
   When DevTest tries to execute a filter, it first creates an instance of the custom class. DevTest then calls the **initialize** method, passing the XML element that defined the filter. You must store the values of the parameter child elements in the new instance.
   For example, the test case can include a filter that looks like the following example:

```
<Filter type="com.mycompany.lisa.ext.filter.FilterFileFirstLine"
    isftp="true" prop="THE_LINE" />
```

   In the **initialize** method, set the **isftp** variable to **true** and set the **prop** variable to **THE_LINE**.

```
static private String ISFTP_PARAM = "isftp";
static private String PROP_PARAM = "prop";
private String prop;
private boolean isftp;
//...
public void initialize( Element e ) throws TestDefException
{
    try {
        String s = XMLUtils.findChildGetItsText( e, ISFTP_PARAM ).
toLowerCase();
        if( s.charAt(0) == 'y' || s.charAt(0) == 't' )
            isftp = true;
        else
            isftp = false;
    }
    catch( Exception ex ) {
        isftp = false;
    }
    prop = XMLUtils.findChildGetItsText( e, PROP_PARAM );
    if( prop == null || prop.length() == 0 )
```

```
    prop = "FILE_FIRST_LINE";
}
```

This code uses a utility class in **lisa-util-*release#*.jar** named **com.itko.util.XMLUtils**. This class finds child tags of the given parent tag and reads the child text of the tag. This approach is useful because DevTest automatically writes the XML representation of filters by making each of the **Parameter** objects in **getParameters** a child tag of the Filter tag. Each parameter key becomes the tag name and the child text of the tag is the value.
If a filter defines the **prop** key, then the default name of the filter is **{{propValue}}.** **propValue** is the value of the **prop** parameter. If a filter does not define the **prop** key, then the default name of the filter is the string that the **getTypeName** method returned. The FilterFileFirstLine sample class defines this key as:

```
static private String PROP_PARAM = "prop";
```

5. Because the filters execute before and after the test step, you get two chances to implement the filter logic.
   Implement the filter logic before node execution with the **subPreFilter** method. The **TestExec** parameter provides access to the test environment, such as logs and events. If the filter sets a new node to execute, the Boolean return type returns **true**. Otherwise, it returns **false**.
   In this example, we are not interested in filtering before the node executes, so the **subPreFilter** method does nothing.

```
public boolean subPreFilter( TestExec ts ) throws TestRunException
{
    // don't have anything to do...
    return false;
}
```

   Implement the filter logic after node execution with the **subPostFilter** method. The **TestExec** parameter provides access to the test environment, such as logs and events. If the test should continue as normal, the Boolean return type returns **false**. Otherwise, it returns **true**.
   In this example, we store the first line of the file that is returned from the node in the new property. ftp:// is prepended if the **isFTP** box is selected.
   A DevTest user can use a filter at either the test step level or the test case level. Add logic to the filter that verifies whether the result is the proper state to run the filter. For example, if your filter assumes that the LASTRESPONSE holds a FileDataObject, then verify that before executing the filter logic.

```
public boolean subPostFilter( TestExec ts ) throws TestRunException
{
    try {
        Object oresponse = ts.getStateObject( "LASTRESPONSE" );
        if (!(oresponse instanceof FileDataObject))
            return false;
        FileDataObject fdo = (FileDataObject)oresponse;
        String firstline = fdo.getFileFirstLine();
        if((firstline ==null) || (firstline.equals("")) {
            ts.setStateValue( prop, "" );
            return false;
        }
        if( isftp )
            firstline="ftp://" + firstline;
        ts.setStateValue( prop, firstline );
        return false;
    }
    catch( Exception e )
    {
        throw new TestRunException
( "Error executing FilterFileFirstLine", e );
    }
}
```

Deploy a Filter

You must make a custom filter available in the model editor before you can use it in a test case.

Follow these steps:

1. Tell DevTest to look for a new custom filter in a **lisaextensions** file, as:

   ```
   filters=com.mycompany.lisa.FilterFileFirstLine
    com.mycompany.lisa.FilterFileFirstLine=com.itko.lisa.editor.
   FilterController,com.itko.lisa.editor.DefaultFilterEditor
   ```

   You can also add this filter to the wizards in the **wizards.xml** file.

2. Copy the JAR file that contains your custom filter and **lisaextensions** file to the **LISA_HOME\hotDeploy** directory.
   If your custom filter depends on any third-party libraries, copy those libraries to the **LISA_HOME\hotDeploy** directory.
   In this example, the FilterFileFirstLine described previously has already been packaged for you in **lisaint-examples.jar**. You can download an **examples.zip** file that contains this file from Using the SDK. This custom filter does not depend on any third-party libraries.

3. If you are already running DevTest, exit and restart the program for this new setting to take effect.

Define and Test a Filter

This procedure describes defining a new filter.

Follow these steps:

1. Change the **Type** of the filter, selecting the text that you specified in the **getTypeName** method.

2. Set the parameters to the custom filter.
   For each parameter in the **Filter Attributes** section, supply a value.

3. Test a custom filter like you would any built-in filter.

Custom Report Generators

This section explains how to extend DevTest Solutions with new reports.

Create a Report Generator

DevTest provides built-in support for custom report generators. The predefined report generators provide output for most situations. However, you can create your own report generator to handle site-specific situations.

Follow these steps:

1. Create a Java class that extends **com.itko.lisa.coordinator.ReportGenerator**.
   This class tells DevTest that your class is a custom report.

   ```
   public class ReportEventsToFile extends ReportGenerator
   {
   }
   ```

2. Implement the required **getTypeName** method.
   This method provides the name that is used to identify the custom report in the Staging Document Editor.

   ```
   public String getTypeName()
   {
       return "Report Events To a File";
   }
   ```

3. Define the parameters to the report.
   For each item in the **Report Attributes** section of the **Reports** tab in the Staging Document Editor. Add a **Parameter** to the **ParameterList** for the report.

4. Initialize the custom report generator object with the **ParameterList** given.
   When DevTest tries to execute a report, it first creates an instance of the custom class. It then calls the initialize method, passing the **ParameterList** that was read from the XML of the staging document. DevTest automatically reads and writes the XML representation of report attributes by making each of the **Parameter** objects in **getParameters** a child tag of the report XML tag. Each parameter key becomes the tag name and the child text of the tag is the value.

5. While the test is running, DevTest invokes the pushEvent method for every event you have not filtered.

6. Implement the **finished** method.
   When DevTest finishes the test, it invokes this method on your report generator. This invocation is your opportunity to complete your processing, like saving the current document.

Deploy a Report Generator

You must make a custom report available before you can use it in a staging document.

Follow these steps:

1. Tell DevTest to look for a new custom report generator in a **lisaextensions** file, as:

   ```
   reportGenerators=com.mycompany.lisa.ReportEventsToFile
   ```

   You can also add the report through the **lisa.properties** file, using **lisa.editor.reportGenerators** key.

2. Copy the JAR file that contains your custom report and the **lisaextensions** file to the **LISA_HOME\hotDeploy** directory.
   If your custom report depends on any third-party libraries, copy those libraries to the **LISA_HOME\hotDeploy** directory.

3. If you are already running DevTest, exit and restart the program for this new setting to take effect.

Use a Report Generator

To use a custom report in the Staging Document Editor, access it the same way as any built-in report.

Custom Report Metrics

This page explains how to extend the DevTest Solutions with a new reporting metric.

Create a Report Metric

When a test case is staged, a subsystem within DevTest samples metric values. DevTest then reports them in the ways that are defined in the staging document. The staging document includes the metrics to be collected. Users can also add them as the test runs.

Two classes must be created for the metric collection:

- The **Metric Integrator** provides a way to view and edit the metrics that you want to collect.
- The **Metric Collector** is the engine that samples values while a test is running.

Follow these steps:

1. Create a Java class that implements **com.itko.lisa.stats.MetricIntegration**.
   This class provides the metrics that you want to access during the staging of a test.

   ```
   public class RandomizerMetricIntegration implements MetricIntegration
   {
   }
   ```

2. Implement the required **getTypeName** method.
   This method that provides the name that is used to identify the custom report metric type in the Staging Document Editor.

   ```
   public String getTypeName()
   {
       return "Randomizer Metric";
   }
   ```

3. Implement the **public MetricCollector[] addNewCollectors(Component parent)** method so that at design time your code can define the requested metrics to be collected.

4. Create a Java class that extends **com.itko.lisa.stats.MetricCollector**.
   DevTest calls on instances of this class to collect the requested metrics. The class must also implement **Serializable**.

   ```
   public class RandomMetricCollector extends MetricCollector implements java.
   io.Serializable
   {
   }
   ```

5. Complete the implementation of your **MetricIntegration** and **MetricCollector** objects.
   See the JavaDocs in the **doc** folder of your installation directory for those classes and the sample code for **RandomizerMetricIntegration** and **RandomMetricCollector** for more information about the individual methods available to extend.

Deploy a Report Metric

You must make a custom report metric available before you can use it in a staging document.

Follow these steps:

1. Tell DevTest to look for a new custom report metric in a **lisaextensions** file, as:

```
metrics=com.mycompany.lisa.metric.RandomizerMetricIntegration
```

You can also add the report metric through the **lisa.properties** file, using **stats.metrics.types** key.

2. Copy the JAR file that contains your metric and the **lisaextensions** file to the DevTest hotDeploy directory at **LISA_HOME\hotDeploy**.
   If your custom metric depends on any third-party libraries, copy those libraries to the **hotDeploy** directory.

3. If you are already running DevTest, exit and restart the program for this new setting to take effect.

Custom Companions

This section explains how to extend DevTest Solutions with a new companion.

You can create custom companions in two different ways. Native companions are created in much the same way that Native test steps are created. And somewhat like the Custom Java test step, there is a simpler way in which DevTest shields you from most of the editor and serialization overhead. That approach is documented here.

Note: If a companion contains duplicate parameter values, the duplicates are filtered out when you save the test case. You must close the test case and then reopen it to see the change in the user interface. This problem is specific to custom companions because of an issue with the implementation of the lifecycle of custom companions. If you have a test case containing a custom companion that wrote duplicate parameters into the .tst file, there is a workaround. Reopen the .tst file, click save, close the .tst file, and then reopen it. The duplicates are removed during the save, but the companion UI does not update to reflect this change. The close and reopen step is required to see the change.

Create a Companion

ge is unversioned. It is available and contains the same content in all versions of this space.

You can create a companion that extends the predefined SimpleCompanion.

Follow these steps:

1. Create a Java class that extends **com.itko.lisa.test.SimpleCompanion**.
   This class provides the information that is required to create, edit, and execute your companion logic.

   ```
   public class AllowedExecDaysCompanion extends SimpleCompanion implements
   Serializable
   {
   }
   ```

   Implement Serializable when your companion is used in remotely staged tests.

2. Implement the required **getTypeName** method.
   This method provides the name that identifies the companion in the model editor.

```
public String getTypeName()
{
    return "Execute Only on Certain Days";
}
```

3. Implement the **getParameters** method.
   This method provides any parameters needed for your companion to execute. Also call the superclass implementation of this method. The model editor allows you to edit the parameters that are shown here. These parameters are given to you at the time of execution. Implement this method only if your companion requires parameters.

```
public ParameterList getParameters()
{
    ParameterList pl = super.getParameters();
    pl.addParameter( new Parameter( "Allowed Days (1=Sunday): ", DAYS,
"2,3,4,5,6", String.class ) );
    return pl;
}
```

4. Implement the **testStarting** method.
   DevTest calls this method with the parameters you request. If an error occurs or you otherwise want to prevent the test from executing normally, throw a **TestRunException**.

```
protected void testStarting( ParameterList pl, TestExec testExec )
    throws TestRunException
```

5. Implement the **testEnded** method.
   DevTest calls this method with the parameters you request. You have an opportunity to perform any post-execution logic for the test.

```
protected void testEnded( ParameterList pl, TestExec testExec )
    throws TestRunException
```

6. (Optional) Implement the **setLongMsg** method to integrate into Portal Test Monitoring facility.
   You can provide a detailed message for the companion.

```
public void setLongMsg( boolean isStarting, String msg )
```

Deploy a Companion

Companions must be explicitly declared to DevTest at startup so that the authoring framework can make the companion available for use in test cases.

Like the Native Test test step, three classes are required for companions, but DevTest provides default implementations for the two classes that are not documented here. The default controller is **com.itko.lisa.editor.CompanionController** and the editor is named **com.itko.lisa.editor.SimpleCompanionEditor**. Notice that the built-in companion named **Set Final Step to Execute Companion** is defined with these classes. Use the registration for that companion as a sample. They are connected in the **lisaextensions** file.

```
companions=com.mycompany.lisa.AllowedExecDaysCompanion
com.mycompany.lisa.AllowedExecDaysCompanion=com.itko.lisa.editor.
CompanionController,com.itko.lisa.editor.SimpleCompanionEditor
```

For more information about the **lisaextensions** file, see Extending DevTest Solutions.

As with all custom test elements, you must make the classes that you have developed and the **lisaextensions** file available to DevTest. The most common way to do make them available is to put a JAR file in the **LISA_HOME\lib** directory.

Note: A custom companion can implement the **StepNameChangeListener** interface and can receive notification when the name of any step is changed. An SDK developer implements this if your custom companion renders a drop-down list of the steps in the test. An example is the **Final**

Step to Execute companion, which lists the step names so a test author can select the teardown step. This code change was added to enable notifying the **Final Step to Execute** companion when a step name changes.

Using Hooks

This section explains how to extend DevTest Solutions with a new hook.

A *hook* is a mechanism that includes test setup logic or test teardown logic for all the tests running in DevTest. An alternative definition of a hook is a system-wide companion.

Hooks are used as follows:

- To configure test environments.
- To prevent the execution of tests that are not properly configured or do not follow defined best practices.
- To provide common operations.

Anything that a hook can perform can also be modeled as a companion. However, there are several differences between hooks and companions:

- Hooks are global in scope. Users do not specifically include a hook in their test case as is the required practice for companions. If you need every test to include the logic and want to prevent users from accidentally not including it, a hook is preferable.
- Companions can have custom parameters and are rendered in the model editor. Hooks are practically invisible to the user and therefore can request no special parameters. Hooks get their parameters from properties in the configuration or from the system.
- Hooks are deployed at the install level, not at the test case level. Assume a test is run on two computers. One computer has a hook that is registered, and the other does not. The hook runs only when the test is staged on the computer where it is explicitly deployed. The defined companions execute regardless of any install-level configuration.

Create a Hook

Follow these steps:

1. Create a Java class that extends **com.itko.lisa.test.Hook**.
   This class provides the information that is required to execute the logic for your hook.

   ```
   public class HeadlineHook extends Hook
   {
   }
   ```

2. Implement the **startupHook** method.
   DevTest calls this method when the test starts. If an error occurs or you otherwise want to prevent the test from executing normally, throw a **TestRunException**.

   ```
   public void startupHook( TestExec testExec ) throws TestRunException
   ```

3. Implement the **endHook** method.
   You have an opportunity to perform any post-execution logic for the test.

   ```
   public void endHook( TestExec testExec )
   ```

4. (Optional) Implement the **setLongMsg** method to integrate into Portal Test Monitoring facility.
   You can provide a detailed message for the hook.

   ```
   public void setLongMsg( boolean isStarting, String msg )
   ```

Deploy a Hook

Hooks are deployed when a class name in the **lisa.properties** file registers them. The system property key that is used for hooks is lisa.hooks. An example follows:

```
# to register hooks with LISA, these are comma-separated
lisa.hooks=com.itko.lisa.files.SampleHook,com.mycompany.lisa.HeadlineHook
```

The preceding **lisa.properties** entry deploys two hooks to be run on every test.

Custom Data Sets

This section explains how to extend DevTest Solutions with a new data set.

Data sets are created in much the same way that test steps are created. DevTest provides a simpler way that shields you from most of the editor and serialization overhead. This approach uses default class implementations for the controller and editor. This simpler approach is documented here.

Data Set Characteristics

Data sets are different from every other extension mechanism in DevTest in that they are inherently remote server objects. Consider a load test that has thousands of virtual users all trying to access the same spreadsheet file. DevTest must create a single object that is serving the spreadsheet for all those virtual users to read.

DevTest provides the infrastructure for remoting your custom data set automatically. There are typically no specific issues that are related to this unique characteristic other than the following: Because they are shared, they have no access to an individual test case or test execution state. This means that you do not have access to a **TestCase** or **TestExec** object. Your class is run in the address space of the coordinator that is staging the test, not necessarily the simulator that is communicating with the system under test.

Create a New Data Set

Follow these steps:

1. Create a Java class that extends **com.itko.lisa.test.DataSetImpl**.
   This class provides the information that is required to execute your data set logic.

   ```
   public class SomeDataSet extends DataSetImpl
    {
    }
   ```

   Your data set object is a Remote RMI object, and is therefore able to throw a **RemoteException** from its constructor and some methods.

2. Implement the required **getTypeName** method.
   This method provides the name that is used to identify the companion in the Test Case Editor.

   ```
   public String getTypeName()
    {
   ```

```
 return "Nifty Data Set";
 }
```

The string that the **getTypeName** method returns is the default name of a new data set.

3. Implement the **getParameters** method.
   This method provides the parameters needed for your companion to execute. You must also call the super class implementation of this method. The Test Case Editor allows you to edit the parameters that are shown here. The parameters are given to you at the time of execution. Implement this method only if your companion requires parameters.

   ```
   public ParameterList getParameters() throws RemoteException
   {
   ParameterList pl = super.getParameters();
   // ...
   return pl;
   }
   ```

   For more information about Parameters and ParameterLists, see Extending DevTest Solutions.

4. Implement the two required **initialize** methods.
   These methods are provided so that the data set can be initialized from either XML or the ParameterList system within DevTest.

   ```
   public void initialize(Element dataset)
    throws TestDefException
    public void initialize(ParameterList pl, TestExec ts)
    throws TestDefException
   ```

5. Implement the **getRecord** method.
   DevTest calls this method when another row is needed from the data source for the data set. If an error occurs or you otherwise want to prevent the test from executing normally, throw a **TestRunException**.

   ```
   synchronized public Map getRecord()
    throws TestRunException, RemoteException
   ```

   If you are out of rows in the data source, you must specifically code for the two possible conditions that the user wants:

   - Restart reading the data source from the top again automatically, or
   - Return a null from this method to indicate that the data source is out of rows so that the test workflow reflects the condition.

   The following example shows possible psuedo-code for this function:

   ```
   Read next row
    If no-next-row, Then
    If there is no "at end" parameter specified, Then
    Re-open the data source
    Read next row
    Else
    Return null
    Return row values
   ```

Note: The API for the DataSet interface includes the public **String getType()** method. This method returns the classname of the class implementing this interface.

Deploy a New Data Set

Data sets must be explicitly declared to DevTest at startup of DevTest Workstation so that the authoring framework can make the data set available for use in test cases. Like the Native Test Node, three classes are required for data sets. DevTest provides default implementations for the two classes that are not documented here. The default controller is **com.itko.lisa.editor.DataSetController** and the editor is named **com.itko.lisa.editor.DefaultDataSetEditor**. Notice that some of the built-in data sets use these classes.

They are connected in the **lisaextensions** file, as follows:

```
datasets=com.itko.examples.dataset.CSVDataSet
 com.itko.examples.dataset.CSVDataSet=com.itko.lisa.editor.DataSetController,com.
itko.lisa.editor.DefaultDataSetEditor
```

See the information about the **lisaextensions** file in Extending DevTest Solutions for more details on how to register your data set.

As with all custom test elements, make the classes that you have developed and the **lisaextensions** file available to DevTest. The most common way to make them available is to bundle them in a jar file that is placed in the hot deploy directory.

Using Jenkins Plugin for CA Service Virtualization

The CA Service Virtualization plugin for Jenkins lets you manage virtual services and provision tests on CA Service Virtualization Enterprise Edition from your Jenkins UI.

Visit the DevTest Community pages to provide feedback and receive community support.

This section contains the following information:

- Prerequisites
- Install
- Configure
- Features
  - Add a new build step
  - Configure existing build steps
  - Use a custom registry endpoint
  - Deploy Virtual Services
  - Deploy Test Cases and Test Suites
  - Start, Stop, or Undeploy Virtual Services
- Pipeline Support
  - Deploy Virtual Service
  - Deploy and Run Test
  - Stop Virtual Services
  - Start Virtual Services
  - Undeploy Virtual Services

Prerequisites

The plugin has the following prerequisites:

- DevTest 10.3 installation or later.
- Jenkins 2.73.3 or later.

Install

The following link will take you to the Jenkins page for CA Service Virtualization:

CA Service Virtualization Jenkins Plugin

> **Note:** The page above includes a link to GitHub. You can download the source code from GitHub if you wish to fork the project and make changes to the code. Otherwise use the Plugin Manager to install the plugin from Jenkins.

Install this plugin with the Jenkins Plugin Manager.

For details on how to use the Jenkins Plugin Manager, see the Jenkins documentation page: Managing Plugins.

Configure

Follow these steps:

1. In the Jenkins Plugin window, select **Manage Jenkins** and then select **Configure System.**
2. Complete the following fields under **CA Service Virtualization Configuration**:

- Registry Host URL (Registry IP address/hostname).
- Port (Port of Registry endpoint).
- Registry Credentials. Use the Jenkins credentials plugin (username/password credentials).
- Select the **Use https** checkbox if your DevTest API is set up to use https security.

> **Note:** The host and port information can be found in the LISA properties file. The username and password are stored in the database.

3. Click **Test Connection.**
4. Click **Save.**

Note: Individual projects can override the global connection settings using the custom registry endpoint option.

Features

The Features of the Jenkins Plugin for CA Service Virtualization can be configured in the **Build** settings of the plugin.

Add a new build step

1. Select **Add build step** under the **Build** section.
2. Choose the step you want to add from the drop-down list.
3. Define the parameters.

Configure existing build steps

1. Choose the existing item that you want to configure from the home screen of the Jenkins plugin.
2. Select the individual step and select **Configure**
3. Define the parameters under the **Build** section.

Use a custom registry endpoint

You can use a custom registry endpoint for each build step to override the default parameters. To use a custom registry, select the **Use custom registry endpoint** checkbox under the build step and complete the required fields.

Features Include:

- Deploy Virtual Services
- Deploy Test Cases and Test Suites
- Start, Stop, or Undeploy Virtual Services
- Pipeline Support

Note: All features support Jenkinks parameters. Select **This Project is parameterized** option in the project configuration to enable the jenkins parameters.

Deploy Virtual Services

To deploy a Virtual Service with a MAR file, enter the **VSE name** and the **MAR file** location.

Example paths to the MAR file:

- **dir/test.mar**
  Relative path to MAR file in job workspace.
- ****/*.mar**
  Wildcard format to match all MAR files within the recursive directory structure.
- **artifacts/*.mar**
  Wildcard format to match all MAR files within a specific directory.
- **file:///C:/something/test.mar**
  Points to the MAR file on file system of the DevTest machine.
- **http://something/test.mar**
  HTTP location of the MAR file.

Note: The MAR File field supports multiple strings delimited by either a separate line or a comma.

Deploy Test Cases and Test Suites

To deploy and start test cases and test suites, select the **Test Type** from the drop-down list and enter the MAR file path in the **MAR file** field.

The chosen Test Type must match the contents of the MAR file. For example, if the Test Type is set to **Test** the MAR file must contain test. If the Test Type is set to **testsuite** the MAR file must contain testsuite.

Here is how Test Case results in CA Service Virtualization map to Build Step results in Jenkins:

Test Case Result	Jenkins Build Step Result
Failed	UNSTABLE
FAILED_TO_STAGE | ABORTED	FAILURE
Others	PASSED

A failed test case maps to UNSTABLE instead of FAILURE because a single test case failure should not necessarily fail the whole build, especially when you could have multiple test cases running in different build steps.

Note: The MAR file must be located in the workspace of the job.

Note: To generate a report, select **CA Service Virtualization Report Publisher** from the drop down under **Post-Build Actions.**

Start, Stop, or Undeploy Virtual Services

To start, stop or undeploy Virtual Services, define the **VSE name** and the **VS names** for the chosen function.

Note: The **VS names** field supports multiple strings delimited by a separate line.

Pipeline Support

Pipeline scripts are also supported to configure the plugin and execute commands.

For more information about the Jenkins Pipeline available commands provided by the plugin see: CA Service Virtualization plugin Pipeline reference documentation.

See the following examples for each build step:

Deploy Virtual Service

Example 1: Deploy the virtual service by specifying a custom Registry endpoint (This also overides the global configuration for the Registry endpoint).

```
node {
    stage('SCM') {
        git credentialsId: 'gitCred', url: 'https://github.com/foo/bar.git'
    }
    stage('CA Service Virtualization Deploy VS') {
        // Overriding global configuration for DevTest Registry endpoint
        svDeployVirtualService marFilesPaths: 'jenkins-plugin/webservices-vs.mar',
vseName: 'VSE' , useCustomRegistry: true, host: 'DevTestHost.com', port:
'1505', secured: true, tokenCredentialId: 'DevTestCred'
    }
}
```

Example 2: Deploy the virtual service with the global configuration for the Registry endpoint.

```
node {
    stage('SCM') {
        git credentialsId: 'gitCred', url: 'https://github.com/foo/bar.git'
    }
    stage('CA Service Virtualization Deploy VS') {
        svDeployVirtualService vseName: 'VSE', marFilesPaths: 'http://share.com/vs.
mar\nfile:///C:/VS/webservices.mar'
    }
}
```

Deploy and Run Test

```
node {
    stage('SCM') {
```

```
        git credentialsId: 'gitCred', url: 'https://github.com/foo/bar.git'
    }
    stage('CA Service Virtualization Start Start') {
        svDeployTest testType: 'tests', marFilePath: 'folder/test.mar'
    }
    //Include the following lines if you want to generate test reports:
    stage('Generate test reports') {
        svPublishTestReport()
    }
}
```

Deploy and Run Test Suite

```
node {
    stage('SCM') {
        git credentialsId: 'gitCred', url: 'https://github.com/foo/bar.git'
    }
    stage('CA Service Virtualization Start TestSuite') {
        svDeployTest testType: 'suites', marFilePath: 'folder/testsuite.mar'
    }
    //Include the following lines if you want to generate test reports:
    stage('Generate test reports') {
        svPublishTestReport()
    }
}
```

Stop Virtual Services

```
node {
    stage('CA Service Virtualization Stop VS') {
        svStopVirtualService  vseName: 'VSE', vsNames: '''WebServicesModel'''
    }
}
```

Start Virtual Services

```
node {
    stage('CA Service Virtualization Start VS') {
        svStartVirtualService vseName: 'VSE', vsNames:
'''WebServicesModel\nwebservices-vs'''
    }
}
```

Undeploy Virtual Services

```
node {
    stage('CA Service Virtualization Undeploy VS') {
        svUndeployVirtualService vseName: 'VSE', vsNames:
'''WebServicesModel\nwebservices-vs'''
    }
}
```

Agents

DevTest Solutions includes the following agents that support various features in the product suite:

- The DevTest Java Agent is geared toward Java-based applications.
- The DevTest CICS Agent and the DevTest LPAR Agent are geared toward mainframe applications.

DevTest Solutions also includes the mainframe bridge, which is a component that enables the registry to communicate with the mainframe agents.

- DevTest Java Agent
- DevTest CICS Agent
- DevTest LPAR Agent
- DevTest Mainframe Bridge

DevTest Java Agent

The DevTest Java Agent is a piece of server-side technology that can be installed inside Java processes, including Java EE containers. The agent enables DevTest to control and monitor server-side activities.

The agent can do what most profilers do: monitor loaded classes/objects, CPU usage, memory usage, threads, track method calls, and so on. However, the agent works across multiple Java virtual machines and is used with DevTest to bring unique features to testing.

In particular, the agent provides visibility into the actions that a test or test step causes the servers to do behind the scenes. This capability can help identify bugs and bottlenecks. This capability is similar to what Application Insight does. However, it works across all protocols that the Java applications use without the need to instrument any code or even any configuration files.

The agent also supports CA Service Virtualization. The agent enables record and replay of traffic and method calls across protocols. The agent gives CA Service Virtualization complete control of the areas of the target application that you want to virtualize. The agent provides a unified framework to accomplish this functionality regardless of the protocol.

This section contains the following pages:

- Java Agent Architecture
- Java Agent Installation
- Start the Broker
- rules.xml File
- Adding a Method to Intercept
- Excluding from Interception and Virtualization
- Java Agent Auto-Response Generation
- Java Agent REST Application Programming Interfaces
- Developing Against the Java Agent
- Java Agent Extensions

- Load Balancers and Native Web Servers
- Java Agent Security
- Java Agent Log Files
- Transaction Dispatchers
- Troubleshooting the Java Agent

Java Agent Architecture

The DevTest Java Agent is a type of Java agent.

Java agents are programs that are embedded in a Java virtual machine (JVM). These programs can be designed to support many functions, such as collecting information about a running application or virtualizing parts of an application.

In the following graphic, the large container represents the JVM. The JVM includes an agent and the class loader, which loads Java class files at run time. The clouds represent resources that the Java application loads.

An agent is packaged in a JAR file. The JVM must be configured with the path to the JAR file. As of Java 1.5, you use a string that starts with **-javaagent** to specify the path and any options. For example:

```
-javaagent:C:\myagent.jar=option1=true,option2=false
```

A single JVM can include multiple agents.

Java Agent Components

The DevTest Java Agent has the following components:

- The agent itself

- The broker
- The consoles
- The database

The **agent** runs embedded in a Java process.

The **broker** is a hub that dispatches Java Message Service (JMS) messages between agents, consoles, and an embedded client.

The embedded client tracks network wide/agent-spanning properties, such as the set of agents, their open queues, or current network connections. As such, the embedded client can assemble partial transaction data into complete transactions for the benefit of consoles. The process of assembling partial transactions into complete transactions is referred to as *stitching*.

After the data is assembled, the data is sent to the consoles (short term) and persisted to the database for long-term storage.

This documentation ignores the distinction and refers to the embedded client as the broker.

The **consoles** are clients that get their finalized data from the broker (if recent) and the database (if older) for display and user interaction. A console does not need to be a GUI component. Consoles include coordinators, simulators, virtual service environments, and DevTest Portal.

Note: If the agents and the broker are on different computers, ensure that the system times are synchronized.

Maximum Transaction Size

By default, the largest transaction that the broker can process is 16 MB. If the broker receives a transaction that is more than 16 MB, the broker discards the transaction.

The **Transaction size limit** property is a broker property that lets you change the maximum size. For example, you can increase the size if you expect larger transactions. You can configure this property from the **Agents** window of the DevTest Portal. The value is in number of bytes. The name of this property in the **rules.xml** file is **lisa.broker.transaction.max.size**.

Java Agent Data Flow

The following graphic shows the Java agent components and how they interact.

The following Java Message Service (JMS) destinations specify the flow of data:

- The **lisa.agent.info** topic is carried over connections 1 and 2, produced by the agents and consumed by the broker and the consoles. This topic lets the broker and the consoles see which agents are currently online and what their basic properties are.
- The **lisa.agent.port** topic is carried over connection 1, produced by the agents and consumed by the broker. This topic lets the broker see the connections that are currently active between multiple agents.
- The **lisa.agent.api** topic is carried over connections 1 and 2, produced by the consoles and consumed (and replied to) by the agents. This topic allows the consoles to invoke agent APIs over JMS.
- The **lisa.broker.api** topic is carried over connection 2, produced by the consoles and consumed (and replied to) by the broker. This topic allows the consoles to invoke broker APIs over JMS.
- The **lisa.stats** topic is carried over connections 1 and 2, produced by the agents and consumed by the broker and the consoles. This topic gives the consoles an idea of what type of load the agents are currently under. This topic also lets the broker persist those to the database.
- The **lisa.vse** topic is carried over connections 1 and 2, produced by the agents and consumed by the consoles. When VSE is enabled, the consoles receive VSE frames (and reply to them in playback mode).
- The **lisa.tx.partial** queue is carried over connection 1, produced by the agents and consumed by the broker. When an agent captures a partial transaction (all the frames that happen in its JVM), the agent sends it to the broker for assembly.
- The **lisa.tx.full** topic is carried over connection 2, produced by the broker and consumed by the consoles. When the broker finishes assembling the partial transactions that are received over **lisa.tx.partial**, the broker sends the full transactions to the consoles.
- The **lisa.tx.incomplete** topic is carried over connection 2, produced by the broker and consumed by the consoles. This topic is similar to **lisa.tx.full**, but is used for transactions that are not fully completed within the allowed timeout.

- **JDBC** connection 3 is used when the broker saves **StatsFrame** objects or fully assembled **TransactionFrame** objects.
- The consoles use **JDBC** connection 4 to perform their queries for transactions or statistics that are no longer held in memory.

The startup order of agents, broker, and consoles does not matter because all communications are asynchronous and can reestablish themselves automatically. This concept is especially important for agents to avoid performance issues because the broker goes down, for example.

When an agent comes online, the agent starts pulsing its information over the **lisa.agent.info** topic at regular, short intervals.

If a broker is not available, the agent does not try to notify any listeners of anything else until a connection is established or reestablished.

If the broker is available, all interested parties are quickly notified of the online agents. The broker and consoles keep a running list of those agents. When they stop pulsing their information, they are expired and removed from the list after a while.

Agent Online Notification

When the agent finishes the startup process, it immediately sends a notification to the broker. Other components can then change the status of the agent. For example, a virtual service environment can put the agent in playback mode.

You can force the agent to wait before sending the notification to the broker. Add the **lisa.agent.agent.report.online.delay** property to the agents section of the **rules.xml** file. Set the value to the number of seconds to wait.

The following example sets the wait time to 60 seconds:

```
<property key="lisa.agent.agent.report.online.delay" value="60"/>
```

Java Agent Database Schema

The broker automatically creates the database schema for the DevTest Java Agent.

To create the schema manually, obtain the DDL statements from one of the following files in the **LISA_HOME\database** directory:

- **db2_cai.ddl**
- **derby_cai.ddl**
- **mysql_cai.ddl**
- **oracle_cai.ddl**
- **sqlserver_cai.ddl**

An alternate approach for obtaining the DDL statements is to run the following command:

```
java -jar LisaAgent.jar -ddl <oracle|sqlserver|mysql|derby|db2> <output-file-name>
```

If you do not include the file name, the DDL statements are displayed on the command line only.

Typically, you create the schema manually for security or process reasons, or possibly migration or documentation concerns.

Java Agent Data Categories

The DevTest Java Agent can capture the following categories of data:

- Client
- EJB
- GUI (AWT, Swing, SWT)
- Java
- JCA
- JDBC
- JMS
- Logging
- REST
- RMI
- SAP
- Thread: a synthetic frame that shows stitching across threads
- Throwable
- TIBCO ActiveMatrix BusinessWorks
- webMethods Integration Server
- WebSphere MQ
- Web (HTTP/S)
- Web service

Java Agent Installation

This section describes how to install the DevTest Java Agent.

The DevTest Java Agent has two versions:

- Pure Java agent
- Native agent

The pure Java agent has no platform-dependent code.

The native agent requires a platform-dependent library module.

You can download DevTest Java Agent files from the DevTest Portal.

Important! Install the pure Java agent unless you need to create TIBCO BusinessWorks baselines. This feature requires the native agent.

More Information:

- Install the Pure Java Agent
- Install the Native Agent
- Options for Agent Parameters String
- Java Agent Install Assistant
- Platform-Specific Agent Configuration
- Wily Introscope Agent

Install the Pure Java Agent

This page describes how to install the pure Java version of the DevTest Java Agent.

Note: Investigate the target environment ahead of time and ensure that it has been tested, or test it yourself. Most Java agent issues are operating system or JVM-dependent, instead of application-dependent. Be sure to follow the instructions in this documentation. If that does not help, review Troubleshooting the Java Agent before contacting Support.

You can download DevTest Java Agent files from the DevTest Portal.

Follow these steps:

1. Obtain the following files from the **LISA_HOME\agent** directory:
   - **LisaAgent.jar**
   - **InsightAgent.jar** (for all platforms other than Oracle WebLogic Server)
   - **LisaAgent2.jar** (for Oracle WebLogic Server)
2. Place the files anywhere on your disk that has read permissions from the Java application and is not automatically loaded in the application classpath (such as **\WEB-INF\lib** directories).

3. Go to the platform-specific section and follow the instructions for the platform that you are using.

    a. For all platforms other than Oracle WebLogic Server, you will specify an agent parameters string in the following format:

    ```
    -javaagent:<path_to_InsightAgent.jar>=name=<agent_name>[,
    url=<broker_url>]
    ```

    b. For Oracle WebLogic Server, you will specify an agent parameters string in the following format:

    ```
    -javaagent:<path_to_LisaAgent2.jar>=name=<agent_name>[,
    url=<broker_url>]
    ```

4. Start the Java application.

Install the Native Agent

This page describes how to install the native version of the DevTest Java Agent.

> **Note:** Investigate the target environment ahead of time and ensure that it has been tested, or test it yourself. Most Java agent issues are operating system or JVM-dependent, instead of application-dependent. Be sure to follow the instructions in this documentation. If that does not help, review Troubleshooting the Java Agent before contacting Support.

If you want to create TIBCO BusinessWorks baselines, install the native agent. In addition, add the **heap** option to the agent parameters string and set the value to true.

You can download DevTest Java Agent files from the DevTest Portal.

The following table contains the file names of the platform-dependent library modules for the native agent:

Module	File Name
Windows (32-bit JVM)	**JavaBinder.dll**
Windows (64-bit JVM)	**JavaBinder.amd64.dll**
OS X (x86/x64) 32/64-bit	**libJavaBinder.jnilib**
Linux (x86) 32-bit	**libJavaBinder.x86.so**
Linux (x64) 64-bit	**libJavaBinder.x64.so**
Solaris (x86) 32-bit	**libJavaBinder.solaris.x86.so**
Solaris (x64) 64-bit	**libJavaBinder.solaris.x64.so**
Solaris (Sparc) 32-bit	**libJavaBinder.solaris.sparc32.so**
Solaris (Sparc) 64-bit	**libJavaBinder.solaris.sparc64.so**
HP-UX (RISC) 32-bit	**libJavaBinder.hp-ux.sl**

Follow these steps:

1. Obtain the following files from the **LISA_HOME\agent** directory:

- **LisaAgent.jar**
- The platform-dependent library module for the target environment.

2. Place the files anywhere on your disk that has read permissions from the Java application and is not automatically loaded in the application classpath (such as **\WEB-INF\lib** directories).
3. Go to the platform-specific section and follow the instructions for the platform that you are using. You will specify an agent parameters string in the following format:

   ```
   -agentpath:<path_to_JavaBinder_file>=jar=file:<path_to_LisaAgent.jar>,
   name=<agent_name>[,url=<broker_url>]
   ```

4. Start the Java application.

Options for Agent Parameters String

When you install the DevTest Java Agent, you specify an agent parameters string. This page describes the options that can be included in the string.

> **Important!** The **name** option is required.

- **name**
  Defines the name of the agent. Use the following format:

  ```
  name=agent-name
  ```

  Be sure to provide a descriptive name.
  Avoid using the same name for multiple agents.

- **url**
  Defines the broker to which the agent connects. Use the following format:

  ```
  url=tcp://broker-host:broker-port
  ```

  Set the **broker-host** portion to the name or IP address of the computer where the broker is deployed.
  Set the **broker-port** portion to the TCP port on which the broker is listening. The default value is 2009.

- **token**
  Makes access to the agent secure. Use the following format:

  ```
  token=admin-token:user-token
  ```

  For more information, see Java Agent Security.

- **heap**
  The **heap** option is available only with the native agent.
  Specifies whether to enable the heap-walking APIs, which are required for TIBCO BusinessWorks baselines. The valid values are true and false. The default value is false.

- **jar**
  The **jar** option is available only with the native agent.
  Defines the path to the **LisaAgent.jar** file.

Java Agent Install Assistant

The Java agent install assistant is a command-line utility that helps you determine the value for the agent parameters string.

The assistant also verifies that the JVM runs properly with the agent installed.

On the JBoss Fuse platform, the assistant also adds the boot delegation information.

The assistant is supported on all platforms except IBM WebSphere Application Server 7.0 on AIX 6.1.

> **Note:** If you intend to create TIBCO BusinessWorks baselines, add the **heap** option to the agent parameters string and set the value to true.

- Automatic Configuration
- Run the Java Agent Install Assistant

Automatic Configuration

The Java agent install assistant provides an option to automatically configure the Java agent on platforms that are based on the user values in the **JavaAgentInstaller.xml** file. This file is a template that contains the platform and the version information that is supported by DevTest. The **JavaAgentInstaller.xml** file is located in the same directory as the **LisaAgent.jar** file.

This procedure describes the prerequisite steps that you must perform *before* starting the assistant.

Follow these steps:

1. Open the **JavaAgentInstaller.xml** file.
2. Locate the platform and operating system that you are using.
3. Define the environment variable, if applicable.
   The environment variable is embedded in ${}, for example **${JBOSS_HOME}**.
4. Replace any customized value, if applicable.
   The customized value is embedded in {}, for example:

   ```
   <ConfigFileName>${ORACLE_HOME}\user_projects\domains\{custom domain name}
   \bin\startWebLogic.cmd<ConfigFileName>
   ```

   Replace it with your project name:

   ```
   <ConfigFileName>${ORACLE_HOME}
   \user_projects\domains\lisa_domain\bin\startWebLogic.cmd</ConfigFileName>
   ```
5. Review all the predefined values and edit values that are based on your system.

Run the Java Agent Install Assistant

The following procedure has separate steps for entering the path of the **java** executable, the broker URL, and the agent name. However, you can specify any of these values in the second step instead by using the following options:

- **-jvm**
  Full path of the **java** executable

- **-broker**
  Broker URL

- **-name**
  Name of the Java agent

To determine if the operating system-specific diagnostic tools should be run, regardless of the status of the system verification, add the **-run-diagnostics** option.

To enable verbose output, add the **-verbose** option.

Follow these steps:

1. Go to the computer where you are installing the agent.
2. Run the **LisaAgent.jar** file with the **-ia** option.
   - If you are installing the pure Java agent, use **java -jar LisaAgent.jar -ia**.
   - If you are installing the native agent, use **java -jar LisaAgent.jar -native**.
3. When prompted, enter the path of the **java** executable. The default value is the path of the **java** executable that is executing.
4. When prompted, enter the broker URL. The default value is **tcp://localhost:2009**.
5. When prompted, enter the agent name. The default value is **{{x}}**, which means that a unique name is generated at runtime.
6. Review the output.
7. (Optional) To configure the agent automatically, enter **Yes** and follow the prompts.

For more information, see the following examples:

- Example 1: Pure Java Agent
- Example 2: Pure Java Agent Using Automatic Configuration
- Example 3: Native Agent

Example 1 - Pure Java Agent

This example is based on the pure Java agent. The automatic configuration option is not used.

```
java -jar LisaAgent.jar -ia -java
Enter the path of the Java executable [C:\Program Files\Java\jre7\bin\java]:


Enter the DevTest broker URL [tcp://localhost:2009]:

NOTE: If the DevTest agent name contains the character sequence

{{x}}

it will be replaced with a unique identifier.

Enter the DevTest agent name [{{x}}]:
```

```
=====

System verification complete.  You can manually add these options to the java comm
and-line to start the DevTest agent:

   -javaagent:C:\DevTest\agent\InsightAgent.jar=url=tcp://localhost:2009,name={{x}}

Do you need assistance to automatically configure the DevTest agent? Please enter
[Yes/Y], if no, enter any other letter, then enter
n


Please manually add these options to the Java command-
line to start the DevTest agent
```

Example 2 - Pure Java Agent Using Automatic Configuration

This example is based on the pure Java agent and WildFly 8.2. The automatic configuration option is used.

As shown in the example, you must set the environment variable *before* starting the install assistant.

```
set JBOSS_HOME=C:\wildfly-8.2.0.Final

java -jar LisaAgent.jar -ia -java -broker tcp://localhost:2009 -name myagent
Enter the path of the Java executable [C:\Program Files\Java\jre7\bin\java]:


=====

System verification complete. You can manually add these options to the java comma
nd-line to start the DevTest agent:

   -javaagent:C:\PROGRA~1\CA\DevTest\agent\InsightAgent.jar=url=tcp://localhost:
2009,name=myagent

Do you need assistance to automatically configure the DevTest agent? Please enter
[Yes/Y], if no, enter any other letter, then enter
y


Please enter the index in [] to choose the platform:
[1] WildFly, JBoss Application Server, JBoss Fuse
[2] IBM WebSphere App Server
[3] Oracle WebLogic Server
[4] TIBCO BusinessWorks
[5] WebMethods Integration Server Windows Service
[X] Exit
1

Unable to automatically detect platform version based on the predefined rules
Please choose the index in [] for your platform version
[1] Wildfly or JBoss 6.2 [2] JBoss 4.2 [3] JBoss Fuse [X] Manually Configure
1

Enter path for Java agent configuration file [C:\wildfly-8.2.0.
Final\bin\standalone.bat]:

Automatic configuration of the Java Agent is complete. Please start your service.

The previous configuration has been saved to file 'C:\wildfly-8.2.0.
Final\bin\standalone.bat.2016-24-15_11-24-55'

The current agent configuration is:
SET "JAVA_TOOL_OPTIONS=-javaagent:C:\PROGRA~1\CA\DevTest\agent\InsightAgent.
jar=url=tcp://localhost:2009,name=myagent %JAVA_TOOL_OPTIONS%"
```

Example 3 - Native Agent

This example is based on the Solaris (Sparc) 32-bit native agent.

```
$ java -jar dist/agent/LisaAgent.jar -ia -native tcp://localhost:2009 MyAgent
Enter the path of the 'java' executable [...]: /usr/jdk/instances/jdk1.6.0/bin/java

=====

System verification complete.  Add these options to the java command-
line to start the DevTest agent:

   -agentpath:/tmp/user/dist/agent/libJavaBinder.solaris.sparc32.so=jar=file:/tmp
/user/dist/agent/LisaAgent.jar,url=tcp://localhost:2009,name=MyAgent

Do you need assistance to automatically configure the DevTest agent? Please enter
[Yes/Y], if no, enter any other letter, then enter

n

Please manually add these options to the Java command-
line to start the DevTest agent
```

Platform-Specific Agent Configuration

This section contains platform-specific information for installing the DevTest Java Agent.

- Configure JBoss to Use the Java Agent
- Configure Jetty to Use the Java Agent
- Configure SAP NetWeaver to Use the Java Agent
- Configure TIBCO BusinessWorks to Use the Java Agent
- Configure WebLogic to Use the Java Agent
- Configure webMethods to Use the Java Agent
- Configure WebSphere to Use the Java Agent

Configure JBoss to Use the Java Agent

This page applies to the following products:

- JBoss Application Server 4.2
- JBoss Application Server 7.3
- JBoss Enterprise Application Platform 6.2
- WildFly 8.2
- JBoss Fuse 6.2

Note: JBoss Enterprise Application Platform 6.2 is built from JBoss Application Server 7.

You can use the agent install assistant to determine the value of the agent parameters string. For example:

```
-javaagent:C:\agent\InsightAgent.jar=name=JBossAgent,url=tcp://localhost:2009
```

JBoss AS 4.2

To configure JBoss AS 4.2 to use the DevTest Java Agent, edit the JBoss startup script.

Follow these steps:

1. Open the JBoss **run.bat** or **run.sh** file in a text editor.
2. Define the following property before calling the Java executable:

   ```
   set JAVA_TOOL_OPTIONS=<agent_parameters_string>
   ```

3. Save the file.

JBoss AS 7.3, JBoss EAP 6.2, and WildFly 8.2

To configure JBoss AS 7.3, JBoss EAP 6.2, or WildFly 8.2 to use the DevTest Java Agent, the startup script that you edit depends on the mode:

- (Standalone server) **standalone.bat** or **standalone.sh**
- (Managed domain) **domain.bat** or **domain.sh**

Follow these steps:

1. Open the startup script in a text editor.
2. Define the following property before calling the Java executable:

   ```
   set JAVA_TOOL_OPTIONS=<agent_parameters_string>
   ```

3. Save the file.

JBoss Fuse 6.2

To configure JBoss Fuse 6.2 to use the DevTest Java Agent, edit the JBoss startup script. The name of the script is **fuse.bat** or **fuse**.

The boot delegation information can be added to the **config.properties** file in the **JBOSS_HOME /etc** directory instead.

Follow these steps:

1. Open the startup script or the **config.properties** file in a text editor.
2. Add **com.itko.*** to the **org.osgi.framework.bootdelegation** property.
3. Save the file.
4. Open the startup script in a text editor.
5. Specify the agent parameters string.
6. Save the file.

The following startup script shows an example configuration.

```
setlocal
set DIRNAME=%~dp0%
set PF_AGENTDIR=C:\agent
set PF_BROKER=mycomputer-w7

set CLASSPATH=D:\pf\g_lisa\lisa-remote\JavaAgent\build-gradle\libs\InsightAgent.
jar;%CLASSPATH%
set BOOTDEL=-Dorg.osgi.framework.bootdelegation=com.itko.*,org.apache.karaf.jaas.
boot,sun.*,com.sun.*,javax.transaction,javax.transaction.*,org.apache.xalan.
processor,org.apache.xpath.jaxp,org.apache.xml.dtm.ref,org.apache.xerces.jaxp.
datatype,org.apache.xerces.stax,org.apache.xerces.parsers,org.apache.xerces.jaxp,
```

```
org.apache.xerces.jaxp.validation,org.apache.xerces.dom
set JAVA_TOOL_OPTIONS=-javaagent:%PF_AGENTDIR%\InsightAgent.jar=name=Jboss_Fuse,
url=tcp://%PF_BROKER%:2009 %BOOTDEL%

rem enable standalone mode

set FABRIC8_ENSEMBLE_AUTO_START = false
set FABRIC8_AGENT_AUTO_START = false

call "%DIRNAME%\karaf.bat" %*
```

Configure Jetty to Use the Java Agent

This page is for Jetty 8, 9.1, and 9.2.

To configure Jetty to use the DevTest Java Agent, specify the agent parameters string in the **JAVA_TOOL_OPTIONS** environment variable.

You can use the agent install assistant to determine the value of the agent parameters string.

```
set JAVA_TOOL_OPTIONS=<agent_parameters_string> "%JAVA_HOME%/bin/java.exe" -DSTOP.
PORT=28282 -DSTOP.KEY=secret -jar start.jar etc/jetty-logging.xml
```

Jetty 9.1 Startup Issue

If you try to start Jetty 9.1 and an error indicates that the system cannot find the file specified, you may have encountered the Jetty bug 425736.

The workaround is to install Java in a folder that does not contain a space.

This error was fixed in Jetty 9.2.

Configure SAP NetWeaver to Use the Java Agent

This page is for SAP NetWeaver 7.3 and 7.4.

Use either of the following approaches to configure the DevTest Java Agent:

- SAP NetWeaver Administrator
- AS Java Config Tool

In both approaches, you add a JVM parameter that contains information about the agent. You can use the agent install assistant to determine the required information.

SAP NetWeaver Administrator

You can configure the DevTest Java Agent on SAP NetWeaver by adding a JVM parameter from SAP NetWeaver Administrator.

Follow these steps:

1. Open the SAP NetWeaver Application Server Java web portal.
2. Click **SAP NetWeaver Administrator**.
3. Click the **Configuration** tab, and then click the **Infrastructure** subtab.
4. Click **Java System Properties**.
5. Click the **Additional VM Parameters** tab.

6. Add a parameter.

   - **Name**
     Specifies the location of the agent. Use the initial portion of the agent parameter string.
     Example: -javaagent:/tmp/user/dist/agent/LisaAgent2.jar

   - **Value**
     Specifies the agent configuration options.
     Example: url=tcp://172.31.255.255:2009,name=myagent

7. Click **Save**.

8. Restart the SAP server.

AS Java Config Tool

You can configure the DevTest Java Agent on SAP NetWeaver by adding a JVM parameter from the AS Java Config Tool.

Use this tool if the SAP NetWeaver Application Server Java web portal is down or unresponsive.

Follow these steps:

1. Open the AS Java Config Tool.

2. Select the instance.

3. Select the **VM Parameters** tab.

4. Click the **Additional** tab.

5. Add a parameter.

   - **Parameter Name**
     Specifies the location of the agent. Use the initial portion of the agent parameter string.
     Example: -javaagent:/tmp/user/dist/agent/LisaAgent2.jar

   - **Value**
     Specifies the agent configuration options.
     Example: url=tcp://172.31.255.255:2009,name=myagent

6. Click **File, Apply Changes**.

7. Restart the SAP server.

Configure TIBCO BusinessWorks to Use the Java Agent

This page is for TIBCO BusinessWorks 5.*x*.

To configure the agent for all applications, you edit the following file:

```
...\tibco\bw\5.x\bin\bwengine.tra
```

Editing this file does not affect applications that have already been created.

To configure the agent for an existing application, edit the following file:

```
...\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:

```
D:\tibco\tra\domain\ca-tibcobw\application\MyBWProjectXml\MyBWProjectXml-caUserCRUD.tra
```

In either file, add the following line:

```
java.extended.properties=<agent-parameters-string>
```

You can use the agent install assistant to determine the string. If you want to create TIBCO BusinessWorks baselines, be sure to add the **heap** option and set the value to **true**.

Configuration Difference for Windows Service

If the TIBCO process is running as a Windows service, you do not add the **java.extended.properties** line to the .tra file. Instead, you specify the required information in the Windows registry.

In the Windows registry editor, locate the registry key for the TIBCO BusinessWorks service for the application. The usual name is **domain-name.application-name**.

Under that registry key, locate the Parameters key. Under the Parameters key, create or edit a string value with the name **java.extended.properties**. Set the value data to the agent parameters string.

The following graphic shows an example of setting the value data.

Configure WebLogic to Use the Java Agent

This page is for Oracle WebLogic Server 10.3 and 12.1.1.

To configure Oracle WebLogic Server to use the DevTest Java Agent, use any of the following approaches:

- Administration console

- config.xml file
- Startup script

For each approach, you specify the agent parameters string as a server startup argument. You can use the agent install assistant to determine the required value. The following example is based on the pure Java agent:

```
-javaagent:/export/home/wls/agent/LisaAgent2.jar=url=tcp://172.24.255.255:2009,
name=wls
```

Administration Console

To follow the officially supported way to add arguments to the JVM, specify the agent parameters string from the WebLogic administration console.

Follow these steps:

1. Open the WebLogic administration console.
2. In the **Domain Structure** panel, expand the **Environments** node and click the **Servers** node.
3. Click the **Configuration** tab and the **Server Start** subtab.
4. In the **Arguments** field, add the agent parameters string.
5. Save your changes.

config.xml File

Another approach is to edit the central configuration file for the domain.

Follow these steps:

1. Go to the **WEBLOGIC_HOME\user_projects\domains\base_domain\config** directory.
2. Open the **config.xml** file.
3. Add the agent parameters string to the **<server>/<server-start>/<arguments>** node of the target server.

Startup Script

You can also edit the WebLogic startup script.

In the following example, the **JAVA_TOOL_OPTIONS** environment variable is used to set the agent parameters string. These lines are located in the **startWebLogic.sh** file after the Java version check and before the actual WebLogic invocation.

```
JAVA_TOOL_OPTIONS=<agent_parameters_string>
export JAVA_TOOL_OPTIONS
```

If the startup script has been customized, the **JAVA_TOOL_OPTIONS** environment variable is not used.

Configure webMethods to Use the Java Agent

This page is for webMethods Integration Server 9.0, 9.5, and 9.6.

Specify the agent parameters string in the **server.bat** or **server.sh** file. You can use the agent install assistant to determine the required value. The following example is based on the pure Java agent:

```
set JAVA_TOOL_OPTIONS=-javaagent:E:/agent/InsightAgent.jar=url=tcp://172.
24.255.255:2009,name=wm
```

On UNIX platforms, use the **export** command instead of the **set** command:

```
export JAVA_TOOL_OPTIONS=-javaagent:/opt/CA/agent/InsightAgent.jar=url=tcp://172.
24.255.255:2009,name=wm
```

Follow these steps:

1. Locate the **bin** directory in the webMethods Integration Server installation.
2. Open the **server.bat** or **server.sh** file.
3. Add the agent parameters string before the line that calls the **java** executable.
4. Save the file.

Windows Service

If webMethods Integration Server is defined as a Windows service, you can edit the **server.bat** file by inserting the agent parameters string in the **/jvmargs** value that is defined in the **if "1%1" =="1-service"** switch. For example:

```
"%IS_DIR%\bin\SaveSvcParams.exe" /svcname %2 /jvm "%JAVA_DIR%\.." /binpath "%
PATH%" /classpath %CLASSPATH% /jvmargs "-javaagent:c:/DevTest/agent/InsightAgent.
jar=name=MyIS %JAVA2_MEMSET%" /progargs "%IS_DIR%\bin\ini.cnf"#"-service %2"#%
PREPENDCLASSES_SWITCH%#%PREPENDCLASSES%#%APPENDCLASSES_SWITCH%#%APPENDCLASSES%#%
ENV_CLASSPATH_SWITCH%#%ENV_CLASSPATH%#%3#%4#%5#%6#%7#%8#%9
```

Configure WebSphere to Use the Java Agent

This page is for IBM WebSphere Application Server 7.0 and 8.5.

To configure IBM WebSphere Application Server to use the DevTest Java Agent, use any of the following approaches:

- server.xml file
- Administrative console
- wsadmin tool

For each approach, you specify the agent parameters string as a generic JVM argument. You can use the agent install assistant to determine the required value. The following example is based on the pure Java agent:

```
-javaagent:/home/itko/agent/InsightAgent.jar=name=was70_linux32,url=tcp://172.
24.255.255:2009
```

Note: In the agent parameters string, do not specify a file path that includes a space. For example, do not include the file path **C:\Program Files\CA\DevTest\agent\InsightAgent.jar**.

server.xml File

In this procedure, you use the **server.xml** file to specify the agent parameters string.

Follow these steps:

1. Go to the **WAS_HOME/AppServer/profiles/AppSrv01/config/cells/<cell_name>/nodes /<node_name>/servers/server1** directory.
2. Open the **server.xml** file.
3. At the bottom of the file, change the **genericJvmArguments** entry.

```
<jvmEntries ... genericJvmArguments="<agent_parameters_string>" .../>
```

Administrative Console

In this procedure, you use the web-based Administrative console to specify the agent parameters string.

Follow these steps:

1. Go to the Administrative console.
2. Navigate to your application server.
3. Expand **Java and Process Management** and click **Process definition** on the Configuration tab.
4. Click **Java Virtual Machine** under **Additional Properties**.
5. Specify the agent parameters string in the **Generic JVM arguments** field. If you paste in the string, be sure to check whether the string has been truncated.

wsadmin Tool

You can use the **wsadmin** tool.

In the following example, the agent parameters string is specified with the modify command of the **AdminConfig** object. The Jacl scripting language is used.

```
C:\IBM\WebSphere70\AppServer\bin>hostname
cam-aa74651f617

C:\IBM\WebSphere70\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 "<agent_parameters_string>"

wsadmin>$AdminConfig save

wsadmin>quit
```

Wily Introscope Agent

> **Note:** This feature has been deprecated and is no longer supported.

The DevTest Java Agent can coexist with the Wily Introscope Agent.

The Wily Introscope Agent is a pure Java agent and is typically configured with the **-javaagent** syntax.

If you are using the **-javaagent** syntax for both agents, specify the DevTest Java Agent *before* the Wily Introscope Agent.

For a typical application server or container, the following example is appropriate:

```
set JAVA_OPTS=-javaagent:E:\DevTest\agent\InsightAgent.jar=url=tcp://localhost:
2009,name=DevTestAgent -javaagent:e:/wily/Agent.jar %JAVA_OPTS%
```

This example is also appropriate:

```
set JAVA_OPTS=-javaagent:e:/wily/Agent.jar %JAVA_OPTS%
set JAVA_TOOL_OPTIONS=-javaagent:E:\DevTest\agent\InsightAgent.
jar=url=tcp://localhost:2009,name=DevTestAgent
```

However, this example is incorrect:

```
set JAVA_OPTS=-javaagent:e:/wily/Agent.jar -javaagent:E:
\DevTest\agent\InsightAgent.jar=url=tcp://localhost:2009,name=DevTestAgent %
JAVA_OPTS%
```

Remember to perform the following somewhere along the line:

```
-Dcom.wily.introscope.agentProfile=e:/wily/core/config/IntroscopeAgent.profile
```

Start the Broker

If the broker is not installed as a Windows service, start the broker manually.

The broker must be installed on the same machine as the registry.

To view agents in the Enterprise Dashboard and DevTest Portal, the broker must be running.

Follow these steps:

1. Ensure that the registry is running.
2. Do one of the following actions:
   - Open a command prompt, navigate to the **LISA_HOME\bin** directory, and run the **Broker** executable.
   - If your installation has a Start menu folder, click **Start menu, All Programs, DevTest Solutions, Broker**.

rules.xml File

The configuration file for the DevTest Java Agent is named **rules.xml**.

The **rules.xml** file is located in the **LISA_HOME** directory on the computer where the broker is running. The **rules.xml** file is generated when the broker is started for the first time. All agents that are connected to the broker use the settings.

By default, the **rules.xml** file includes only a sample set of configuration properties. All the properties and their default values are maintained internally.

You can view these properties in a file named **rules.xml.sample**. The **rules.xml.sample** file is generated when the broker is started for the first time. The **rules.xml.sample** file is located in the same directory as the **rules.xml** file.

Each property includes a comment, the name, and the value. For example:

```
<property comment="Keep track of file descriptors" key="lisa.agent.tracking.
disabled" value="false"/>
```

The following XML shows the general format of the **rules.xml** file. The **agent** element contains the properties for an agent. The **broker** element contains the properties for the broker. The **console** element contains the properties for the consoles.

```
<?xml version="1.0" encoding="UTF-8"?>
<rules>
   <agent guid="0" name="A1">
      ...
   </agent>
   <broker>
      ...
   </broker>
   <console>
      ...
   </console>
</rules>
```

You can also place a **rules.xml** file on the agent side or the console side. The settings in these files override any information for that agent or console on the broker side.

The following graphic shows an example configuration. The broker accesses the **rules.xml** file that is on the same computer. Multiple agents and consoles are connected to the broker. One of the agents has its own **rules.xml** file.

The following properties can be configured only on the agent side because the agent reads them *before* making a connection to the broker:

- lisa.agent.agent.log
- lisa.agent.log.max.size
- lisa.agent.log.max.archives
- lisa.agent.security.manager.disabled
- lisa.agent.transaction.auto.start
- lisa.agent.transaction.auto.start.max.delay
- lisa.broker.encryption.token
- lisa.log.level

When Updates Are Applied

If you update the **rules.xml** file for the broker, you must restart the broker. The broker sends the new settings to the agents.

If you update the **rules.xml** file for an agent, you must restart the agent JVM.

Directives

You can add *directives* to the **rules.xml** file to perform certain functions. For information about the directives, see the following pages:

- **category** directive: Category Settings

- **database** directive: Configure a Database Sink
- **exclude** directive: Excluding from Interception and Virtualization
- **feature** directive: Protocol Weight Configuration
- **intercept** directive: Adding a Method to Intercept
- **loadbalancer** directive: Load Balancers and Native Web Servers
- **response** directive: Java Agent Auto-Response Generation
- **track** directive: Add a Class for Tracking
- **virtualize** directive: Instrumentation Rules for VSE

The **rules.xml.sample** file contains examples of the directives.

Managing Agents in Groups

You can create a *group* that contains two or more agents.

You can then apply configuration changes at the group level, instead of making the same changes to each agent individually.

> **Note:** An agent cannot belong to more than one group.

An agent that does not belong to a group is referred to as a *stand-alone* agent.

The following XML shows the general format of a **rules.xml** file that has a group. The **group** element contains the properties for the group. The **agent** element contains the properties for an agent. If the agent is part of a group, then the **agent** element appears within the **group** element. The **broker** element contains the properties for the broker. The **console** element contains the properties for the consoles.

```
<?xml version="1.0" encoding="UTF-8"?>
<rules>
   <group name="G1">
      ...
      <agent name="G1A1">
         ...
      </agent>
   </group>
   <agent guid="0" name="A1">
      ...
   </agent>
   <broker>
      ...
   </broker>
   <console>
      ...
   </console>
</rules>
```

If an **agent** element within a **group** element has one or more properties, the properties override the group settings.

Do not place the **broker** element or the **console** element inside the **group** element.

When you update an agent property from DevTest Portal, the setting is saved to the stand-alone agent section of the **rules.xml** file.

> **Note:** For detailed information about DevTest Portal, see Using Application Insight.

Example: Configure One Group in rules.xml File

The following XML shows a **rules.xml** file that has one group. The group has three agents. This file also has a stand-alone agent.

```
<?xml version="1.0" encoding="UTF-8"?>
<rules>
   <group name="group1">
      <property key="lisa.agent.jms.poll.int" value="5000"/>
      <property key="lisa.agent.transaction.auto.start.max.delay" value="60000"/>
      <property key="lisa.agent.virtualize.jit.enabled" value="false"/>
      <intercept class="com.itko.examples.entity.Account" method="
setName" signature="(Ljava/lang/String;)V"/>
      <agent name="agent1" guid="1234567">
         <property key="lisa.agent.stats.alarm.threshold.permgen" value="90"/>
         <property key="lisa.agent.stats.sampling.interval" value="1000"/>
      </agent>
      <agent name="agent2" guid="2345678" />
      <agent name="agent3" guid="3456789" />
   </group>
   <agent guid="-2032180703" name="DEFAULT">
      ...
   </agent>
   <broker>
      ...
   </broker>
   <console>
      ...
   </console>
</rules>
```

Protocol Weight Configuration

The **feature** directive lets you modify the capture level for each protocol that the DevTest Java Agent can capture.

> **Note:** Setting different capture levels is not supported for queue-based client and server communication, for example, WebSphere MQ and JMS.

You can also modify the capture levels from the DevTest Portal.

The **feature** directive has the following format:

```
<feature name="protocol_name" weight="weight"/>
```

The **feature** directive can be placed within the **group** element or the **agent** element of the **rules.xml** file.

Set the **weight** attribute to 0, 4, or 8. The value 0 corresponds to the **Counts** level. The value 4 corresponds to the **Counts and Paths** level. The value 8 corresponds to the **Full Data** level.

The following example sets the JDBC protocol to the **Full Data** level:

```
<feature name="JDBC" weight="8"/>
```

Signature Specification

The following directives in the **rules.xml** file include a signature specification:

- **exclude**
- **intercept**
- **respond**
- **virtualize**

You use the signature specification to describe the characteristics of a Java method signature.

The first portion of the specification is the word **signature**, followed by an equal sign and a quotation mark.

The second portion of the specification contains the arguments, surrounded by parentheses. If the method has no arguments, you must still include the parentheses.

The third portion of the specification contains the return type, followed by a quotation mark. If the return type is void, use the letter V.

Do not include any spaces within the specification.

To specify a primitive type in the arguments or the return type, use one of the following letters:

Letter	Primitive Type
Z	boolean
B	byte
C	char
D	double
F	float
I	int
J	long
S	short

To specify a fully qualified class, do the following steps:

- Add the letter L at the beginning.
- Use a forward slash as the separator, instead of a dot.
- Add a semicolon at the end.

For example:

```
Ljava/lang/String;
```

Example: One Argument, Returns Void

Assume that you want to intercept the **onMessage()** method of the **javax.jms.MessageListener** interface. This method has the following signature:

- The argument is a **javax.jms.Message** object.
- The return type is void.

The signature specification in the intercept rule would be:

```
signature="(Ljavax/jms/Message;)V"
```

Example: No Arguments, Returns Primitive Type

Assume that you want to intercept the **getPriority()** method of the **javax.jms.MessageProducer** interface. This method has the following signature:

- The method has no arguments.
- The return type is an integer.

The signature specification in the intercept rule would be:

```
signature="()I"
```

Instrumentation Rules for VSE

You can customize the VSE functionality by adding the **virtualize** directive to the **rules.xml** file of the agent.

The **virtualize** directive can be placed within the **group** element or the **agent** element of the **rules.xml** file.

Adding a Class for Virtualization (Both Recording and Playback Modes)

```
<virtualize class="class_name"/>
```

Example:

```
<virtualize class="javax.ejb.SessionBean"/>
```

Telling the Agent how to Determine What Constitutes a Session for a Given Protocol

You perform this task by providing a code snippet that returns a session identifier.

```
<virtualize>
   <track class="class_name" method="method_name" signature="signature" push="
true|false">
      <code><![CDATA[" and ends with "]]></code>
   </track>
</virtualize>
```

The format of the signature is described in Signature Specification.

Examples:

```
<virtualize>
   <track class="javax.servlet.http.HttpServlet" method="service" signature="
(Ljavax/servlet/http/HttpServletRequest;Ljavax/servlet/http/HttpServletResponse;)
V" push="false">
      <code><![CDATA["return $1.getSession().getId();"]]></code>
   </track>
</virtualize>
```

```
<virtualize>
   <track class="javax.ejb.SessionBean" method="setSessionContext" signature="
(Ljavax/ejb/SessionContext)V" push="true">
      <code><![CDATA["return $1.getEJBObject().getHandle().toString();"]]></code>
   </track>
</virtualize>

<virtualize>
   <track class="javax.ejb.EntityBean" method="setEntityContext" signature="(Ljavax
/ejb/EntityContext)V" push="true">
      <code><![CDATA["return $1.getPrimaryKey().toString();"]]></code>
   </track>
</virtualize>
```

These examples are hard-coded in the agent and thus unnecessary in your **rules.xml** file. However, these lines show how the process works so it can be implemented for any number of protocols, other than HTTP and EJB without recompiling the agent.

The value of the **class** attribute is the class from which we gain access to a session.

The value of the **method** and **signature** attributes determine the method that, when invoked, computes the session identifier. This computation uses the value of the **code** attribute. **$0** represents the source object. **$1**, **$2**, and so on, are the method arguments.

The **push** attribute determines how we store that session for later use by VSE frames.

- **push="true"** specifies that the container sets the session, and we keep a mapping of object to session.
- **push="false"** specifies that session is thread-scoped and we store in a thread-local variable.

The innermost session (identifier) is served back through VSE in the **com.itko.lisa.remote.vse.VSEFrame getSessionId()** method. For more information, see the JavaDocs in the **LISA_HOME\doc** directory.

Configure a Database Sink

The **database** directive lets you configure database access for console applications such as the Application Insight command-line tool. Console applications do not access the database through the broker.

The **database** directive can be added to the **broker** element of the **rules.xml** file.

This setting stays in use for the lifetime of the agent.

```
<database driver="myDriver" url="myURL" user="myUser" password="myPassword"/>
```

Example:

```
<database driver="oracle.jdbc.driver.OracleDriver" url="jdbc:oracle:thin:
@localhost:1521:ORCL" user="system" password="myPassword"/>
```

Category Settings

The DevTest Java Agent assigns a category to each transaction frame. If the agent cannot determine the category, the default category is assigned.

You can configure the agent to associate an intercepted class or interface with one of the non-default categories. Add the **category** directive to the **rules.xml** file of the agent. Specify the class or interface name and the category number.

The **category** directive has the following format:

```
<category class="class_or_interface_name" value="category_number"/>
```

The **category** directive must be placed within the **broker** element.

The valid category numbers are:

- CATEGORY_DEFAULT = 0
- CATEGORY_THREAD = 5
- CATEGORY_THROWABLE = 10
- CATEGORY_GUI = 7
- CATEGORY_GUI_SWT = 9
- CATEGORY_LOGGING = 15
- CATEGORY_WEB_HTTP = 20
- CATEGORY_WEB_HTTPS = 21
- CATEGORY_WS_HTTP = 22
- CATEGORY_WS_HTTPS = 23
- CATEGORY_REST_HTTP = 24
- CATEGORY_RMI = 30
- CATEGORY_RMI_HTTP = 31
- CATEGORY_RMI_SSL = 32
- CATEGORY_EJB = 40
- CATEGORY_JDBC = 50
- CATEGORY_JCA = 55
- CATEGORY_JMS = 60
- CATEGORY_MQ = 61
- CATEGORY_WM = 62
- CATEGORY_TIBCO = 64
- CATEGORY_AMX = 70
- CATEGORY_FRAMEWORK = 80
- CATEGORY_CLIENT = 90
- CATEGORY_CICS = 100
- CATEGORY_WPS = 120

- CATEGORY_SAP = 130
- CATEGORY_SYNTHETIC_ROOT = 140
- CATEGORY_DN_DEFAULT = 1000
- CATEGORY_DN_REMOTING = 1010
- CATEGORY_DN_SQL = 1020

Example:

```
<category class="com.mycompany.GuiClass" value="7"/>
```

Note: While possible, it is not recommended to change the categories of transaction frames that are already assigned a non-default category by the agent.

Add a Class for Tracking

The **track** directive is used for monitoring and debugging the Java virtual machine (JVM). The directive keeps track of the instance creation of the specified class.

The **track** directive can be placed within the **group** element or the **agent** element of the **rules.xml** file.

```
<track class="class_name"/>
```

Example:

```
<track class="java.io.File"/>
```

JNDI Priority

You can specify the order of priority for JNDI name prefixes.

Add the **transaction_jndi_name_prefix_priority** property to the **agent** element of the **rules.xml** file.

The following line shows the default settings. Notice that the **java:global** prefix has the highest priority.

```
<property key="transaction_jndi_name_prefix_priority" value="java:global=100;ejb:
=70;java:app=30;java:module=20;java:comp=10;#DEFAULT#=70"/>
```

The valid values of the priorities are integers.

Separate each priority setting with a semicolon.

The **#DEFAULT#** setting applies to any prefix that is not specified.

If you add or update this property, the agent needs to be restarted for the changes to take effect.

You cannot configure this property from the DevTest Portal.

Adding a Method to Intercept

You can add a method for the DevTest Java Agent to intercept by adding the **intercept** directive to the **rules.xml** file of the agent.

The **intercept** directive has the following format:

```
<intercept class="class_name" method="method_name" signature="signature"/>
```

The **intercept** directive can be placed within the **group** element or the **agent** element of the **rules.xml** file.

The format of the signature is described in Signature Specification.

Class hierarchies are considered. Assume that class B extends class A, and both classes define method M. If you intercept method M of class A, then method M of class B is also captured.

By default, Application Insight does not capture getter and setter methods. To add a getter or setter method, include the **delay** attribute with the value set to true.

Examples

The following example adds the **setName()** method of the **com.itko.examples.entity.Account** class. The method takes in a **java.lang.String** object as an argument. The method returns void.

```
<intercept class="com.itko.examples.entity.Account" method="setName" signature="
(Ljava/lang/String;)V" delay="true"/>
```

The following example adds the **getTransactions()** method of the **com.itko.examples.entity.Account** class. The method takes in no arguments. The method returns a **java.util.Collection** object.

```
<intercept class="com.itko.examples.entity.Account" method="
getTransactions" signature="()Ljava/util/Collection;" delay="true"/>
```

The following example adds the **getRequestTypes()** method of the **com.itko.examples.airline.ws.jaxws.Request** class. The method takes in no arguments. The method returns an array of **java.lang.String** objects.

```
<intercept class="com.itko.examples.airline.ws.jaxws.Request" method="
getRequestTypes" signature="()[Ljava/lang/String;" delay="true"/>
```

For more examples, see the **rules.xml.sample** file.

Excluding from Interception and Virtualization

You can prevent the DevTest Java Agent from intercepting or virtualizing a method, class, package, or URL by adding the **exclude** directive to the **rules.xml** file of the agent.

The **exclude** directive can be placed within the **group** element or the **agent** element of the **rules.xml** file.

If you add or remove the **exclude** directive, you must restart the agent.

- Excluding Methods, Classes, and Packages
- Excluding URLs

Excluding Methods, Classes, and Packages

To exclude a method, class, or package, use the following format:

```
<exclude class="class_name" method="method_name" signature="signature"/>
```

The format of the signature is described in Signature Specification.

Class hierarchies are not considered. Assume that class B extends class A, and both classes define method M. If you exclude class A, then method M of class A is not captured. However, method M of class B is captured.

If you want to exclude a class or package irrespective of method or signature, you can do either:

- Specify **method="*" signature="*"**
- Omit the **method** and **signature** attributes

By default, Application Insight does not capture getter and setter methods. You do not need to exclude getter and setter methods.

Examples

The following example excludes the **serviceComposition()** method of the **com.itko.examples.ejb3.OrdinaryBean** class. The method takes in no arguments. The method returns a **java.lang.String** object.

```
<exclude class="com.itko.examples.ejb3.OrdinaryBean" method="
serviceComposition" signature="()Ljava/lang/String;"/>
```

The following example excludes the **com.itko.examples.ejb3.OrdinaryBean** class.

```
<exclude class="com.itko.examples.ejb3.OrdinaryBean"/>
```

The following example excludes the **com.itko.examples.ejb** package. Notice the use of a single wildcard character.

```
<exclude class="com.itko.examples.ejb.*"/>
```

The following example excludes the **com.itko.examples** package and all its subpackages. Notice the use of two wildcard characters.

```
<exclude class="com.itko.examples.**"/>
```

For more examples, see the **rules.xml.sample** file.

Excluding URLs

To exclude a URL, use the following format:

```
<exclude url="url"/>
```

The URL can contain the following wildcards:

- Single asterisk (*). You can use this wildcard in place of one or more characters. You can also use this wildcard in place of an entire path segment.
- Double asterisk (**). You can use this wildcard only at the end of the URL. This wildcard applies to the current path segment and all subsequent path segments.

Examples

Assume that you specify the following **exclude** directive:

```
<exclude url="http://mycomputer:8080/datamanagement/design/applications/*
/server_types/*/components"/>
```

If the agent encounters the following URLs, the agent does not capture them:

- http://mycomputer:8080/datamanagement/design/applications/65/server_types/128 /components
- http://mycomputer:8080/datamanagement/design/applications/72/server_types/134 /components
- http://mycomputer:8080/datamanagement/design/applications/88/server_types/141 /components
- http://mycomputer:8080/datamanagement/design/applications/93/server_types/150 /components

However, the agent does capture this URL:

- http://mycomputer:8080/datamanagement/design/applications/65/server_types/128 /architectures

The following example does not use wildcards.

```
<exclude url="http://mycomputer:8080/lisabank/buttonclick.do"/>
```

For more examples, see the **rules.xml.sample** file.

Java Agent Auto-Response Generation

You can use the DevTest Java Agent to record transactions even when the back-end is not available.

To configure this feature, add the **respond** directive to the **rules.xml** file. The **respond** directive has the following format:

```
<respond class="class_or_interface_name" method="method_name" signature="
signature" source="string_value" args="string_value" return="class_name"/>
```

The **respond** directive must be placed within the **agent** element of the **rules.xml** file.

The **class** attribute is the only required attribute.

Use the following attributes to specify the method or methods that you want to the agent to intercept:

- **class**: Defines the name of the class or interface that defines the method or methods. If you specify an interface, any class that implements the interface is also included.
- **method**: Defines the name of a method. If you do not include this attribute, all methods of the class or interface are included.
- **signature**: Defines the signature of the method. The format is described in Signature Specification.

- **source**: If calling the **toString()** method on the object contains this string value, then intercept the method.
- **args**: If calling the **toString()** method on the arguments contains this string value, then intercept the method.

When the agent intercepts a method call that matches the specification, the agent performs the following actions:

- Makes an educated guess as to what type of object should be returned.
- Returns a generic version of the object. The object contains random values.

You can use the **return** attribute to specify the object type, thus overriding the first action.

If a value contains a less than sign, replace the less than sign with the following text:

```
&lt;
```

If a value contains a greater than sign, replace the greater than sign with the following text:

```
&gt;
```

Example Auto-Response Generation for EJB Calls

The following Java method creates a JNDI context object, calls a server that is able to run EJBs, and invokes APIs.

```
public void doEJBCall() throws Exception
    {
        Properties props = new Properties();
        props.put(Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.
NamingContextFactory");
        props.put(Context.PROVIDER_URL, "jnp://localhost:1099");

        /* 1st (optionally) remote call */
        Context ctx = new InitialContext(props);

        /* 2nd remote call */
        EJB3UserControlBeanRemote remote = (EJB3UserControlBeanRemote) ctx.lookup
("EJB3UserControlBean/remote");

        /* 3rd remote call */
        User u = remote.getUser("lisa_simpson");

        /* Local calls... */
        System.out.println(u.getEmail());
    }
```

Suppose that the server is not available. If you run this method, the code does not succeed because it cannot obtain a connection.

You can use the following **respond** directives to force the code to succeed. The first directive intercepts the creation of the **InitialContext** object. The second directive intercepts the **lookup()** method of the **Context** object. The third directive intercepts any method of the **EJB3UserControlBeanRemote** object.

```
<respond class="javax.naming.InitialContext" method="&lt;init&gt;" args="1099"/>

<respond class="javax.naming.Context" method="lookup" args="EJB3UserControlBean
/remote"/>

<respond class="com.itko.examples.ejb3.EJB3UserControlBeanRemote"/>
```

Java Agent REST Application Programming Interfaces

The DevTest Java Agent includes a set of REST Application Programming Interfaces that you can use to manage the agent.

When you invoke an API, the response is in JSON format.

- REST API Categories
- View the REST APIs
- Example: Retrieve Performance Data

REST API Categories

The REST APIs are divided into the following categories:

- **Agents**
  Perform basic tasks for an agent, such as retrieving statistics and starting and stopping capture.

- **Artifacts**
  Perform tasks for the adding of extra steps to baselines.

- **CaseRecording**
  Create and manage transaction recordings.

- **NoiseFilter**
  Manage the exclusion of data from agent capture.

- **Tickets**
  View and update tickets that have been created with the agent.

- **Transaction Service**
  Manage business transactions and transaction frames.

View the REST APIs

You can view the REST APIs from a web browser. The following graphic shows the main components of the user interface.

Screen capture of REST APIs in web browser.

The left area contains the base path, a list of API categories, and a list of objects.

If you select an API category, the center area displays the APIs for the category.

If you select an object, the center area displays information about the object.

Follow these steps:

1. Ensure that the registry is running.
2. Enter **http://localhost:1505/lisa-pathfinder-invoke/restApiDoc/index** in a supported browser.
   If the registry is on a remote computer, replace **localhost** with the name or IP address of the computer.
3. Enter **/lisa-pathfinder-invoke/restApiDoc/api** in the text area.
4. Click **Get documentation**.
   The APIs appear.
5. To invoke an API from the browser:
   a. Select an API category.
   b. Click the API name.
   c. Specify any path parameters, query parameters, and body objects in the right area. The API information in the center area indicates whether each parameter is required or optional.
   d. Click **Submit**.

Example: Retrieve Performance Data

This example shows how to retrieve performance data for an agent. The API requires one parameter: the agent ID.

```
GET http://mycomputer:1505/lisa-pathfinder-invoke/api/v1/agents/1088413135
/performance
```

The response contains the agent ID and a set of performance values.

```
{
   "id": 1088413135,
   "cpuUsage": "0",
   "nonHeapUsage": "79",
   "heapUsage": "223",
   "ioIn": "0",
   "ioOut": "0",
   "txn": "0"
}
```

Developing Against the Java Agent

The main interface to the DevTest Java Agent from the client side is **com.itko.lisa.remote.client.AgentClient**. This class has methods to invoke APIs on the agents or the broker, to discover agents and to be notified of their status changes (online/offline).

This class also gives access to the classes responsible for the 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**

Note: You can view detailed information about these classes in the JavaDocs for the agent. The JavaDocs are located in the **LISA_HOME\doc** directory.

More Information:

- Agent General APIs
- Agent Discovery APIs
- Agent Transaction APIs
- Agent VSE APIs
- Agent API Examples

Agent General APIs

Most of the client APIs must specify an agent as their target. This specification is accomplished by passing a parameter of type **com.itko.lisa.remote.IAgentInfo**. This parameter represents an object that uniquely identifies an agent and some of its basic information.

Note: You can view detailed information about the interface and class that this page describes in the JavaDocs for the agent. The JavaDocs are located in the **LISA_HOME\doc** directory.

```
/** A unique identifier for this agent or console. If it is named the guid is pers
istent 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 prop
erties */
 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 ag
ent 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 covers 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 i
n all other API calls
  * @param tokens a map of agent guids or names to tokens, null if no agent has to
ken-enabled security
  * @return a set of IAgentInfo objects representing agents that are currently onl
ine
  */
  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 i
nstance.
  * @return the value returned by the code
  * @throws JMSInvocationExceptionthrown if the code throws on the agent
  */
 public Object eval(IAgentInfo info, String input) throws JMSInvocationException
```

Agent Discovery APIs

The **com.itko.lisa.remote.client.DiscoveryClient** class provides methods pertaining to discovering data on an agent. You can get the **DiscoveryClient** class with the following call: **AgentClient.getInstance().getDiscoveryClient()**.

> **Note:** You can view detailed information about this class in the JavaDocs for the agent. The JavaDocs are located in the **LISA_HOME\doc** directory.

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

 /**
  * Returns a list of StatsFrames recorded for the specified agent between the fro
m 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 (star
ting 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 fro
m 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 (star
ting 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 t
ree
  * @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, IOExce
ption;

 /**
  * 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 w
ith 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 loca
tion and return a handle to it
  * @param info
  * @param file
  * @return
  * @throws JMSInvocationException
  */
 public File getFile
(IAgentInfo info, String file) throws JMSInvocationException, IOException;
```

Agent Transaction APIs

A *transaction* is a code path executed by one or more servers as the result of a client request. A transaction is represented by a tree structure that is rooted at the client initiating the request. The nodes of the tree are **com.itko.lisa.remote.transactions.TransactionFrame** objects. These objects encapsulate information about a class, method, and arguments that were invoked as part of the server processing. Frames also contain ancillary information, such as the duration, the time of execution, and the thread in which it occurred.

You can think of a transaction as a method call stack. One difference is that transactions cross thread, process, or even computer boundaries. Another difference is that stacks contains all of the methods involved in the code execution of a thread, whereas transactions skip some levels and have frames only for chosen methods of interest. Such methods are referred to as *intercepted methods*.

Note: You can view detailed information about these classes in the JavaDocs for the agent. The JavaDocs are located in the **LISA_HOME\doc** directory.

The main way to obtain and work with **TransactionFrame** objects 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 ser
ver 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 serv
er 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 cl
ient transaction created using
  * the specified parameters.
  * Note: you must call installSocketTracker before you initiate the client side t
ransaction you're adding here
  * and it is recommended you call uninstallSocketTracker after you're done with t
he network call.
  * @param stamp     a unique string you can later use to identify the root transa
ction
  *      (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 e
mpty)
  * @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 satsfy the
 filters passed as arguments
  * @param offset    offset
  * @param limit     max number of results
  * @param category  filter by transaction category (see TranactionFrame.
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 t
his is empty
  * @param minTime   duration below which transactions are pruned out of the resul
ts
  * @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)
```

The parameters to these APIs do not specify an **Agent** or **AgentInfo** because transactions can span multiple agents.

Those APIs return lists of **com.itko.lisa.remote.transactions.TransactionFrame** (or trees thereof), so let us 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 talki
ng from */
 public String getLocalIP();
 public void setLocalIP(String ip);
```

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


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


/** Network incoming or outgoing frames set this to tell us what port they are tal
king 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 intercep
tion 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 recorde
d in */
 public String[] getArguments();
 public void setArguments(Object[] arguments);


/** Formatted representation of the result of the method this frame was recorded i
n */
 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 Tran
sactionFrame.FLAG_XXX */
 public long getFlags();
 public void setFlags(long flags);
```

Agent VSE APIs

VSE enables you to stub out processes, services, or parts of them along well-defined boundaries. The internal elements of these processes and services can be replaced with a layer run by DevTest according to custom user-defined rules. These layers that DevTest runs are known as a model. Usually, a default starting point for those rules is obtained from a recording of live system interactions.

DevTest 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. Virtualization can be enabled for HTTP, JMS, and JDBC but also for RMI, EJB, or any custom Java objects.

DevTest (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()**.

> **Note:** You can view detailed information about this class in the JavaDocs for the agent. The JavaDocs are located in the **LISA_HOME\doc** directory.

```
/**
  * 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 sy
ntax:
  * 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 i
nvoked
  * 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);

 /**
  * Unegister 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 ne
eded
  * 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 ne
eded
  * 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 playb
ack 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);
```

Agent API Examples

This page shows examples of agent APIs.

Agent API Example 1

The following code generates a transaction from client code and retrieves the transaction tree that 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!");
 }
```

The preceding code uses the following utility function, which has nothing to do with the agent but is listed for completeness:

```
/** 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();
 }
```

Agent API Example 2

The following code registers 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) {}
 });
```

Java Agent Extensions

The following pages describe various aspects of creating extensions for the DevTest Java Agent:

- Extending the Agent
- Adding Tags to Transaction Frames
- Extending the Broker
- Extending Java VSE

Extending the Agent

You can 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. For more information, see the JavaDocs in the **doc** folder of your installation directory.

```
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 signa
ture, Object[] args, Object ret);

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

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

If you want to stop data from being captured, overwrite the **block()** method. This technique is similar to adding the **exclude** directive to the **rules.xml** file, but provides more flexibility.

If you want the agent to perform logic immediately before it captures a method, overwrite the **preProcess()** method.

If you want the agent to perform logic immediately after it captures a method, overwrite the **postProcess()** method.

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 documented syntax in the **rules.xml** file. You can also programmatically specify the interception in the constructor of the extension class. The advantage of the latter approach is that the extension is self-contained.

The first argument to the **block()** method is **boolean wayIn**. The **block()** method is called twice per method: once on entry, once on exit. When the entry call is made, the value of the **wayIn** argument is true. When the exit call is made, the value of the **wayIn** argument is false.

The following table describes the arguments that are common to the **block()**, **preProcess()**, and **postProcess()** methods:

Argument	Description
Object src	The object that the method is being called on.
String spec	The class or interface name that was specified to instrument the API.
String clazz	The name of the class that defines the intercepted method.
String method	The name of the intercepted method.
String signature	The signature (in JVM format) of the intercepted method.
Object[] args	The arguments being passed to the intercepted method.
Object ret	The return value of the intercepted method. When the wayIn argument is true, the return value is null.

The difference between the **src**, **spec**, and **clazz** arguments can be explained with an example.

Assume that you have the following interface and class definitions:

```
public interface A {
     void m();
}

public class B implements A {
     void m() {}
}

public class C extends B {
}
```

If the rules specify **intercept("A", "m", "*")** and the code calls **C.m()**, then the following information is true:

- **spec** is A
- **clazz** is B
- **src** is an instance of C

Deployment

To deploy an extension, compile the extension and package it in a JAR file with a manifest file that contains the following entry:

```
Agent-Extension: extension-class-name
```

When you drop this JAR into the Agent JAR directory, the agent automatically picks it up. If you add the file after the agent has started, a hot load mechanism helps to ensure that the file is applied. If you update the extension JAR while the agent is running, the JAR classes are reloaded dynamically. Dynamic reloading makes it easy and fast to test your extension code without restarting the server.

The extension JARs get loaded by a classloader that can see all the classes that are used in the extension class. You can compile your extension source against **LisaAgent.jar** and all container JARs that define classes you want to use. You do not need to use reflection in your extension.

Examples

The following example uses the **block()** method to prevent the agent from capturing any method whose thread name starts with **Event Sink Thread Pool**.

```
public class MyInterceptor extends AbstractInterceptor {

   /** Returns true if this call should not be intercepted */
   public boolean block
(boolean wayIn, Object src, String spec, String clazz, String method, String signa
ture, Object[] args, Object ret) {
      if (Thread.currentThread().getName().startsWith("Event Sink Thread Pool")) {
         return true;
      }
      return super.block(wayIn, src, spec, clazz, method, signature, args, ret);
   }

   ...

}
```

The following example uses the **postProcess()** method to capture data for the Response row in the Transactions window. This example calls the **setResponse()** method of the **com.itko.lisa.remote.transactions.TransactionFrame** class. For more information, see the JavaDocs in the **doc** folder of your installation directory.

```
public class MyInterceptor extends AbstractInterceptor {

   ...

   public boolean postProcess
(TransactionFrame frame, Object src, String spec, String clazz, String method, Str
ing signature, Object[] args, Object ret) {
      if (clazz.equals("com.itko.lisa.training.NotCaptured") && method.equals
("doRequest")) {
         frame.setResponse((String) ret);
      }
      return super.postProcess
(frame, src, spec, clazz, method, signature, args, ret);
   }
```

The following example shows how to print the XML contents of a WebMethods **com.wm.data.IData** object as it gets invoked in a flow of Integration Server. The agent already supports WebMethods, so an extension is not necessary for it.

```
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, Str
ing 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, Str
ing signature, Object[] args, Object ret) {
     if (ServiceManager.class.getName().equals(clazz) && "invoke".equals(method)) {
         doCustomLogic((BaseService) args\[0\], (IData) ret, true);
     }

     return super.postProcess
(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, compile this code against **LisaAgent.jar**, **wm-isclient.jar**, and **wm-isserver.jar** then JAR the class file with a manifest containing the following line:

```
Agent-Extension: com.itko.lisa.ext.IDataInterceptor
```

Adding Tags to Transaction Frames

You can associate a transaction frame with arbitrary key/value pairs. The key/value pairs are known as *tags*.

TransactionFrame objects provide a **setTag()** method. The method has two string arguments: the key and the value.

The following code shows how to add a tag from within the **postProcess()** method of an agent extension:

```
// somewhere in postProcess
frame.setTag("tagname", "tagvalue");
```

For example, you can create an extension that upon login to a web site grabs the value of the request parameter **username** and calls **frame.setTag("username", value)**.

The tags are stored in the **FRAME_TAGS** table in the database.

You can view the tags of a frame in DevTest Portal. For more information, see Frame Information.

You can use tags as search criteria in the command-line tool.

Extending the Broker

Broker extensions are loaded and invoked by the broker after a transaction fragment is received from an agent.

Broker extensions let you perform the following tasks:

- Change data that is contained in frames.
- Add or eliminate certain frames.
- Customize the stitching algorithm. The stitching algorithm defines how transaction fragments are assembled.

To extend the broker, implement the **com.itko.lisa.remote.plumbing.IAssemblyExtension** interface. This interface defines the **onTransactionReceived()** method. For more information, see the JavaDocs in the **doc** folder of your installation directory.

```
public interface IAssemblyExtension {
     /**
      * The method invoked prior to assembly
      * @param frame the partial transaction root received from an agent
      * @return true to bypass normal assembly (i.
e. if the extension wants to take care of it itself)
      */
     public boolean onTransactionReceived(TransactionFrame frame);
 }
```

Deployment

To deploy a broker extension, compile the extension and package it in a JAR file with a manifest file that contains the following entry:

```
Broker-Extension: extension-class-name
```

For example:

```
Broker-Extension: com.itko.lisa.remote.ext.JMSPayloadCorrelationBroker
```

When you drop this JAR into the broker directory, the broker picks it up automatically. If you add the file after the broker has started, a hot load mechanism helps to ensure that the file is applied. If you update the extension JAR while the broker is running, the JAR classes are reloaded dynamically. Dynamic reloading makes it easy and fast to test your extension code without restarting the broker.

Example: Customize the Stitching Algorithm

A typical usage example is when the normal stitching algorithm that uses TCP/IPs and ports is confused by a load balancer sitting between agents and is assigned a virtual IP, or when agents use a native library to do IO and we do not have direct access to the IPs and ports in use. In this situation, the address and port fields of frames are either left blank or incorrect, and the frames must be assembled in an extension.

The following example shows the basic structure of an extension that addresses a load balancer issue.

```
import com.itko.lisa.remote.plumbing.IAssemblyExtension;
import com.itko.lisa.remote.transactions.TransactionFrame;
import com.itko.lisa.remote.utils.UUID;

public class LoadBalancerExtension implements IAssemblyExtension {

    private TransactionFrame m_lastAgent1Frame;
    private TransactionFrame m_lastAgent2Frame;

    public boolean onTransactionReceived(TransactionFrame frame) {

        if (frame.getClassName().equals("Class1") && frame.getMethod().equals
("method1")) {
            m_lastAgent2Frame = frame;
            if (m_lastAgent1Frame.getFrameId() == m_lastAgent2Frame.getParentId())
 {
                stitch(m_lastAgent1Frame, m_lastAgent2Frame);
            }
        }

        if (frame.getClassName().equals("Class2") && frame.getMethod().equals
("method2")) {
            m_lastAgent1Frame = frame;
            if (m_lastAgent1Frame.getFrameId() == m_lastAgent2Frame.getParentId())
 {
                stitch(m_lastAgent1Frame, m_lastAgent2Frame);
            }
        }

        return false;
    }

    /*
     * Link the parent frame and the child frame
     */
    private void stitch(TransactionFrame parent, TransactionFrame child) {
        String newFrameId = UUID.newUUID();
        parent.setFrameId(newFrameId);
        child.setParent(parent);
        parent.getChildren().add(child);
```

```
    }
}
```

Extending Java VSE

In a Java VSE extension, you can overwrite any of the following methods:

- **onPreRecord():** This method is called during recording before the virtualized method starts executing.
- **onPostRecord():** This method is called during recording after the virtualized method stops executing.
- **onPreHijack():** This method is called during playback before the virtualized method goes to VSE.
- **onPostHijack():** This method is called during playback after the virtualized method returns from VSE.
- **onNewStream():** This method is called every time that an object is converted to XML.

Example

The following example uses the **onPostRecord()** method to change the name of a class during recording. This example calls the **setClassName()** method of the **com.itko.lisa.remote.vse.VSEFrame** class. For more information, see the JavaDocs in the **doc** folder of your installation directory.

```
public class MyInterceptor extends AbstractVSEInterceptor {

   ...

   public boolean onPostRecord
(VSEFrame frame, Object src, String clazz, String method, String signature, Object
[] args, Object ret) {
      frame.setClassName(clazz.replaceAll("\\.", "_"));
      return super.onPostRecord(frame, src, clazz, method, signature, args, ret);
   }

}
```

Load Balancers and Native Web Servers

The process of assembling partial transactions into complete transactions is referred to as *stitching*.

When agents are deployed on a network that has load balancers or native web servers, the stitching algorithm that Application Insight uses might not function properly.

The following graphic shows a load balancer that is between two agents.

To help the broker to perform the stitching correctly, add the **loadbalancer** directive to the **rules.xml** file of the broker. For each appliance installed between agents, specify the IP address of the appliance. For example:

```
<loadbalancer ip="172.16.0.0"/>
<loadbalancer ip="172.31.255.255"/>
```

The **loadbalancer** directive must be placed within the **broker** element.

An alternate approach is to enable the **Always wait** property.

You can configure this property from the Agents window of DevTest Portal. The property appears in the **Settings** tab.

Java Agent Security

The DevTest Java Agent provides a security mechanism for extensions and remote code invocation.

The agent installs its own security manager. All custom agent code runs under special permissions that the security manager enforces. If the application already uses its own security manager, then the agent security manager wraps the existing security manager and delegates to it for regular application code.

- Extensions
- Remote Code Invocation
- Configure the Java Agent to Use SSL

Extensions

Agent extensions for Application Insight and CA Service Virtualization run with the following restrictions:

- File access is limited to the agent directory and the temp directory.
- Process creation is disabled.
- Process exit is disabled.

As a result, the system under test is sandboxed from the rest of the computer.

Some applications can explicitly check for the security manager being null and take a different code path depending on the result. In this situation, the agent security manager can cause insurmountable issues. You can disable the agent security manager by using either of the following approaches:

- Specify **-Dsecurity.manager.disabled=true** on the command line.
- Add the **lisa.agent.security.manager.disabled** property to the **rules.xml** file and set the value to **true**.

Note: Disabling the security manager disables authorization checks. However, you can still enable token authentication.

Remote Code Invocation

All agent APIs work through remote code invocation, because the agent is located in a remote system.

Security for remote code invocation is handled with tokens.

You use the **token** option to define one or two tokens for an agent. The first token represents an admin role. The second token represents a user role. If you specify two tokens, use a colon to separate them.

Examples:

- **token=asdf1** specifies an admin token with a value of **asdf1**.
- **token=asdf1:asdf2** specifies an admin token with a value of **asdf1**, and a user token with a value of **asdf2**.

The admin or user token can include any character except for commas, equal signs, and space characters. The maximum length of a token is 16 characters.

The token concept can be used with consoles. You can specify it with the command-line syntax **-Dlisa.token=xxxx** or **-Dlisa.token=xxxx:xxxx**. For consoles, there is no custom security manager, so both tokens have all permissions. Security is achieved by the fact that not supplying the console token prevents you from doing anything.

Configure the Java Agent to Use SSL

You can use the Secure Sockets Layer (SSL) to secure communication between the agent and the broker.

The broker inherits the settings that the registry has for SSL. The agent uses the same encrypted password and the same keystore.

Choose one of the following approaches.

Default Keystore

This approach uses the default keystore in DevTest.

Follow these steps:

1. Open the **local.properties** file in the **LISA_HOME** directory and uncomment the following line:

   ```
   lisa.net.default.protocol=ssl
   ```

2. Save the **local.properties** file.

3. Start or restart the registry.

4. When specifying the broker URL, use the **ssl** scheme. For example:

   ```
   ssl://localhost:2009
   ```

Custom Keystore

This approach is based on a custom keystore.

> **Note:** For detailed information about the DevTest Portal, see Using Application Insight.

Follow these steps:

1. Configure a custom keystore by following the instructions in Using SSL to Secure Communication.

2. Place the keystore in the **LISA_HOME\agent** directory.

3. Open the DevTest Portal.

4. Select **Settings**, **Agents** in the left navigation menu.

5. In the left portion, select the agent.

6. Click the **Settings** tab.

7. Set the following security properties to configure a custom keystore that mirrors the DevTest properties. You can copy the encrypted passwords from the **local.properties** file.

   - **Keystore location**
   - **Keystore password (encrypted)**
   - **Truststore location**
   - **Truststore password (encrypted)**

8. In the left portion, select the broker.

9. Set the same properties as for the agent. The path to the keystore will be different.

10. When you specify the broker URL, use the **ssl** scheme. For example:

    ```
    ssl://localhost:2009
    ```

Java Agent Log Files

The following log files can help you troubleshoot issues:

- The agent log file is named **devtest_agent_pid.log**. This file is written to the same directory as the **LisaAgent.jar** file.
- Each time the broker is started, a log file named **devtest_broker_pid.log** is created. In addition, the broker has a consolidated log file named **pfbroker.log**. These files are written to the same directory as the main DevTest log files.
- The console log files are named **devtest_agent_console_pid.log**. These files are written to the same directory as the main DevTest log files.

The **pid** portion of the file name is the process identifier of the process that is doing the logging.

> **Note:** For information about the main DevTest log files and their location, see Logging.

The following configuration properties let you control the logging behavior:

- **lisa.agent.log.max.size**
  Sets the maximum size of an agent or broker log file before it gets archived. The value is in bytes.
- **lisa.agent.log.max.archives**
  Sets the maximum number of archived log files to keep.
- **lisa.agent.java.logging.level**
  Sets the minimum level at which logging statements are captured. This setting is not applicable to the broker. For more information about the levels, see the Java API documentation for the **java.util.logging.Level** class.
- **lisa.agent.agent.log**
  Sets the location of the agent or broker log files.

You can also configure these properties from the DevTest Portal.

Transaction Dispatchers

The transaction dispatchers control the capturing of transactions by agents.

You can improve performance by disabling some or all of the dispatchers.

To disable all of the dispatchers, add the following line to the **rules.xml** file:

```
<property key="com.itko.lisa.remote.transactions.TransactionDispatcher.disabled"
value="true"/>
```

To disable individual dispatchers, add one or more of the following lines to the **rules.xml** file:

```
<property key="com.itko.lisa.remote.transactions.dispatchers.SocketDispatcher.
disabled" value="true"/>
<property key="com.itko.lisa.remote.transactions.dispatchers.FileDispatcher.
disabled" value="true"/>
<property key="com.itko.lisa.remote.transactions.dispatchers.StreamInDispatcher.
disabled" value="true"/>
<property key="com.itko.lisa.remote.transactions.dispatchers.StreamOutDispatcher.
disabled" value="true"/>
<property key="com.itko.lisa.remote.transactions.dispatchers.URLDispatcher.
disabled" value="true"/>
<property key="com.itko.lisa.remote.transactions.dispatchers.ConnectionDispatcher.
disabled" value="true"/>
<property key="com.itko.lisa.remote.transactions.dispatchers.JNDIDispatcher.
disabled" value="true"/>
<property key="com.itko.lisa.remote.transactions.dispatchers.JMSDispatcher.
disabled" value="true"/>
<property key="com.itko.lisa.remote.transactions.dispatchers.MQDispatcher.
disabled" value="true"/>
<property key="com.itko.lisa.remote.transactions.dispatchers.ExceptionDispatcher.
disabled" value="true"/>
<property key="com.itko.lisa.remote.vse.VSEDispatcher.disabled" value="true"/>
```

Disabling all of the dispatchers is easier than disabling individual dispatchers.

The **Agents** window in DevTest Portal lets you control whether an agent is capturing transactions. This action is the same as disabling all of the dispatchers. However, the change is not persistent.

You can also perform these actions from the Application Trace Kit:

- Display the properties for an agent and select **transactions/dispatchers**. Update one or more properties and click **Save**. This change is persistent.
- Select an agent, click the magnifying glass icon , and click **Turn Pathfinder Off**. This action is the same as disabling all of the dispatchers. However, the change is not persistent.

None of these settings affect any virtual services that are deployed. Even with transaction capture turned off, the agent still sends the correct transactions to the virtual service environment.

A related action that you can take is to add the following line to the **local.properties** file:

```
lisa.pathfinder.on=false
```

This action disables the capturing of transactions from tests that are executed on the coordinator. It can reduce the load on the broker.

Troubleshooting the Java Agent

> **Important:** Investigate the target environment ahead of time and ensure that it has been tested, or test it yourself. Most Java agent issues are OS or JVM-dependent, instead of application-dependent. Be sure to follow the instructions in this documentation. If that does not help, this page reviews the most common issues.

Most serious issues involving the DevTest Java Agent occur at start-up. For example, the agent is not found, or the process crashes or hangs. Generally, after you get past issues at start-up, you are in good shape. From there, it is a matter of tweaking the configuration or writing extensions.

Notes:

- Bouncing the servers to try various things in the agent environment can be difficult. Typically, it is simpler, and a worthwhile exercise, to test running **java <agent options> -version** on the target computer and the JVM. This exercise indicates whether the issue is OS/JVM-specific or container/application-specific.
- For a command-line utility that can help you with the installation process, see Java Agent Install Assistant .

- Error at startup: Error occurred during initialization of VM. Could not find agent library in absolute path...
- Agent exits immediately (or shortly after starting the process).
- Agent exits with the message "GetEnv on jvmdi returned -3 (JNI_EVERSION)".
- Agent exits with the message "UTF ERROR" ["../../../src/solaris/instrument /EncodingSupport_md.c":66]: ..."
- Agent hangs or throws numerous exceptions at startup (LinkageErrors, CircularityErrors, and so on).
- Agent throws java.lang.VerifyErrors or hot swapping has no effect.
- Agent starts but the consoles or broker cannot see the agent.
- Agent causes some operations to time out.
- java.lang.NoClassDefFoundError on com/itko/lisa/remote/transactions /TransactionDispatcher.class
- Security-related exceptions are thrown when the agent is enabled.
- Abnormal resource consumption (CPU, memory, file handles, ...).
- I can see the agent started, but Application Insight data is missing or incomplete.
- I turned on Java VSE recording or playback, but VSE does not receive any requests from the agent.
- Other issues with Java VSE.
- Ticket functionality is not working.
- DevTest is configured to use an IBM DB2 database. In DevTest Portal, I selected a JDBC node in a path graph. The Statements and Connection Url columns are blank.
- Received the following exception: com.itko.lisa.remote.transactions. TransactionSizeExceededException: could not XStream ... Serialization for this frame will be disabled.
- The VSE log includes the following error: com.itko.lisa.vse.stateful.protocol.java.listen. DefaultJavaPlaybackCallback - Timed out waiting on response.

Error at startup: Error occurred during initialization of VM. Could not find agent library in absolute path...

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

Also verify that the library is not missing any dependencies. For example, use **depends.exe** on Win32, **otool -L** on OS X, or **ldd -d** on Linux and UNIX. If libraries are missing, ensure that **LD_LIBRARY_PATH** is set to include the directories where they reside. Containers can override **LD_LIBRARY_PATH** in their start-up scripts. Do not assume that it is correctly set if you set it in a shell instead of from inside the script or a container-specific administration tool.

If **ldd** does not return cleanly, the agent does not run properly. Therefore, verifying **ldd** is the first thing to get right. If you cannot find an agent version for the operating system, consider using the pure Java agent.

Agent exits immediately (or shortly after starting the process).

If you see the message **LISA AGENT: VM terminated**, it is likely that the process ended normally. Several containers have launcher processes, and it is normal for them to exit quickly.

If you do not see this message or a crash dump happens (after **ldd** returns cleanly), you could have a legitimate agent bug. Notify Support and try the pure Java agent. If it still crashes or produces a dump, you could have a JVM bug. One such occurrence is for the IBM JVM 1.5 on some operating systems, caused by trying to instrument threads. In that case, try supplying the command-line JVM argument **-Dlisa.debug=true**.

Agent exits with the message "GetEnv on jvmdi returned -3 (JNI_EVERSION)".

Try specifying the command-line option: **-Xsov** to instruct the JVM to use its debug-enabled libraries. If that does not work (for example, you get an invalid option error message), then this operating system is not supported.

Agent exits with the message "UTF ERROR" ["../../../src/solaris/instrument /EncodingSupport_md.c":66]: ..."

The message can vary depending on the version of the operating system, the JVM, or both. The message typically has one of the following elements: UTF ERROR ["../../../src/solaris/instrument /EncodingSupport_md.c":66]: Failed to complete iconv_open() setup.

This issue is caused by a bug in some Solaris JVMs when some language packs are not installed. To fix this issue, install the en-US language pack by running **pkg install SUNWlang-enUS** and then **export LANG=en_US.UTF-8**. Alternatively, you can try using the native agent.

Agent hangs or throws numerous exceptions at startup (LinkageErrors, CircularityErrors, and so on).

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

We have eliminated all known occurrences of these issues for all combinations of JVMs and operating systems. However, it is possible that you have encountered an untested combination. Notify Support of this issue. If this issue is a hang, include a thread dump with your support issue. Produce a thread dump by entering Ctrl+Break on Windows, Ctrl+\ or kill -3 <pid> on UNIX/Linux. You can also try the Java agent, as the class-loading order is slightly different. If a class or package seems involved every time in the hung thread, try adding an **exclude** directive for it in the **rules.xml** file.

Agent throws java.lang.VerifyErrors or hot swapping has no effect.

Some older 1.4 JVMs have bugs in their support for hot swapping (instrumenting classes after they are loaded).

In that case, turn off hot swapping by disabling the **Enable hot instrumentation** setting.

You can configure this property from the Agents window of DevTest Portal. The property appears in the **Settings** tab.

You must determine in advance the classes, methods, or both that you want to intercept or virtualize, add the rules for these classes or methods in the **rules.xml** file, and bounce the server. This process is more tedious than doing it on a live server, but it is the only known way to work around this issue.

Sometimes the **java.lang.VerifyError** is not even seen in the logs, but the agent behaves as though the designated class has not been instrumented or yields random results, including crashes.

Agent starts but the consoles or broker cannot see the agent.

Typically, the cause is a firewall or port issue between the agent and the broker.

The top of the agent log can include the following warning: **Can't connect to broker at tcp://ip:port**. Verify that the IP address and port that the agent is using are correct. Then ensure that the broker is listening on the specified port on the specified IP address. **netstat -ano | grep port** should show a port listening on the supplied ip or 0.0.0.0. Finally, look for firewall issues by running **telnet ip port** from the agent computer to ensure that it can see the broker. If it can see the broker, the broker is likely in a bad state. Review the registry and broker logs (and restart it if necessary).

Agent causes some operations to time out.

Some JVMs (IBM JVMs in particular) do not behave properly after some of their networking classes are instrumented. As a result, networking calls can fail or time out without apparent reason.

To prevent those classes from being instrumented, try supplying the command-line JVM argument **-Dlisa.debug=true**.

If the issue is not resolved, edit the **rules.xml** file to exclude the following networking packages:

```
<exclude class="java.net.**"/>
<exclude class="java.nio.**"/>
<exclude class="sun.nio.**"/>
```

java.lang.NoClassDefFoundError on com/itko/lisa/remote/transactions/TransactionDispatcher.class

Verify that **LisaAgent.jar** is available and has read permissions and that it is not corrupted. The easiest way to verify is to run **java -jar LisaAgent.jar -v**.

Verify whether the application uses OSGi. If it does (as in JBoss 7), add **com.itko** to the system or bootstrap packages. The method for adding **com.itko** is container-dependent, which makes it difficult to provide specific instructions. However, it is typically a configuration file with a property that specifies a list of packages or a similar JVM argument.

Security-related exceptions are thrown when the agent is enabled.

You may see SecurityExceptions or PermissionExceptions thrown by the application only when the agent is turned on with the security enabled. This setting is now the default setting.

The reason and workaround are explained in Java Agent Security.

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

If the CPU usage is abnormally high or spikes periodically, note the period of the spikes because it will help determine the faulty thread or threads. Also try turning off Application Insight and VSE.

If you get OutOfMemory errors, monitor the Java heap usage. If it exceeds the **-Xmx** limit, then increase that limit. However, avoid increasing the limit if it is already high and well in excess of normal application usage without the agent. In that case, generate a heap dump. You can use the WAS HeapDump utility for WebSphere 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 is likely a leak in native code. In this case, notify Support.

If you get unexplained IOExceptions (such as too many file handles), especially on UNIX or Linux, verify the ulimit on the box (**ulimit -n -H** and **ulimit -n -S**). If it is low, consider asking the administrator of the box to raise it (under 4096 is considered low for modern J2EE apps). Do not forget to restart afterward.

The agent tries to keep the number of file handles under that limit by triggering garbage collection when the limit is approached. If the agent cannot read the limit at startup, the agent uses a default value of 1024. Using this default value can result in excessive garbage collection and semi-random, frequent CPU spikes. The following messages in the logs indicate this situation:

```
Max (or preferred) handles limit approaching - triggering GC...
```

In that case, increase the value of the **Maximum number of handles** setting.

You can configure this property from the Agents window of DevTest Portal. The property appears in the **Settings** tab.

I can see the agent started, but Application Insight data is missing or incomplete.

The data lifecycle process includes the following stages:

- Transaction capture in the agent
- Transfer to the broker
- Partial transaction assembly in the broker
- Transfer to the consoles
- Persistence to the database
- Retrieval from the database

Missing or incomplete data can be the result of an issue in any of these stages.

Review the agent log for capture exceptions. Review the broker and console logs for transfer or persistence exceptions.

If there are no exceptions, and you are still unable to locate the missing data, turn on debug or dev logging in the agents. If you do not see statements such as "Sent partial transaction," then Application Insight is probably not turned on.

If none of these steps provide conclusive results, contact Support.

I turned on Java VSE recording or playback, but VSE does not receive any requests from the agent.

First, verify that the agent is in VSE record or playback mode. Open the agent log and search for "Starting VSE record/playback..."

Then look for exceptions in the agent logs and the VSE logs. If they are clean, it is possible that the class you think is virtualized is not virtualized. Look in the agent log for a statement such as "Virtualized com.xxx...." If the statement is missing, it is possible that the application has not yet loaded the class. Another possibility is that if you are using an early version of Java 1.4, some flavors do not support hot swapping. Java VSE uses hot swapping by default. In that case, specify the virtualized classes in the **rules.xml** file of the agent and restart it.

Other issues with Java VSE.

If you encounter any issues (functional or abnormal resource usage) while doing Java VSE, turn off Application Insight on the agent side.

You can turn off Application Insight by disabling the **Auto-start** property. Then restart the JVM.

You can configure this property from the Agents window of DevTest Portal. The property appears in the **Settings** tab.

Do not turn off Application Insight on the broker side or Java VSE stops working altogether.

Ticket functionality is not working.

First, ensure that the agent is connected to a broker.

If the agent is connected to a broker, review the HTML source of the page that has the issue. Verify that the bottom of the page has a block of JavaScript that is easily identifiable by the variable names it uses (for example, **com_itko_pathfinder_defectcapture_xxx**). If the block is missing, review the agent log for possible exceptions. Other reasons for the absence of the block include:

- The page HTML is unusual. For example, HTML tags are missing.
- The agent has issues capturing it entirely, which can happen with untested containers or static pages.

If the block is present, verify whether a native web server (such as Apache) or a load balancer is present in front of the Java container. If it is the case, configure it to forward the request of the Application Insight JavaScript and resource files to the Java container. Those will have the word **defectcapture** as part of their URL. IT administrators generally know how to perform this task.

DevTest is configured to use an IBM DB2 database. In DevTest Portal, I selected a JDBC node in a path graph. The Statements and Connection Url columns are blank.

Disable progressive streaming by adding the string **progressiveStreaming=2** to the JDBC connection URL. For example:

```
lisadb.pool.common.url=jdbc:db2://myhostname:50000/dbname:progressiveStreaming=2;
```

Received the following exception: com.itko.lisa.remote.transactions.TransactionSizeExceededException: could not XStream ... Serialization for this frame will be disabled.

This error means that the agent exceeded the maximum buffer size while trying to serialize an object graph.

Go to DevTest Portal and increase the value of the **Maximum graph size** setting.

You can also configure this setting in the **rules.xml** file. The corresponding property name is **lisa.agent.transaction.max.graph.size**. For example:

```
<property key="lisa.agent.transaction.max.graph.size" value="2000000"/>
```

The VSE log includes the following error: com.itko.lisa.vse.stateful.protocol.java.listen.DefaultJavaPlaybackCallback - Timed out waiting on response.

Go to DevTest Portal and increase the value of the **Reply timeout** setting. The default value is 5000.

You can also configure this setting in the **rules.xml** file. The corresponding property name is **lisa.agent.virtualize.reply.timeout**. For example:

```
<property key="lisa.agent.virtualize.reply.timeout" value="60000"/>
```

DevTest CICS Agent

The DevTest CICS Agent is a z/OS CICS resident agent that provides functions to support the VSE. The agent is implemented as a combination of long-running task TKOA and CICS exits XPCREQ, XPCREQC, XEIIN, and XEIOUT. The agent can be activated automatically through the CICS PLT, or manually through the TKOI transaction to install and TKOR transaction to remove. The only configuration that is required is the location of the DevTest LPAR Agent.

The following graphic shows the components of the DevTest LPAR Agent and DevTest CICS Agent.

Install the CICS Agent

To get the z/OS agents (CICS and LPAR agents), contact CA Customer Support.

The DevTest CICS Agent package consists of the following files:

- iTKOCICSAgentDoc.pdf
- iTKOCICSAgentLoadCICS42
- iTKOCICSAgentLoadCICS51
- iTKOCICSAgentLoadCICS52
- iTKOCICSAgentLoadCICS53
- iTKOCICSAgentLoadCICS54
- iTKOCICSAgentCntl

Follow these steps:

1. FTP the files to the target z/OS system.
   Use the load library that matches your CICS version:

   - For CICS Version 4.2, use iTKOCICSAgentLoadCICS42
   - For CICS Version 5.1, use iTKOCICSAgentLoadCICS51
   - For CICS Version 5.2, use iTKOCICSAgentLoadCICS52
   - For CICS Version 5.3, use iTKOCICSAgentLoadCICS53
   - For CICS Version 5.4, use iTKOCICSAgentLoadCICS54

   A simple method to determine your version of CICS is to run the following transaction and check the value of CICSTslevel. CICSTslevel has the version/release/maintenance level in the format vvrrmm: CECI INQUIRE SYSTEM CICSTslevel.

   The files iTKOCICSAgentLoadCICSvvv and iTKOCICSAgentCntl are FTP'd to the target z/OS system in binary mode with the z/OS attributes indicated in the example that follows.

   The following example of command-line FTP shows the proper transfer to z/OS of distribution files data sets **iTKOCICSAgentCntl** and **iTKOCICSAgentLoadCICS42** to z/OS data sets ITKO.CICSAGNT.CNTL.UNLOAD and ITKO.CICSAGNT.LOAD.UNLOAD. You can change the z/OS file names for your data set naming conventions.

```
ftp> quote SITE LRECL=80 RECFM=FB BLKSIZE=3120 TRACKS PRI=10 SEC=1
200 SITE command was accepted
ftp> bin
200 Representation type is Image
ftp> put iTKOCICSAgentLoadCICS42 'ITKO.CICSAGNT.LOAD.UNLOAD'
200 Port request OK.
125 Storing data set ITKO.CICSAGNT.LOAD.UNLOAD
250 Transfer completed successfully.
ftp> put iTKOCICSAgentCntl 'ITKO.CICSAGNT.CNTL.UNLOAD'
200 Port request OK.
125 Storing data set ITKO.CICSAGNT.CNTL.UNLOAD
250 Transfer completed successfully.
```

2. Restore the Partitioned Datasets (PDSs).

   The CICS Agent load library PDS and control (JCL) library are restored with the TSO RECEIVE command, which typically runs in a batch job.

   The following sample JCL restores the PDSs from the FTP'd files data sets **ITKO.CICSAGNT.LOAD** and **ITKO.CICSAGNT.CNTL** on volume VOL001. The data set names can be changed for your site data set naming conventions.

```
//job
//*
//* Unload the LOAD Library with TSO RECEIVE
//*
//UNLOAD EXEC PGM=IKJEFT01,DYNAMNBR=10
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSTSIN DD *
RECEIVE INDSNAME('ITKO.CICSAGNT.LOAD.UNLOAD')
   DATASET('ITKO.CICSAGNT.LOAD') VOLUME(VOL001)
RECEIVE INDSNAME('ITKO.CICSAGNT.CNTL.UNLOAD')
   DATASET('ITKO.CICSAGNT.CNTL') VOLUME(VOL001)
/*
```

3. Ensure that the TCP/IP interface is installed and started in CICS.
   The CICS Agent requires that the CICS TCP/IP interface is installed and started. To verify that the TCP/IP interface is installed, use the transaction: EZAO, INQUIRE, CICS.

4. Create the CICS Agent configuration file.
   The CICS agent communicates with the LPAR Agent through TCP/IP. The CICS Agent configuration file identifies the IP address and TCP port number of the LPAR Agent and other configurable items. This file is accessed as a Transient Data (TD) Queue during agent startup. This data set has attributes: DSORG=PS, RECFM=FB, LRECL=80.
   Rules for coding the configuration are:

   a. Blank lines (all spaces) are ignored.

   b. Lines with the first character of **#** or ***** are considered as comments and ignored.

   c. Lines processed must be in the format: *keyword = value.*

   d. Spaces are allowed before and after the keyword, equal sign and value.

   e. Character case in the *keyword* is ignored. Upper or lower case may be used as desired for readability.

   f. Text after the space following the value is considered comments and ignored.

   Required Configuration
   The keywords that are used in the required configuration are:

   - **LPARAgentIP**
     The IP address of the LPAR agent in dotted decimal notation (nnn.nnn.nnn.nnn). Maximum length is 15 characters.

   - **LPARAgentPort**
     The TCP/IP port number, specified as 1 to 5 digits.

   Optional Configuration - Timeout Values
   The keywords that you can use for timeout values are:

   - **PlaybackTimeout**
     The number of seconds that the CICS Agent waits for a playback response from VSE . Only code this configuration item when you require a timeout that is different from the default.
     Specified as 1 to 5 digits.
     Default: 30 seconds.

   - **SelectTimeout**
     The number of seconds that the CICS Agent is inactive while waiting for work. Normally the CICS Agent is interrupt driven. When this timeout occurs, the CICS Agent monitors various queues. Only code this configuration item when you require a timeout that is different from the default.
     Specified as 1 to 5 digits.
     Default: 2 seconds.

- **AppTestTimeout**
  The number of seconds that the CICS Agent waits for an Application Test request to be processed.
  Specified as 1 to 5 digits.
  Default: 30 seconds.

Optional Configuration - System Identifier Values
The keywords that you can use for system identifier values are:

- **CICS_SYSID**
  The 1-4 character CICS System ID of the CICS region*.

- **CICS_APPLID**
  The 1-8 character VTAM APPLID used by the CICS region*.

- **CICS_JOBNAME**
  The 1-8 character job or started task name of the CICS region*.

- **LPARName**
  The 1-8 character name of the z/OS LPAR the CICS region is running in*.

- **SYSPLEXName**
  The 1-8 character name of the z/OS SYSPLEX the CICS region is running in*.

- **CICSPLEXName**
  The 1-8 character name of the CICSPLEX the CICS region is running in. The default for this value is spaces. This value may also be set in user exit 1.

- **UserGroupName**
  The 8 uppercase character name of the user defined group this CICS region is in. The default for this value is spaces. This value may also be set in user exit 1.

- **AppTestTran**
  The CICS transaction ID to be started to process an Application Test request. This value is specified as 1 to 4 characters. The default is “TKOT”.

* These values should not be changed unless the rare case of more than one CICS region connecting to an LPAR agent with the same value exists.

Optional Configuration - Agent Control
The keywords in this section modify the default behavior of the CICS Agent. The keywords that you can use for Agent Control are:

- **BypassByCaller**
  A 1-8 character CICS program name. If this program is the calling program on a LINK , virtualization is bypassed. This allows a test harness to invoke the program when a virtual service is deployed. The recommended value is **ITKOTST**, which is the program used by the CICS Agent to LINK to a program run by a DevTest baseline. There is no default value.

The member CONFIG of the control PDS contains the following example:

```
# CA DevTest CICS Agent Configuration
#
# Required configuration
LPARAgentIP = 192.168.0.100 LPAR Agent IPv4 Address
LPARAgentPort = 3998 Connect to LPAR Agent on this port
#
```

```
# Optional timeout configuration
#PlaybackTimeout = 10 10 minute playback timeout
#SelectTimeout = 2 2 second select timeout
#AppTestTimeout = 60 60 second apptest timeout
#
# Optional system identifier override values
# These should normally not be coded (leave them commented)
#CICS_SYSID = xxxx 4 character override CICS System ID
#CICS_APPLID = xxxxxxxx 8 character override CICS APPLID
#CICS_JOBNAME = xxxxxxxx 8 character override CICS Job Name
#LPARName = xxxxxxxx 8 character override LPAR Name
#SYSPLEXName = xxxxxxxx 8 character override Sysplex Name
#CICSPLEXName = xxxxxxxx 8 character override CICSPlex Name
#UserGroupName = xxxxxxxx 8 character User Group Name
#AppTestTran = TKOT 4 character Transaction ID for AppTest
#
# Optional agent control configuration
BypassByCaller = ITKOTST   8 character program name. If this is the
#                          calling program on a LINK, bypass VSE.
#                          This allows a test harness to invoke the
#                          program when a virtual service is deployed.
```

5. Define CICS Agent resources to CICS.
   The following resources must be defined to CICS:

   - **TKOC**
     Configuration Transient Data Queue.
     Type: TDQUEUE

   - **TKOI**
     Transaction to install CICS exits and start agent transaction (TKOA). Typically run during PLTPI processing.
     Type: TRANSACTION

   - **TKOR**
     Transaction to remove CICS exits and stop agent transaction (TKOA). Typically run during PLTSD processing.
     Type: TRANSACTION

   - **TKOA**
     Agent transaction. Typically started by TKOI and stopped by TKOR, although it may be started and stopped by the monitor transaction (TKOM).
     Type: TRANSACTION

   - **TKOM**
     Monitor transaction. Used to manually start and stop exits and agent and monitor agent processing.
     Type: TRANSACTION

   - **ITKOAGNT**
     Agent program. Started by TKOA.
     Type: PROGRAM

   - **ITKOINST**
     Exit installation program. Started by TKOI.
     Type: PROGRAM

   - **ITKOREMV**
     Exit removal program. Started by TKOR.
     Type: PROGRAM

- **ITKPCRQ**
  CICS XPCREQ (before CICS LINK exit). Enabled by TKOI, disabled by TKOR.
  Type: PROGRAM

- **ITKPCRC**
  CICS XPCREQC exit (after CICS LINK exit). Enabled by TKOI, disabled by TKOR.
  Type: PROGRAM

- **ITKOXEIN**
  CICS XEIIN exit (before CICS Command exit) Enabled by TKOI, disabled by TKOR.
  Type: PROGRAM

- **ITKOXEIO**
  CICS XEIOUT exit (after CICS Command exit) Enabled by TKOI, disabled by TKOR.
  Type: PROGRAM

- **ITKOTABN**
  CICS XPCABND exit (Task Abend exit) Enabled by TKOI, disabled by TKOR.
  Type: PROGRAM

- **ITKOMON**
  Monitor program.
  Type: PROGRAM

- **TKOT**
  Application Test transaction.
  Type: TRANSACTION

- **ITKOTST**
  Application Test driver program.
  Type: PROGRAM

JCL for a job to install these resources through DFHCSDUP is in the control PDS named DFHCSDUP.

6. Add exit enable/disable to the CICS PLT.
   You can start the CICS Agent and exits manually with the TKOI transaction.
   You can start the CICS Agent and exits manually with the TKOR transaction.
   A simple way to enable the CICS exits and start the agent running is to place entries in the PLTSI (CICS startup). A PLTPI example follows. If the SIP has PLTPI=SI, then the following table would be assembled as DFHPLTSI. The CICS TCP/IP interface is started before the DevTest exits and agent.

```
DFHPLT TYPE=INITIAL,SUFFIX=SI
*
* The following programs are run in the second pass of PLTPI
*
         DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
* The following programs are run in the third pass of PLTPI
*
         DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
         DFHPLT TYPE=ENTRY,PROGRAM=ITKOINST
         DFHPLT TYPE=FINAL
         END
```

   A way to disable the DevTest exits and stop the agent when it shuts down is to include an entry in the PLTSD (CICS shutdown). A PLTSD example follows. If the SIP has PLTSD=SD, then the following table is assembled as DFHPLTSD. The DevTest exits and agent are stopped before the CICS TCP/IP interface.

```
DFHPLT TYPE=INITIAL,SUFFIX=SD
*
* The following programs are run in the first pass of PLTSD
*
         DFHPLT TYPE=ENTRY,PROGRAM=ITKOREMV
         DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
*
         DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
* The following programs are run in the second pass of PLTSD
*
         DFHPLT TYPE=FINAL
         END
```

7. Modify CICS startup JCL.
   Add the DevTest CICS Agent load modules to CICS by copying them to an existing DFHRPL load library. Alternatively, you could add the DevTest CICS Agent load library to the DFHRPL concatenation.
   JCL must also be added to define the agent configuration file (from Step 5). If the data set is named ITKO.AGENT.CONFIG and the TD Queue TKOC was defined with DDNAME (ITKOACFG), the following JCL statement could be used:

   ```
   //ITKOACFG DD DSN=ITKO.AGENT.CONFIG,DISP=SHR
   ```

CICS Agent Storage Requirements

The DevTest CICS agent is a long running task. The agent requires approximately 16K of program storage in ESDSA and approximately 1K of dynamic storage in ECDSA.

The DevTest CICS exits require approximately 34K of program storage in ERDSA and approximately 3K of dynamic storage (for each transaction) in ECDSA.

The recorded images of a CICS LINK are built in ECDSA when using the VSE recorder.

The exits build recorded images and pass to the CICS agent. The CICS agent sends them to the LPAR agent and then frees them, so they are short lived.

The size of a recorded image is approximately 330 + 2 * ((COMMAREA length or sum of all CONTAINER lengths) + INPUTMSG length).

The amount of ECDSA required to accommodate the recorded images depends on the average size that is calculated in the previous paragraph, and the transaction rate.

Playback images of a CICS LINK are also built in ECDSA when a Virtual Service is deployed.

At the beginning of a virtualized CICS LINK, the exits build a playback request and pass it to the CICS agent. The CICS agent forwards it to the LPAR agent.

When the CICS agent receives the playback response, the response is matched to the original request. Both the request and response are passed to the exits, which free them and complete the CICS LINK.

The size of a playback request and response is approximately 300 + (COMMAREA length or sum of all CONTAINER lengths).

The amount of ECDSA required to accommodate the playback images depends on the average size that is calculated in the previous paragraph, and the transaction rate.

CICS Exit Considerations

The DevTest CICS Agent uses the following CICS exit points:

- XPCREQ (before CICS LINK exit)
- XPCREQC (after CICS LINK exit)
- XEIIN (before CICS Command exit)
- XEIOUT (after CICS Command exit)
- XPCABND (Transaction ABEND)

XPCREQ and XPCREQC are used for CICS LINK virtualization.

XEIIN and XEIOUT are used for CICS Distributed Transaction Processing (DTP) virtualization.

XPCABND (and the other exits) is used for Application Insight.

XPCREQ and XPCREQC are required. If you do not require DTP virtualization or Application Insight, you can remove XEIIN and XEIOUT. To remove them, delete ITKOXEIN and ITKOXEIO from the load library (or rename them).

If you do not require Application Insight, you can remove XPCABND. To remove it, delete ITKOTABN from the load library (or rename it).

Coexistence with Other Exits

If any exits of type XPCREQ, XPCREQC, XEIIN, XEIOUT, or XPCABND are already installed, the DevTest exits may be able to coexist with them.

CICS calls the exits in the order in which they are enabled. You can control the order with the placement of the ITKOINST entry in the PLT.

DevTest CICS Agent Exit Behavior Summary

- The DevTest CICS Agent exits can be entered with a previous return code (addressed by UEPCRCA) that is nonzero. In this case, the exit returns with the return code (R15) set to the previous return code (which another exit set). The DevTest exits can coexist with other exits if they are not the first exit that CICS invokes.
- The DevTest CICS LINK exits (XPCREQ and XPCREQC) use the token (UEPPCTOK) passed by CICS to the exits to pass a DevTest control block address. If another XPCREQ or XPCREQC corrupts that address, then an abend can occur, or the DevTest XPCREQC exit can write a one-time message: **ITKO0014 - ERROR - Corrupt token in ITKPCRC Exit**.
- If the DevTest CICS Agent **before** exits (XPCREQ and XEIIN) must bypass a command to virtualize it, R15 returns a return code of **bypass** (UERCBYP). If CICS invokes another exit of the same type after the DevTest exit, the exit should propagate the bypass return code back to CICS. If it does not, the command is not bypassed, and virtualization fails.
- If the **recursion** indicator (addressed by UEPRECUR) is nonzero, the DevTest XPCREQ and XPCREQC exits return immediately. This return eliminates issues that can arise if another exit of this type uses a CICS LINK command.

- The XPCABND (Transaction ABEND) exit always returns with UEPCRCA in R15, so it should coexist with other XPCABND exits. The only exception to this rule is the event of an XPI function returning **task purged** (see the following).

- If any of the DevTest CICS Agent exits receive a return code of **task purged** from the CICS XPI (Exit Programming Interface), it returns with **purge** (UERCPURG) in R15. The exit overrides the previous return code (addressed by UEPCRCA) with UERCPURG. If CICS invokes another exit of the same type after the DevTest exit, the exit should propagate the purge return code back to CICS.

CICS Agent User Exits

This topic describes the CICS Agent user exits that are available.

- ITKOUEX1 - DevTest CICS Agent Initialization Exit
- ITKOUEX2 - DevTest CICS Agent Termination Exit
- ITKOUEX3 - LISA CICS COMMAREA Consolidation Exit

ITKOUEX1 - DevTest CICS Agent Initialization Exit

This exit is called during the DevTest CICS Agent initialization. The exit can be used to:

- Indicate the user group that this CICS region is a member of

- Indicate the CICSPlex that this CICS region is a member of

- Allocate storage and pass the address/length to other exits using the token parameter

- Perform other initialization as required by other user exits

This exit is invoked through a CICS LINK, so it can be written in any language that CICS supports. During initialization, a CICS LINK to ITKOUEX1 is attempted.

If this exit is not necessary, take one of the following actions:

- Do not include a module that is named ITKOUEX1 in the DFHRPL concatenation.

- Disable the exit in the PPT.

Parameters are passed in COMMAREA.

The following table describes the parameters:

8 bytes that the user exits can use. Maintained across exit calls.

Parameter	Length in Bytes	Input or Output	Data Type	Description
Token	8	Input /Output	Unknown	
Group	8	Output	Character	Set by exit. Identifies a user group of which this CICS is a member.

CICSPlex	8	Output	Character	Set by exit. Identifies a CICSPlex of which this CICS is a member.
Sysid	4	Input /Output	Character	CICS SYSID from CICS ASSIGN SYSID().*
Applid	8	Input /Output	Character	CICS APPLID from CICS ASSIGN APPLID().*
Jobname	8	Input /Output	Character	CICS job name from CICS INQUIRE SYSTEM JOBNAME().*
LPAR	8	Input /Output	Character	LPAR name from CVTSNAME in which this CICS resides.*
Sysplex	8	Input /Output	Character	SYSPLEX name from ECVTSPLX in which this CICS resides.*

*These parameters should only be changed by the exit in the rare case that multiple CICS regions connect to the same DevTest LPAR Agent and with duplicate values. These values have no effect on the operation of the CICS region or the DevTest CICS Agent and are only used by the DevTest registry to establish the uniqueness of DevTest CICS Agents.

ITKO.CICSAGNT.CNTL(SAMP1EX1) contains a sample COBOL ITKOUEX1. This sample sets the user group to the first four characters of the CICS job name.

ITKO.CICSAGNT.CNTL(ITKOEX1C) contains the COBOL COMMAREA copybook.

ITKO.CICSAGNT.CNTL(ITKOEX1A) contains the Assembler COMMAREA copybook.

ITKO.CICSAGNT.CNTL(COMPUEX1) contains COBOL compile JCL.

ITKOUEX2 - DevTest CICS Agent Termination Exit

This exit is called during the DevTest CICS Agent termination. It can be used to perform any cleanup that user exits require (for example, free storage and close files).

This exit is invoked through a CICS LINK, so it can be written in any language that CICS supports. During the agent termination, a CICS LINK to ITKOUEX2 is attempted. If this exit is not necessary, take one of the following actions:

- Do not include a module that is named ITKOUEX2 in the DFHRPL concatenation.
- Disable the exit in the PPT.

Parameters are passed in COMMAREA.

The following table describes the parameters:

Parameter	Length in Bytes	Input or Output	Data Type	Description
Token	8	Input /Output	Unknown	8 bytes that the user exits can use. Maintained across exit calls.

ITKO.CICSAGNT.CNTL(SAMP1EX2) contains sample COBOL ITKOUEX2.

ITKO.CICSAGNT.CNTL(ITKOEX2C) contains the COBOL COMMAREA copybook.

ITKO.CICSAGNT.CNTL(ITKOEX2A) contains the Assembler COMMAREA copybook.

ITKO.CICSAGNT.CNTL(COMPUEX2) contains the COBOL compile JCL.

ITKOUEX3 - LISA CICS COMMAREA Consolidation Exit

This exit is used to manage a COMMAREA that contains addresses that refer to storage outside of the COMMAREA. This exit consolidates the storage fragments with the COMMAREA to create a single contiguous COMMAREA.

This exit is called:

- During the CICS LINK recording, for both the before and after images, to create a replacement COMMAREA with no addresses referencing data outside the COMMAREA. The exit is responsible for GETMAINing the new storage area. The DevTest CICS Agent will FREEMAIN the area. Addresses in the COMMAREA that point to locations within the COMMAREA are also problematic. In this case, the exit should convert the addresses to relative offsets during recording and should revert them to real addresses during playback.
- During CICS LINK playback (virtualization) to restore the original COMMAREA from the DevTest response.
- During CICS Agent initialization (after user exit 1) so the exit can perform any additional initialization tasks.
- During CICS Agent termination (before user exit 2) so that the exit can perform any additional termination tasks.

This exit is invoked through a CICS LINK, so it can be written in any language that CICS supports. However, assembler language is best suited to address manipulation and variable length data movement. During initialization, a CICS LINK to ITKOUEX3 is attempted.

If this exit is unnecessary, take one of the following actions:

- Do not include a module that is named ITKOUEX3 in the DFHRPL concatenation.
- Disable the exit in the PPT.

Parameters are passed in COMMAREA.

The following table describes the parameters:

Parameter	Length in Bytes	Input or Output	Data Type	Description
Token	8	Input /Output	Unknown	8 bytes that the user exits can use. Maintained across exit calls.
Call Type	1	Input	Binary	12 indicates Playback 16 indicates Termination
Image Type	1	Input	Binary	4 indicates a Before Image 8 indicates an After Image
COMMAREA Address	4	Input	Address	Address of COMMAREA passed on the application CICS LINK.
	2	Input	Binary	

COMMAREA Length				Length of COMMAREA passed on the application CICS LINK.
Calling Program	8	Input	Character	Name of program issuing the CICS LINK command.
Target Program	8	Input	Character	Name of program being LINKed to.
New COMMAREA Address	4	Input /Output	Address	For recording: The address of the new consolidated COMMAREA is placed here by this exit. This exit typically GETMAINs the area. For Playback: The address of the new consolidated COMMAREA from the VSE response is placed here. This exit copies the contents to the application COMMAREA.
New COMMAREA Length	2	Input /Output	Binary	For recording: The length of the new consolidated COMMAREA is placed here by this exit. For Playback: The length of the new consolidated COMMAREA from the VSE response is placed here for this exit to use.
Return Code	1	Output	Binary	0 indicates that the COMMAREA was successfully consolidated. The DevTest CICS Agent will use the **New COMMAREA Address** and **New COMMAREA Length** to create the recorded image. 4 indicates that this exit was successful; however there is no new COMMAREA created. The DevTest CICS Agent uses the COMMAREA from the CICS LINK to create the recorded image. 8 indicates that this exit failed and there is no new COMMAREA. The DevTest CICS Agent returns to the application from the CICS LINK with EIBRCODE, EIBRESP, and EIBRESP2 as indicated in the following parameters. 12 indicates all the processing of return code 8, and the indication that this exit should be removed from future DevTest CICS Link recording and playback.
EIBRCODE	6	Output	Binary	The EIBRCODE value to be used when the **Return Code** is 8 or 12.
EIBRESP	4	Output	Binary	The EIBRRESP value to be used when the **Return Code** is 8 or 12.
EIBRESP2	4	Output	Binary	The EIBRRESP2 value to be used when the **Return Code** is 8 or 12.

There are three samples of ITKOUEX3:

Assembler language sample SAMP1EX3 is in ITKO.CICSAGNT.CNTL(SAMP1EX3).

This sample consolidates a COMMAREA that contains pointers and addresses so that the COMMAREA presented during recording and playback is a concatenation of the original COMMAREA and the blocks of storage addressed by the pointers and addresses. See the sample for a full description.

Assembler language sample SAMP2EX3 is in ITKO.CICSAGNT.CNTL(SAMP2EX3).

This sample manages a COMMAREA in the format that the Software AG Natural language produces when calling COBOL with the Natural parameter CALLRPL=(ALL,3) or CALLRPL=(ALL,4). With this format of COMMAREA, the COMMAREA length is 12 (when CALLRPL=(ALL,3) is used) and 16 (when CALLRPL=(ALL,4) is used). The first fullword in the COMMAREA is the address of a parameter list. The high-order bit that is set to 1 in the address indicates the last parameter. The third fullword is the address of a list of parameter lengths. See the sample for a full description.

Assembler language sample SAMP3EX3 is in ITKO.CICSAGNT.CNTL(SAMP3EX3).

This sample handles virtualizing a CICS program that has data that is passed in or out with a Temporary Storage Queue (TSQ). See the sample for a full description.

ITKO.CICSAGNT.CNTL(ITKOEX3A) contains the Assembler COMMAREA copybook.

ITKO.CICSAGNT.CNTL(ASMBUEX3) contains the assembly JCL.

CICS Agent Operation

Starting the DevTest CICS Agent and Exits

The CICS Agent and exits are started during CICS startup PLT processing. The agent and exits can also be started manually by running the TKOI transaction or with the monitor transaction TKOM. The TKOI transaction installs the exits and starts the agent transaction, TKOA.

Stopping the DevTest CICS Agent and Exits

The CICS Agent and exits are stopped during CICS shutdown PLT processing. The agent and exits can also be stopped manually by running the TKOR transaction or with the monitor transaction TKOM.

In rare circumstances, the CICS Agent transaction, TKOA, can fail to terminate. If you suspect this is the case, use the CEMT I TAS command to confirm whether TKOA is still running. If it is, use the CEMT SET TAS(nnn) PURGE or CEMT SET TAS(nnn) FORCEPURGE command as appropriate to terminate TKOA.

Impact on Virtualized CICS LINK Commands

In rare cases, virtualization of a CICS LINK can fail. In this event, the CICS LINK returns to the application with the "Invalid Request" (INVREQ) condition: EIBRCODE=x'E00000000000' and EIBRESP=16. This situation typically results in a transaction abend, unless the application has INVREQ condition handling implemented. You can use EIBRESP2 to determine the exact cause of the virtualization failure:

- **500 / x'1F4'**
  No playback response was received from the VSE. This is most likely due to a timeout. The timeout interval is set to 10 seconds. Contact Customer Support if the timeout value must be changed.

- **501 / x'1F5'**
  DevTest CICS Agent is down. The "After CICS LINK" exit detected that the DevTestCICS Agent is down. No playback response can be expected.

- **502 / x'1F6'**
  Virtualization failed. A playback response was received from the Service Virtualization and updating the CICS LINK COMMAREA or CHANNEL/CONTAINERs failed. See the CICS job log for a CICS command failure message.

- **503 / x'1F7'**
  The DevTest LPAR Agent was shut down, resulting in cancellation of all pending playback requests.

Impact on Virtualized CICS DTP/MRO Commands

In rare cases, virtualization of a CICS DTP/MRO command (ALLOCATE, FREE, SEND, RECEIVE, CONVERSE, or EXTRACT ATTRIBUTES) can fail. In this event, the CICS command returns to the application with the "Invalid Request" (INVREQ) condition: EIBRCODE=x'E00000000000' and EIBRESP=500. This situation typically results in a transaction abend, unless the application has INVREQ condition handling in place. You can use EIBRESP2 to determine the exact cause of the virtualization failure:

- **501 / x'1F5'**
  DevTest CICS Agent is down. The "After CICS LINK" exit detected that the DevTest CICS Agent is down. No playback response can be expected.

- **502 / x'1F6'**
  No response was received from Service Virtualization.

- **503 / x'1F7'**
  An invalid response was received from the Service Virtualization. See the CICS job log for a CICS command failure message.

- **504 / x'1F8'**
  The session table was full.

- **505 / x’1F9’**
  An attempt to add the playback request to the playback queue failed.

- **506 / x’1FA’**
  The XEIIN WAIT timeout expired before a response was received from the Service Virtualization.

- **507 / x’1FB’**
  A GETMAIN for the request or response failed.

Monitor Transaction (TKOM)

With the Monitor Transaction (TKOM), you can:

- Check the status of the agent and exits
- View the Global Work Area (GWA) in hexidecimal
- View some of the GWA fields formatted/decoded
- Check the recording and playback counts
- Check the program table counts

- View the program table
- Start and stop the agent
- Start and stop the exits
- Check the current depth (count) and high watermarks (Max) of the three main queues:
  - **Recording Queue (RecQ):**
  - Defines the queue of recorded images.
  - **Playback Queue (PlbQ):**
  - Defines the queue of playback images to be sent to the DevTest LPAR Agent.
  - **LPAR Agent Queue (LPAQ):**
  - Defines the queue of playback images waiting for a response from the DevTest LPAR Agent.
- Determine the task number of the agent transaction

The Monitor Transaction displays as follows:

```
                              ITKO AGENT MONITOR                CICS 040200
XPCREQ Started   XPCREQC Started   XEIIN  Started   XEIOUT  Started
GWA:00167534  Length:00F2  EYE:ITKOEXITGWA V090500C040200 Agent ECB:807BE548
0000 C9E3D2D6 C5E7C9E3 C7E6C140 E5F0F9F0 F5F0F0C3 F0F4F0F2 F0F00000 000000F2
0020 03050000 001F0000 00000000 00000000 807BE548 00000002 00000000 00000000
0040 00000000 00000013 00000000 00000000 00000000 00000000 00000000 00000000
0060 00000000 00000000 00000000 1B79C000 00000000 00000000 00000000 00000000
0080 000001BA 00000002 00000204 00000000 00000003 00000000 1B7A6810 1B7A8760
00A0 1B7B4300 E2F6F7F0 C3C9C3E2 E3E2F140 C3C9C3E2 F1404040 E2F0E6F1 40404040
00C0 E2E5E2C3 D7D3C5E7 40404040 40404040 40404040 40404040 00000000 00000000
00E0 00000000 00000000 00000000 00000000 0303
STATUS 1: EXITS INITALIZED,RUNNING
STATUS 2: AGENT RUNNING,CONNECTED,PF OFF
RecQ Seq: 00000002 Head: 00000000 Tail: 00000000 Count: 00000000 Max: 00000013
PLbQ Seq: 00000000 Head: 00000000 Tail: 00000000 Count: 00000000 Max: 00000000
LPAQ               Head: 00000000 Tail: 00000000 Count: 00000000 Max: 00000000
TstQ Seq: 00000000 Head: 00000000 Tail: 00000000 Count: 00000000 Max: 00000000
CTBL:01F4 Token: 00000000 Used:0000 XTBL:01F4 Used:0000  FMHT:01F4 Used:0000
Prog List @ 1B79C000 U Use: 00000000 Rec Ct: 00000000 Plb Ct: 00000000
Record Count: 00000204  Playback Count: 00000000  Event Count: 00000003
Agent ENQ: TKOA  Task 00000036
Press CLEAR to exit
 PF1     PF2     PF3     PF4       PF5      PF6       PF7       PF8      PF9
Unused Refresh Pgm List Unused  Strt Agnt Stop Agnt Strt Exit Stop Exit   PF
```

CICS Agent Messages

Messages are written to the operator console and CICS job log by the DevTest CICS Agent and exits.

The DevTest exit installer (Transaction TKOI, Program ITKOINST) writes the following messages:

- ITKO0001 - iTKO exit init failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0002 - iTKO exit [XPCREQ | XPCREQC] at xxxxxxxx GWA at xxxxxxxx Len 9999 is enabled and started

- ITKO0003 - iTKO exit [XPCREQ | XPCREQC] was enabled but not started
- ITKO0004 - iTKO exit GWA error - Address is xxxxxxxx Length is 9999

The DevTest exit uninstaller (Transaction TKOR, Program ITKOREMV) writes the following messages:

- ITKO0005 - iTKO exit remove failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0006 - iTKO exit [XPCREQ | XPCREQC] disabled

The CICS XPCREQ (before CICS LINK) Exit (module ITKPCRQ) or the CICS XPCREQC (after CICS LINK) exit (module ITKPCRC) write the following messages:

- ITKO0007 - GWA missing or too small for [ITKPCRQ | ITKPCRC] Exit
- ITKO0008 - Program program-name Error EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0009 - Recording Abandoned of calling-program LINK to called-program Tran xxxx Task 9999999 Term xxxx
- ITKO0010 - [Recorded Image | Playback Image | LPAR Agent] PLO failed in [ITKPCRQ | ITKPCRC] Exit
- ITKO0011 - BEFORE/AFTER image creation failed with reason code xxx
- ITKO0012 - Image update failed with reason code 999
- ITKO0013 - PDU Parsing Error in ITKPCRC Exit
- ITKO0014 - ERROR - Corrupt token in ITKPCRC Exit

The DevTest CICS Agent (Transaction TKOA, Program ITKOAGNT) writes the following messages:

- ITKO0101 - Agent waiting for exits
- ITKO0102 - Agent initialization complete
- ITKO0103 - Agent shutting down
- ITKO0104 - iTKO [Recorded Image | Playback Image | LPAR Agent Img] PLO failed in Agent
- ITKO0105 - iTKO Agent started while another agent is running
- ITKO0106 - iTKO Agent connected to LPAR Agent
- ITKO0107 - Agent Config LPAR Agent IP: nnn.nnn.nnn.nnn Port 99999
- ITKO0108 - iTKO Agent GWA error- Address is xxxxxxxx Length is 9999
- ITKO0109 - Socket func xxxxxxxx failed. RETCODE=xXXXXXXXX ERRNO=999999
- ITKO0110 - Agent CICS cmd failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999

- ITKO0111 - iTKO Agent disconnected from LPAR Agent
- ITKO0112 - iTKO Agent error sending recorded image to LPAR Agent
- ITKO0113 - iTKO Agent error sending playback image to LPAR Agent
- ITKO0114 - iTKO Agent error sending response to LPAR Agent
- ITKO0115 - iTKO Agent error receiving request from LPAR Agent
- ITKO0116 - iTKO Agent error - invalid LPAR Agent IP address
- ITKO0117 - iTKO Agent released nnnnnn recorded images
- ITKO0118 - iTKO Agent canceled nnnnnn playback images
- ITKO0119 - iTKO Agent canceled nnnnnn pending playback images
- ITKO0120 - ITKO Agent released playback image: token xxxxxxxxxxxxxxxx Reason xxx
- ITKO0121 - iTKO Agent error reading configuration
- ITKO0122 - Agent TCP/IP API Timeout
- ITKO0123 - Agent waiting for TCP/IP API
- ITKO0124 - ITKO Agent response timeout for Token xxxxxxxxxxxxxxxx
- ITKO0125 - Call to ITKOUEX1/ITKOUEX2/ITKOUEX3 failed
- ITKO0126 - Handshake error encountered. Shutting down Agent
- ITKO0127 - ITKO Agent Test request failed -- Token: xxxxxxxxxxxxxxxx Reason xxx
- ITKO0128 - ITKO Agent Test Req timeout for Token xxxxxxxxxxxxxxxx
- ITKO0129 - ITKO Agent Test Req timeout fail for Token xxxxxxxxxxxxxxxx
- ITKO0130 - ITKO Agent cancelled 999999 pending Lisa Test requests
- ITKO0131 - Incompatible LPAR agent vvvv. Shutting down Agent.
- ITKO0132 - Config line xxxxx error-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

The CICS XEIIN (before CICS command) Exit (module ITKOXEIN) or the CICS XEIOUT (after CICS command) exit (module ITKOXEIO) write the following messages:

- ITKO200 - ITKOXEIN XPI INQ_APPLICATION_DATA Failed
- ITKO201 - ITKOXEIN/ITKOXEIO XPI Getmain Dynamic Stg Failed
- ITKO202 - ITKOXEIN/ITKOXEIO XPI Freemain Dynamic Stg Failed
- ITKO203 - ITKOXEIN/ITKOXEIO XPI Getmain Image Stg Failed
- ITKO204 - ITKOXEIN/ITKOXEIO XPI Freemain Image Stg Failed
- ITKO205 - ITKOXEIN Task table full - discarding image

- ITKO206 - ITKOXEIN XPI WAIT Failed
- ITKO207 - ITKOXEIN XPI Freemain plbk req/rsp failed
- ITKO208 - ITKOXEIO Add to recording queue failed
- ITKO209 - ITKOXEIN/ITKOXEIO XPI Freemain SET() Stg Failed
- ITKO210 - ITKOXEIN XPI Getmain SET() Stg Failed
- ITKO0300 - Test CICS cmd failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0301 - DevTest CICS exit not running.
- ITKO0302 - DevTest CICS GWA not available.
- ITKO0303 - DevTest CICS Agent not running/connect.
- ITKO0304 - ITKOLDEL failed for test request queue.
- ITKO0305 - No TEST request found.
- ITKO0306 - ITKOTST terminated due to errors.

More Information:

- DevTest CICS Agent Message Descriptions

DevTest CICS Agent Message Descriptions

This section lists and describes **DevTest** CICS agent messages.

- ITKO0001 - iTKO exit init failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0002 - iTKO exit [XPCREQ | XPCREQC] at xxxxxxxx GWA at xxxxxxxx Len 9999 is enabled and started
- ITKO0003 - iTKO exit [XPCREQ | XPCREQC] was enabled but not started
- ITKO0004 - iTKO exit GWA error - Address is xxxxxxxx Length is 9999
- ITKO0005 - iTKO exit remove failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0006 - iTKO exit [XPCREQ | XPCREQC] disabled
- ITKO0007 - GWA missing or too small for [ITKPCRQ | ITKPCRC] Exit
- ITKO0008 - Program program-name Error EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0009 - Recording Abandoned of calling-program LINK to called-program Tran xxxx Task 9999999 Term xxxx
- ITKO0010 - [Recorded Image | Playback Image | LPAR Agent list] PLO failed in [ITKPCRQ | ITKPCRC] Exit
- ITKO0011 - BEFORE/AFTER image creation failed with reason code xxx

- ITKO0012 - Image update failed with reason code 999
- ITKO0013 - PDU Parsing Error in ITKPCRC Exit
- ITKO0014 - ERROR - Corrupt token in ITKPCRC Exit
- ITKO0101 - AGENT waiting for exits
- ITKO0102 - AGENT initialization complete
- ITKO0103 - AGENT shutting down
- ITKO0104 - iTKO [Recorded Image | Playback Image | LPAR Agent Img] PLO failed in Agent
- ITKO0105 - iTKO Agent started while another agent is running
- ITKO0106 - iTKO Agent connected to LPAR Agent
- ITKO0107 - Agent Config LPAR Agent IP: nnn.nnn.nnn.nnn Port 99999
- ITKO0108 - iTKO Agent GWA error- Address is xxxxxxxx Length is 9999
- ITKO0109 - Socket func [INITAPI | SELECTEX | SEND | RECV | CONNECT | CLOSE | PTON | TERMAPI] failed. RETCODE=xXXXXXXXX ERRNO=999999
- ITKO0110 - Agent CICS cmd failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0111 - iTKO Agent disconnected from LPAR Agent
- ITKO0112 - iTKO Agent error sending recorded image to LPAR Agent
- ITKO0113 - iTKO Agent error sending playback image to LPAR Agent
- ITKO0114 - iTKO Agent error sending response to LPAR Agent
- ITKO0115 - iTKO Agent error receiving request from LPAR Agent
- ITKO0116 - iTKO Agent error - invalid LPAR Agent IP address
- ITKO0117 - iTKO Agent released nnnnnn recorded images
- ITKO0118 - iTKO Agent canceled nnnnnn playback images
- ITKO0119 - iTKO Agent canceled nnnnnn pending playback images
- ITKO0120 - iTKO Agent released playback image with token xxxxxxxxxxxxxxxx
- ITKO0121 - iTKO Agent error reading configuration
- ITKO0122 - Agent TCP/IP API Timeout
- ITKO0123 - Agent waiting for TCP/IP API
- ITKO0124 - ITKO Agent response timeout for Token xxxxxxxxxxxxxxxx
- ITKO0125 - Call to ITKOUEX1/ITKOUEX2/ITKOUEX3 failed
- ITKO0126 - Handshake error encountered. Shutting down Agent.
- ITKO0127 - ITKO Agent Test request failed -- Token: xxxxxxxxxxxxxxxx Reason xxx
- ITKO0128 - ITKO Agent Test Req timeout for Token xxxxxxxxxxxxxxxx
- ITKO0129 - ITKO Agent Test Req timeout fail for Token xxxxxxxxxxxxxxxx
- ITKO0130 - ITKO Agent cancelled 999999 pending Lisa Test requests
- ITKO0131 - Incompatible LPAR agent vvvv. Shutting down Agent.
- ITKO0132 - Config line xxxxx error-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
- ITKO200 - ITKOXEIN XPI INQ_APPLICATION_DATA Failed
- ITKO201 - ITKOXEIN/ITKOXEIO XPI Getmain Dynamic Stg Failed
- ITKO202 - ITKOXEIN/ITKOXEIO XPI Freemain Dynamic Stg Failed
- ITKO203 - ITKOXEIN/ITKOXEIO XPI Getmain Image Stg Failed
- ITKO204 - ITKOXEIN/ITKOXEIO XPI Freemain Image Stg Failed
- ITKO205 - ITKOXEIN Task table full - discarding image
- ITKO206 - ITKOXEIN XPI WAIT Failed
- ITKO207 - ITKOXEIN XPI Freemain plbk req/rsp failed
- ITKO208 - ITKOXEIO Add to recording queue failed
- ITKO209 - ITKOXEIN/ITKOXEIO XPI Freemain SET() Stg Failed

- ITKO210 - ITKOXEIN XPI Getmain SET() Stg Failed
- ITKO0300 - Test CICS cmd failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0301 - DevTest CICS exit not running.
- ITKO0302 - DevTest CICS GWA not available.
- ITKO0303 - DevTest CICS Agent not running/connect.
- ITKO0304 - ITKOLDEL failed for test request queue.
- ITKO0305 - No TEST request found.
- ITKO0306 - ITKOTST terminated due to errors.

ITKO0001 - iTKO exit init failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999

Message Type:

Fatal Error

Description:

A CICS command failed and ITKOINST was aborted. The hexidecimal EIBFN value identifies the CICS command that failed. The command that failed is:

EIBFN CICS Command

x0C02 GETMAIN

x1008 START

x2202 ENABLE PROGRAM

x2206 EXTRACT EXIT

x6C02 WRITE OPERATOR

x8802 INQUIRE EXITPROGRAM

Action:

Using the EIBRCDE, EIBRESP, and EIBRESP2 codes, determine the cause of the failure and correct it. Some possible causes of failure:

- The user running the TKOI transaction does not have the authority to execute CICS system programming interface (SPI) commands (such as EXTRACT EXIT).
- A START command for the agent transaction, TKOA, failed because TKOA was not defined or was defined incorrectly.

ITKO0002 - iTKO exit [XPCREQ | XPCREQC] at xxxxxxxx GWA at xxxxxxxx Len 9999 is enabled and started

Message Type:

Informational

Description:

The indicated exit (XPCREQ or XPCREQC) was installed at the indicated address with a Global Work Area at the indicated address of the indicated length. The exit was enabled and started.

Typically, two of these messages appear: one for XPCREQ and one for XPCREQC. Both have the same GWA address and length.

Action:

None

ITKO0003 - iTKO exit [XPCREQ | XPCREQC] was enabled but not started

Message Type:

Fatal Error

Description:

The indicated exit (XPCREQ or XPCREQC) was enabled; however, it was not possible to start the exit. The ITKO recording and playback capability do not function properly.

An ITKO0001 message typically accompanies this message.

Action:

Diagnose the accompanying ITKO0001 message and correct the issue.

Remove the exits with TKOR and reinstall with TKOI.

ITKO0004 - iTKO exit GWA error - Address is xxxxxxxx Length is 9999

Message Type:

Fatal Error

Description:

The ITKO exits were allocated a Global Work Area that is too small.

An ITKO0001 message can accompany this message.

Action:

Report the error to CA support.

ITKO0005 - iTKO exit remove failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999

Message Type:

Fatal Error

Description:

A CICS command failed and ITKOREMV was aborted. The hexidecimal EIBFN value identifies the CICS command that failed. The command that failed is:

EIBFN CICS Command

x0C02 FREEMAIN

x1004 DELAY

x2204 DISABLE PROGRAM

x2206 EXTRACT EXIT

x6C02 WRITE OPERATOR

x8802 INQUIRE EXITPROGRAM

Action:

Using the EIBRCDE, EIBRESP, and EIBRESP2 codes, determine the cause of the failure and correct it. This failure can occur if the user running the TKOR transaction lacks the authority to execute CICS system programming interface (SPI) commands (such as EXTRACT EXIT).

ITKO0006 - iTKO exit [XPCREQ | XPCREQC] disabled

Message Type:

Informational

Description:

The indicated exit (XPCREQ or XPCREQC) was disabled.

Action:

None

ITKO0007 - GWA missing or too small for [ITKPCRQ | ITKPCRC] Exit

Message Type:

Fatal Error

Description:

The exit Global Work Area for the exits does not exist or is too small.

Action:

Report the error to CA support.

ITKO0008 - Program program-name Error EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999

Message Type:

Fatal Error

Description:

A CICS command failed in the indicated exit program. The hexidecimal EIBFN value identifies the CICS command that failed. The command that failed is:

EIBFN CICS Command

0202 ADDRESS

0208 ASSIGN INVOKINGPROG

0C02 GETMAIN

0C02 FREEMAIN

3414 GET CONTAINER

3416 PUT CONTAINER

6C02 WRITE OPERATOR

9626 STARTBROWSE CONTAINER

9628 GETNEXT CONTAINER

Action:

Using the EIBRCDE, EIBRESP, and EIBRESP2 codes, determine the cause of the failure and correct it. One possible cause of the failure is that a GETMAIN failed due to a CICS short on storage condition.

ITKO0009 - Recording Abandoned of calling-program LINK to called-program Tran xxxx Task 9999999 Term xxxx

Message Type:

Warning

Description:

When recording a CICS LINK from calling program to called program for the indicated transaction, task, and terminal, no resource was available to handle the recorded image. An accompanying ITKO0008 message can identify the resource (such as a GETMAIN failure for the recorded image). The DevTest LPAR Agent could have disconnected from the DevTest CICS Agent.

Action:

Investigate the condition causing the recorded image to be dropped.

ITKO0010 - [Recorded Image | Playback Image | LPAR Agent list] PLO failed in [ITKPCRQ | ITKPCRC] Exit

Message Type:

Warning

Description:

The PLO (z/OS Perform Locked Operation) command failed.

PLO is used to serialize the manipulation of three queues/lists used to communicate between the exits and the agent. The Recorded Image queue contains recorded images to for the agent to process. The Playback Image queue contains playback requests for the agent to process. The LPAR Agent List contains the playback responses that the agent receives to be passed to the exits.

For a Recorded Image queue entry, the exit discards the recorded image.

For a Playback Image queue entry, the image is discarded and the transaction continues without playback (virtualization).

For an LPAR Agent List entry (playback response), the image is orphaned on the LPAR Agent List by the exit.

Action:

Report the error to CA support.

ITKO0011 - BEFORE/AFTER image creation failed with reason code xxx

Message Type:

Error

Description:

In exit XPCREQ (module ITKPCRQ) or exit XPCREQC (module ITKOPCRC), the creation of a before or after image failed with the indicated reason code.

The following reason codes are possible:

8 - No image was passed to the create routine.

12- Neither a before or after image in the create routine.

16- Exceeded PDU length inserting INPUTMSG in the create routine.

20- Exceeded PDU length inserting COMMAREA in the create routine.

24- Exceeded PDU length inserting Container in the create routine.

28- Exceeded PDU length inserting Origin Data in the create routine.

32- MVCL logic error in the create routine.

36- No TKOCLNK in the image that was passed to the create routine.

40- CPDUVARN is zero in the create routine.

44- Exceeded PDU length inserting EIB in the create routine.

Action:

Report the error to CA Support.

ITKO0012 - Image update failed with reason code 999

Message Type:

Error

Description:

In the exit XPCREQ (module ITKPCRQ) or exit XPCREQC (module ITKOPCRC), the update an after image failed with the indicated reason code.

The following reason codes are possible:

4- A CICS command error occurred.

5- No image was passed to the update routine.

6- No after EIB in the image that was passed to the update routine.

7- No TKOCLNK in the image that was passed to the update routine.

8- CVARTYP not CVARLINK in the image that was passed to the update routine.

9- CLNKEYE not 'CLNK' in the image that was passed to the update routine.

10-CVARLEN is zero in the image that was passed to the update routine.

11-CTNREYE not 'CTNR' in the image that was passed to the update routine.

Action:

Report the error to CA support.

ITKO0013 - PDU Parsing Error in ITKPCRC Exit

Message Type:

Error

Description:

In the exit XPCREQC (module ITKPCRC), the update of a COMMAREA that is expanded in User Exit 3 failed.

Action:

Report the error to CA support.

ITKO0014 - ERROR - Corrupt token in ITKPCRC Exit

Message Type:

Error

Description:

In the CICS LINK after exit XPCREQC (module ITKPCRC), a token (UEPPCTOK) was passed that does not address the proper DevTest CICS Agent control block. The cause is probably a conflicting XPCREQ or XPCREQC exit. This message is only written once.

Action:

Investigate the XPCREQ and XPCREQC exits installed in the CICS region.

ITKO0101 - AGENT waiting for exits

Message Type:

Informational

Description:

The DevTest CICS Agent was started before the exits were enabled. The agent waits for the exits to be enabled before continuing.

Action:

None.

ITKO0102 - AGENT initialization complete

Message Type:

Informational

Description:

The DevTest CICS Agent has completed initialization and starts connecting to the DevTest LPAR Agent.

Action:

None.

ITKO0103 - AGENT shutting down

Message Type:

Informational

Description:

The DevTest CICS Agent is shutting down.

Action:

None.

ITKO0104 - iTKO [Recorded Image | Playback Image | LPAR Agent Img] PLO failed in Agent

Message Type:

Warning

Description:

The PLO (z/OS Perform Locked Operation) command failed.

PLO is used to serialize the manipulation of three queues/lists used to communicate between the exits and the agent. The Recorded Image queue contains recorded images for the agent to process. The Playback Image queue contains playback requests for the agent to process. The LPAR Agent List contains the playback responses that the agent receives to be passed to the exits.

For Recorded and Playback Image queue entries, the image is skipped and the PLO is retried on the next pass of the Recorded/Playback Image queues.

For an LPAR Agent List entry (playback response), the image is discarded and the user transaction issuing the CICS LINK receives an INVREQ response (EIBRCODE=x'E00000000000', EIBRESP=16 and EIBRESP2=500).

Action:

Report the error to CA support.

ITKO0105 - iTKO Agent started while another agent is running

Message Type:

Warning

Description:

A DevTest CICS Agent (transaction TKOA) was started while another instance of the agent is running. Only one agent can be running in a CICS region, so the second agent stops.

Action:

None.

ITKO0106 - iTKO Agent connected to LPAR Agent

Message Type:

Informational

Description:

The DevTest CICS Agent has established a TCP/IP connection to the DevTest LPAR Agent.

Action:

None.

ITKO0107 - Agent Config LPAR Agent IP: nnn.nnn.nnn.nnn Port 99999

Message Type:

Informational

Description:

The DevTest CICS Agent read the configuration (TD Queue TKOC) and is using the indicated IP address and port number to connect to the DevTest LPAR Agent.

Action:

None

ITKO0108 - iTKO Agent GWA error- Address is xxxxxxxx Length is 9999

Message Type:

Fatal Error

Description:

The exit Global Work Area (GWA) does not exist or has an incorrect length. The DevTest CICS Agent stops.

Action:

Report the error to CA support.

ITKO0109 - Socket func [INITAPI | SELECTEX | SEND | RECV | CONNECT | CLOSE | PTON | TERMAPI] failed. RETCODE=xXXXXXXXX ERRNO=999999

Message Type:

Warning

Description:

An error occurred on the TCP/IP connection to the LPAR Agent.

The message **Socket func RECV failed. RETCODE=x00000000 ERRNO=000000** typically indicates a disconnect from the LPAR Agent. An ITKO0111 message accompanies this message. An error in the SEND function can also indicate an LPAR Agent disconnect.

The ERRNO values are documented in the IBM publication z/OS Communications Server IP CICS Sockets Guide.

Action:

Determine if the error is due to a DevTest LPAR Agent disconnect or other environmental issue and correct the issue. Otherwise, report the error to Support.

ITKO0110 - Agent CICS cmd failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999

Message Type:

Fatal Error

Description:

A CICS command failed in the DevTest CICS Agent. The hexidecimal EIBFN value identifies the CICS command that failed. The command that failed is:

EIBFN CICS Command

0202 ADDRESS

0208 ASSIGN

0C02 GETMAIN/ FREEMAIN

0E02 LINK

1204 ENQ

1206 DEQ

2206 EXTRACT EXIT

4E02 INQUIRE PROGRAM

5402 INQUIRE SYSTEM

6C02 WRITE OPERATOR

8802 INQUIRE EXITPROGRAM

Action:

Using the EIBRCDE, EIBRESP and EIBRESP2 codes, determine the cause of the failure and correct it if possible. One possible cause of the failure is that the user ID running the TKOA transaction does not have the authority to execute CICS system programming interface (SPI) commands (such as EXTRACT EXIT).

ITKO0111 - iTKO Agent disconnected from LPAR Agent

Message Type:

Informational

Description:

The DevTest CICS Agent has disconnected from the DevTest LPAR Agent.

Action:

None.

ITKO0112 - iTKO Agent error sending recorded image to LPAR Agent

Message Type:

Error

Description:

The DevTest CICS Agent was in the process of sending a recorded image and the TCP/IP connection with the DevTest LPAR Agent was broken. The recorded image is dropped.

Action:

Investigate the cause of the DevTest LPAR Agent disconnect.

ITKO0113 - iTKO Agent error sending playback image to LPAR Agent

Message Type:

Error

Description:

The DevTest CICS Agent was in the process of sending a playback request image and the TCP/IP connection with the LPAR Agent was broken. The recorded image is dropped, and the CICS LINK in progress completes with an INVREQ response (EIBRCODE=x'E00000000000', EIBRESP=16 and EIBRESP2=500).

Action:

Investigate the cause of the DevTest LPAR Agent disconnect.

ITKO0114 - iTKO Agent error sending response to LPAR Agent

Message Type:

Error

Description:

The DevTest CICS Agent was in the process of sending a response to the DevTest LPAR Agent and the TCP/IP connection was broken.

Action:

Investigate the cause of the CICS LPAR Agent disconnect.

ITKO0115 - iTKO Agent error receiving request from LPAR Agent

Message Type:

Error

Description:

The DevTest CICS Agent was in the process of receiving a request from the DevTest LPAR Agent and the TCP/IP connection was broken.

Action:

Investigate the cause of the DevTest LPAR Agent disconnect.

ITKO0116 - iTKO Agent error - invalid LPAR Agent IP address

Message Type:

Fatal Error

Description:

The DevTest LPAR Agent IP address in the DevTest CICS Agent configuration file (TD Queue TKOC) is not a valid dotted-decimal notation IPv4 address.

Action:

Correct the configured IP address.

ITKO0117 - iTKO Agent released nnnnnn recorded images

Message Type:

Warning

Description:

The DevTest LPAR Agent disconnected from the DevTest CICS Agent while recorded images were queued. The DevTest CICS Agent removes the pending recorded images from the queue and releases them.

Action:

Investigate the cause of the DevTest LPAR Agent disconnect.

ITKO0118 - iTKO Agent canceled nnnnnn playback images

Message Type:

Warning

Description:

The DevTest LPAR Agent disconnected from the DevTest CICS Agent while playback request images were queued. The DevTest CICS Agent removes the pending playback request images from the queue and releases them. Each playback request image represents a CICS LINK to be virtualized. These CICS LINK commands complete with an INVREQ response (EIBRCODE=x' E00000000000', EIBRESP=16 and EIBRESP2=500).

Action:

Investigate the cause of the DevTest LPAR Agent disconnect.

ITKO0119 - iTKO Agent canceled nnnnnn pending playback images

Message Type:

Warning

Description:

The DevTest LPAR Agent disconnected from the DevTest CICS Agent while pending playback response images were expected. The DevTest CICS Agent removes the pending playback response images from the queue and releases them. Each pending playback response image represents a CICS LINK to be virtualized. These CICS LINK commands complete with an INVREQ response (EIBRCODE=x'E00000000000', EIBRESP=16 and EIBRESP2=500).

Action:

Investigate the cause of the DevTest LPAR Agent disconnect.

ITKO0120 - iTKO Agent released playback image with token xxxxxxxxxxxxxxxx

Message Type:

Warning

Description:

The DevTest CICS Agent received a playback response from the DevTest LPAR Agent that does not match a pending playback request. The CICS Transaction waiting on the CICS LINK virtualization could not be identified. The playback image is released (discarded).

The following reason codes are possible. Codes 5 and 6 indicate that a playback response was received from the Virtual Service Environment after the CICS Agent 30-second timeout. An ITKO0124 message typically accompanies this message.

1- PDU too short.

2- TKOCPDU Format Error.

3- TKOCPDU Token prefix is not ITKO.

4- TKOCPDU Token suffix is x'00000000'.

5- LPAR Agent Queue is empty - request may have timed out.

6- Not found on LPAR Agent Queue - request may have timed out.

7- CVAR Format error.

8- No CICS Command header.

9- No after image EIB.

10-Request CPDU error.

11-Matching image on LPAR Agent Queue already posted.

12-CMALEYE not 'CMAL'.

13-CMFREYE not 'CMFR'.

14-CMSDEYE not 'CMSD'.

15-CMRCEYE not 'CMRC'.

16-CMCVEYE not 'CMCV'.

17-CMEXEYE not 'CMEX'.

18-The Receive command has no received data block.

19-The Converse command has no received data block.

20-The Receive command data block too short.

21 Converse command data block too short.

Action:

For reason codes 5 and 6, investigate the timeout. For all other reason codes, report the error to Support.

ITKO0121 - iTKO Agent error reading configuration

Message Type:

Fatal Error

Description:

An error occurred reading the configuration TD Queue TKOC.

Action:

Check the definition and contents of TD Queue TKOC.

ITKO0122 - Agent TCP/IP API Timeout

Message Type:

Fatal Error

Description:

The DevTest CICS Agent waits for the CICS TCP/IP interface expired and the DevTest CICS Agent stopped.

Action:

Ensure that the CICS TCP/IP interface is started and restart the agent.

ITKO0123 - Agent waiting for TCP/IP API

Message Type:

Warning

Description:

The DevTest CICS Agent is waiting for the CICS TCP/IP interface to initialize. The DevTest CICS Agent retries 30 times in 10-second intervals (5 minutes). If the CICS TCP/IP interface does not initialize before the wait times out, the DevTest CICS Agent stops.

Action:

No action is required.

ITKO0124 - ITKO Agent response timeout for Token xxxxxxxxxxxxxxxx

Message Type:

Warning

Description:

A playback image was sent to the Virtual Service Environment and a response was not received before the 30-second timeout. The playback request is purged and the CICS command terminates with an INVREQ response.

Action:

Check the VSE for errors.

ITKO0125 - Call to ITKOUEX1/ITKOUEX2/ITKOUEX3 failed

Message Type:

Warning

Description:

The CICS LINK to ITKOUEX1, ITKOUEX2, or ITKOUEX3 failed. See the accompanying ITKO0110 message.

Action:

Correct the issue that the ITKO0110 message identified.

ITKO0126 - Handshake error encountered. Shutting down Agent.

Message Type:

Error

Description:

The handshake between the DevTest CICS Agent and the DevTest LPAR agent failed. This failure is typically due to an incorrectly configured port number for the DevTest LPAR Agent in the DevTest CICS Agent configuration. The error causes the DevTest CICS Agent to connect to a server that is not a DevTest LPAR Agent.

Action:

Correct the DevTest CICS Agent configuration.

ITKO0127 - ITKO Agent Test request failed -- Token: xxxxxxxxxxxxxxxxx Reason xxx

Message Type:

Warning

Description:

A DevTest CICS Application Test request failed.

Action:

Report the error to CA support.

ITKO0128 - ITKO Agent Test Req timeout for Token xxxxxxxxxxxxxxxxx

Message Type:

Warning

Description:

A DevTest CICS Application Test request was not processed before the timeout expired. This might be due to system load or a failure in the Application Test components of the DevTest CICS Agent.

Action:

Investigate any CICS issues at the time of this message. If necessary, report the error to CA support.

ITKO0129 - ITKO Agent Test Req timeout fail for Token xxxxxxxxxxxxxxxx

Message Type:

Warning

Description:

A DevTest CICS Application Test request was not processed before the timeout expired, and the timeout mechanism failed.

Action:

Report the error to CA support.

ITKO0130 - ITKO Agent cancelled 999999 pending Lisa Test requests

Message Type:

Warning

Description:

During DevTest CICS Agent shutdown, the indicated number of pending Application Test requests were cancelled.

Action:

None.

ITKO0131 - Incompatible LPAR agent vvvv. Shutting down Agent.

Message Type:

Fatal Error

Description:

A DevTest LPAR Agent of the indicated incompatible version was connected. The connection is closed.

Action:

Investigate the version of the DevTest LPAR Agent.

ITKO0132 - Config line xxxxx error-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Message Type:

Fatal Error

Description:

The DevTest CICS Agent configuration has the indicated error in the indicated line.

Action:

Correct the DevTest CICS Agent configuration.

ITKO200 - ITKOXEIN XPI INQ_APPLICATION_DATA Failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) received an error in the XPI INQ_APPLICATION_DATA command.

Action:

Report the error to CA support.

ITKO201 - ITKOXEIN/ITKOXEIO XPI Getmain Dynamic Stg Failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) or XEIOUT exit (module ITKOXEIO) received an error in the XPI GETMAIN command while allocating dynamic storage.

Action:

Investigate the CICS short on storage condition.

ITKO202 - ITKOXEIN/ITKOXEIO XPI Freemain Dynamic Stg Failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) or XEIOUT exit (module ITKOXEIO) received an error in the XPI FREEMAIN command while releasing dynamic storage.

Action:

Report the error to CA support.

ITKO203 - ITKOXEIN/ITKOXEIO XPI Getmain Image Stg Failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) or XEIOUT exit (module ITKOXEIO) received an error in the XPI GETMAIN command while allocating storage for a recorded image.

Action:

Investigate the CICS short on storage condition.

ITKO204 - ITKOXEIN/ITKOXEIO XPI Freemain Image Stg Failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) or XEIOUT exit (module ITKOXEIO) received an error in the XPI FREEMAIN command while releasing image storage.

Action:

Report the error to CA support.

ITKO205 - ITKOXEIN Task table full - discarding image

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) tried to allocate an entry in the task table and the table was full.

Action:

Report the error to CA support.

ITKO206 - ITKOXEIN XPI WAIT Failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) received an error in the XPI WAIT command while waiting for a playback response.

Action:

Report the error to CA support.

ITKO207 - ITKOXEIN XPI Freemain plbk req/rsp failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) received an error in the XPI FREEMAIN command while releasing a playback request or response.

Action:

Investigate the CICS short on storage condition.

ITKO208 - ITKOXEIO Add to recording queue failed

Message Type:

Error

Description:

The XEIOUT exit (module ITKOXEIO) tried to add a recorded image to the recording queue and the attempt failed.

Action:

Report the error to CA support.

ITKO209 - ITKOXEIN/ITKOXEIO XPI Freemain SET() Stg Failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) or XEIOUT exit (module ITKOXEIO) received an error in the XPI FREEMAIN command while releasing storage allocated for the SET parameter in a RECEIVE or CONVERSE command.

Action:

Report the error to CA support.

ITKO210 - ITKOXEIN XPI Getmain SET() Stg Failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) received an error in the XPI GETMAIN command while allocating storage for the SET parameter in a RECEIVE or CONVERSE command.

Action:

Report the error to CA support.

ITKO0300 - Test CICS cmd failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999

Message Type:

Fatal Error

Description:

A CICS command failed in the DevTest CICS Application Test driver program. The hexadecimal EIBFN value identifies the CICS command that failed.

Action:

Using the EIBRCDE, EIBRESP and EIBRESP2 codes, determine the cause of the failure and correct it if possible. If necessary, report the error to CA support.

ITKO0301 - DevTest CICS exit not running.

Message Type:

Informational

Description:

The DevTest CICS Application Test transaction was run when the DevTest CICS exits were not enabled. This is typically a case where the transaction was started by a mechanism other than the DevTest CICS Agent. This message could also indicate a timing issue where the DevTest CICS Agent was cancelled while an Application Test request was in-flight.

Action:

Ensure that the DevTest CICS Agent is running.

ITKO0302 - DevTest CICS GWA not available.

Message Type:

Informational

Description:

The DevTest CICS Application Test transaction was run when the DevTest CICS exit Global Work Area was not allocated. This is typically a case where the transaction was started by a mechanism other than the DevTest CICS Agent. This message could also indicate a timing issue where the DevTest CICS Agent was cancelled while an Application Test request was in-flight.

Action:

Ensure that the DevTest CICS Agent is running.

ITKO0303 - DevTest CICS Agent not running/connect.

Message Type:

Informational

Description:

The DevTest CICS Application Test transaction was run when the DevTest CICS Agent was not running or was not connected to the DevTest LPAR agent. This message could indicate a timing issue where the DevTest LPAR Agent was shut down while an Application Test request was in-flight.

Action:

Ensure that the DevTest CICS Agent is running and connected to the DevTest LPAR Agent.

ITKO0304 - ITKOLDEL failed for test request queue.

Message Type:

Fatal Error

Description:

The DevTest CICS Application Test transaction attempted to remove an Application Test request from the queue, and the operation failed.

Action:

Report the error to CA support.

ITKO0305 - No TEST request found.

Message Type:

Fatal Error

Description:

The DevTest CICS Application Test transaction did not find an Application Test request and terminated. This is typically a case where the transaction was started by a mechanism other than the DevTest CICS Agent.

Action:

None.

ITKO0306 - ITKOTST terminated due to errors.

Message Type:

Informational

Description:

The DevTest CICS Application Test transaction ended abruptly. The Application Test response is incomplete. This message accompanies a DevTest error message with more information regarding the issue (ITKO0128),

Action:

None.

DevTest LPAR Agent

The DevTest LPAR Agent is a z/OS resident agent that runs as a started task. The agent can be installed on any LPAR in the z/OS Sysplex. The agent provides a single point of contact for the DevTest registry (and therefore DevTest clients) to communicate with other DevTest z/OS agents. The agent enables DevTest to know which z/OS CICS agents are up and running and some basic configuration information about those agents. The LPAR Agent also provides a means to group DevTest z/OS CICS agents if multiple DevTest LPAR agents are defined to the z/OS Sysplex.

The agent can operate in client mode, where it makes the connection to the DevTest registry.

The agent can operate in server mode, where it accepts connections from DevTest registries.

The following graphic shows the components of the DevTest LPAR Agent and DevTest CICS Agent.

Install the LPAR Agent

To get the z/OS agents (CICS and LPAR Agents), contact CA Customer Support.

LPAR Agent Storage Requirements

The DevTest LPAR agent runs as a standalone address space with a minimum REGION of 3 MB.

A larger REGION may be required if the recorded images are excessively large or the arrival rate is high.

LPAR Agent Installation

The LPAR Agent package consists of three files:

- iTKOLPARAgentDoc.pdf
- iTKOLPARAgentLoad
- iTKOLPARAgentCntl

Follow these steps:

1. FTP the files to the target system.
   The files iTKOLPARAgentLoad and iTKOLPARAgentCntl are FTP'd to the target z/OS system in binary mode with the z/OS attributes indicated in the example that follows.

   The following example of command-line FTP shows the proper transfer to z/OS of data sets ITKO.LPARAGNT.CNTL.UNLOAD and ITKO.LPARAGNT.LOAD.UNLOAD. You can change the z/OS file names for your data set naming conventions.

   ```
   ftp> quote SITE LRECL=80 RECFM=FB BLKSIZE=3120 TRACKS PRI=40 SEC=5
   200 SITE command was accepted
   ftp> bin
   200 Representation type is Image
   ftp> put iTKOLPARAgentLoad 'ITKO.LPARAGNT.LOAD.UNLOAD'
   200 Port request OK.
   125 Storing data set ITKO.LPARAGNT.LOAD.UNLOAD
   250 Transfer completed successfully.
   ftp> quote SITE LRECL=80 RECFM=FB BLKSIZE=3120 TRACKS PRI=10 SEC=1
   200 SITE command was accepted
   ftp> put iTKOLPARAgentCntl 'ITKO.LPARAGNT.CNTL.UNLOAD'
   200 Port request OK.
   125 Storing data set ITKO.LPARAGNT.CNTL.UNLOAD
   250 Transfer completed successfully.
   ```

2. Restore the Extended Partitioned Datasets (PDSEs) and Partitioned Dataset (PDS).
   The LPAR Agent load library PDSE and control (JCL) library PDS are restored with the TSO RECEIVE command, which typically runs in a batch job.

   The following sample JCL restores the Extended Partitioned Data Sets (PDSE) and Partitioned Data Set (PDS) from the FTP'd data sets ITKO.LPARAGNT.LOAD and ITKO. LPARAGNT.CNTL on volume VOL001. The data set names can be changed for your site data set naming conventions.

   ```
   //job
   //*
   //* Unload the LOAD Library with TSO RECEIVE
   //*
   //UNLOAD EXEC PGM=IKJEFT01,DYNAMNBR=10
   //SYSTSPRT DD SYSOUT=*
   //SYSPRINT DD SYSOUT=*
   //SYSOUT DD SYSOUT=*
   //SYSTSIN DD *
   RECEIVE INDSNAME('ITKO.LPARAGNT.LOAD.UNLOAD')
      DATASET('ITKO.LPARAGNT.LOAD') VOLUME(VOL001)
   RECEIVE INDSNAME('ITKO.LPARAGNT.CNTL.UNLOAD')
      DATASET('ITKO.LPARAGNT.CNTL') VOLUME(VOL001)
   /*
   ```

3. Set up the LPAR Agent to run as a started task.
   The LPARAGNT member in ITKO.LPARAGNT.CNTL can be used as a model for the proc that runs the LPAR Agent as a started task. After the new proc member is created, it must be copied to a PROCLIB from where tasks can be started. Four JCL statements make up the proc:

   - **PROC:**
     Statement that specifies the name of the started task.

   - **EXEC:**
     Statement that specifies the program name and parameters.

   - **STEPLIB:**
     The PDSE load library that contains the LPAR Agent load module.

   - **config:**
     The DD statement that specifies a PDS member where the LPAR Agent configuration is defined.

   The PARM specified in the EXEC statement defines environment variables. The PARM also specifies the DD name of the statement that points to the LPAR Agent configuration data member. A forward slash ('/') delimits the PARM data. Values before the slash specify run-time options and values after the slash specify the command-line arguments to the application. The environment variable is a run-time option and the LPAR Agent configuration data is a command-line argument.

   Currently the only environment variable that the LPAR Agent recognizes is a time zone. The variable is optional and if not specified, defaults to UTC. This variable is used to calculate date and time values in the messages that the LPAR Agent produces. The valid values are the standard UNIX TZ strings; for example, EST5EDT, CST6CDT, MST7MDT, PST8PDT.

   The following sample proc is in the LPARAGNT member of ITKO.LPARAGNT.CNTL:

```
//LPARAGNT    PROC
//AGENT       EXEC PGM=LPARAGNT,PARM='ENVAR("TZ=CST6CDT")/"//DD:CONFIG"'
//STEPLIB     DD DISP=SHR,DSN=ITKO.LPARAGNT.LOAD
//CONFIG      DD DISP=SHR,DSN=ITKO.LPARAGNT.CNTL(CONFIG)
//SYSUDUMP    DD SYSOUT=*
```

4. Create the LPAR Agent configuration member.
   The configuration data that the DevTest LPAR Agent requires is specified as keyword /value pairs in plain text. Specify the PDS member in a DD statement in the proc that is used to start the LPAR Agent. Specify the DD name in the PARM value in the EXEC statement. The file is parsed as follows:

   - A hash character ('#') is used as a comment indicator.

   - Any blank lines or lines starting with a hash character are ignored.

   - An equal sign separates the keywords and values.

   - Any white space is trimmed from the beginning and ending of both the keywords and values.

   - Keywords are not case-sensitive, while most values have the case preserved.

   - A list exists of valid keywords that the LPAR Agent understands.

The keywords that can be used to configure the LPAR Agent are:

- **Name**
  Specifies the name of the LPAR Agent. This name is used for things like logging.
  Default: DevTest LPAR Agent

- **Mode**
  Specifies whether the LPAR Agent is in **server** (keyword: Server) or **client** (keyword: Client) mode. In Server mode, the agent accepts connections from the DevTest registry on the port *PortClient*. This value must be the opposite of the **lisa.mainframe.bridge.mode** property in **lisa.properties**.

- **PortClient**
  When running in server mode, defines the value of the well-known port to which clients connect. This port number must match the value that the **lisa.mainframe.bridge.server.port** property in **lisa.properties** specifies.
  Default: 3997

- **Server**
  When running in client mode, specifies the IP address of the server to which to connect. This IP address typically matches the IP address of the DevTest Server host running the registry.

- **ServerPort**
  When running in client mode, defines the port number that the DevTest Server registry uses. This port must match the value that the **lisa.mainframe.bridge.port** property in **lisa.properties** specifies.

- **PortAgent**
  Defines the value of the well-known port that CICS agents connect to. This port number must match the value that all CICS agents connecting to this LPAR Agent configure.
  Default: 3998

- **InitialTrace**
  Specifies whether packet tracing is initially turned on.
  Values:

  - **True:** Tracing is turned on for all connections

  - **False:** Tracing is turned off for all connections

  Limits: These values are not case-sensitive.
  Default: False

- **BufferSize**
  Defines the initial buffer size allocated. This value is typically set to twice the size of the largest data payload expected. The buffer size can be reallocated up to the configured maximum when packets of a larger size than this value are received.
  Default: 65536

- **MaxBufferSize**
  Defines the maximum packet size to be accepted. A packet larger than this value is discarded and a message is logged. A value of 0 disables the size verification, and packets of any size from a client are accepted.
  Default: 0

The following sample configuration file for running in **server** mode is in the CONFIGSV member of ITKO.LPARAGNT.CNTL:

```
#
# A sample server mode configuration file
#
Name              = DevTest LPAR Agent

#
# Mode can be "Client" or "Server"
#
Mode              = Server
PortClient        = 3997
#

#
# The port used as a server for the agent connections
#
PortAgent         = 3998

#
BufferSize        = 65536
MaxBufferSize     = 65536
#
```

The following sample configuration file for running in **client** mode is found in the CONFIGCL member of ITKO.LPARAGNT.CNTL:

```
#
# A sample client mode configuration file
#
Name               = DevTest LPAR Agent

#
# Mode can be "Client" or "Server"
#
Mode               = Client
Server             = 10.0.0.1
ServerPort         = 61617

#
# The port used as a server for the agent connections
#
PortAgent          = 3998

#
BufferSize         = 65536
MaxBufferSize      = 65536
#
```

LPAR Agent Operation

Starting the DevTest LPAR Agent

In a typical operation, the LPAR Agent is set up as a started task and is started when the LPAR IPLs. The standard MVS START operator command ('S') can be used to start the LPAR Agent explicitly.

Stopping the DevTest LPAR Agent

The LPAR Agent responds to the standard MVS STOP operator command ('P'). You can also stop the agent by issuing the "SHUTDOWN" message through the management socket connection or through the MVS MODIFY operator command ('F'). The LPAR Agent terminates identically for each technique.

Monitoring and Managing the LPAR Agent

The LPAR Agent can be monitored and managed to a limited extent through standard MVS operator commands.

To use standard MVS operator commands, issue the requests to the LPAR Agent using the MODIFY command ('F'). The response is written to the console log. The supported LPAR Agent commands are:

- **?**
  Displays a list of available commands.
  Usage: F LPARAGNT,?

- **L**
  Lists the current connections and the associated connection IDs.
  Usage: F LPARAGNT,L

- **T**
  Toggles packet tracing.
  Usage: F LPARAGNT,T <id> <state>
  <id>
  Defines the connection ID found in the "L" command.

- **<state>**
  (Optional) Specifies the state of the packet trace for a connection.
  Issuing the T command without the state returns the tracing state for that connection.
  Values: "ON" and "OFF"

- **SHUTDOWN**
  Shuts down the LPAR Agent.
  Usage: F LPARAGNT,SHUTDOWN

The following output is an example of the **L** command:

```
F LPARAGNT,L
+ITKO9003 - Command accepted
+ID    Type IP              Port         In       Out  Connected
+3     AT   192.168.0.100   6891          1         0  12/15/11 09:38:44
+4     A    192.168.0.100   6892          1         0  12/15/11 09:38:45
```

The Type is **A** for "agent connection" or **C** for "client connection". If a **T** follows the **A** or **C**, it means that packet tracing is enabled for that connection. The IP address and Port number identify the source of the connection. In and Out show the number of packets that were received from and sent to the peer, respectively. The Connected date and time shows when the connection was created. The Connected date and time use the time zone value that is specified on the PARM of the proc EXEC statement.

The following output is an example of the T command:

```
F LPARAGNT,T 3
+ITKO9003 - Command accepted
+Tracing for ID 3 is ON
F LPARAGNT,T 3 OFF
+ITKO9003 - Command accepted
+Tracing for ID 3 is now OFF
```

LPAR Agent Messages

Messages are written to the operator console and to data sets in the JES log for the started task.

The following messages can be written to the operator console. These messages are informational only.

- ITKO9001 - LPAR Agent process initialized and waiting for connections
- ITKO9002 - The Operator command was too large
- ITKO9003 - The command was accepted
- ITKO9004 - The Start command was accepted
- ITKO9005 - The Stop command was accepted
- ITKO9006 - LPAR Agent process terminating

A sample JES log (DD SYS00001 of the started task):

```
2016:09:30 12:00:32 - Starting via operator command.
2016:09:30 12:00:32 - LPAR Agent started. Version: VA00 Compiled: May 19 2016 12:
22:31
2016:09:30 12:00:32 - Using configuration file: //DD:CONFIG
2016:09:30 12:00:32 - ---: Name = Dev LPAR Agent
2016:09:30 12:00:32 - ---: InitialTrace = False
2016:09:30 12:00:32 - ---: Mode = Server
2016:09:30 12:00:32 - ---: PortClient = 3997
2016:09:30 12:00:32 - ---: PortAgent = 4997
2016:09:30 12:00:32 - ---: BufferSize = 65536
2016:09:30 12:00:32 - ---: MaxBufferSize = 65536
2016:09:30 12:00:32 - ---: AppKASeconds = 600
2016:09:30 12:00:32 - ---: AppKALimit = 5
2016:09:30 12:00:32 - Configuration file processing successful.
2016:09:30 12:00:32 - LPAR Agent named Dev LPAR Agent
2016:09:30 12:00:32 - Agent socket ready.
2016:09:30 12:00:32 - Client socket ready.
2016:09:30 12:00:32 - Memory for message buffers allocated.
2016:09:30 12:00:32 - Known interfaces: 192.168.0.100
2016:09:30 12:00:32 - Initial configuration complete.
2016:09:30 12:00:35 - ID: 2 - Accepted agent connection from 192.168.0.100.
2016:09:30 12:00:35 - ID: 3 - Accepted agent connection from 192.168.0.100.
2016:09:30 12:00:35 - ID: 4 - Accepted agent connection from 192.168.0.100.
2016:10:03 14:19:09 - ID: 5 - Accepted client connection from 192.168.0.106.
2016:10:03 14:30:48 - ID: 5 - Client connection reset.
2016:10:03 14:30:48 - ID: 5 - Disconnected.
```

DevTest Mainframe Bridge

The mainframe bridge is a component that enables DevTest clients to communicate with DevTest mainframe agents.

The following graphic shows how the mainframe bridge interacts with other components.

DevTest Mainframe Bridge Properties

The following mainframe bridge properties must be added to the **local.properties** file in the **LISA_HOME** directory.

- **lisa.mainframe.bridge.enabled**
  Specifies whether the mainframe bridge is started when the registry is started. The bridge runs inside the registry and not as an external process.
  Default: false

- **lisa.mainframe.bridge.mode**
  Specifies the mode in which the mainframe bridge is run. The mode defines the peer relationship with the LPAR agents. The bridge can be run in client mode or server mode:

  - In client mode, the LPAR Agent must be configured to run in server mode. The bridge initiates the connection to the LPAR Agent.

  - In server mode, the LPAR Agent must be configured to run in client mode. The bridge waits for connections from the LPAR Agent.

  Values:

  - client

  - server

  Default: server

- **lisa.mainframe.bridge.port**
  In server mode, this property specifies the “well-known port” that the bridge listens on for connections from the LPAR Connection. This property is ignored in client mode.
  Values: Any valid port number.
  Default: 61617

- **lisa.mainframe.bridge.server.host**
  In client mode, this property defines the host to which the bridge initiates a connection. This property is ignored in server mode.
  Values: Any valid port number.
  Default: 127.0.0.1

- **lisa.mainframe.bridge.server.port**
  In client mode, this property defines the port to which the bridge initiates a connection on the specified host. This property is ignored in server mode.
  Values: Any valid port number.
  Default: 3997

- **lisa.mainframe.bridge.connid**
  Two-character unique ID for each client.
  Default: AA

DevTest Mainframe Bridge Rules

The following agent properties must be configured for the mainframe bridge:

- **Code page**
  Defines the codepage that is used to convert between ASCII and EBCDIC.
  Default: Cp1047

- **Initial packet buffer size**
  Specifies the size of the initial buffer to handle packets from the mainframe, in bytes. The buffer size increases if packets of a larger size are received.
  Default: 1024*65.

- **Packet trace LPAR**
  Specifies whether packets from and to the LPAR Agent are traced.
  Default: Tracing is disabled.

- **Packet trace client**
  Specifies whether packets from and to the workstations are traced.
  Default: Tracing is disabled.

- **Client connect timeout**
  Specifies the number of seconds to wait between retry connects if the bridge is started in client mode and the connection to the LPAR Agent fails.
  Default: 30

You can configure these properties from the Agents window of DevTest Portal. The properties appear in the **Settings** tab.

Administering

This section provides administrators with procedures to help them perform regular administrative duties in DevTest Solutions.

- General Administration
- Database Administration
- License Administration
- Security
- Logging
- Monitoring
- Telemetry

General Administration

This section contains the following topics that provide information on general administration of DevTest:

- Default Port Numbers
- Separating the Data Directory from the Installation Directory
- Directory Structure
- Running Server Components as Services
- Running DevTest with Ant and JUnit
- Memory Settings
- Third-Party File Requirements
- Enabling Additional Scripting Languages

Default Port Numbers

This page defines each default port number that the following components use:

- DevTest Server
- DevTest Workstation
- Demo Server

This page also describes how to change the port numbers for DevTest Portal.

> **Note:** As with any socket communication, ephemeral ports are consumed as necessary when local endpoints are created.

DevTest Server Default Port Numbers

The following table lists the default port numbers that various components in DevTest Server use. The table also includes the property that sets each port number.

Port	DevTest Component	Property
1505	Embedded web server	lisa.webserver.port
1506	Enterprise Dashboard (embedded web server)	dradis.webserver.port
1507	DevTest Portal	devtest.port
1508	Broker (embedded web server)	broker.webserver.port
1528	Derby database	lisadb.internal.port, lisadb.pool.common.url
1529	Demo server	Not applicable
1530	Enterprise Dashboard Derby database port	dradisdb.internal.port
2003	Enterprise Dashboard network port	lisa.net.15.port
2004	VSE Manager network port	lisa.net.9.port
2005	Test Runner network port	lisa.net.5.port
2006	ServiceManager network port	lisa.net.11.port
2008	Workstation network port	lisa.net.0.port
2009	Broker network port	lisa.pathfinder.broker.port
2010	Registry network port	lisa.net.3.port
2011	Coordinator network port	lisa.net.2.port
2012	JUnit exec network port	lisa.net.4.port
2013	VSE network port	lisa.net.8.port
2014	Simulator network port	lisa.net.1.port
2999	JDBC simulation driver	Not applicable
3128	HTTP proxy	laf.httpproxy.port
3997	LPAR agent	lisa.mainframe.bridge.server.port
61617	LPAR agent	lisa.mainframe.bridge.port

If more than one simulator is created on the same computer, the port numbers for the new simulators are increased from 2014. For example, three more simulators would have the port numbers 2015, 2016, and 2017.

Typically, there is one coordinator for each lab. However, if you create more coordinators, then you define the port on the command line. For example:

```
CoordinatorServer --name=tcp://hostname:2468/Coordinator2
```

The **lisa.properties** file defines most of the default port numbers.

If you want to change one or more port numbers after installation, add the property to the **local.properties** file and set the new value. Do not update the **lisa.properties** file.

The only time **site.properties** must be updated because of a port update is when the Enterprise Dashboard port changes. In itself, that is configured in **local.properties** like everything else. However, the registries must be configured to talk to the Enterprise Dashboard through **devtest.enterprisedashboard.host** and **devtest.enterprisedashboard.port**, which should be set in **site.properties**.

You can change the port number for the JDBC simulation driver by editing the **Virtual JDBC Listener** step in DevTest Workstation.

DevTest Workstation Default Port Numbers

The default port numbers in a DevTest Workstation installation are a subset of the default port numbers in DevTest Server.

Port	DevTest Component	Property
1505	Embedded web server	lisa.webserver.port
2004	VSE Manager	lisa.net.9.port
2005	Test Runner	lisa.net.5.port
2006	ServiceManager	lisa.net.11.port
2009	Broker	lisa.pathfinder.broker.port
2999	JDBC simulation driver	Not applicable
3128	HTTP proxy	laf.httpproxy.port

Demo Server Default Port Numbers

The following table lists the default port numbers that the demo server uses.

Port	DevTest Component
1098	JNDI
1099	JNDI
1528	Derby database
8080	HTTP

The following files define the default port numbers:

- lisa-demo-server/jboss/server/default/conf/jboss-service.xml
- lisa-demo-server/jboss/server/default/deploy/jboss-web.deployer/server.xml

If you want to change one or more port numbers after installation, you can update these files directly.

Change Port Number for DevTest Portal

To change the port number of DevTest Portal, you must configure both of the following property files:

- **local.properties**
- **phoenix.properties**

Follow these steps:

1. Go to the **LISA_HOME** directory.
2. If the **LISA_HOME** directory does not contain a **local.properties** file, copy **_local.properties** to **local.properties**.
3. Open the **local.properties** file.
4. Add the following property and set the value to the new port number.

   ```
   # DevTest UI
   devtest.port=port-number
   ```

5. Save the **local.properties** file.
6. If the **LISA_HOME** directory does not contain a **phoenix.properties** file, copy **_phoenix.properties** to **phoenix.properties**.
7. Open the **phoenix.properties** file.
8. Uncomment the **phoenix.port** property and set the value to the new port number.
9. Save the **phoenix.properties** file.

Separating the Data Directory from the Installation Directory

Different users of the DevTest application can have their own data directory, while sharing a common DevTest installation. In this scenario, users only need read access to the DevTest programs directory.

When installing, the default data directory is a subfolder that is named *DevTest* within the user's home folder. Example: %userprofile%/DevTest

Using the %userprofile% (or '~') user home variables is useful for system administrators who are installing this product on behalf of other users. These variables resolve to the user's home folder of whatever user is using the product. The specified data directory must exist and the user must have write permissions to it. If it does not exist, the installer creates it and tests to be sure it is writable.

Server Components

When installing Server components, the installer prompts will only prompt for a data directory if it detects a previous upgrade from a Shared Installation.

Note: We strongly recommend that the installation directory and the data directory are the same for server components. If you have used a shared installation previously, you can continue to do so; refer to previous versions of the user documentation for support.

If the data directory is different from the installation directory, DevTest creates a lisa.user.properties file in both the installation directory and in the user's home folder. The **lisa.data.dir** property in this property file tells the product where the data directory is located. The user must have write permissions to the specified data directory. If the data directory does not exist, the installer creates it and test to be sure it is writable.

The components included in the data directory are:

- Workstation Components: _local.properties, _logging.properties, _de-identify.xml, hotdeploy, library for customizations, projects with assets, examples folders
- Server Components: hotdeploy, CVS Monitor, VSE Deploy

On initial startup of DevTest Workstation, Projects, Examples, and Templates (_local.properties, _logging.properties, and _de-identify.xml) are copies or refreshed from the install directory to the data directory.

You can also specify the location of the file that defines the **lisa.data.dir** property by setting the **lisa.user.properties** system property or the **LISA_USER_PROPERTIES** environment variable. If the property has been defined as a system property, the system property definition takes precedence.

Adding User-specific JAR and ZIP Files to the Workstation Classpath

When using DevTest Workstation with a data directory that is separate from the DevTest install directory (also known as LISA_HOME), JAR and ZIP files can be added in the lib folder of the data directory. These are added to the end of the Workstation's classpath. This is helpful when you do not have write access to the LISA_HOME\lib directory and there are third-party libraries or custom code that you want to make accessible to DevTest Workstation.

The LISA_HOME\bin directory contains the following scripts that set the LISA_POST_CLASSPATH environment variable of the current terminal session to a classpath string containing all the JAR and ZIP files that are located in the lib folder of the data directory:

- updateUserPostClasspathEnv.bat (Windows)
- updateUserPostClasspathEnv.sh (Mac OS, UNIX)

Before executing these scripts, set the LISA_HOME environment variable to the location of your DevTest installation. The default location for DevTest installations is:

- C:\Program Files\CA\DevTest (Windows)
- /opt/CA/DevTest (Mac OS, UNIX)

Export Environment Variables on Mac OS and UNIX

When using the updateUserPostClasspathEnv.sh UNIX shell script, run it with the source command (or the "." alias) so that the LISA_POST_CLASSPATH variable is exported from the script into the current session. Here is an example of running the script, exporting the environment variable, and launching DevTest Workstation:

```
cd $LISA_HOME/bin
source ./updateUserPostClasspathEnv.sh
./Workstation
```

Export Environment Variables on Windows

Using a command prompt:

```
cd LISA_HOME/bin
updateUserPostClasspathEnv.bat Workstation.exe
```

Using PowerShell:

```
cd LISA_HOME/bin
.\updateUserPostClasspathEnv.bat .\Workstation.exe
```

When the LISA_POST_CLASSPATH environment variable has been set by the script, launch DevTest Workstation from the same terminal session. Any classes from the JAR and ZIP files from the lib directory will be accessible.

LISA_POST_CLASSPATH Environment Variable Scope

For DevTest Workstation to be aware of JAR and ZIP files from the lib folder of the data directory, the LISA_POST_CLASSPATH environment variable must be set in the same environment that DevTest Workstation is running in. If the LISA_POST_CLASSPATH environment variable has only been set in the current terminal session by the updateUserPostClasspathEnv script, then DevTest Workstation must be run from the same terminal session.

If new JAR or ZIP files have been added to the lib folder of the directory data changes since the LISA_POST_CLASSPATH environment variable was set, then you must run the updateUserPostClasspathEnv again.

If you do not want to run the updateUserPostClasspathEnv script every time before running DevTest Workstation in a terminal session, you can use the output of the script to help you set the LISA_POST_CLASSPATH environment variable at the system or user level for your operating system. Setting the LISA_POST_CLASSPATH environment variable at the system or user level causes all new instances of DevTest Workstation to use it unless it has been overwritten at the terminal session level.

Directory Structure

This page describes the directory structure of DevTest Workstation and DevTest Server.

- DevTest Workstation Directories
- DevTest Server Directories

This page assumes that you did not specify a separate data directory for the installation of DevTest Solutions. If you have a separate data directory, the following directories can be in a location other than **LISA_HOME**:

- cvsMonitors
- database
- hotdeploy

- locks
- tmp
- vseDeploy
- local.properties
- logging.properties
- de-identify.xml
- _local.properties
- _logging.properties
- _de-identify.xml
- library for customizations
- projects with assets
- examples folders

DevTest Workstation Directories

The **LISA_HOME** directory for DevTest Workstation contains the following directories:

> **Note:** The **locks** directory must have read/write permissions.

- **.install4j**
  Contains the installer.
- **addons**
  Contains the DevTest add-ons.
- **agent**
  Contains the files for the DevTest Java Agent.
- **bin**
  Contains the executable files for DevTest Workstation and other components.
- **defaults**
  Contains the audit document and staging documents that are common across all projects. Project-specific staging documents are located in the **Projects** directory.
- **doc**
  Contains the license agreement, JavaDocs for the DevTest Java Agent, and JavaDocs for the Software Development Kit (SDK).
- **examples**
  A default project containing examples that use the demo server.

- **examples_src**
  Examples source files and the Kiosk-related files.

- **hotDeploy**
  A directory that DevTest monitors. This file contains Java classes and JAR files. Java classes and JAR files in this directory are on the DevTest classpath. Any new files or directories added to this directory are dynamically added to the DevTest classpath.

  > Putting a custom extension into hotDeploy still requires rebooting the DevTest component (for example, VSE or DevTest Workstation) before the extension is available for use. The HotDeployClassLoader loads the extension's class files, but the "lisaextension" file is not discovered automatically, so the extension is still unknown to DevTest. The "lisaextension" files are only discovered during start-up.

- **incontainer**
  Contains the incontainer testing instructions.

- **jre**
  Contains the required JRE.

- **lib**
  Contains the required JAR files.

- **licenses**
  Contains the license files that are required for running DevTest Solutions.

- **locks**
  Contains the lock files that are used for interprocess concurrency.

- **Projects**
  Contains example projects, and any projects that you create.

- **reports**
  Contains the XML-based test reports that DevTest creates.

- **snmp**
  Contains the SNMP-related files.

- **tmp**
  Contains the logging files that DevTest creates. If you communicate with Support on an issue, you could be asked to send one or more files from this directory. The tmp directory must be 1.5 times the size of the installer.

- **umetrics**
  Contains the files that are related to collecting metrics.

- **webserver**
  Contains the web server files.

DevTest Server Directories

The **LISA_HOME** directory for DevTest Server contains the following directories:

> **Note:** The **locks** directory must have read/write permissions.

- **.install4j**
  Contains the installer.

- **addons**
  Contains the DevTest add-ons.

- **agent**
  Contains the files for the DevTest Java Agent.

- **bin**
  Contains the executable files for the registry, coordinator, simulator, VSE, DevTest Workstation, and other components. This directory also contains the following batch files:

  - startdefservers.bat - starts the default servers.

  - stopdefservers.bat - stops the default servers.

- **cvsMonitors**
  Contains the deployed CVS monitors.

- **database**
  Contains the DDL files and Application Insight upgrade scripts for various databases.

- **defaults**
  Contains the audit document and staging documents that are common across all projects. Project-specific staging documents are located in the **Projects** directory.

- **doc**
  Contains the license agreement, JavaDocs for the DevTest Java Agent, and JavaDocs for the Software Development Kit (SDK).

- **examples**
  A default project containing examples that use the demo server.

- **examples_src**
  Examples source files and the Kiosk-related files.

- **hotDeploy**
  A directory that DevTest monitors. This file contains Java classes and JAR files. Java classes and JAR files in this directory are on the DevTest classpath. Any new files or directories added to this directory are dynamically added to the DevTest classpath.

- **incontainer**
  Contains the incontainer testing instructions.

- **jre**
  Contains the required JRE.

- **lib**
  Contains the required JAR files.

- **licenses**
  Contains the license files that are required for running DevTest Solutions.

- **locks**
  Contains the lock files that are used for interprocess concurrency.

- **Projects**
  Contains example projects, and any projects that you create.

- **reports**
  Contains the XML-based test reports that DevTest creates.

- **snmp**
  Contains the SNMP-related files.

- **tmp**
  Contains the logging files that DevTest creates. If you communicate with Support on an issue, you could be asked to send one or more files from this directory.

- **umetrics**
  Contains the files that are related to collecting metrics.

- **webserver**
  Contains the web server files.

Running Server Components as Services

If you plan to leave the server components running most of the time, you can use the service executables in the **LISA_HOME\bin** directory.

The default names of the server components are used when started from the command line without a specific name. The lisa.properties file is installed with the following properties and default values:

- lisa.registryName=Registry

- lisa.coordName=Coordinator

- lisa.simulatorName=Simulator

- lisa.vseName=VSE

If you want to override a default value for any of these properties, specify a new property value in your **local.properties** file.

We recommend that you start and stop the components in the order shown.

To start server components as services:

Enter the following commands:

```
IdentityAccessManager start
EnterpriseDashboardService start
RegistryService start
PortalService start
BrokerService start
CoordinatorService start
SimulatorService start
VirtualServiceEnvironmentService start
```

To stop server components as services:

Enter the following commands:

```
SimulatorService stop
CoordinatorService stop
VirtualServiceEnvironmentService stop
BrokerService stop
PortalService stop
RegistryService stop
EnterpriseDashboardService stop
IdentityAccessManager stop
```

On UNIX, to configure the services to start automatically, consult your system administrator.

If DevTest Workstation started the report database (Derby), the report database shuts down when DevTest Workstation shuts down.

If the coordinator started the database (Derby), the database does not shut down when coordinator shuts down.

Running DevTest with Ant and JUnit

You can execute DevTest tests as JUnit tests within the execution path of an Ant build.

This feature provides real automated build and test integration opportunities. You can use the flexibility of Ant and the simplicity of JUnit with the power of DevTest Solutions to meet the following automation needs:

- Integration
- Load
- Stress
- Production monitoring

Ant is a Java-based, open source automated software building tool. Ant is extended with support for many third-party tools, including JUnit. For more information, see http://ant.apache.org/.

DevTest supports Ant 1.9.x and above with a 1.8 JDK. Refer to System Requirements for requirements.

JUnit is a Java-based, open source unit-testing framework. JUnit is widely supported by third-party testing tools, including DevTest Solutions. For more information, see http://www.junit.org.

- The Execute JUnit Test Case/Suite step can execute a JUnit test. DevTest Ant Tasks can run any DevTest test or suite, making the DevTest tests look like they JUnit tests.
- The JUnit reports and log files are generated only when the **junitlisa** Ant task is used to run tests.
- The JUnit step enables existing JUnit test cases to be integrated into the testing workflow. You can use DevTest to wrap the JUnit tests and execute them through DevTest.
- The **junitlisa** task enables automating testing without going through the DevTest user interface. The **junitlisa** task generates an HTML report because there is no user interface that it can send results to.

Run DevTest Tests as JUnit Tests

The JUnit step supports JUnit3 and JUnit4 test cases and test suites. This procedure describes the steps that are required to execute DevTest tests as JUnit tests and have them report as native JUnit tests.

Follow these steps:

1. Verify that 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 DevTest install directory.
4. Use the **junitlisa** task to execute DevTest tests as JUnit tests.
5. (Optional) Use the **junitlisareport** task to create HTML-based reports from the JUnit XML output.

Logging output is written to a **junitlisa_log.log** file in the **user.home\lisatmp** directory.

The logging level that is used is the same level that is 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 **user.home\lisatmp\junit\index.html**.

DevTest Solutions Ant Tasks

This section describes DevTest Ant tasks.

- junitlisa Ant Task
- junitlisareport Ant Task

junitlisa Ant Task

The **junitlisa** task is a "'drop in" replacement for the JUnit task available with Ant, but it executes DevTest tests instead of JUnit tests. Most continuous build systems recognize the XML output files and integrate the build dashboard with the test results.

This task is a direct subclass of the JUnit task. Therefore, the junitlisa task has the same attributes and nested elements as the JUnit task. However, be aware of the following differences in behavior:

- You cannot set the **fork** attribute to false.
- You cannot add nested **test** elements. Use the **test** attribute instead.

- You cannot add nested **batchtest** elements. Use the **suite** attribute instead.
- An implied classpath consisting of **LISA_HOME\bin*.jar**, **LISA_HOME\lib*.jar, *.zip**, and **LISA_HOME\lib\endorsed*.jar** is added.
- An implied **java.endorsed.dirs** system property pointing to **LISA_HOME\lib\endorsed** is added.
- If no formatter is specified, then a default formatter of type **xml** is added.
- The **printsummary** attribute is defaulted to true.
- The **maxmemory** attribute is defaulted to 1024m.
- The **showoutput** attribute is defaulted to true.

In addition to the attributes inherited from the JUnit task, the **junitlisa** task has the following attributes:

- **suite**
  The file name of a suite document.
  Example: suite="AllTestsSuite.ste"
- **test**
  The file name of a test case.
  Example: test="multi-tier-combo.tst"
- **stagingDoc**
  The file name of a staging document.
  Example: stagingDoc="Run1User1Cycle.stg"
- **config**
  A named internal configuration set or a file name.
  Example: config="project.config"
- **outfile**
  The file name that is used to write reporting data. If the value does not comply with the standard naming scheme for junitlisareport, specify a fully configured junitreport task instead.
  Example: outfile="report"
- **registry**
  A pointer to the registry to use when you want to stage the test cases remotely.
  Example: registry="tcp://testbox:2010/Registry"
- **preview**
  Lets you write out the name and description of each test case, without executing the test cases.
  Example: preview="true"
- **user**
  The user name for ACL
  Example: user="admin"
- **password**
  The password for ACL
  Example: password="admin"

- **mar**
  The file name of a MAR document.
  Example: mar=”example.mar”

- **mari**
  The file name of a MAR info document.
  Example: mari=”example.mari”

The **junitlisa** task includes a nested element named **lisatest**. This element has the following attributes:

- **suite**
  The file name of a suite document.
  Example: suite="AllTestsSuite.ste"

- **test**
  The file name of a test case.
  Example: test="multi-tier-combo.tst"

- **stagingDoc**
  The file name of a staging document.
  Example: stagingDoc="Run1User1Cycle.stg"

- **mar**
  The file name of a MAR document.
  Example: mar=”example.mar”

- **mari**
  The file name of a MAR info document.
  Example: mari=”example.mari”

The attribute values can use curly braces, which are resolved in the usual way.

You are required to specify at least one test or suite. You can specify a test or suite in a **lisatest** nested element or in the **test** or **suite** attribute. You can specify multiple tests and suites by adding more **lisatest** elements. The tests and suites are executed in the order in which they appear in the XML.

When you run a single test with the **test** attribute, the test has the following default behavior:

- Staged with a single vuser.

- Run once.

- 100 percent think time.

To change this default behavior, wrap the test in a suite and specify an alternative staging document.

junitlisareport Ant Task

You can produce HTML-based reports with the **junitlisareport** task or the regular **junitreport** task.

The **junitlisareport** task is a subclass of the regular **junitreport** element, except that sensible defaults are specified. This task is equivalent to the following code:

```
<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. Because the task is a direct subclass of **junitreport**, all the attributes and nested elements that **junitreport** has are supported.

We recommend specifying the inherited **toDir** attribute usually, although it defaults to the current working directory.

You can specify the path where the reports from your Ant tasks are created by defining the **lisa.reports.dir** property in either site.properties or local.properties. If you do not define this property, the reports are created in the reports folder of your DevTest installation directory (**LISA_HOME\reports**).

Ant and JUnit Usage Examples

This section provides examples of Ant and JUnit use with DevTest.

Example Complete Ant Build File

The **build.xml** file in the **LISA_HOME\examples** directory is a complete Ant build file.

The build file contains two targets:

- The **lisaTests** target runs a suite as JUnit tests. The suite is specified with the **lisatest** nested element.
- The **oneTest** target runs a test case as a JUnit test. The test case is specified with the **test** attribute.

The build file assumes that access control (ACL) is enabled. Therefore, the **user** and **password** attributes are included.

Example junitlisa Task for a Test Case

The following **junitlisa** task is configured to run a single test case on a remote registry.

```
<junitlisa test="MyTest.tst"
    config="dev"
    registry="tcp://testbox:2010/Registry"
    toDir="${testReportDir}"
    haltOnError="no"
    errorProperty="test.failure">
    <jvmarg value="-DmySystemProp=someValue"/>
  </junitlisa>
```

JUnit Usage Examples

This page includes examples of how to use JUnit tests with DevTest.

- JUnit 3 Test Cases
- JUnit 3 Test Suites
- JUnit 4 Test Cases

- JUnit 4 Test Suites

JUnit 3 Test Cases

For JUnit 3 test cases, follow the JUnit conventions for writing your test case. In particular:

- Your test class should extend **junit.framework.TestCase**.
- Your method names start with **test**.

For example:

```
import junit.framework.TestCase;

public class JUnit3TestCase extends TestCase {

   public void testOneIsOne() {
      assertEquals (1, 1);
   }

   public void testTwoIsThree() {
      assertEquals (2, 3);
   }

}
```

JUnit 3 Test Suites

For JUnit 3 test suites, your suite is not required to extend **junit.framework.TestSuite**. However, it must implement the **suite()** method, with test cases wrapped by **JUnit4TestAdapter**.

For example:

```
import junit.framework.Junit4TestAdapter;
import junit.framework.TestSuite;

public class JUnit3VanillaTestSuite {

   public static TestSuite suite() {
      TestSuite suite = new TestSuite();
      suite.addTest ( new JUnit4TestAdapter ( MyJUnit3TestCase.class) ) ;
      return suite;
   }

}
```

JUnit 4 Test Cases

For JUnit 4 test cases, the test methods must have the **@org.junit.Test** annotation on the methods, as required by JUnit 4.

For example:

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class JUnit4TestCase {

    @Test
    public void oneIsOne() { assertEquals (1, 1); }

    @Test
    public void twoIsThree() { assertEquals (2, 3); }

}
```

JUnit 4 Test Suites

To implement a JUnit 4 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 ( { JUnit4TestCase.class } )

public class JUnit4VanillaTestSuite { // empty }
```

Note: If loading your JUnit test returns an **IllegalArgumentException** on the JUnit step, confirm that you added **.class** to the end of the class name. You can manually verify the spelling of the classname or use the classpath browser to locate the class.

Integration with CruiseControl

If you are using CruiseControl to provide some continuous integration, you can configure it to include test failures in its Control Panel.

In the JUnit Ant task that is shown in Ant and JUnit Usage Examples, **${testReportDir}** is defined as **${LISA_HOME}\bin\lisa-core.jar**.

The following sample shows a typical **CruiseControl config.xml**:

```
<project name="myproj " buildafterfailed="false">
  <log>
    <merge dir="/home/cruise/build"/>
    <merge dir="/path/to/lisajunit/output"/>
  </log>
.
.
.
</project>
```

More Example Build Files

In addition to the **build.xml** file described in Ant and JUnit Usage Examples, the **LISA_HOME\examples** directory contains the following example files for automating DevTest projects:

- **automated-build.xml:** The master build file that you place at the top of the project hierarchy.
- **lisa-project-build.xml:** The build file that you place in each project. The file name must be changed to **build.xml**.
- **common.xml:** The file that you place in each subdirectory between the root and the project.
- **common-macros.xml:** This file contains macros for performing various tasks with DevTest Server components. For example, you can start the registry, run a suite, or deploy a virtual service. This file is not a standalone build file and is meant to be incorporated into existing Ant-based frameworks.

For more information, see the comments in each file.

Memory Settings

The **.vmoptions** files are used to pass more parameters to a Java process to modify the default settings that are used for the JVM. These files let you customize the memory allocation settings for each of the DevTest processes used in the server. The **LISA_HOME\bin** folder contains **.vmoptions** files with the same name as each executable file.

> **Note:** A full list of JVM parameters is available from the Oracle website under Configuring the Default JVM and Java Arguments.

On Windows operating systems, the default memory limit for DevTest Workstation is 512 MB (-Xmx512m).

For a Windows system, it is recommended that DevTest Server is on Windows 64-bit to use more memory when needed.

For Application Insight, the registry service may require more heap memory to process millions of business transactions. The recommended maximum heap memory setting for the registry is 1 GB per five million transactions.

- Change the Java Heap Size on Windows and UNIX
- Change the Java Heap Size on OS X
- Adjust the Memory for Individual Processes

Change the Java Heap Size on Windows and UNIX

Follow these steps:

1. Open the target file with a **.vmoptions** extension. For example:

   ```
   # Enter one VM parameter per line
   # For example, to adjust the maximum memory usage to 512 MB, uncomment the f
   ollowing line:
   # -Xmx512m
   # To include another file, uncomment the following line:
   # -include-options [path to other .vmoption file]
   ```

2. To change the maximum memory to be allocated (**Xmx**), uncomment the following line:

   ```
   # -Xmx512m
   ```

3. To change the minimum memory to be allocated (**Xms**), add the VM argument on another line.

   ```
   -Xms128M
   ```

   The file would then specify the memory allocation range as in the following example.

   ```
   -Xms128M
   -Xmx512M
   ```

4. Save the text file in the **LISA_HOME\bin** folder.

Change the Java Heap Size on OS X

Browse to the **LISAWorkstation.app** and edit the **Info.plist** file.

The path is **LisaHome/bin/LISAWorkstation.app/Contents/Info.plist**. You can modify the **Info.plist** file, but your changes apply only to DevTest Workstation.

Adjust the Memory for Individual Processes

You can adjust memory for the following programs:

- ATK
- Broker
- BrokerService
- CoordinatorServer
- CoordinatorService
- CVSManager
- EnterpriseDashboard
- EnterpriseDashboardCIC
- EnterpriseDashboardCICService
- EnterpriseDashboardService
- MakeMar
- PFAdapter
- PFCmdLineTool
- Portal
- PortalService
- Registry
- RegistryService
- ServiceImageManager
- ServiceManager
- Simulator
- SimulatorService
- TestRunner
- VirtualServiceEnvironment

- VirtualServiceEnvironmentService
- VSEManager
- Workstation

Some of these programs are available in installations of DevTest Server, but not DevTest Workstation.

To tweak the memory for each process, you can edit the individual script files. For example, you can allow the registry to have 128M but the simulator to have 1024M.

You can also copy these files and edit the copies. For example, you can copy **Registry** to **MyRegistry** and tweak **MyRegistry**.

Third-Party File Requirements

To use DevTest Solutions with various third-party applications, you must make JAR files from the third-party application available to DevTest.

> For information about adding user-specific JAR and ZIP files to the classpath, see Adding User-specific JAR and ZIP Files to the Workstation Classpath.

Unless otherwise specified in the following sections, you can use any of these approaches:

- Place the files in the **LISA_HOME\hotDeploy** directory
- Place the files in the **LISA_HOME\lib** directory
- Define the **LISA_POST_CLASSPATH** variable

The following example shows the **LISA_POST_CLASSPATH** variable on a Windows computer. The files in this example are WebSphere MQ files.

```
LISA_POST_CLASSPATH="C:\Program Files\IBM\MQSeries\Java\lib\com.ibm.mq.allclient.
jar;C:\Program Files\IBM\MQSeries\Java\lib\com.ibm.mq.commonservices.jar;C:
\Program Files\IBM\MQSeries\Java\lib\javax.jms-api-2.0.jar"
```

CA does *not* provide the third-party files that are listed in this page.

The JAR files must support Java 8.

The **hotDeploy** directory can be in a location other than **LISA_HOME**. For an explanation of what files may be in a separate data directory, see Installation Options.

- ActiveMQ File Requirements
- Apache Kafka
- JCAPS File Requirements
- JMS Messaging File Requirements
- Oracle OC4J File Requirements
- RabbitMQ Requirements
- SAP File Requirements

- SAP IDoc Virtualization Requirements
- SAP RFC Virtualization Requirements
- SonicMQ File Requirements
- TIBCO File Requirements
- WebLogic File Requirements
- webMethods File Requirements
- WebSphere MQ File Requirements

ActiveMQ File Requirements

To use CA DevTest Solutions with various third-party applications, ensure that JAR files from the third party are available to DevTest:

1. Ensure that the following file exists in the LISA_HOME\hotDeploy directory:

   `activemq-client-X.X.X.jar`

2. The functioning of activemq-client JAR file depends on hawtbuf JAR file. An appropriate version of hawtbuf JAR file is also required for DevTest to work with ActiveMQ broker. You can find the relevant version of hawtbuf JAR file for the ActiveMQ client jar version in ActiveMQ::Client. The correct version of hawtbuf JAR file version is mentioned in the Compile Dependencies for the corresponding activemq-client version. Place the hawtbuf JAR file in the LISA_HOME\hotDeploy directory.

Apache Kafka

DevTest Solutions requires Apache Kafka version 0.8.2 or later. Kafka 1.1.0 requires the following jars from KAFKA_HOME/libs directory:

- kafka-clients-1.1.0.jar
- lz4-java-1.4.jar
- slf4j-log4j12-1.7.25.jar
- snappy-java-1.1.7.1.jar

See the Apache Kafka documentation for more information about the JAR files that you might need.

Note: The **LISA_HOME\hotDeploy** option is not supported for Apache Kafka. You must place these files in the **LISA_HOME\lib** directory.

JCAPS File Requirements

If you are using the JCAPS Messaging (Native) step, see the JCAPS documentation for information about the JAR files that you might need.

If you are using the JCAPS Messaging (JNDI) step, the **com.stc.jms.stcjms.jar** file is required. See the JCAPS documentation for information about other JAR files that you might need.

The JAR files are available in the **lib** directory of the JCAPS installation.

JMS Messaging File Requirements

If you are using the JMS Messaging (JNDI) step, see the JMS provider documentation for information about the JAR files that you might need.

Oracle OC4J File Requirements

The following JAR files are required:

- dms.jar
- oc4j.jar
- oc4jclient.jar

See the OC4J documentation for more information about JAR files that you might need. The JAR files are available in the **lib** directory of the OC4J installation.

RabbitMQ Requirements

Copy the JAR file to the **LISA_HOME\lib** directory or define the **LISA_POST_CLASSPATH** variable. Do not copy the file to the **LISA_HOME\hotDeploy** directory.

The following JAR file is required:

- amqp-client-x.x.x.jar

SAP File Requirements

The following JAR files are required:

- sapjidoc3.0.8/sapidoc3.jar - required for the SAP IDoc steps and virtualization
- sapjco3.0.9/sapjco3.jar
- sapjco3.0.9/*platform*/ a native library file of the platform, where *platform* is your computer system (osx_64, windows_x86, and others) - required for the SAP RFC step and the SAP IDoc steps and virtualization

SAP IDoc Virtualization Requirements

1. Create an RFC Destination of type T on both the client and server SAP systems. DevTest uses this RFC Destination to receive IDocs from both systems.
2. Update the outbound partner profile on the client and server SAP system. Update the outbound parameter in the partner profile for the respective IDoc type to send to the port number associated with the RFC Destinations that are created in Step 1. This allows DevTest to receive IDocs from the client and server SAP systems.
3. Create and import the connection property files in your project under the Data directory.
    a. Create the RFC Connection (with properties from .jcoServer) for the client RFC Destination created in Step 1.
    b. Create the System Connection property file (with properties from .jcoDestination) for the client SAP system.

c. Create the RFC Connection (with properties from .jcoServer) for the server RFC Destination created in Step 1.

d. Create the System Connection property file (with properties from .jcoDestination) for the server SAP system.

Before you begin recording, you must also create and import the connection property files in your project under the **Data** directory. You need these files to start JCo servers to receive IDocs from and forward IDocs to the client and server SAP systems. You need the RFC Connection (with properties from .jcoServer) and System Connection (with properties from .jcoDestination) property files. You need these files to start JCo servers to receive IDocs from and forward IDocs to the client and server SAP systems. Consult the SAP administrator about populating these property files.

SAP RFC Virtualization Requirements

1. Create an RFC Destination of type T on the client SAP system. DevTest uses this RFC Destination to intercept remote function calls that the client system makes.

2. Update the ABAP code that makes the remote function call to use the new destination.

3. Create and import the connection property files in your project under the Data directory. Consult the SAP administrator about populating these property files.

   a. Create the RFC Connection (with properties from .jcoServer) for the client RFC destination that was created in Step 1. This file MUST NOT specify jco.server. repository_destination.

   b. Create the Repository Connection property file (with properties from the . jcoDestination) for the system that acts as the RFC repository. This is typically the same as the system on which the RFC was run.

   c. Create the System Connection property file (with properties from the . jcoDestination) for the system where the RFC runs. If this is the same as the Repository connection, only one properties file is needed.

SonicMQ File Requirements

The following JAR files are required:

- mfcontext.jar
- sonic_Client.jar
- sonic_XA.jar

The JAR files are available in the **lib** directory of the SonicMQ installation.

TIBCO File Requirements

The requirements for TIBCO vary depending on the application.

Copy the JAR files to the **LISA_HOME\lib** directory or define the **LISA_POST_CLASSPATH** variable. Do not copy the files to the **LISA_HOME\hotDeploy** directory.

TIBCO Rendezvous Messaging

The following JAR files are required:

- tibrvj.jar
- tibrvjms.jar
- tibrvjsd.jar
- tibrvjweb.jar
- tibrvnative.jar
- tibrvnativesd.jar

TIBCO Rendezvous .dll files are also required. Copy all .dll files from the TIBCO Rendezvous **bin** directory to the **LISA_HOME\bin** directory. Reference the **LISA_HOME\bin** location in your path environment.

In addition, add the TIBCO Rendezvous **bin** directory to your **PATH** environment variable.

TIBCO EMS Messaging or TIBCO Direct JMS

The following JAR files are required:

- tibcrypt.jar
- tibjms.jar
- tibjmsadmin.jar
- tibjmsapps.jar
- tibrvjms.jar

TIBCO Hawk Metrics

The following JAR files are required:

- console.jar
- talon.jar
- util.jar

TIBCO Rendezvous and/or TIBCO EMS JAR files, depending on which transport TIBCO Hawk is using, also must be copied to the **LISA_HOME\lib** directory.

WebLogic File Requirements

The **weblogic.jar** file is required. If you are using security or JMX, other JAR files might be required.

See the WebLogic documentation for more information about JAR files that you might need. The JAR files are available in the **lib** directory of the WebLogic installation.

webMethods File Requirements

The following JAR files are required:

- (webMethods Integration Server 8.2) Copy the wm-isclient.jar from your webMethods installation to your DevTest installation_directory\lib\shared directory.
- (webMethods Integration Server 7.1 and later) installation_directory\lib\shared\wm-isclient.jar
- (webMethods Integration Server 7.0 and earlier) installation_directory\lib\client.jar
- wm-enttoolkit.jar
- wmbrokerclient.jar
- wmjmsadmin.jar
- wmjmsclient.jar
- wmjmsnaming.jar

The JAR files are available in the **lib** directory of the webMethods installation.

WebSphere MQ File Requirements

Copy the JAR files to the **LISA_HOME\lib** directory or define the **LISA_POST_CLASSPATH** variable. Do not copy the files to the **LISA_HOME\hotDeploy** directory.

> **Note:** If the operating system is a Japanese version, use the files that are listed for WebSphere MQ 7.

The JAR files are available in the **MQ_HOME\java\lib** directory.

WebSphere MQ 5.2

The following JAR files are required for WebSphere MQ 5.2:

- com.ibm.mqjms.jar
- com.ibm.mqbind.jar
- com.ibm.mq.pcf.jar
- com.ibm.mq.jar
- connector.jar

WebSphere MQ 6

The following JAR files are required for WebSphere MQ 6:

- com.ibm.mq.jar

- com.ibm.mq.pcf.jar
- com.ibm.mqjms.jar
- connector.jar
- dhbcore.jar

WebSphere MQ 7

The following JAR files are required for WebSphere MQ 7:

- com.ibm.mq.commonservices.jar
- com.ibm.mq.headers.jar
- com.ibm.mq.jar
- com.ibm.mq.jmqi.jar
- com.ibm.mq.pcf.jar
- com.ibm.mqjms.jar
- connector.jar
- dhbcore.jar

WebSphere MQ 8 and 9

The following JAR files are required for WebSphere MQ 8 and 9:

- com.ibm.mq.allclient.jar
- com.ibm.mq.commonservices.jar

Alternately, you can use the following JAR files:

- com.ibm.mq.commonservices.jar
- com.ibm.mq.headers.jar
- com.ibm.mq.jar
- com.ibm.mq.jmqi.jar
- com.ibm.mq.pcf.jar
- com.ibm.mqjms.jar

In addition, if you are using WebSphere MQ in JMS mode, the javax.jms-api-2.0.jar file is required.

Enabling Additional Scripting Languages

To let users use more JSR-223 languages in the Execute Script test step, the Scriptable assertion, the match script editor, and the Scriptable data protocol, you can enable additional languages.

To enable an additional scripting language, put the language's jar file into the **hotdeploy** directory of the remote coordinator, server, or CA Service Virtualization. At next startup, the additional language will be in the **Language** drop-down in DevTest Workstation.

Database Administration

Apache Derby is provided as an out-of-the-box database so that you have a functioning system as you prepare to move to the enterprise database solution for your organization. Derby is *not* supported as an enterprise database solution. The Derby database is located in the **LISA_HOME\database\lisa.db** directory.

To avoid manual data migration issues, we recommend that you configure enterprise databases as a post-installation task.

The following components interact with a database:

- Reporting
- Access control (ACL)
- Java Agent broker
- Virtual service environment (VSE)
- Enterprise Dashboard

Use your site-specific procedures to back up the enterprise databases that you use for DevTest Solutions.

For information about database system requirements, see System Requirements.

- External Registry Database Configuration
- External Java Agent Database Configuration
- External Enterprise Dashboard Database Configuration
- Database Maintenance

External Registry Database Configuration

You can configure the registry to use an external database by editing the **site.properties** file in the **LISA_HOME** directory.

> **Note:** For information about database system requirements, see System Requirements .

The **site.properties** file includes properties for each of the supported databases:

- IBM DB2
- Derby
- MySQL
- Oracle
- Microsoft SQL Server

If you are running DevTest Server, you do not need to reconfigure each DevTest Workstation installation. The configuration in **site.properties** is propagated to each workstation, VSE, coordinator, simulator server, and any other DevTest component that connects to the registry.

The **LISA_HOME\database\dropSchema** directory contains the following files for the registry database:

- **db2_drop.ddl**
- **derby_drop.ddl**
- **mysql_drop.ddl**
- **oracle_drop.ddl**
- **sqlserver_drop.ddl**

These files can be useful if you need to revert the schema.

Configure DevTest to Use DB2

This page describes how to configure DevTest to use an IBM DB2 database.

The following properties in the **site.properties** file are relevant to database configuration:

```
lisadb.acl.poolName=common
lisadb.broker.poolName=common
lisadb.reporting.poolName=common

lisadb.pool.common.driverClass=com.ibm.db2.jcc.DB2Driver
lisadb.pool.common.url=jdbc:db2://[HOSTNAME]:[PORT]/[DATABASENAME]:
progressiveStreaming=2;
lisadb.pool.common.user=[USER]
lisadb.pool.common.password=[PASSWORD]

lisadb.pool.common.minPoolSize=0
lisadb.pool.common.initialPoolSize=0
lisadb.pool.common.maxPoolSize=10
lisadb.pool.common.acquireIncrement=1
lisadb.pool.common.maxIdleTime=45
lisadb.pool.common.idleConnectionTestPeriod=5
```

To minimize the number of connections to the database, connection pooling is used. The underlying implementation of the pooling functionality is c3p0. For detailed information about the settings, see http://www.mchange.com/projects/c3p0/index.html#configuration_properties. By default, all the components use the **common** connection pool. However, you can define a separate pool for each component or mix and match.

For the reporting database, we recommend at least 10 GB for optimal performance. A database for VSE is required only if you are working with legacy images created in a release earlier than 6.0.

The value of any property that ends with **password** is automatically encrypted at startup.

> **Note:** The remote installations of DevTest Workstation, coordinator, simulator server, VSE, or any other remote DevTest component do not require more configuration. They all receive **site.properties** from the registry when they connect and configure their database access accordingly.

Follow these steps:

1. Ensure that the code page of the DB2 database is 1208.
2. Ensure that the page size of the DB2 database is at least 8 KB.
3. In the **LISA_HOME** directory, locate the **_site.properties** file.
4. Change the file name to **site.properties**.
5. Open the **site.properties** file.
6. Configure the database configuration properties.
   You typically update the following properties:
   - lisadb.pool.common.url
   - lisadb.pool.common.user
   - lisadb.pool.common.password
7. The schema is automatically created in the database when the registry starts for the first time. However, if you do not want the DevTest user to have DBA privileges, you can manually create the schema beforehand. The **db2.ddl** file in the **LISA_HOME\database** directory contains SQL statements that can serve as the basis for creating the reporting tables and indexes.
8. Ensure that the **lisadb.internal.enabled** property is commented out or set to false.
9. Save the **site.properties** file.
10. Start the registry.

Configure DevTest to Use MySQL

This page describes how to configure DevTest to use a MySQL database.

The following properties in the **site.properties** file are relevant to database configuration:

```
lisadb.acl.poolName=common
lisadb.broker.poolName=common
lisadb.reporting.poolName=common
```

```
lisadb.pool.common.driverClass=com.mysql.jdbc.Driver
lisadb.pool.common.url=jdbc:mysql://DBHOST:DBPORT/DBNAME
lisadb.pool.common.user=database_username
lisadb.pool.common.password=database_password

lisadb.pool.common.minPoolSize=0
lisadb.pool.common.initialPoolSize=0
lisadb.pool.common.maxPoolSize=10
lisadb.pool.common.acquireIncrement=1
lisadb.pool.common.maxIdleTime=45
lisadb.pool.common.idleConnectionTestPeriod=5
```

To minimize the number of connections to the database, connection pooling is used. The underlying implementation of the pooling functionality is c3p0. For detailed information about the settings, see http://www.mchange.com/projects/c3p0/index.html#configuration_properties. By default, all the components use the **common** connection pool. However, you can define a separate pool for each component or mix and match.

The MySQL database must provide collation and characters set supporting UTF-8; double-byte characters are stored in the ACL and reporting tables. The default code page for the database has to be UTF-8; it is not enough only to define your database as UTF-8. If the MySQL database is not UTF-8, then runtime errors occur.

For the reporting database, we recommend at least 10 GB for optimal performance. A database for VSE is required only if you are working with legacy images created in a release earlier than 6.0.

The value of any property that ends with **password** is automatically encrypted at startup.

> **Note:** The remote installations of DevTest Workstation, coordinator, simulator server, VSE, or any other remote DevTest component do not require more configuration. They all receive **site.properties** from the registry when they connect and configure their database access accordingly.

Follow these steps:

1. In the **LISA_HOME** directory, locate the **_site.properties** file.
2. Change the file name to **site.properties**.
3. Open the **site.properties** file.
4. Configure the database configuration properties.
   You typically update the following properties:
   - lisadb.pool.common.url
   - lisadb.pool.common.user
   - lisadb.pool.common.password
5. The schema is automatically created in the database when the registry starts for the first time. However, if you do not want the DevTest user to have DBA privileges, you can manually create the schema beforehand. The **mysql.ddl** file in the **LISA_HOME\database** directory contains SQL statements that can serve as the basis for creating the reporting tables and indexes.

6. Ensure that the **lisadb.internal.enabled** property is commented out or set to false.

7. Save the **site.properties** file.

8. Add the MySQL JDBC driver to the **LISA_HOME\lib\shared** directory. The minimum version of the MySQL JDBC driver is 5.1.25.

9. Start the registry.

Configure DevTest to Use Oracle

This page describes how to configure DevTest to use an Oracle database.

The following properties in the **site.properties** file are relevant to database configuration:

```
lisadb.acl.poolName=common
lisadb.broker.poolName=common
lisadb.reporting.poolName=common

lisadb.pool.common.driverClass=oracle.jdbc.driver.OracleDriver
## Select one of the two connection URLs depending on usage of SID or SERVICE
lisadb.pool.common.url=jdbc:oracle:thin:@[HOST]:1521:[SID]
lisadb.pool.common.url=jdbc:oracle:thin:@//[HOST][:PORT]/SERVICE
lisadb.pool.common.user=[USER]
lisadb.pool.common.password=[PASSWORD]

lisadb.pool.common.minPoolSize=0
lisadb.pool.common.initialPoolSize=0
lisadb.pool.common.maxPoolSize=10
lisadb.pool.common.acquireIncrement=1
lisadb.pool.common.maxIdleTime=45
lisadb.pool.common.idleConnectionTestPeriod=5
```

To minimize the number of connections to the database, connection pooling is used. The underlying implementation of the pooling functionality is c3p0. For detailed information about the settings, see http://www.mchange.com/projects/c3p0/index.html#configuration_properties. By default, all the components use the **common** connection pool. However, you can define a separate pool for each component or mix and match.

For the reporting database, we recommend at least 10 GB for optimal performance. A database for VSE is required only if you are working with legacy images created in a release earlier than 6.0.

The value of any property that ends with **password** is automatically encrypted at startup.

> **Note:** The remote installations of DevTest Workstation, coordinator, simulator server, VSE, or any other remote DevTest component do not require more configuration. They all receive **site.properties** from the registry when they connect and configure their database access accordingly.

Follow these steps:

1. Ensure that the character set of the Oracle database supports Unicode. In addition, for the initial creation of the database, the Oracle user must have the **CREATE VIEW** system privilege. Without this privilege, the following error might be generated during installation: **Schema creation failed: ORA-01031: insufficient privileges**.

2. In the **LISA_HOME** directory, locate the **_site.properties** file.

3. Change the file name to **site.properties**.

4. Open the **site.properties** file.

5. Configure the database configuration properties.
   You typically update the following properties:

   - lisadb.pool.common.url
   - lisadb.pool.common.user
   - lisadb.pool.common.password

   For the lisadb.pool.common.url property, you can use the service name.

6. The schema is automatically created in the database when the registry starts for the first time. However, if you do not want the DevTest user to have DBA privileges, you can manually create the schema beforehand. The **oracle.ddl** file in the **LISA_HOME\database** directory contains SQL statements that can serve as the basis for creating the reporting tables and indexes. All DevTest users will share the same schema and privileges.

7. Ensure that the **lisadb.internal.enabled** property is commented out or set to false.

8. Save the **site.properties** file.

9. Ensure that **LISA_HOME\lib\shared** contains the JDBC driver. In most cases, **ojdbc7.jar** is the correct driver for Java 1.7. If you are using Java 1.6, then use **ojdbc6.jar**. If you are using Java 1.5, then use **ojdbc5.jar**.

10. Start the registry.

Configure DevTest to Use SQL Server

This page describes how to configure DevTest to use a Microsoft SQL Server database.

The following properties in the **site.properties** file are relevant to database configuration:

```
lisadb.acl.poolName=common
lisadb.broker.poolName=common
lisadb.reporting.poolName=common

lisadb.pool.common.driverClass=com.microsoft.sqlserver.jdbc.SQLServerDriver
lisadb.pool.common.url=jdbc:sqlserver://SERVER:PORT;databaseName=DATABASENAME
lisadb.pool.common.user=database_username
lisadb.pool.common.password=database_password

lisadb.pool.common.minPoolSize=0
lisadb.pool.common.initialPoolSize=0
lisadb.pool.common.maxPoolSize=10
lisadb.pool.common.acquireIncrement=1
lisadb.pool.common.maxIdleTime=45
lisadb.pool.common.idleConnectionTestPeriod=5
```

Note: If you are running DevTest in a Linux environment while using MS SQL Server on Windows with integrated/Windows security, use the following property configurations.

```
lisadb.internal.enabled=false
lisadb.pool.common.driverClass=net.sourceforge.jtds.jdbc.Driver
lisadb.pool.common.url=jdbc:jtds:sqlserver://<DB_IP_ADDR>:<DB_IP_PORT>;
```

```
instance=MSSQLSERVER;databaseName=devtest;domain=<USER_DOMAIN>;user=<USER_ID>;
password=<USER_PASSWORD>;
lisadb.pool.common.user=<USER_DOMAIN>\<USER_PASSWORD>
lisadb.pool.common.
password_enc=l335e298cc850c4146cf71b3b47afaf025cd9bbda7a40c4ddc13d64c0b6b5ad7521cd
630fa379617be582a0
```

To minimize the number of connections to the database, connection pooling is used. The underlying implementation of the pooling functionality is c3p0. For detailed information about the settings, see http://www.mchange.com/projects/c3p0/index.html#configuration_properties. By default, all the components use the **common** connection pool. However, you can define a separate pool for each component or mix and match.

For the reporting database, we recommend at least 10 GB for optimal performance. A database for VSE is required only if you are working with legacy images created in a release earlier than 6.0.

The value of any property that ends with **password** is automatically encrypted at startup.

> **Note:** The remote installations of DevTest Workstation, coordinator, simulator server, VSE, or any other remote DevTest component do not require more configuration. They all receive **site.properties** from the registry when they connect and configure their database access accordingly.

Follow these steps:

1. In the **LISA_HOME** directory, locate the **_site.properties** file.
2. Change the file name to **site.properties**.
3. Open the **site.properties** file.
4. Configure the database configuration properties.
   You typically update the following properties:
   - lisadb.pool.common.url
   - lisadb.pool.common.user
   - lisadb.pool.common.password
5. The schema is automatically created in the database when the registry starts for the first time. However, if you do not want the DevTest user to have DBA privileges, you can manually create the schema beforehand. The **sqlserver.ddl** file in the **LISA_HOME\database** directory contains SQL statements that can serve as the basis for creating the reporting tables and indexes.
6. Ensure that the **lisadb.internal.enabled** property is commented out or set to false.
7. Save the **site.properties** file.
8. (Optional) For trusted connections, download **sqljdbc_auth.dll** from Microsoft and put it in $LISA_HOME\lib\native. To use a trusted connection, you must include *integratedSecurity=true* in the lisadb.pool.common.url URL.

9. Start the registry.

External Java Agent Database Configuration

You can configure the DevTest Java Agent to use an external database by editing the **site.properties** file in the **LISA_HOME** directory.

- Scenario 1: Same Schema for Registry and Agent
- Scenario 2: Different Schema for Registry and Agent
- DDL Statements for Creating and Deleting the Schema

Scenario 1: Same Schema for Registry and Agent

If you want the registry and the agent to use the same database schema, follow the instructions in External Registry Database Configuration and the applicable subtopic.

Scenario 2: Different Schema for Registry and Agent

If you want the registry and the agent to use different database schemas, follow the instructions in External Registry Database Configuration and the applicable subtopic with the following modifications.

In the **site.properties** file, change the value of the **lisadb.broker.poolName** property from **common** to a new value. For example:

```
lisadb.broker.poolName=cai
```

When you configure the database configuration properties for your database, use the new pool name. For example:

```
lisadb.pool.cai.driverClass=com.microsoft.sqlserver.jdbc.SQLServerDriver
lisadb.pool.cai.url=jdbc:sqlserver://localhost:1433;databaseName=cai
lisadb.pool.cai.user=sa
lisadb.pool.cai.password=xyz
```

DDL Statements for Creating and Deleting the Schema

If you want to manually create the agent schema, obtain the DDL statements from one of the following files in the **LISA_HOME\database** directory:

- **db2_cai.ddl**
- **derby_cai.ddl**
- **mysql_cai.ddl**
- **oracle_cai.ddl**
- **sqlserver_cai.ddl**

The **LISA_HOME\database\dropSchema** directory contains the **drop_cai.ddl** file. This file can be useful if you need to revert the schema.

External Enterprise Dashboard Database Configuration

You can configure Enterprise Dashboard to use an external database by editing the **dradis.properties** file in the **LISA_HOME** directory.

> **Note:** For information about database system requirements, see System Requirements .

The **_dradis.properties** file includes properties for each of the supported databases:

- IBM DB2
- Derby
- MySQL
- Oracle
- Microsoft SQL Server

The **LISA_HOME\database\dropSchema** directory contains the following files for the Enterprise Dashboard database:

- **db2_drop_enterprisedashboard.ddl**
- **derby_drop_enterprisedashboard.ddl**
- **mysql_drop_enterprisedashboard.ddl**
- **oracle_drop_enterprisedashboard.ddl**
- **sqlserver_drop_enterprisedashboard.ddl**

These files can be useful if you need to revert the schema.

The following video shows how to configure Enterprise Dashboard to use an Oracle database. You can use the approach that is presented to configure the other supported databases.

Configure Enterprise Dashboard to Use DB2

This page describes how to configure Enterprise Dashboard to use an IBM DB2 database.

In the following procedure, you configure the DB2 version of the **dradis.properties** file. The following properties in this file are relevant to the database configuration:

```
dradis.db.driverClass=com.ibm.db2.jcc.DB2Driver
dradis.db.url=jdbc:db2://[HOSTNAME]:[PORT]/[DATABASENAME]
dradis.db.user=[USER]
dradis.db.password=[PASSWORD]
```

The value of any property that ends with *password* is automatically encrypted at startup.

Follow these steps:

1. Ensure that the code page of the DB2 database is 1208.
2. Ensure that the page size of the DB2 database is at least 16 KB.
3. In the **LISA_HOME** directory, locate the **_dradis.properties** file.
4. Change the file name to **dradis.properties**.
5. Open the **dradis.properties** file.
6. Configure the database configuration properties.
   You typically update the following properties:
   - dradis.db.url
   - dradis.db.user
   - dradis.db.password

   The schema is automatically created in the database when the registry starts for the first time. However, if you do not want the Enterprise Dashboard user to have DBA privileges, you can manually create the schema beforehand. The **db2_enterprisedashboard.ddl** file in the **LISA_HOME\database** directory contains SQL statements that can serve as the basis for creating the Enterprise Dashboard tables and indexes.
7. Set the **dradis.db.internal.enabled** property to false.
8. Start Enterprise Dashboard.

Configure Enterprise Dashboard to Use MySQL

This page describes how to configure Enterprise Dashboard to use a MySQL database.

In the following procedure, you configure the MySQL version of the **dradis.properties** file. The following properties in this file are relevant to the database configuration:

```
dradis.db.driverClass=com.mysql.jdbc.Driver
dradis.db.url=jdbc:mysql://[DBHOST]:[DBPORT]/[DBNAME]
dradis.db.user=[USER]
dradis.db.password=[PASSWORD]
```

The MySQL database must provide collation and characters set supporting UTF-8. The default code page for the database has to be UTF-8; it is not enough only to define your database as UTF-8. If the MySQL database is not UTF-8, then runtime errors occur.

The value of any property that ends with *password* is automatically encrypted at startup.

Follow these steps:

1. In the LISA_HOME directory, locate the _dradis.properties file.
2. Change the file name to **dradis.properties**.
3. Open the **dradis.properties** file.
4. Configure the database configuration properties.
   You typically update the following properties:

- dradis.db.url
- dradis.db.user
- dradis.db.password

The schema is automatically created in the database when the registry starts for the first time. However, if you do not want the Enterprise Dashboard user to have DBA privileges, you can manually create the schema beforehand. The **mysql_enterprisedashboard.ddl** file in the **LISA_HOME\database** directory contains SQL statements that can serve as the basis for creating the Enterprise Dashboard tables and indexes.

5. Set the **dradis.db.internal.enabled** property to *false*.
6. Add the MySQL JDBC driver to the **LISA_HOME\lib\dradis** directory. The minimum version of the MySQL JDBC driver is 5.1.25.
7. Start Enterprise Dashboard.

For more information, see Tech Tips: Using SQL Server with Windows Authentication for DevTest Solutions.

Configure Enterprise Dashboard to Use Oracle

This page describes how to configure Enterprise Dashboard to use an Oracle database.

In the following procedure, you configure the Oracle version of the **dradis.properties** file. The following properties in this file are relevant to the database configuration:

```
dradis.db.driverClass=oracle.jdbc.driver.OracleDriver
dradis.db.url=jdbc:oracle:thin:@[HOST]:1521:[SID]
dradis.db.user=[USER]
dradis.db.password=[PASSWORD]
```

The value of any property that ends with *password* is automatically encrypted at startup.

Follow these steps:

1. Ensure that the character set of the Oracle database supports Unicode. In addition, for the initial creation of the database, the Oracle user must have the **CREATE VIEW** system privilege. Without this privilege, the following error might be generated during installation: **Schema creation failed: ORA-01031: insufficient privileges**.
2. In the **LISA_HOME** directory, locate the **_dradis.properties** file.
3. Change the file name to **dradis.properties**.
4. Open the **dradis.properties** file.
5. Configure the database configuration properties.
   You typically update the following properties:
   - dradis.db.url

- dradis.db.user
- dradis.db.password

For the **dradis.db.url** property, you can use the service name.
The schema is automatically created in the database when Enterprise Dashboard starts for the first time. However, if you do not want the Enterprise Dashboard user to have DBA privileges, you can manually create the schema beforehand. The **oracle_enterprisedashboard.ddl** file in the **LISA_HOME\database** directory contains SQL statements that can serve as the basis for creating the Enterprise Dashboard tables and indexes.

6. Set the **dradis.db.internal.enabled** property to *false*.
7. Ensure that **LISA_HOME\lib\dradis** contains the JDBC driver. In most cases, **ojdbc7.jar** is the correct driver for Java 1.7. If you are using Java 1.6, then use **ojdbc6.jar**. If you are using Java 1.5, then use **ojdbc5.jar**.
8. Start Enterprise Dashboard.

Configure Enterprise Dashboard to Use SQL Server

This page describes how to configure Enterprise Dashboard to use a Microsoft SQL Server database.

In the following procedure, you configure the SQL Server version of the **dradis.properties** file. The following properties in this file are relevant to the database configuration:

```
dradis.db.driverClass=com.microsoft.sqlserver.jdbc.SQLServerDriver
dradis.db.url=jdbc:sqlserver://[SERVER]:[PORT];databaseName=[DATABASENAME]
dradis.db.user=[USER]
dradis.db.password=[PASSWORD]
```

The value of any property that ends with *password* is automatically encrypted at startup.

Follow these steps:

1. In the LISA_HOME directory, locate the _dradis.properties file.
2. Change the file name to **dradis.properties**.
3. Open the **dradis.properties** file.
4. Configure the database configuration properties.
   You typically update the following properties:
   - dradis.db.url
   - dradis.db.user
   - dradis.db.password

   The schema is automatically created in the database when the registry starts for the first time. However, if you do not want the Enterprise Dashboard user to have DBA privileges, manually create the schema beforehand. The **sqlserver_enterprisedashboard.ddl** file in the **LISA_HOME\database** directory contains SQL statements that can serve as the basis for creating the Enterprise Dashboard tables and indexes.
5. Set the **dradis.db.internal.enabled** property to *false*.

6. (Optional) For trusted connections, download **sqljdbc_auth.dll** from Microsoft and put it in $LISA_HOME\lib\native. To use a trusted connection, you must include *integratedSecurity=true* in the dradis.db.url URL.

7. Start Enterprise Dashboard.

Database Maintenance

Various processes are available for maintenance of the DevTest database.

- Automatic Reporting Maintenance
- Automatic Deletion of Audit Log Entries
- Automatic Deletion of Transactions
- Automatic Deletion of Tickets

Automatic Reporting Maintenance

The following background processes affect reporting:

- Performance Summary Calculator

- Report Cleaner

Performance Summary Calculator

This process calculates the statistics for a test or suite run. Many of the graphs in the reporting portal depend on this process, so it cannot be disabled. You can vary the scheduling of the performance summary calculator by changing the following properties:

- **rpt.summary.initDelayMin=7**
  Determines how many minutes to wait after starting the registry before starting this process.

- **rpt.summary.pulseMin=1**
  Determines how long to wait between runs of the process.

Report Cleaner

This process is responsible for deleting "expired" report runs from the database.

- **perfmgr.rvwiz.whatrpt.autoExpire=true**
  You can enable or disable the report cleaner by setting this property. If disabled, the report cleaner process does not run, and the remaining properties have no effect.

- **perfmgr.rvwiz.whatrpt.expireTimer=30d**
  Sets the expiration value of a suite or test. Once a suite or test is older than this value, it is deleted. The property value is a number followed by a suffix. The following suffixes are valid, with **t** being the default:

  - t=milliseconds

  - s=second

  - m=minutes

- h=hours
- d=days
- w=weeks

Keep in mind that tests and suites are deleted at approximately the same time of day that they were created. Adding several hours can minimize delete processes running during the day. However, there is no absolute way to ensure the times that a delete process runs.

- **perfmgr.rvwiz.whatrpt.forceCompleteTimer=24h**
  To force the completion of a test (in the reporting database). Some suites or tests that fail to finish cleanly are never marked as "finished" in the database. Tests that are not marked "finished" are not deleted by the report cleaner and do not have performance summary calculations performed. Any test that is older than this value is marked as completed in the database. The completed tests are only removed from the database after the expireTimer has also been accounted for. If you have tests that run longer than 24 hours by default, increase the value for this process.

You can vary the scheduling of the report cleaner by changing the following properties:

- **rpt.cleaner.initDelayMin=10**
  Determines how many minutes to wait after starting the registry before starting this process.
- **rpt.cleaner.pulseMin=60**
  Determines how long to wait between runs of the process.

Automatic Deletion of Audit Log Entries

The access control (ACL) feature stores the audit log in the database. On a periodic basis, the registry runs a process to delete old audit log entries from the database.

The following properties control this behavior:

- **lisa.acl.audit.logs.delete.frequency**
  Specifies how often to run the automatic deletion process. The default value is **1d**, which means that the process runs once a day. The valid units of time are d, h, m, and s (for days, hours, minutes, and seconds).
- **lisa.acl.audit.logs.delete.age**
  Specifies the minimum age of audit log entries that are deleted by the automatic deletion process. The default value is 30d, which means that entries are considered to be old after 30 days. The valid units of time are d, h, m, and s (for days, hours, minutes, and seconds).

The default value of each property is located in the **lisa.properties** file. To change the default value, add the property to the **local.properties** file.

Automatic Deletion of Transactions

You can configure Application Insight to delete old transactions automatically from the database.

By default, the automatic deletion of transactions is enabled.

The following broker properties are available:

- **Enable cleaner**
  Controls whether the automatic deletion of transactions is enabled.

- **Cleanup frequency**
  Specifies how often the cleaner process runs. The value is in number of minutes.

- **Maximum age**
  Specifies the age of transactions that the cleaner process deletes. The value is in number of minutes.

You can configure these properties from the Agents window of DevTest Portal. The properties appear in the **Settings** tab.

Automatic Deletion of Tickets

You can configure Application Insight to delete old tickets automatically from the database.

By default, the automatic deletion of tickets is enabled.

The following properties are available:

- **Enable cleaner**
  Controls whether the automatic deletion of tickets is enabled.

- **Cleanup frequency**
  Specifies how often the cleaner process runs. The value is in number of minutes.

- **Maximum age**
  Specifies the age of tickets that the cleaner process deletes. The value is in number of minutes.

You can configure these properties from the Agents window of the DevTest Portal. The properties appear in the **Settings** tab.

License Administration

This page gives an overview of transaction consumption based licensing, the license requirements and how to obtain a new license for DevTest 10.4 or later.

- License Requirements
- How to Request a New License File
- Track Reporting and Consumption

Beginning with version 10.3, DevTest Solutions has a new transaction consumption licensing model.

License Requirements

Upgrading to DevTest 10.4

- To upgrade to DevTest 10.4, you must request a new license file.

- A new license file does not change your existing licensing model.

Renew a Current License

When you renew your current current license, the new licensing model is applied.

New Customers

The new licensing model is applied that tracks transaction consumption for new customers.

How to Request a New License File

To use DevTest version 10.4, you must request a new license. There are several scenarios when you will be required to request a new license.

- To upgrade from DevTest 10.3 or earlier.
- Your license file is expired.
- You want to install multiple Enterprise Dashboards with pre-defined Mac addresses.

To request a new license file, please follow the steps below:

1. Collect the Mac address of the machine where you will be installing the Enterprise Dashboard.
2. Identify the name and email address to send the license file.
3. Contact the Customer Assistance Team to request a new license in one of the following ways:
   - Call the Customer Care Hotline at **(800) 225-5224**.
     a. Select **Option 2**, Customer Care.
     b. Select **Option 1**, License Keys.
   - Fill out a Generic Customer Care Web Form online:
     a. Fill out the Customer Care form.
     b. Select **CA Licensing** as your request type.
   - Go to CA Support.
     a. Click **Sign In** and enter your credentials.
     b. Click **Case Management**.
     c. Select **Licensing** in the **Product** field.
     d. Complete the rest of the form and click **Submit**.
4. The Customer Assistance Team will create a new license file (**devtestlic.xml**) and send it to the email address identified in step 2.

You can download DevTest Solutions directly from the Download Management section of the CA Support site.

Track Reporting and Consumption

Licensing reporting and consumption are shown on the The Enterprise Dashboard.For more information about how to track consumption of your transactions see: Features and Functionality of the Enterprise Dashboard

DevTest Solutions Usage Audit Report

Your contract determines your licensing model. If you are using a license model that is based on power users rather than transaction consumption, the DevTest Solutions Usage Audit Report provides details on compliance with your license agreement. For more information about usage data for a license model that is based on transaction consumption, see Features and Functionality of the Enterprise Dashboard.

A DevTest Solutions Usage Audit Report is generated as an Excel workbook with the following tabs:

- Overview
- User Type Chart
- SV for Performance Chart
- Historical Usage
- Component by User

Overview

The Overview tab provides details on the extent an enterprise is using the number of concurrent users of each user type that their license permits. A given report provides data from all registries for the specified date range. The Count data is shown by user type.

	A	B	C
1	DevTest Solutions	Usage Audit Report	
2	Company:	AppDelBU	
3	License Key ID:	1fbe856	
4	License Key Expiration:	December 08, 2016	
5	Enterprise Dashboard Product Version	9.5.0	
6	Reporting Period Start:	February 14, 2016	
7	Reporting Period End:	March 15, 2016	
8	Generated On:	March 22, 2016	
9	User Type	Maximum Concurrent Use Counts	Instances of Max Use Count
10	Continuous Application Insight Power User	2	1
11	Service Virtualization Power User	2	2
12	Application Test Power User	3	1
13	DevTest Runtime User	2	1
14	Maximum Concurrent Instances	Maximum Concurrent Use Counts	Instances of Max Use Count
15	Service Virtualization for Performance	2	1
16			
17	Usage Audit Documentation		
18			
19			

Usage Audit Report Overview Tab

- **Report Information**
  The report information section lists licensing information and the date range for this audit report.
  - **Company**: Name of the company that generated this report
  - **License Key ID**: The license key identifier

- **License Key Expiration**: The date the license expires
- **Enterprise Dashboard Product Version**: The version of DevTest
- **Reporting Period Start**: The starting date for this report
- **Reporting Period End**: The ending date for this report
- **Generated On**: The date on which the report was generated

- **User Type**
  Reported user types include Continuous Application Insight Power User, Service Virtualization Power User, Application Test Power User, and DevTest Runtime User.
  If the user is granted permissions that are associated with more than one user type, the higher user type is used. The user type hierarchy, from highest to lowest, follows:

  1. a. Continuous Application Insight Power User or Service Virtualization Power User (equivalent user types)

     b. Application Test Power User

     c. DevTest Runtime User
     See User Types.

- **Maximum Concurrent Use Counts**
  All registries continuously collect usage data from the services that support the UIs and CLIs. Raw data is kept for concurrent usage by the same user type across the enterprise, where the concurrent login sessions are recorded by registry. Concurrent usage occurs when two or more users of the same user type access a DevTest UI or DevTest CLI where their sessions overlap in time. In this sample report, the highest concurrent usage was at least once where 3 users with a user type of Application Test Power User were logged on simultaneously.

- **Instances of Max Use Count**
  The instance calculations for the report are based on group counts where the maximum concurrency count is reached. In this sample report, there were two points in time between February 14 and March 15, 2016, where two Service Virtualization Power Users were logged in simultaneously. See the Count Calculation Example.

- **Maximum Concurrent Instances**
  Counts concurrent VSEs that have the VSE Performance option enabled. If the VSE Performance option is not enabled, the count is 0. In this sample report, between February 14 and March 15, 2016, there was one time when two VSEs with the Performance option enabled were running simultaneously.

The **Usage Audit Documentation** link at the bottom of the report links to this page of the product documentation.

User Type Chart

The User Type chart shows, by date, the concurrent usage by user type of the product during the time period for which the report was requested. The chart is derived from the data on the Historical Usage tab.

Usage Audit Report User Type Chart

To customize the chart, click on it.

To customize the chart style and color, click the paintbrush icon.

To drill into a specific usage type, click the funnel icon and select a component.

For more information about customizing charts, see Microsoft Excel Help.

SV for Performance Chart

The SV for Performance Usage: Summary chart shows, by date, the maximum concurrent usage of SV for Performance during the time period for which the report was requested.

Usage Audit Report SV for Performance Chart

To customize the chart, click on it.

To customize the chart style and color, click the paintbrush icon.

To drill into a specific usage type, click the funnel icon and select a component.

For more information about customizing charts, see Microsoft Excel Help.

Historical Usage Tab

The Historical Usage report shows, by date, the highest concurrent usage by user type of the product for each day during the time period for which the report was requested. This tab contains the raw data that is shown on the User Type Chart.

Date	SV for Performance	Application Insight Power User	Service Virtualization Power User	Application Test Power User	DevTest Runtime User
Feb 14, 2016	0	0	0	0	0
Feb 15, 2016	0	0	0	0	0
Feb 16, 2016	0	0	0	0	1
Feb 17, 2016	0	0	1	2	0
Feb 18, 2016	2	0	0	0	0
Feb 19, 2016	0	0	0	0	0
Feb 20, 2016	0	0	0	0	0
Feb 21, 2016	0	0	0	1	0
Feb 22, 2016	0	0	0	0	0
Feb 23, 2016	0	0	0	0	0
Feb 24, 2016	0	0	0	0	0
Feb 25, 2016	0	1	1	0	0
Feb 26, 2016	0	0	0	0	0
Feb 27, 2016	0	0	0	0	0
Feb 28, 2016	0	0	0	0	0
Feb 29, 2016	0	0	0	0	0
Mar 01, 2016	0	0	0	0	0
Mar 02, 2016	0	0	1	1	0
Mar 03, 2016	0	0	0	1	0
Mar 04, 2016	1	2	2	2	2
Mar 05, 2016	1	0	0	1	0
Mar 06, 2016	1	0	0	1	0
Mar 07, 2016	1	0	0	3	0
Mar 08, 2016	0	0	0	0	0
Mar 09, 2016	0	0	0	0	0
Mar 10, 2016	0	0	0	0	0
Mar 11, 2016	0	0	0	0	0
Mar 12, 2016	0	0	0	0	0
Mar 13, 2016	0	0	0	0	0

User Audit Report Historical Usage Tab

Component by User

The Component by User tab reports user session data from each registry that is connected to the Enterprise Dashboard, sorted by registry, with a column for each UI or CLI with access during this time period. Entries on this tab include all access, not only concurrent access.

Registry	User Name	User Type	Total Sessions	Console	invoke2-api	Test Runner	Workstation
Registry@lenba0113068-2.ca.com:2010	admin	PF_POWER_USER	1	0	0	0	1
Registry@lenba0113068-2.ca.com:2010	admin	SV_POWER_USER	1	0	0	0	1
Registry@lenba0113068-2.ca.com:2010	tpower	TEST_POWER_USER	2	0	0	0	2
Registry@lenba0113933-2.ca.com:2010	admin	PF_POWER_USER	1	0	0	0	1
Registry@lenba0113933-2.ca.com:2010	admin	SV_POWER_USER	1	0	0	0	1
Registry@LENBA01.ca.com:2010	tpower	TEST_POWER_USER	1	0	0	0	1
Registry@lenba0113638-2.ca.com:2010	admin	ADMIN_USER	3	3	0	0	0
Registry@lenba0113638-2.ca.com:2010	barb	ADMIN_USER	4	4	0	0	0
Registry@lenba0113638-2.ca.com:2010	barb	PF_POWER_USER	6	0	2	0	4
Registry@lenba0113638-2.ca.com:2010	barb	SV_POWER_USER	6	0	2	0	4
Registry@lenba0113638-2.ca.com:2010	guest	SVT_RUNTIME_USER	3	0	1	0	2
Registry@lenba0113638-2.ca.com:2010	svpower	ADMIN_USER	5	5	0	0	0
Registry@lenba0113638-2.ca.com:2010	svpower	SV_POWER_USER	6	5	0	0	1
Registry@lenba0113638-2.ca.com:2010	tpower	ADMIN_USER	3	3	0	0	0
Registry@lenba0113638-2.ca.com:2010	tpower	TEST_POWER_USER	14	3	3	2	6

Usage Audit Report Component by User Tab

- **Registry**
  Registry names are in the format:

  `registry@FQDN:2010`

- **User Name**
  Names of users who logged in to a user interface or command-line interface for one or more sessions.

- **User Type**
  The user type that is assigned to the associated user.

- **Total Sessions**
  The sum of counts from the reported columns for the associated user, where the columns represent either a UI session or a session using a CLI. User interfaces include DevTest Workstation. Examples of CLIs include the DevTest Portal, the REST Invoke API and Test Runner. Columns may vary by report, depending on where users log in. Some examples follow:

- - invoke2-api: The number of times this user initiated the DevTest Invoke API.
  - Workstation: The number of times this user opened a DevTest Workstation, connected to the associated registry, and logged in.

Note: For details on generating this report, see Export Usage Audit Data.

Count Calculation Example

Calculating use counts for the DevTest Solutions Usage Audit Report is a two-step process.

1. Each time a concurrency is detected across the enterprise, a total is recorded for all involved registries. Consider the following example for the Application Test Power User user group.

   In this example, there were totals recorded at five different times for three different registries. The largest number of concurrent logins by Application Test Power users was 3 and is highlighted below.
   For reporting purposes, the totals with maximum concurrency are used. In this sample example, row 1 is used, because its total concurrent users is the peak count for the selected time interval.

Application Test Power User					
Row	**Date**	**Registry 1**	**Registry 2**	**Registry 3**	**Total**
1	***10/07/2016***	***1***	***0***	***2***	***3***
2	10/08/2016	0	0	2	2
3	10/09/2016	0	2	0	2
4	10/10/2016	0	0	1	1
5	10/11/2016	1	1	0	2

2. This data is summarized on the Overview page of the report as User Type, Maximum Concurrent Use Counts, and Instances of Max Use Count. The count represents the largest total for that user type. The instances of max use count represent the number of times over the time period that this maximum concurrency occurred. For the example above, the instance count would be 1 (the number of times that the Application Test Power Users reached its maximum concurrent use count of 3).

User Type	Maximum Concurrent Use Count	Instances of Max Use Count
Application Test Power User	**3**	1

User Types

ACL has the following user types:

- SV Power User
- CAI Power User
- Test Power User
- Runtime User

A user type can be associated with one or more roles. Each role is associated with only one user type. For example, the Runtime User user type has three roles (System Administrator, Runtime, and Guest), but the Runtime role is associated with only the Runtime User user type.

A user can be granted one or more roles. When granted multiple roles, the role that is associated with the highest user type is used for usage auditing purposes. The user type hierarchy is made up of the following user types:

- CAI Power User and SV Power User are equal in the hierarchy and both are the highest user type.
- Test Power User is lower than CAI Power User and SV Power User but higher than Runtime User.
- Runtime User is the lowest user type.

Example of How the User Type Is Determined for Auditing Purposes

The permissions that are associated with a role assigned to a user determine the tasks that a user is authorized to perform. An administrator can assign one or more roles to a user. Typically, administrators assign a single role to each user because the permissions for the built-in roles are assigned with the user type hierarchy in mind. For example, the PF Power role is also assigned permissions that are associated with Test Power and Runtime.

To grant users permissions that are not part of the higher role, you can assign them multiple roles.

Consider an example where a user is assigned the following roles:

- Test Administrator
- Runtime

Different user types are associated with the assigned roles:

- The Test Administrator role is associated with the SV Power User user type.
- The Runtime role is associated with the Runtime User user type.

> **Note:** See Standard User Types and Standard Roles for details.

User types are hierarchical.

For purposes of the DevTest Solutions Usage Audit Report, the user type under which a user is counted is the highest user type that is associated with an assigned role. For example, when the *fakeName* user logs in to a DevTest UI or CLI, the user session is audited as belonging to the SV Power User user type, even if *fakeName* performs only tasks that are associated with the Runtime role, such as report administration.

User Type Inheritance Chart

The following diagram shows all cases of how the user type of a logged in user is determined if the user is assigned multiple roles from different user types. The "highest" user type takes precedence over all lower user types.

Usage Audit Report FAQs

This section addresses some of the Frequently Asked Questions about the Usage Audit Report and license administration.

- Licensing
- Account Sharing
- Reporting
- Administration

Licensing

- **What is concurrent licensing?**

"Concurrent User" means a software license that is based on the number of simultaneous (concurrent) users accessing the program. DevTest does not prohibit more users from access, but does log license usage for auditing purposes. You are entitled to have the specified number of Concurrent Users access the software simultaneously. For licensing purposes, each Concurrent User is entitled to have one active session that is connected to the server either directly or

indirectly. Each active session that an individual, device, or process initiates is counted as a Concurrent User. An active session means that either a workstation or Web browser is connected to the registry server, DevTest Portal, or both, or a custom application session accessing a virtualized service.

- **What types of licenses and how many licenses am I licensed for?**

Contact your account team for questions about your licensing agreement.

- **How many licenses are consumed when I use DevTest Workstation and DevTest Portal on the same machine?**

If you are using both DevTest Workstation and DevTest Portal on the same machine, one concurrent usage count is logged. The same is true for any combination of UIs and CLIs, on the same or on different machines.

- **Does my license limit the number of** Service Virtualization for Performance **instances that I can run? Is that also honor-based?**

You enable the Performance Option for VSE with the VSE Manager command or by setting a property. This number is also honor-based.

Account Sharing

- **How are concurrent licenses calculated when we share a user: for example, *admin*?**

If more than one user uses the same user ID simultaneously on different machines, each is counted toward the concurrent use. If a user uses the same user ID to access different DevTest applications on the same machine, it is counted as a single use .

For example, if you have six users who are logged on to DevTest Workstation with the *admin* user ID, the Usage Report shows a concurrent use count of 6 for the user type that is associated with the *admin* user ID. Note: We do not recommend sharing of user IDs.

To prevent users from sharing user IDs, we recommend connecting DevTest to your AD/LDAP datasource. DevTest can enable AD/LDAP authentication. DevTest continues to provide authorization. This technique does not bind a user ID or login to a DevTest Workstation . This technique does, however, provide some control because users are less likely to share their AD /LDAP credentials.

Reporting

- **What type of licenses and how many concurrent licenses am I consuming?**

To get this information, run a DevTest Solutions Usage Audit Report by exporting usage audit data from the Enterprise Dashboard to Excel. See the Overview tab to see the maximum concurrent sessions that are used by each license type for the time period you choose.

- **How do we calculate *runtime* users? Are there limitations, for example *runtime* users that do not log into DevTest?**

The license counts are all based on login and logout records.

Every DevTest user login creates a user session and increments the license count by one. Every DevTest user logout (including implicit logouts through timeouts) closes a user session and decrements the license count by one.

Note: Anything that connects to the virtual services that are deployed on a VSE server consumes a license, but these connections cannot be tracked in the usage audit report because they do not log in to DevTest.

- **How are concurrent licenses calculated when I'm logged into one DevTest application (for example, DevTest Workstation) on one machine and logged into another DevTest application (for example, DevTest Portal) on a different machine?**

In this example, because you are using two separate machines, you would consume two licenses. The license type that is consumed is based on the actions that are performed during the user session.

- **How frequently is usage polled?**

The information is collected continuously based on login and logout times. When an audit report is requested, the data is examined to determine when the concurrent access occurs.

- **If a user is logged in, but is not performing any activity, is that user counted in the usage count?**

Yes.

- **What about command-line interfaces like VSE Manager? Are those users counted in the usage count?**

Yes. Any utility that requires credentials is included in the usage count.

- **I am licensed for 10 SV Power Users, but I see as many as 15 concurrent users at one time. Is there a problem with my license?**

No, licensing for DevTest Solutions is honor-based. Use the Usage Audit Report to monitor your compliance with the terms of your license.

- **My Component by User report shows a user type for ADMIN_USER. What is that user type and do I need to be concerned about licensing for it?**

While the ADMIN_USER type is included in the report, you do not need to consider it when evaluating your compliance to your licensing agreement. The ADMIN_USER user type, which is displayed on the Roles page as a combination CAI Power User and SV Power User types, is included in the DevTest Solutions Usage Audit Report for your information, and is counted as an SV_POWER user for licensing purposes.

- **The Overview tab indicates that I have 12 Concurrent users. When I click on the Component by User tab, I only see four different user IDs. Can you explain why we need to get licenses for more Power Users when we see only four different user IDs using the product?**

On the Overview tab, you see that CAI Power User and Service Virtualization Power User have each had 12 concurrent users during the reporting period, once for CAI Power User and twice for SV Power User.

DevTest Solutions	Usage Audit Report	
Company:	Forward, Inc.	
License Key ID:	1fbe856	
License Key Expiration:	December 08, 2016	
Enterprise Dashboard Product Version	9.5.0	
Reporting Period Start:	February 14, 2016	
Reporting Period End:	March 1, 2016	
Generated On:	March 15, 2016	
User Type	Maximum Concurrent Use Counts	Instances of Max Use Count
Continuous Application Insight Power User	12	1
Service Virtualization Power User	12	2
Application Test Power User	0	0
DevTest Runtime User	0	0
Maximum Concurrent Instances	Maximum Concurrent Use Counts	Instances of Max Use Count
Service Virtualization for Performance	2	1
Usage Audit Documentation		

DevTest Usage Audit Report

On the Historical Usage tab, you can see when those high concurrent usage days occurred: for CAI Power User, on March 7, and for SV Power User, on March 5 and March 13.

Date	SV for Performance	Continuous Application Insight Power User	Service Virtualization Power User	Application Test Power User	DevTest Runtime User
Mar 01, 2016	0	0	0	0	0
Mar 02, 2016	0	0	1	0	0
Mar 03, 2016	0	0	0	0	0
Mar 04, 2016	1	2	2	0	0
Mar 05, 2016	1	0	12	0	0
Mar 06, 2016	1	0	0	0	0
Mar 07, 2016	1	12	3	0	0
Mar 08, 2016	0	0	0	0	0
Mar 09, 2016	0	0	0	0	0
Mar 10, 2016	0	0	0	0	0
Mar 11, 2016	0	0	0	0	0
Mar 12, 2016	0	0	0	0	0
Mar 13, 2016	0	0	12	0	0
Mar 14, 2016	0	1	1	0	0
Mar 15, 2016	0	0	0	0	0

Historical Usage Tab

The Component by User tab does not reflect any information about concurrent usage. It does indicate that in this example, there are only four different users who have used the product during this time period: *admin*, *barb*, *rob*, and *svpower*. They have used various interfaces to connect to three different registries.

Registry	User Name	User Type	Total Sessions	Console	invoke2-api	Test Runner	Workstation
Registry@forward13068-2.fw.com:2010	admin	CAI_POWER_USER	1	0	0	0	1
Registry@forward13068-2.fw.com:2010	admin	SV_POWER_USER	56	16	5	34	1
Registry@forward13933-2.fw.com:2010	rob	CAI_POWER_USER	10	9	0	0	1
Registry@forward13933-2.fw.com:2010	admin	SV_POWER_USER	1	0	0	0	1
Registry@forward13638-2.fw.com:2010	barb	CAI_POWER_USER	36	7	16	9	4
Registry@forward13638-2.fw.com:2010	barb	SV_POWER_USER	38	32	2	0	4
Registry@forward13638-2.fw.com:2010	svpower	SV_POWER_USER	14	5	8	0	1

Component by User Tab

In this example, we can tell that users at this company are sharing user IDs, thus accounting for the concurrent usage that is higher than the number of user IDs. To track product usage accurately, we recommend that your users do not share user IDs.

Administration

- **Where can I specify timeout values for DevTest Portal, so that users who are not active will time out and not be counted in the Usage Audit Report?**

This property specifies how many seconds the SSO security token is kept in memory to bypass authentication for DevTest Portal.

The **registry.max.user.lifetime.seconds** property is the main session lifetime property that determines how long a user session is valid after the last activity done by a user in a session.

- **Can we implement ACL so that a user can log in from only one DevTest Workstation instance and thus can occupy only one license per user ID?**

No , once a user ID is given access, DevTest does not perform actions that "bind" a given user ID to a DevTest Workstation IP address. User IDs are shown in the Usage Audit report.

- **When I perform an action (for example, log in to DevTest Workstation) how can I tell which license type is consumed?**

Each permission (listed in the Role UI, right-hand side) has a license type that is associated with it, identified within parentheses.

- **If a user has multiple user types, how can I determine the effective user type for a new or existing role?**

You can view or change the permissions and essentially update the effective user type of a role, within access control.

- **How does the Usage Audit Report differ from the User Activity Report?**
  The User Activity Report is useful to show an administrator what users are performing which activities in real time.

Security

This section contains the following pages that describe security options in DevTest :

- Identity and Access Manager
- Access Control (ACL)
- Using SSL to Secure Communication
- SSL/TLS Protocol Configuration
- Using HTTPS Communication with the invoke APIs
- Using Kerberos Authentication

Identity and Access Manager

Identity and Access Manager is the component that provides user authentication for all DevTest components. Use Identity and Access Manager to do the following:

- Manage users and roles
- Manage sessions and events
- Define realm settings

- Federate external user databases (LDAP)

> **Note:** Use Identity and Access Manager to create and manage users and roles. Use Access Control (ACL) to assign permissions to individual roles.

Before you configure Identity and Access Manager, carefully consider how each user will use DevTest and the corresponding access that is required for each type of user. By default, only the admin role has administrative access to Identity and Access Manager. For more information about the relationships between Identity and Access Manager, Enterprise Dashboard, and the registry, see DevTest Process Relationships.

- IAM Administration
- Using the Lightweight Directory Access Protocol (LDAP) with DevTest Solutions.
- Authentication
- Authorization
- User Sessions
- IAM Database
- Define Realm Settings
- Add SSL Certificates to the Trust Store
- IAM and JAVA_HOME

IAM Administration

The Identity and Access Manager Administrator is responsible for the following activities:

- Creating and managing users in IAM.
- Creating and managing roles in IAM.
- Assigning roles to users, based on the responsibilities and required access of each user.

The Administrator must have a thorough understanding of the various ACL roles and their associated privileges to assign the appropriate roles to each user.

> **Important!** Do not use the default admin user to manage IAM. Use the default admin user to create new users with identical roles to the default admin. Use the new user for managing IAM, and change the passwords for the default admin user to prevent unauthorized access.

Using the Lightweight Directory Access Protocol (LDAP) with DevTest Solutions.

You can also choose to manage users and passwords through LDAP, especially if an LDAP or Active Directory system is already available. When you use LDAP, user password changes are made by the LDAP administrator. The IAM administrator is no longer able to perform password changes. For more information, see Configure User Federation (LDAP).

Important! The implementation of IAM, and any integration with an LDAP service, remain the responsibility of the customer. If you need assistance with these implementation activities, contact CA Services. User administration through IAM is the responsibility of the IAM or LDAP administrator. CA Support is unable to progress cases where view access to the roles table is not available.

For example, a customer that reports a problem staging a test due to permission issues must ensure that the IAM or LDAP administrator is available when engaging CA support. IAM administrators must take these factors into account when assigning roles to their users and groups.

The following video demonstrates using Identity and Access Manager with LDAP.

Authentication

You determine how DevTest authenticates users.

- You can manually add users to the IAM database and specify credentials. The credentials are the user ID and password used to log in to the DevTest user interface or command-line interface.
- If your users are already defined with credentials in an LDAP database, you can use the LDAP server for authentication.

Note: DevTest lets you limit the features that individual users can access by the assignment of roles. The roles are created and assigned to users in IAM, but the permissions for each role are defined in ACL.

Authorization

DevTest limits what features individual users can access based on their business role. DevTest is installed with more than a dozen standard roles. The roles are created and assigned to users in Identity and Access Manager, but the permissions are assigned to each role in ACL.

When you manually add users to the IAM database, you assign a role to each user. The role grants a set of permissions. It is possible to assign multiple roles, but is rarely necessary as roles with more responsibility include permissions from lower related roles. When you use LDAP, IAM is automatically populated with a row for each user.

The following activities are examples of the activities that you can control with permissions:

- Create a test case
- Create a staging document
- Stage a test case

User Sessions

When an authorized user logs in to a DevTest UI or CLI, a user session is created. User sessions are audited and form the basis of Usage Audit Reports. These reports include metrics and statistics on maximum concurrent user sessions by user type, where user types are categories that include multiple roles.

IAM Database

The IAM data is stored in the default internal Derby database after the installation. As outlined in *Installing*, the Derby database should be replaced with an enterprise database. For more information, see Database Administration.

The following video demonstrates using Identity and Access Manager with Database Users.

Define Realm Settings

Select Configure/Realm Settings on the left navigation bar of Identity and Access Manager.

- The **General** tab displays information about how to get started and a link to this documentation
- The **Tokens** tab displays the available parameters to configure your security realm. Click the gray circles with white question marks beside each field for help about that field.
  - The **SSO Session Idle** parameter lets you set the time a session can remain idle before it expires.
  - The **Access Token Lifespan** parameter lets you set the expiration time for your access tokens.

> The timeout parameters do not affect timeouts for DevTest Workstation and CA Service Virtualization Community Edition.

Add SSL Certificates to the Trust Store

Identity and Access Manager provides a default trust store that you can use to import your SSL certificates.

To add SSL certificates to the trust store and make them available to Identity and Access Manager, follow these steps:

1. Navigate to the **IdentityAccessManager** folder in your installation directory.
2. Run the following command, where:
   - **HOSTDOMAIN** = Any name to identify the certificate.
   - **host-certificate.cer** = The certificate that needs to be imported.

```
keytool -import -alias HOSTDOMAIN -keystore iam-truststore.ks -storepass
passphrase -file host-certificate.cer
```

3. Restart Identity and Access Manager.

IAM and JAVA_HOME

To run Identity and Access Manager in an environment without JAVA_HOME defined, you must modify **standalone.sh** in the **bin** folder. Failure to make this change prevents IAM from starting and results in the following error:

```
=========================================================================
JBoss Bootstrap Environment
JBOSS_HOME: /home/dc-user/CA/DevTest/IdentityAccessManager
JAVA: java
JAVA_OPTS: -server -Djboss.http.port=51112 -Djboss.https.port=51111 -Djboss.bind.
address.management=0.0.0.0 -Djboss.bind.address=0.0.0.0 -Diam.truststore=../iam-
truststore.ks -Diam.truststore.pass=passphrase -Dkeycloak.migration.action=import -
Dkeycloak.migration.provider=iam-updater -Dkeycloak.migration.dir=../realms-to-
import -Dkeycloak.migration.strategy=IGNORE_EXISTING -Diam.version=1.2.0
=========================================================================
```

To correct this error, open **standalone.sh** in the **IdentityAccessManager/bin** folder of your installation directory and make the following change.

```
# Setup the JVM
if [ "x$JAVA" = "x" ]; then
   if [ "x$JAVA_HOME" != "x" ]; then
       JAVA="$JAVA_HOME/bin/java"
   else

JAVA="java"
```

Change **JAVA=”java”** to **JAVA="../../jre/bin/java"**.

More Information:

- Upgrading the Authentication Component
- Manage Roles and Users
- Configure User Federation (LDAP)
- Access Control (ACL)

Start Identity and Access Manager

Note: The host address and port number are user-definable. The defaults are ***localhost*** and **51111**, but you can change the defaults in your lisa.properties file.

Follow these steps:

1. Start Identity and Access Manager by doing one of the following:
   - Double-click the **Identity and Access Manager** desktop icon.

- Launch the **IdentityAccessManager** executable from the ***Install Directory*** **/IdentityAccessManager/bin** folder.

2. Start the Identify and Access Manager UI by doing one of the following:

   - Double-click the **Identity and Access Manager UI** desktop icon.

   - Enter https://localhost:51111/ in a browser.

   > The browser will display a warning that the connection is not secure or that the certificate for the website is invalid. This connection is flagged because it is using a self-signed certificate. Proceed to the site.

   The Identity and Access Manager login page opens.

3. Enter your username and password and click **Login**.

   > The first time you log in, use username of *admin* and password of *admin*. Once logged in, you should then create a new username with admin authority and change the password for the default *admin* username.

   The General tab opens.

Configure User Federation (LDAP)

Identity and Access Manager can connect to existing LDAP servers. Select **User Federation** to configure a provider.

When you add or edit an LDAP configuration, the details of your configuration are stored in a file named **authentication-providers.xml**. When you assign roles to LDAP groups, those role mappings are stored in a file named **ldap-mappings.xml**. If you ever need to access these files directly, both files are both located in your DevTest installation directory.

- Add an LDAP System
- Import Existing DevTest LDAP Authentication
- Define LDAP Group Settings
- Assign Roles to LDAP Groups
- Map LDAP Attributes
- Edit Your LDAP System
- LDAP Configuration Example

Add an LDAP System

Follow these steps:

1. Click **User Federation** under the Configure menu.
   The User Federation page opens.

2. Select **ldap** from the **Add provider...** list at the top of the table.
   The Required Settings page opens. Click the question mark beside each field for additional information about that field.

3. Leave the **Priority** field set to 0.

4. Enter your LDAP values for required fields, which are indicated with a red asterisk.

5. For the **Connection URL** field, use the **Test connection** button to the right of the field to validate your entry.

6. For the **Bind credential** field, use the **Test authentication** button to the right of the field to validate your entry.

7. Use the **Custom User LDAP Filter** field to specify a filter to narrow the results that are returned from your search.

8. Scroll to the bottom of the page and click **Save**.

Import Existing DevTest LDAP Authentication

The Import LDAP Authentication lets you import an existing LDAP configuration from a DevTest registry.

Follow these steps:

1. Click **User Federation** under the Configure menu.
   The User Federation page opens.

2. Click **Import LDAP Authentication** at the top of the table.
   The Required Settings page opens.

3. Click **Choose a File** to select an existing authentication provider XML file.
   The authentication-providers.xml file contains the details for you authentication provider and is located in the DevTest installation directory. This is a required field.

4. Click **Choose a File** to select an existing LDAP mapping XML file.
   The ldap-mappings.xml file assign roles to groups of LDAP users and is located in the DevTest installation directory. This is an optional field.

5. Click **Import**.
   The authentication provider information displays in the User Federation page.

Define LDAP Group Settings

The Group Settings tab lets you define the LDAP groups that you want to pull into Identity and Access Manager.

Follow these steps:

1. Click the **Group Settings** tab.

2. Enter the details for your LDAP group attributes and classes.

3. Click **Save** to save your changes.

4. Click **Sync LDAP Groups To Identity and Access Manager** to make these LDAP groups available for role mapping in IAM.

Assign Roles to LDAP Groups

Follow these steps:

1. Click the **Group(s) to Role Mapping** tab.
2. Select a group from the **LDAP Groups** list.
3. To assign one or more roles to the selected LDAP group:
   a. Click the role that you want to assign in the **Available Roles** column.
   Available Roles are roles that are defined within DevTest, but have not been assigned to this user by an administrator or by LDAP.
   b. Click **Add Selected** to assign the role and move it to the Assigned Roles column.
4. To remove one or more roles from the selected LDAP group:
   a. Click the role that you want to remove in the **Assigned Roles** column.
   Assigned Roles are roles that have been assigned to this user by an Identity and Access Manager administrator.
   b. Click **Remove Selected** to remove the role and move it to the Available Roles column.

For more information about roles see Standard User Types and Standard Roles.

Map LDAP Attributes

Identity and Access Manager provides a set of default attributes that you can map to your LDAP attributes. For example, if your LDAP attribute for username is something other than the default "cn," you can map the username LDAP attribute "username" attribute in IAM.

Follow these steps:

1. Click the **Mappers** tab.
2. Click the attribute that you want to map in the **Name** column.
   The details for that attribute open in a separate page.
3. Make the desired changes.
4. Click **Save**.

Edit Your LDAP System

Follow these steps:

1. Click **User Federation** under the Configure menu.
   The User Federation page opens.
2. Click **Edit** in the **Actions** column.
   The Settings tab opens.

3. Make any desired changes on this tab.
   When LDAP users log into DevTest Solutions, a record is created in the Identity and Access Manager database so that those users display in the Users tab in IAM. Click **Remove Imported** to remove these users from the IAM database.

4. Refer to the sections above for additional details about each of the tabs on this page:

   - Group Settings
   - Group(s) to Role Mapping
   - Mappers

LDAP Configuration Example

Following is an example of one way to configure your LDAP system within Identity and Access Manager.

LDAP Configuration

Manage Roles and Users

Identity and Access Manager administrators can create and manage the users and roles that control access to DevTest Solutions. DevTest automatically installs a variety of default users and roles, but you can also create custom roles and manually add users to the IAM database. For more information about configuring IAM to work with an LDAP server, see Configure User Federation (LDAP).

- Manage Roles
- Manage Users

Manage Roles

Select **Roles** under the **Configure** menu to display the Lookup tab of the Roles page. The list of all available roles displays.

To search for a specific role, enter a full or partial role name or description in the **Search** field.

To add a role:

1. Select **Roles** under the **Configure** menu.
   The Lookup tab opens on the Roles page.

2. Click **Add Role**.
   The Add Role page opens.

3. Enter the name of the new role in the **Role Name** field.

4. Enter a description for the role.

5. Click **Save**.
   The Add Role page closes, and the new role is added to the Lookup tab.

To edit a role:

1. Select **Roles** under the **Configure** menu.
   The Lookup tab opens on the Roles page.

2. Click **Edit** in the **Actions** column for the role that you want to edit.

3. The Edit Role page opens.

4. Make your desired changes.

   Note: You cannot change the name of a role once it has been created.

5. Click **Save**.

To delete a role:

1. Select **Roles** under the **Configure** menu.
   The Lookup tab opens on the Roles page.

2. Click **Delete** in the Actions column for the role that you want to delete.

3. A confirmation window opens.

   Note: Deleting a role permanently removes it from all registries that are connected to this instance of Identity and Access Manager.

4. Click **Delete**.

Manage Users

Select **Users** under the **Manage** menu to display the Lookup tab of the Users page. Click **View All Users** to see a list of all manually created users and all LDAP users who have logged into DevTest Solutions.

To search for a specific user, enter a full or partial name or email address in the **Search** field.

To add a user:

1. Select **Users** under the **Manage** menu.
   The Lookup tab opens on the Users page.

2. Click **Add User**.
   The Add User page opens.

3. Enter the user details.

   - Required fields are noted with a red asterisk.

   - Click the question mark beside each field for more information about that field.

4. Click **Save**.

5. Follow the steps in edit a user below to define the credentials and role mappings for the user.

To edit a user:

1. Select **Users** under the **Manage** menu.
   The Lookup tab opens on the Users page.

2. Click **View All Users** to see a list of users.

3. This page displays all manually created users and all LDAP users who have logged into DevTest Solutions.

4. Click **Edit** in the **Actions** column for the user that you want to edit.
   The Edit User page opens.

5. Make the desired changes.

   - The **Credentials** tab allows you to reset a user's password.

   - The **Role Mappings** tab lets you give a user access to specific roles.

     - **Available Roles** are roles that are defined within DevTest, but which have not been assigned to this user by an administrator or by LDAP. You can select a role and click **Add selected** to assign the role to this user.

     - **Assigned Roles** are roles that have been assigned to this user by an Identity and Access Manager administrator. You can select a role and click **Remove selected** to remove this role from a user.

     - **Effective Roles** are roles that have been assigned to this user as a result of the user being associated with an LDAP group or groups.

- On the **Sessions** tab, you can see all active sessions and log out selected sessions or all sessions.

To delete a user:

1. Select **Users** under the **Manage** menu.
   The Lookup tab opens on the Users page.
2. Click **View All Users** to see a list of users.
3. This page displays all manually created users and all LDAP users who have logged into DevTest Solutions.
4. Click **Delete** in the Actions column for the user that you want to delete.
   A confirmation window opens.
5. Click **Delete**.

To unlock user accounts that have been locked because of too many failed login attempts:

1. Select **Users** under the **Manage** menu.
   The Lookup tab opens on the Users page.
2. Click **Unlock Users**.
3. This unlocks all locked users.

Standard User Types and Standard Roles

DevTest Solutions creates standard user types with associated roles. Each role is associated with a unique set of permissions. You can change the permissions that DevTest assigns to a standard role.

- **CAI Power User and SV Power User**
  The CAI Power User / SV Power User combination user types includes the following roles:
  - Super User
  - DevTest Administrator
- **CAI Power User**
  The CAI Power User user type includes the following role:
  - CAI Power
- **SV Power User**
  The SV Power User user type includes the following roles:
  - Test Administrator
  - SV Power
  - Test Runner
- **Test Power User**
  The Test Power User user type includes the following roles:
  - Test Power

  - Test Observer
  - Load Tester
  - User

- **Runtime User**
  The Runtime User user type includes the following roles:
  - Runtime
  - Guest

> **Note:** The default **Super User** role does not include access to Enterprise Dashboard or Virtual Service Catalog. For a user to access Enterprise Dashboard, you must assign either the **Enterprise Dashboard User** or **Enterprise Dashboard Administrator** role. For a user to access Virtual Service Catalog, you must assign either the **Virtual Service Catalog User** or **Virtual Service Catalog Administrator** role.

Standard Users

When you start the registry for the first time, standard users are created. Each standard user is assigned a role within a user type, and default password. See Standard User Types and Standard Roles for permissions that are associated with each role.

> **Important!** To help prevent unauthorized access, we recommend that you change the default passwords for these users as soon as possible.

- **admin**
  The admin user has the Super User role, a CAI Power User and SV Power User combination user type. The default password is **admin**.
- **caipower**
  The caipower user has the CAI Power role, a CAI Power User user type. The default password is **caipower**.
- **svpower**
  The svpower user has the SV Power role, an SV Power User user type. The default password is **svpower**.
- **tpower**
  The tpower user has the Test Power role, a Test Power User user type. The default password is **tpower**.
- **devtest**
  The devtest user has the Runtime role, a Runtime User user type. The default password is **devtest**.

- **sysadmin**
  The sysadmin user has the System Administrator role, a Runtime user type. The default password is **sysadmin**.

- **guest**
  The guest user has the Guest role. The default password is **guest**.

Change Passwords for Standard Users

When you start the registry for the first time, seven standard users are created. Each standard user is assigned a role, user type, and default password. To help prevent unauthorized access, we recommend that you change the default passwords for these users as soon as possible.

Follow these steps:

1. Log into Identity and Access Manager.
   For more information about starting IAM and accessing the UI, see Start Identity and Access Manager.

2. Click **Users** under the **Manage** menu.
   The Lookup tab opens on the Users page.

3. Click **View All Users**.
   A list of available users displays. This list includes all manually created users and all LDAP users who have logged into DevTest Solutions.

4. Click **Edit** in the **Actions** column for the user that you want to update.
   The Edit User page opens.

5. Click the **Credentials** tab.

6. Enter the new credentials for the user and click **Reset Password**.
   A confirmation window opens.

7. Click **Change Password**.

8. Repeat these steps for each standard user.

User Activity Report

The User Activity report contains a series of entries for user activities that ACL controls. You access the report from the Administration window in DevTest Portal. Open the window by selecting **Reporting, Administration** from the left navigation menu. System administrators can search and filter the report for a specific event and user.

The User Activity report provides a record of permission checks made to ACL and shows the actions allowed and denied for each user.

For more information about viewing and manipulating reports see View and Navigate Reports.

More Information:

- Testing Reports

- Virtual Service Metrics Reports
- CAI Top N and Metrics Reports

Manage Sessions and Events

Intro

- Manage Sessions
- Manage Events
  - View Events
  - Remove Events

Manage Sessions

Click Manage/Session on the left navigation bar to display the **Manage Sessions** window.

The **Realm Sessions** tab displays all active session in your realm, and lets you click a button to log them all out.

Use the **Revocation** tab to set a date to indicate that the system should revoke all tokens that are issued before this date.

Manage Events

View Events

Click Manage/Events on the left navigation bar to display the **Manage Events** window.

The **Login Events** tab displays all events related to users logging into and out of Service Virtualization components. Use the buttons on the upper right part of the tab to:

- indicate how many events to see on the tab
- filter on event type (LOGIN, LOGOUT, and LOGIN_ERROR), user, and a date range
- update the display when you have changed the number of events or the filter
- reset the values back to the default values

The **Admin Events** tab displays activities performed by administrators of Identity and Access Management. For example, the CREATE USER event indicates the ID and username of the administrator that created the user and the ID and username of the user that was created.

Remove Events

The **Config** tab lets you set an expiration time in minutes, hours, or days to govern how long to keep login events in the event log. The default is 2 days.

The **Config** tab also lets you manually clear admin events by clicking **Clear admin events**.

When you have completed your changes, click **Save**.

Update the IAM Database

Identity and Access Manager includes a utility for updating the database that Identity and Access Manager uses. You can use this utility to change the database that is used or to change the password for the database.

To update the Identity and Access Manager utility:

1. Launch the **DatasourceUpdater** executable in the ***Install Directory /IdentityAccessManager/bin*** folder.

2. Follow the prompts to enter the details for your database.

Configure IAM Logging

Identity and Access manager writes logs to <INSTALL_DIR>/vscatalog/IdentityAccessManager /standalone/log/server.log by default. You can change the location of the log file by editing the **logging.properties** file in the <INSTALL_DIR>/vscatalog/IdentityAccessManager/standalone /configuration directory.

Access Control (ACL)

Access control (ACL) is the mechanism that DevTest uses to authorize users and enforce roles. ACL grants users access to only the application functionality they require to perform their role-based activities.

You administer ACL using DevTest Portal.

- ACL Administration
- Authentication
- Authorization

ACL Administration

The ACL Administrator is responsible for the following activities:

- Assigning permissions to individual roles.

- Restricting access to various parts of DevTest by assigning components to Resource Groups.

Note: The creation of users and roles and the assignment of roles to users are performed in Identity and Access Manager.

The Administrator must have a thorough understanding of the various roles to assign the associated privileges to each role.

> **Important!** Do not use the default Super User or the System Administrator users to manage the ACL system. Use the default Super User to create new users with identical roles to the default Super User and System Administrator. Use the new users for managing the ACL system, and change the passwords for the default Super User and System Administrator users to prevent unauthorized access.

Authentication

You can determine how DevTest authenticates users. Users and roles are created in Identity and Access Manager, where you can also assign the roles to specific users. You can manually add users to the IAM database or use your LDAP server for authentication.

Authorization

DevTest limits what features individual users can access based on their business role. DevTest is installed with more than a dozen standard roles. You can experience how users with different roles experience DevTest by logging on as a standard user. Standard users are assigned unique standard roles.

The roles are created and assigned to users in Identity and Access Manager, but the permissions are assigned to each role in ACL. The role grants a set of permissions. It is possible to assign multiple roles, but is rarely necessary as roles with more responsibility include permissions from lower related roles.

The following activities are examples of the activities that you can control with permissions:

- Create a test case
- Create a staging document
- Stage a test case

Permission Types

Permissions can be divided into the following types:

- Boolean
- Numeric Limit

Boolean

Most of the permissions are Boolean. These permissions indicate whether an activity is allowed or not allowed.

For example, the Stop Registry permission indicates whether a user can stop the registry.

Numeric Limit

A permission can represent a numeric limit.

For example, the Stage Test permission lets a user stage a test case with the specified maximum number of virtual users.

The value of a numeric limit permission must be -1, 0, or a positive integer.

If the value is -1, the permission is allowed and there is no numeric limit.

If the value is 0, the permission is denied.

Standard Permissions

A permission controls whether a user can perform a specific activity.

If a parent permission is allowed, then all its child permissions are allowed.

The following top-level permissions are automatically created.

- User and Role Administration Permission
- Resource Administration Permission
- Test/Suite Administration Permission
- Virtual Services Administration Permission
- DevTest Server Administration Permission
- CVS Administration Permission
- Report Administration Permission
- Metric/Event Administration
- CAI Administration Permission
- DevTest Workstation Permission

User and Role Administration Permission

The User and Role Administration permission is a Boolean permission that allows a user to manage access control (ACL).

Resource Administration Permission

The Resource Administration permission is a Boolean permission that allows a user to manage resource groups.

Test/Suite Administration Permission

The Test/Suite Administration permission has the following child permissions.

Child Permission	Type	Description
Stage Test	Numeric Limit	Allows a user to stage a test case with the specified maximum number of virtual users.
Stage Suite	Numeric Limit	Allows a user to stage a suite with the specified maximum number of virtual users.
Stop Test	Boolean	Allows a user to stop a test.
Kill Test	Boolean	Allows a user to kill a test.
Optimize Test	Boolean	Allows a user to optimize a test.
View Test	Boolean	Allows a user to view a test.
Quick Stage Test	Boolean	Allows a user to stage a quick test.
Stage Local Suite	Boolean	Allows a user to run a suite locally.

Virtual Services Administration Permission

The Virtual Services Administration permission has the following child permissions.

Child Permission Level 1	Child Permission Level 2	Child Permission Level 3	Type	Description
View VSE Dashboard			Boolean	Allows a user to monitor VSEs.
VSE Server Administration			Boolean	
	Configure VSE Tracking Data Cleanup		Boolean	Allows a user to configure the process that deletes tracking data older than a specified amount of time.
	Stop VSE Server		Boolean	Allows a user to shut down a Virtual Service Environment.
	Reset VSE Server		Boolean	Allows a user to reset a Virtual Service Environment.
	Monitor VSE Server		Boolean	Allows a user to monitor a Virtual Service Environment.
VSE Service Administration			Boolean	
	VSE Service Deployment		Boolean	Allows a user to deploy a virtual service.
	VSE Service Archive Retrieval		Boolean	Allows a user to download the Model Archive (MAR) associated with a virtual service.
	VSE Service Execution		Boolean	
		Start VSE Service	Boolean	Allows a user to start a virtual service.
		Stop VSE Service	Boolean	Allows a user to stop a virtual service.
	Update Deployed VSE Service		Boolean	
		Update Virtual Service Capacity	Boolean	Allows a user to update the concurrent capacity for a deployed virtual service.
		Update Virtual Service Think Scale	Boolean	Allows a user to update the think time scale for a deployed virtual service.
		Update Virtual Service Auto-Restart	Boolean	Allows a user to update the auto-restart option for a deployed virtual service.
		Update Virtual Service Group Tag	Boolean	Allows a user to update the group tag on a deployed virtual service

	Set Virtual Service Execution Mode	Boolean	Allows a user to select a new execution mode for a deployed virtual service.
	Reset Virtual Service Transaction Counts	Boolean	Allows a user to reset the transaction and error counts for a deployed virtual service.
	Heal Virtual Service Image	Boolean	Allows a user to perform model healing.

DevTest Server Administration Permission

The DevTest Server Administration permission has the following child permissions.

Child Permission	Type	Description
Debug DevTest Server	Boolean	Allows a user to create heap dumps, create thread dumps, and perform garbage collection for a server component.
Monitor Registry	Boolean	Allows a user to access the Registry Monitor.
Reset Registry	Boolean	Allows a user to reset the registry.
Stop Registry	Boolean	Allows a user to stop the registry.
Monitor Coordinator	Boolean	Allows a user to view a coordinator status message.
Reset Coordinator	Boolean	Allows a user to reset a coordinator.
Stop Coordinator	Boolean	Allows a user to stop a coordinator.
Monitor Simulator	Boolean	Allows a user to view a simulator status message.
Reset Simulator	Boolean	Allows a user to reset a simulator.
Stop Simulator	Boolean	Allows a user to stop a simulator.

CVS Administration Permission

The CVS Administration permission has the following child permissions.

Child Permission	Type	Description
View CVS Dashboard	Boolean	Allows a user to access the CVS tab in DevTest Portal.
Deploy or re deploy monitor to CVS	Boolean	Allows a user to deploy and redeploy CVS monitors.
Delete monitor from CVS	Boolean	Allows a user to delete CVS monitors.

Run monitor immediately on CVS	Boolean	Allows a user to run CVS monitors immediately, regardless of when they are scheduled to run.
Activate/Deactivate monitor on CVS	Boolean	Allows a user to activate and deactivate CVS monitors.

Report Administration Permission

The Report Administration permission has the following child permissions.

Child Permission	Type	Description
Access reporting console	Boolean	Allows a user to access the Reporting navigation menu.
View report data	Boolean	Allows a user to view his or her own report data.
View report data of other users	Boolean	Allows a user to view report data of other users.
View PDF report data	Boolean	Allows a user to view his or her own report data as a PDF file.
View PDF report data of other users	Boolean	Allows a user to view report data of other users as a PDF file.
Export Excel report data	Boolean	Allows a user to export his or her own report data as an Excel file.
Export Excel report data of other users	Boolean	Allows a user to export report data of other users as an Excel file.

Metric/Event Administration

The Metric/Event Administration permission has the following child permissions.

Child Permission	Type	Description
Add metric at runtime	Boolean	Allows a user to add a metric at runtime.
Remove metric at runtime	Boolean	Allows a user to remove a metric at runtime.
Pause metrics collection	Boolean	Allows a user to pause metric collection.
Save interval data	Boolean	Allows a user to save interval data.
Save metric data	Boolean	Allows a user to save metric data.
Save event data	Boolean	Allows a user to save event data.

CAI Administration Permission

The CAI Administration permission has the following child permissions.

Child Permission	Type	Description
View Paths	Boolean	Allows a user to view paths in DevTest Portal.
Create Baselines	Boolean	Allows a user to create baselines in DevTest Portal.
Create Virtual Service Models	Boolean	Allows a user to create virtual service models and raw traffic files in DevTest Portal.
Extract Data	Boolean	Allows a user to create data sets in DevTest Portal.

View Tickets	Boolean	Allows a user to view tickets in DevTest Portal.
Edit Tickets	Boolean	Allows a user to edit tickets in DevTest Portal.
View Agents	Boolean	Allows a user to view agents in DevTest Portal.
Agent Administration	Boolean	Allows a user to stop and start dispatching for an agent in DevTest Portal.
Import Paths	Boolean	Allows a user to import paths into DevTest Portal.
Export Paths	Boolean	Allows a user to export paths from DevTest Portal.
Delete Paths	Boolean	Allows a user to delete paths from DevTest Portal.
Create Document	Boolean	Allows a user to create a document from transactions in DevTest Portal.
View Documented Transactions	Boolean	Allows a user to view documented transactions in DevTest Portal.
Create Documented Transactions	Boolean	Allows a user to document transactions for testing in DevTest Portal.
Edit Point of Interest	Boolean	Allows a user to edit Point of Interest transactions in DevTest Portal.
View Point of Interest	Boolean	Allows a user to view Point of Interest transactions in DevTest Portal.

DevTest Workstation Permission

The DevTest Workstation permission has the following child permissions.

Child Permission	Type	Description
Start DevTest Workstation	Boolean	Allows a user to start DevTest Workstation.
View a Configuration	Boolean	Allows a user to view configurations.
Edit a Configuration	Boolean	Allows a user to edit configurations.
Create a New Configuration	Boolean	Allows a user to create configurations.
View a Staging Document	Boolean	Allows a user to view staging documents.
Edit a Staging Document	Boolean	Allows a user to edit staging documents.
Create a New Staging Document	Boolean	Allows a user to create staging documents.
View a Suite	Boolean	Allows a user to view suites.
Edit a Suite	Boolean	Allows a user to edit suites.
Create a New Suite	Boolean	Allows a user to create suites.
View a Test Case	Boolean	Allows a user to view test cases.
Edit a Test Case	Boolean	Allows a user to edit test cases.
Create a New Test Case	Boolean	Allows a user to create test cases.
View an Audit Document	Boolean	Allows a user to view audit documents.
Edit an Audit Document	Boolean	Allows a user to edit audit documents.
Create a New Audit Document	Boolean	Allows a user to create audit documents.
View Virtual Service Model and Data Sets	Boolean	Allows a user to view virtual service models and data sets.

Edit Virtual Service Model and Data Sets	Boolean	Allows a user to edit virtual service models and data sets.
Create a New Virtual Service Model	Boolean	Allows a user to create virtual service models.
View Virtual Service Image	Boolean	Allows a user to view service images.
Edit Virtual Service Image	Boolean	Allows a user to edit service images.
Create a New Virtual Service Image	Boolean	Allows a user to create service images.
View Copybook Bundle	Boolean	Allows a user to view copybook bundles.
Edit Copybook Bundle	Boolean	Allows a user to edit copybook bundles.
Create a New Copybook Bundle	Boolean	Allows a user to create copybook bundles.
Execute ITR	Boolean	Allows a user to start the Interactive Test Run utility.
Execute Instant Replay	Boolean	Allows a user to replay a test case to a specific point in the Interactive Test Run utility.
View ITR Properties	Boolean	Allows a user to view the Properties tab in the Interactive Test Run utility.
View ITR Test Events	Boolean	Allows a user to view the Test Events tab in the Interactive Test Run utility.

View User Information from DevTest Workstation

When you are logged in to DevTest Workstation, you can open a dialog that lets you view the following user information:

- The unique ID of your user name
- The roles that are assigned to you
- The permissions that you have

Follow these steps:

1. Select **System**, **View Security Permissions** from the main menu.
   The **User Security Permissions** dialog opens.
2. When you finish viewing the information, close the dialog.

Manage Role Permissions

You use **DevTest Portal** to add or remove permissions that are assigned to each role.

> **Note:** The creation of roles and the assignment of roles to users are performed in Identity and Access Manager.

DevTest Portal displays the roles and permissions in the **Roles** tab.

- Each role is shown with its effective user type. This identifies the user type that is reported in the Enterprise Dashboard. Selecting a role displays the **Permissions** tab, with available permissions shown in the left column and permissions that are assigned to the role in the right column.

- When a user has more than one role, all the permissions from each role are aggregated into a single list of permissions from which actions are interrogated.

When you select a role, the **Permissions** tab on the right is populated. On this tab, if you select a parent permission, the child permissions are automatically selected. If you clear a parent permission, the child permissions are automatically cleared. If you clear a child permission, the parent check becomes a dash to indicate partial child selection. If you unselect the last child permission, the parent becomes unselected.

You can show or hide any of the columns on the **Roles** tab. Click the drop-down arrow in a column. Select **Hide Column** to hide the column. To show a hidden column, select the **Filter** icon and click the name of the hidden column.

To update the permissions for a role:

1. In DevTest **Portal**, click **Settings**, **Access Control**, **Roles**.
   The Roles tab opens.

   Roles Tab

2. Click the name of the role that you want to update in the panel on the left.
   The details for the selected role display in the Role Information panel on the right. The available and assigned permissions display on the Permissions tab. Click the **Expand** icon for any permission to see its child permissions.

   > The **All Effective User Types** field identifies all the user types that this role consumes. For example, there is at least one permission from each of the identified user types assigned to the role.

3. To add permissions to this role:

   a. Select the check box for each permission that you want to add from the **Available** column on the Permissions tab.

b. If the permission includes a numeric limit, type the numeric limit in the text box next to the permission.

c. Click the right arrow (Grant Selected Permissions) to add the selected permissions to the **Assigned** column.

4. To remove permissions from this role:

   a. Select the check box for each permission that you want to remove from the **Assigned** column on the Permissions tab.

   b. Click the left arrow (Revoke Selected Permissions) to remove the selected permissions and move them to the **Available** column.

5. To assign a user type to a role quickly, click the **Quick Selection** button and select a user type from the drop-down.

6. Click **Save**.

Resource Groups

Resource groups are defined for one or more DevTest Servers or VSEs and determine the resources that a user or a project can access. You can then associate roles with resource groups to determine which roles can act on which resources.

> **Tip:** If you have a small environment, such as a site that has a local registry and a VSE, you must set up resource groups to control access such as restricting VSE-specific access. However, you may want to set up Resource Groups to limit the access between your Production and your Development group.

Manage Resource Groups

Use the **DevTest Portal** to add resource groups, change the details for a resource group, delete resource groups, and remove a resource from a resource group.

You can show or hide any of the columns on the **Resource Groups** tab. Click the drop-down arrow in a column. Select **Hide Column** to hide the column. To show a hidden column, select the **Filter** icon and click the name of the hidden column.

To add a resource group:

1. In the **DevTest Portal**, click **Settings**, **Access Control**, **Resource Groups**.

2. Click **Add Resource Group**. The **Add New Resource Group** dialog appears.

3. In the **Name** field, enter a unique name for the resource group. The name must contain only alphanumeric characters, spaces, hyphens, or underscores.

4. In the **Description** field, enter a description of the resource group.

5. From the **Resources** tab, select a resource or resources by highlighting the group's row.

6. Click the arrow between the columns to move a resource from the **Available** column to the **Assigned** column.
   To filter the lists of resources, type in the **Type your filter** field at the top of the column.

7. Click **Save**.
   The Resource Groups tab shows the resource with the associated resource group displayed in the **Resource Groups** column.

To display resource groups:

1. To refresh the display, click **Refresh** at the upper right corner of the window.

2. Resources that are inactive, but still associated with a resource group, display with a Disconnected status.

To delete a resource group or groups:

1. In the **Action** column for the resource group, click the **Delete** icon.

2. From the drop-down list, select **Delete Resource Group**.

3. Click **Delete** in the **Please Confirm** dialog.

4. To delete multiple resource groups, select the check box to the left of the groups you are deleting and click the **Delete Resource Group(s)** button.

To copy a resource group:

1. In the **Action** column for the resource group, click the **Copy** icon.

2. A copy of the resource group appears in the **Add New Resource Group** dialog.

3. Continue as if you are adding a resource group.

To remove a resource from a resource group:

1. From the **Resources** tab, highlight the resource in the **Assigned** column and click the left arrow at the top of the column.

2. Click **Save**.

Grant Roles to Resource Groups

To grant roles to resource groups, follow these steps.

1. In the **DevTest Portal**, click **Settings**, **Access Control**, **Resource Groups**.

2. Select a resource group.

3. Select the **Roles in Resource Group** tab.

4. In the **Available** column, select a role, then click the right arrow between the columns to move it into the **Assigned** column.

5. Click **Save**.

Use Resource Groups to Control Access

You can use resource groups to control access to resources in addition to using Access Control (ACL).

Follow these steps:

1. Add the resource groups that are appropriate for your organization. For instructions, see *Manage Resource Groups*.
2. To open a configuration file in DevTest Workstation, double-click it from the **Project Panel** .
3. Click **Add** at the bottom of the panel.
4. Click the **Key** column in the **Properties Editor**.
5. From the list of properties, select **RESOURCE_GROUP**.
6. Click the **Value** column.
7. Select the resource group that you want to associate with this configuration.
   You can only select one resource group from the drop-down list, but you can edit the **Value** column to enter a list of resource groups, which are separated by commas.
8. To save the configuration, click **Save**.

Any test case, test suite, or VSM that you stage using this configuration is restricted to using the resources that are in the resource group.

> **Note:** The resource group association is only in effect when the configuration is the active configuration.

Resources

Resources are coordinators, servers, and VSEs.

Manage Resources

Use the **DevTest Portal** to delete disconnected resources and modify the assignment of resource groups for a selected resource.

You can show or hide any of the columns on the **Resources** tab. Click the drop-down arrow in a column. Select **Hide Column** to hide the column. To show a hidden column, select the **Filter** icon and click the name of the hidden column.

To delete a resource or resources:

You can only delete resources that are disconnected.

1. In the **DevTest Portal**, click **Settings**, **Access Control**, **Resources**.
2. In the **Action** column for the resource, click the **Delete** icon.

3. From the drop-down list, select **Delete Resource**.

4. Click **Delete** in the **Please Confirm** dialog.

5. To delete multiple resources, select the check box to the left of the resources you are deleting and click the **Delete Resource(s)** button.

To change the assignment of resource groups for a resource:

1. In the **DevTest Portal**, click **Settings**, **Access Control**, **Resources**.

2. Select a resource by highlighting the resource's row.
   The **Resource Information** tab displays.

3. Click the arrow between the columns to move a resource group from the **Available** column to the **Assigned** column.
   To filter the lists of resource groups, type in the **Type your filter** field at the top of the column.

4. Click **Save**.

Using SSL to Secure Communication

By default, communication between components uses an unencrypted protocol. If necessary, the Secure Sockets Layer (SSL) can encrypt the network traffic. For example, if you run a lab in a public cloud and you want to ensure the traffic that is transmitted from your workstation is encrypted.

The easiest way to enable SSL is to set a DevTest property.

```
lisa.net.default.protocol=ssl
```

You cannot specify this property in **site.properties**; that is too late in the bootstrap phase. This property must be specified in **local.properties** (or on the command line).

If you then start a registry with no extra parameters - for example, it is listening on port 2010 (the usual port) but it expects clients to use the SSL protocol - the service name for the registry is *ssl://hostname:2010/Registry*.

If you want to connect to that registry from DevTest Workstation, use *ssl://hostname:2010/Registry* instead of the usual *tcp://hostname:2010/Registry*. If you start a simulator on the same computer, it is available on *ssl://hostname:2014/Simulator*, and it automatically connects to the registry at *ssl://hostname:2010/Registry* with no property changes.

You can also mix and match SSL and normal TCP protocols. If you leave the **lisa.net.default.protocol** property at its default setting (tcp), you can enable specific services for SSL by specifying the name of the individual service with the "ssl:" protocol prefix, instead of the default "tcp:" prefix. For example, to start a registry in SSL mode:

```
Registry --name=ssl://reghost.company.com:2010/Registry
```

To enable the SSL, use "ssl" in the service names instead of "tcp". For example:

```
Registry --name=ssl://reghost.company.com:2010/Registry
```

starts the registry with SSL enabled.

To connect a simulator to this registry, start the simulator with the fully qualified registry address:

```
Simulator --name=ssl://simhost.company.com:2014/Simulator --registry=ssl://reghost.
company.com:2010/Registry
```

This command tells the simulator to use SSL to talk to the registry while also securing the simulator. If you want the simulator itself to be unsecured, do this:

```
Simulator --registry=ssl://reghost.company.com:2010/Registry
```

Mixing secured and unsecured servers is not common. However, you may want to have unsecured servers inside your firewall and secured servers in a public cloud. SSL encryption costs some overhead, which varies considerably depending on the hardware.

The **lisa.net.default.protocol** property defines the default protocol for ActiveMQ connections. The property does not influence the protocol that is used when DevTest components start.

If you have DevTest Enterprise Dashboard configured to use SSL, the **lisa.enterprisedashboard.server.url** property must be configured to have the value of **ssl**. Set this property value in the **local.properties** file.

All communication with the Enterprise Dashboard is done through the webserver port (1506 by default), and uses either HTTP or HTTPS.

To configure the Enterprise Dashboard to use HTTPS, create the **dradis.properties** file (by copying the **_dradis.properties** file), and uncomment the **dradis.webserver.https.enabled** property.

```
dradis.webserver.https.enabled=true
```

To configure the registry to know that the Enterprise Dashboard is using HTTPS, uncomment the **devtest.enterprisedashboard.https.enabled** property in the **site.properties** file.

```
devtest.enterprisedashboard.https.enabled=true
```

To configure the registry and all other components to use SSL by default, uncomment the **lisa.net.default.protocol** value in the **local.properties** file.

lisa.net.default.protocol=ssl

SSL Certificates

By default, a self-signed certificate encrypts and decrypts the messages between components. VSE also uses this certificate when recording https:// style web traffic. The certificate is in the **LISA_HOME\webreckeys.ks** file.

When you set the default protocol to SSL and you do not change anything else, you use an "internal DevTest" certificate. All DevTest users (not only your organization) share this internal certificate. Using this certificate encrypts the network traffic. Using this certificate does not prevent an unauthorized user from connecting to your simulator on a public cloud. To prevent this type of unauthorized access, use your own certificate. You can also continue to use the "well-known" DevTest certificate, but enable access control.

If you want to use your own certificate, you can override the certificate keystore by specifying the following properties:

```
lisa.net.keyStore=/path/to/keystore.ks
lisa.net.keyStore.password=plaintextPassword
```

The first time DevTest reads the plain text password, it converts the password to an encrypted property:

```
lisa.net.keyStore.password_enc=33aa310aa4e18c114dacf86a33cee898
```

Use Your Own Self-Signed Certificate

This example uses the keytool utility, which is in the Java Runtime Environment (JRE).

To create your own self-signed certificate:

1. Work with your security administrator to create a self-signed certificate.

2. Copy the certificate file to LISA_HOME and update **local.properties**:

   ```
   lisa.net.keyStore={{LISA_HOME}}keystore.ks
   lisa.net.keyStore.password=MyNewSecretPassword
   ```

3. The first time DevTest reads the plain text password, it converts the password to an encrypted property:

   ```
   lisa.net.keyStore.password_enc=33aa310aa4e18c114dacf86a33cee898
   ```

   The server side of the connection configuration is complete.

4. Configure the client.
   Because this certificate is self-signed, you explicitly tell the clients to trust the certificate. Typically, when you connect to an SSL service (for example, using a browser to https://www.MyBank.com) a trusted Certification Authority certifies the certificate. Because a trusted third party does not certify self-signed certificates, you must add the certificate to a trust store:

   ```
   lisa.net.trustStore={{LISA_HOME}}trustStore.ts
   lisa.net.trustStore.password=MyNewSecretPassword
   ```

   The same keytool utility manipulates trust stores. In general, a keystore contains one certificate and a trust store contains one or more certificates.

5. Export the certificate from the server keystore:

   ```
   keytool -exportcert -rfc -alias serverA -keystore keyStore.ks -file serverA.
   cer
   ```

   The **-rfc** means to export the certificate as ASCII text instead of binary, to make it easier to copy and paste. In our example, the resulting **serverA.cer** file looks like the following example:


   ```
   -----BEGIN CERTIFICATE-----
   MIICEzCCAXygAwIBAgIEThZnYzANBgkqhkiG9w0BAQUFADBOMQswCQYDVQQGEwJDQjELMAkGA1UE
   CBM420IxCzAJBgNVBAcTAkNCMQswCQYDVQQKEwJDQjELMAkGA1UECxMCQ0IxCzAJBgNVBAMTAkNC
   MB4XDTExMDcwODAyMTE0N1oXDTEyMDcwNzAyMTE0N1owTJELMAkGA1UEBhMCQ0IxCzAJBgNVBAgT
   AkNCMQswCQYDVQQHEwJDQjELMAkGA1UECDMCQ0IxCzAJBgNVBAsTAkNCMQswCQYDVQQDEwJDQjCB
   nzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAhYfaN+dCrKQwYZ+KeaaPUI8DeXNiqQ/mS+KGnXnh
   Pz08vdX/7HDLW4pzFhntjmkxxOi9dMwlO2thTD1cOxI571PotenMENo4nyiUAEnMK9MTiWEYr2cQ
   b6/TUueBCjRJ9I0GPCI0WPS+0Na2Q/wq8gPCHmDRpw1Xgo4uZ1v6C/ECAwEAATANBgkqhkiG9w0B
   AQUFAAOBgQByCsX9EoBFIGhcSwoRwEvapIrv8wTaqQPOKKyeIevSmbnERRu6+oi+cJftbdEfw6GG
   CBddJH+dGZ9VeqLU8zBGasbU+JPzG5ElOgOXcUGeQQEaM1YMv6XWrIwNSljQk/MPZSt3ROtJOlae
   JPKJXSQ610xof9+yLHH0ebUGhUjdlQ==
   -----END CERTIFICATE-----
   ```


6. Add this certificate to the client trust store.
   Because you are creating a trust store file, you enter the password twice. If you add further certificates to this client trust store, you enter the password once.

   ```
   prompt> keytool -importcert -file serverA.cer -keystore trustStore.ts
   Enter keystore password:
   Re-enter new password:
   Owner: CN=serverA, OU=dev, O=itko, L=Dallas, ST=Texas, C=US
   ```

```
Issuer: CN=serverA, OU=dev, O=itko, L=Dallas, ST=Texas, C=US
Serial number: 4e155338
Valid from: Thu Jul 07 16:33:28 EST 2011 until: Wed Oct 05 17:33:28 EST 2011
Certificate fingerprints:
     MD5:  5B:10:F6:C8:02:3E:36:F5:AA:6D:FC:10:EF:F5:7F:54
     SHA1: 09:DA:8E:71:7C:D5:BB:44:89:14:13:07:F4:A1:C7:06:35:CD:BE:B1
     Signature algorithm name: SHA1withRSA
     Version: 3
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

Now you have a cryptographically strong way of talking to your DevTest servers in the public cloud. You must have the certificate on both sides for two DevTest components to talk to each other.

7. If your client talks to more than one remote SSL server, run the same keytool command to import the certificate to the trust store.

> **Note:** In addition to the transport level security (the SSL), you can still enable fine-grain Access Control Lists (ACL). Access Control Lists let you require users to authenticate by user name and password. This type of security is similar to a banking website that uses HTTPS but still requires you to identify yourself.

Use SSL with Multiple Certificates

To have your local copy of DevTest configured to talk securely with multiple server certificates, add each server certificate to your local trustStore file.

In this example, we have serverA, serverB, and workstation.

The administrator of serverA wants to export the certificate using keytool:

```
serverA> keytool -exportcert -alias lisa -file serverA.cer -keystore serverA.ks
```

Similarly, the administrator for serverB wants to export the serverB certificate:

```
serverB> keytool -exportcert -alias lisa -file serverB.cer -keystore serverB.ks
```

Acquire a copy of **serverA.cer** and **serverB.cer,** and then import them into your client trust store:

```
workstation>keytool -importcert -alias serverA -file serverA.cer -
keystore trustStore.ts
workstation>keytool -importcert -alias serverB -file serverB.cer -
keystore trustStore.ts
```

Enter the password to your trustStore to modify it.

Ensure that your workstation is using your trustStore, which now contains certificates for both serverA and serverB.

Copy this file to **LISA_HOME** and update **local.properties** as follows:

```
lisa.net.trustStore={{LISA_HOME}}trustStore.ts
lisa.net.trustStore.password_enc=33aa310aa4e18c114dacf86a33cee898
```

When you run DevTest Workstation, you can select registries.

ssl://serverA:2010/Registry

and

ssl://serverB:2010/Registry

If you try to connect to **ssl://serverC:2010/Registry**, DevTest refuses the connection because you do not have the required certificate.

Mutual (Two-Way) Authentication

You can configure DevTest so that the server and client both need to authenticate each other. This type of authentication requires you to set a property on the server side:

```
lisa.net.clientAuth=true
```

In addition to each client needing a server certificate in the client trustStore, the server component needs a client certificate for each client in the server trustStore:

```
serverA>keytool -importcert -alias clientX -file clientX.cer -keystore trustStore.
ts
serverA>keytool -importcert -alias clientY -file clientY.cer -keystore trustStore.
ts
```

If clientZ attempts to connect to serverA, the connection fails because serverA does not have the clientZ certificate in the serverA trustStore. This failure occurs even though clientZ has the serverA certificate in the clientZ trustStore.

SSL/TLS Protocol Configuration

During HTTPS recording, playback, and live invocation, DevTest enables the following SSL/TLS protocols by default: TLS 1.0, SSL 3.0, and SSL 2.0.

> If your installation of DevTest is using an IBM JVM, the default SSL/TLS protocols are only TLS 1.0 and SSL 3.0, as the IBM JVM does not support SSL 2.0.

To override these defaults, DevTest looks for the **https.protocols** system property to define which SSL/TLS protocols to enable. This property represents a comma-separated list of SSL/TLS protocols that the JVM supports. The values in this list must not have any quotes or white space before or after the commas.

Supported SSL/TLS Protocols and the Equivalent Values Expected by the JVM

SSL/TLS Protocol	Value Expected by JVM
TLS 1.2	TLSv1.2
TLS 1.1	TLSv1.1
TLS 1.0	TLSv1
SSL 3.0	SSLv3
SSL 2.0	SSLv2Hello

For example, setting the property to **TLSv1**,**SSLv3**,**SSLv2Hello** enables the TLS 1.0, SSL 3.0, and SSL 2.0 protocols.

There are two options for configuring the **https.protocols** system property.

Use the local.properties File to Configure All DevTest Components

Set the **https.protocols** system property in the **local.properties** file of your DevTest installation. Each component that is part of your DevTest installation (DevTest Workstation, VSE, Service Image Manager, for example) use this property when handling a SSL/TLS connection during HTTPS recording, playback, and live invocation.

For example, to make TLS 1.2, 1.1, and 1.0 the enabled SSL/TLS protocols for all components, add the following property to the **local.properties** file:

```
https.protocols=TLSv1.2,TLSv1.1,TLSv1
```

Use . vmoptions Files to Configure the SSL/TLS Protocols for Each Component

Configure the **https.protocols** system property on a per-component level by using the **.vmoptions** files in the **bin** directory of your DevTest installation.

For example, to make TLS 1.1 and 1.2 the enabled SSL/TLS protocols for the Virtual Service Environment, add the following line to the **bin/VirtualServiceEnvironment.vmoptions** file:

```
-Dhttps.protocols=TLSv1.1,TLSv1.2
```

Using HTTPS Communication with the invoke APIs

Complete the following tasks to enable HTTPS communication with the invoke APIs.

1. (Optional) Generate a New Key Pair and Certificate
2. Configure the Properties Files

(Optional) Generate a New Key Pair and Certificate

The simplest way to generate keys and certificates is to use the **keytool** application that comes with the JDK. This application generates keys and certificates directly into the keystore.

For more information, see http://www.eclipse.org/jetty/documentation/current/configuring-ssl.html.

Follow these steps:

1. Open a command prompt window.
2. Navigate to the **JAVA_HOME\bin** directory.
3. Type the following command:

   ```
   keytool -keystore keystore -alias jetty -genkey -keyalg RSA
   ```

   > **Note:** You must use **jetty** as the alias.

   This command prompts you for information about the certificate and for passwords to protect both the keystore and the keys within it.

4. Complete the following prompts.

- **Enter the keystore password:**
  The password is case-sensitive. The text of your password does not display.

- **Re-enter new password:**
  The password is case-sensitive. The text of your password does not display.

- **What is your first and last name?**
  [Unknown]:
  Enter the same machine name that is used in the registry name. Typically, this is the unqualified host name of the server. For example, for a computer that is named jetty.eclipse.org, you would enter **jetty.eclipse.org.**
  However, you can start the registry with the -m command line parameter, using an IP address or a fully qualified host name. In these cases, the host name in the SSL certificate must match to prevent certificate errors in the web browser.

  > **Note:** This is the only mandatory prompt.

- **What is the name of your organizational unit?**
  [Unknown]:

- **What is the name of your organization?**
  [Unknown]:

- **What is the name of your City or Locality?**
  [Unknown]:

- **What is the name of your State or Province?**
  [Unknown]:

- **What is the two-letter country code for this unit?**
  [Unknown]:

A confirmation of your entries displays.

5. Type **yes** to confirm.
   The following prompt displays.

   - **Enter key password for <jetty>**
     <RETURN if same as keystore password>:

6. Press Enter.
   The utility creates a new file named **keystore** in the current directory.

7. Copy the new keystore file to your LISA_HOME directory.

8. Rename the keystore file to **webserver.ks**.

> **Note: webserver.ks** is the default file specified in the properties files. If you want to use a different file name, be sure to update the keystore location properties in the next section.

Configure the Properties Files

In this procedure, you configure the following files:

- **dradis.properties**
- **phoenix.properties**

Both files are in the **LISA_HOME** directory.

> **Note:** The first time the system reads a password, it converts the password to an encrypted value.

Follow these steps:

1. Make a copy of the **_dradis.properties** file and save it as **dradis.properties**.
2. Uncomment and configure the following properties:
   - **dradis.webserver.https.enabled**
     The default value is **true**. Do not change the value.
   - **dradis.webserver.ssl.keystore.location**
     The default value is **{{LISA_HOME}}webserver.ks**. Change the value if you want to use a keystore file with a different name or in a different directory.
   - **dradis.webserver.ssl.keystore.password**
     Set the value to the password that you defined when generating the keystore file.
   - **dradis.webserver.ssl.keymanager.password**
     Set the value to the key manager password that you defined when generating the keystore file. Unless you specified a different password, this password is the same as the keystore password.

     > **Note:** The key manager password must match the password of the first certificate that is found in the keystore file.
3. Uncomment and configure the following properties:
   - **lisa.webserver.https.enabled**
     The default value is **true**. Do not change the value.
   - **lisa.webserver.ssl.keystore.location**
     The default value is **{{LISA_HOME}}webserver.ks**. Change the value if you want to use a keystore file with a different name or in a different directory.
   - **lisa.webserver.ssl.keystore.password**
     Set the value to the password that you defined when generating the keystore file.

- **lisa.webserver.ssl.keymanager.password**
  Set the value to the key manager password that you defined when generating the keystore file. Unless you specified a different password, this password is the same as the keystore password.

- **lisa.portal.url.prefix**
  Set the value to **https://**.

4. Save the dradis.properties file.

5. Make a copy of the **_phoenix.properties** file and save it as **phoenix.properties**.

6. Uncomment and configure the following properties:

   - **registry.https.enabled**
     The default value is **true**. Do not change the value.

   - **phoenix.https.enabled**
     The default value is **true**. Do not change the value.

   - **phoenix.ssl.keystore**
     The default value is **{{LISA_HOME}}webserver.ks**. Change the value if you want to use a keystore file with a different name or in a different directory.

   - **phoenix.ssl.keystore.password**
     Set the value to the password that you defined when generating the keystore file.

   - **phoenix.ssl.keymanager.password**
     Set the value to the key manager password that you defined when generating the keystore file. Unless you specified a different password, this password is the same as the keystore password.

7. Save the **phoenix.properties** file.

8. Restart the registry.

Using Kerberos Authentication

Kerberos support is similar to Basic Authentication and NTLM support in DevTest. DevTest uses Kerberos support when an application or resource that DevTest accesses through some of the steps is protected with Kerberos authentication. For example, HTTP/HTTPS, Web Service XML - the same steps as NTLM and Basic Authentication. Kerberos support uses the following properties in the local.properties file:

- **lisa.java.security.auth.login.config**
  The location of the login configuration file.

- **lisa.java.security.krb5.conf**
  The location of the Kerberos configuration file that is used to override any preset locations.

- **lisa.http.kerberos.principal**
  The name of the principal that is used for logging in when using DevTest support for principal + password authentication. When DevTest Workstation starts, it encrypts this principal.

- **lisa.http.kerberos.pass**
  The password that is used for logging in when using DevTest support for principal + password authentication. When DevTest Workstation starts, it encrypts this principal.

You can authenticate with only the **lisa.java.security.auth.login.config** and the **lisa.java.security.krb5.conf** settings. These files and their settings vary depending on the operating system where DevTest runs. Consult the appropriate documentation about how to configure those two files for authentication that does not use DevTest support for principal + password authentication.

DevTest Support for Principal + Password Authentication

To support logging a user in by giving DevTest the credentials, the user must configure their login configuration file to use the DevTest login configuration file. The following example illustrates the contents of the file:

```
com.sun.security.jgss.initiate {
  com.itko.lisa.http.LisaKrb5LoginModule required doNotPrompt=false;
};
```

The custom **LisaKrb5LoginModule** is an extension of the standard **com.sun.security.auth.module.Krb5LoginModule** with one change. This extension submits the credentials in **lisa.http.kerberos.principal** and **lisa.http.kerberos.pass** instead of prompting the user for credentials.

Sample krb5.conf File

```
[libdefaults]
        default_realm = EXAMPLE.COM
        allow_weak_crypto = true

[realms]
        EXAMPLE.COM = {
                kdc = kdc.fakedomain.com:60088
        }
[domain_realm]
        .example.com = EXAMPLE.COM
        example.com = EXAMPLE.COM
[login]
        krb4_convert = true
```

Sample krb5.conf File with Active Directory as KDC

```
[libdefaults]
        default_realm = FAKEDOMAIN.COM
        allow_weak_crypto = false
        default_tkt_enctypes = arcfour-hmac-md5
        default_tgs_enctypes = arcfour-hmac-md5
        permitted_enctypes = RC4-HMAC arcfour-hmac-md5
[realms]
        FAKEDOMAIN.COM = {
            kdc = kdc.fakedomain.com
            master_kdc = kdc.fakedomain.com
            admin_server = kdc.fakedomain.com
            default_domain = FAKEDOMAIN.COM
        }
[domain_realm]
        fakedomain.com = FAKEDOMAIN.COM
[login]
        krb4_convert = true
```

Sample login.config File

```
com.sun.security.jgss.initiate {
  com.itko.lisa.http.LisaKrb5LoginModule required doNotPrompt=false;
};
```

Logging

This section contains the following pages that provide details about logging in DevTest:

- Logging Properties File
- Status Messages for Server Components
- Automatic Thread Dumps
- Test Step Logger
- VSE Logging

Main Log Files

The main log files include:

- **coordinator.log:** the logging output for the coordinator.
- **cvsmgr.log:** the logging output for the Continuous Validation Service (CVS).
- **devtest_broker_pid.log:** the logging output for the broker component of the DevTest Java Agent.
- **devtest_console_pid.log:** the logging output for the console component of the DevTest Java Agent.
- **marmaker.log:** the logging output for the Make Mar command-line utility.
- **pfbroker.log:** the cumulative logging output for the broker component of the DevTest Java Agent.
- **portal.log**: the logging output for DevTest Portal.
- **registry.log:** the logging output for the registry.
- **simulator.log:** the logging output for the simulator.
- **svcimgmgr.log:** the logging output for the Service Image Manager command-line utility.
- **svcmgr.log:** the logging output for the Service Manager command-line utility.
- **trunner.log:** the logging output for the Test Runner command-line utility.
- **vse.log:** the logging output for the VSE server.
- **vsemgr.log:** the logging output for the VSE Manager command-line utility.
- **vse_xxx.log:** the logging output for VSE conversations and service image navigation, where *xxx* is the service image name.
- **workstation.log:** the logging output for DevTest Workstation.

Generally, looking at the main log for the component in question or suspected to have an issue should be enough to get an idea of the problem. The main component logs are: Registry, Coordinator, Simulator, and VSE.

The **lisa.tmpdir** property controls the location of these log files. To see the value of the **lisa.tmpdir** property from DevTest Workstation:

1. Click **Help**, **DevTest Runtime Info** from the main menu.
2. In the **System Properties** tab, locate **lisa.tmpdir**.

If the registry, coordinator, simulator, or VSE is running as a Windows service, the log files are located in the **LISA_HOME\lisatmp** directory.

> **Note:** Although **lisa.tmpdir** allows you to change the location of where you can store your temporary files, we do not recommend that you change this property to save the temporary files out to an external mount point or external share. If you encounter issues with product instability, and you are using an external share for temp file storage, Support might instruct you to go back to using a local disk for temp file storage for continued support of your environment.

Demo Server Log Files

The demo server has its own log files, which are located in the **lisa-demo-server/jboss/server/default/log** directory.

The **lisa.tmpdir** property does not control this location.

The demo server log files are:

- boot.log
- server.log

Logging Properties File

DevTest uses the Apache log4j logging framework. The **logging.properties** file, located either in the user-specified data directory (DevTest Workstation only) or in the **LISA_HOME** directory, lets you configure the logging behavior. This file is your main source for manipulating what is being logged across the multiple components of DevTest.

> **DevTest Workstation and logging.properties**
>
> The DevTest application uses logging.properties file to initialize the logging. When the Workstation starts up in 10.1 or lower, it looks for the logging.properties file in the installation directory also known as LISA_HOME. In 10.2, if you specify a different data directory than LISA_HOME during installation, then the Workstation looks for logging.properties in the user-specified data directory. If it doesn't find a logging.properties file in the data directory that was specified, then the application falls back to LISA_HOME. This change doesn't affect the server components like Registry, VSE, Portal, Simulator, and Coordinator.

To get more logging information from DevTest, you can change the logging level in **log4j.rootCategory**:

```
log4j.rootCategory=INFO,A1
```

The **logging.properties** file contains a set of loggers for third-party components that are included in DevTest. The default log levels for these loggers are intended to prevent the third-party components from flooding the log files with too many messages. You typically do not need to change the log levels.

```
log4j.logger.com.teamdev=WARN
log4j.logger.EventLogger=WARN
...
```

The default appender is **com.itko.util.log4j.TimedRollingFileAppender**. Log statements for a component are appended to a file that is backed up when it reaches a certain size. The default maximum file size is 10 MB. The default number of backup files is 5.

```
log4j.appender.A1=com.itko.util.log4j.TimedRollingFileAppender
log4j.appender.A1.File=${lisa.tmpdir}/${LISA_LOG}
log4j.appender.A1.MaxFileSize=10MB
log4j.appender.A1.MaxBackupIndex=5
log4j.appender.A1.layout=org.apache.log4j.EnhancedPatternLayout
log4j.appender.A1.layout.ConversionPattern=%d{ISO8601}{UTC}Z (%d{HH:mm}) [%t] %-5
p %-30c - %m%n
```

To perform log rotation by date or time, use the **DailyRollingFileAppender.** For more information, see the log4j documentation.

```
log4j.appender.A1=
org.apache.log4j.DailyRollingFileAppender

log4j.

appender.A1.File

=${lisa.tmpdir}/${LISA_LOG}


log4j.appender.A1.Append=true


log4j.appender.A1.DatePattern='.
'
yyyy-MM-dd
```

The backup files are placed in the same directory as the log file. For example, if the log file for the registry has been backed up three times, the directory contains the following files:

- registry.log
- registry.log.1
- registry.log.2
- registry.log.3

Layouts control the format of log statements. The default layout is **org.apache.log4j.EnhancedPatternLayout**. The default conversion pattern is **%d{ISO8601}{UTC}Z (%d{HH:mm}) [%t] %-5p %-30c - %m%n**. This conversion pattern specifies that a log statement includes the date, thread, priority, category, and message. For example:

```
2014-11-20 14:09:08,152Z (07:09) [main] INFO  com.itko.lisa.net.
ActiveMQFactory - Starting amq broker
```

The date uses Coordinated Universal Time (UTC). This convention makes it easier to follow log events when the registry is running in a different time zone than DevTest Workstation.

For information about the thread dump properties, see Automatic Thread Dumps.

Status Messages for Server Components

The following server components write status messages to their log files at a specified interval:

- Registry
- Coordinator
- Simulator
- VSE

The following example was written to a log file by the registry.

```
2016-09-27 18:28:03,082Z (11:28) [Event Sink Thread Pool Thread 1] INFO  com.itko.
lisa.coordinator.TestRegistryImpl -
Coordinator Servers: 0 Simulator Servers: 0 VSEs: 0 Running vusers: 0 Labs: 1
Memory used 444mb, allocated 578mb, max 892mb (49%)
Our cpu usage 0%, system cpu used 1% GC time 0% db ping: 1,010 ms
```

The default interval of the status messages is 30 seconds. You can change the interval by adding the following properties to the **local.properties** file:

- lisa.defaultRegistry.pulseInterval
- lisa.coordinator.pulseInterval
- lisa.simulator.pulseInterval
- lisa.vse.pulseInterval

Automatic Thread Dumps

You can use the **logging.properties** file to enable automatic thread dumps, which are helpful in debugging performance issues.

Locate the following property and change **WARN** to **INFO**.

```
log4j.logger.threadDumpLogger=WARN, THREAD_DUMPS
```

To disable the thread dumps, change **INFO** back to **WARN**.

The default interval of the thread dumps is 30 seconds. You can change the interval by editing the **lisa.threadDump.interval** property in the **lisa.properties** file.

> You can change the setting from WARN to 1 for more granular logging without generating a dump

Test Step Logger

As described in Elements of a Test Step, each test step includes a log message element.

To send the log message to a specific file at runtime, add the following properties to the **logging.properties** file in the **LISA_HOME** directory.

```
log4j.logger.com.itko.lisa.test.StepLogger=DEBUG, A2
log4j.additivity.com.itko.lisa.test.StepLogger=false
log4j.appender.A2=org.apache.log4j.RollingFileAppender
log4j.appender.A2.File=${lisa.tmpdir}/log.log
log4j.appender.A2.MaxFileSize=10MB
log4j.appender.A2.MaxBackupIndex=5
log4j.appender.A2.layout=org.apache.log4j.PatternLayout
log4j.appender.A2.layout.ConversionPattern=%d [%t] %-5p %-30c - %m%n
```

The values of the **File**, **MaxFileSize**, and **MaxBackupIndex** properties can be different from the values that are shown in the preceding example.

Note: After you add the properties, restart DevTest Workstation (if currently running).

The resulting message in the log file is similar to the following example:

```
2012-07-11 17:32:59,390 [basic-test/basic-test [QuickStageRun]/0] DEBUG com.itko.
lisa.test.StepLogger  -
LOG basic-test,basic-test [QuickStageRun],local,0,3,my log message
```

The portion of the message after **LOG** consists of six components:

- The name of the test case.
- The name of the staging document.
- The name of the simulator.
- The instance/vuser number. In a ten vuser test, the number varies from 1 to 10.
- The cycle number. This number applies to situations in which the staging document is configured to run the test over and over until some condition occurs. The value is the number of times this particular vuser executed the test.
- The log message that is set in the test step.

VSE Logging

VSE_Matches File

DevTest generates a log file with the services matches into a file named VSE_matches. You can use this log file to debug errors at the request-response level. It is located in the **lisatmp** directory. It contains the matching characteristics for the request being processed.

If you have multiple VSEs on the same computer, each must have a unique VSE_matches file name.

VSE.log File

The VSE.log file contains the errors found when matching requests and responses.

Monitoring

You can monitor DevTest Solutions by using the Server Health Monitor in DevTest Portal, the ServiceManager command-line utility, the Registry Monitor, or the Enterprise Dashboard.

- Use the Server Health Monitor
- Use the ServiceManager
- Use the Registry Monitor
- Use the Enterprise Dashboard

Use the Server Health Monitor

The Server Health Monitor in DevTest Portal provides an indicator of the overall health of running components.

To access the Server Health Monitor, select **Monitor**, **Server Health** from the left navigation menu.

If you do not have permission to view the health of a component, the component displays with the message "No permission to view status of any components of this type." To remove those components from the display, clear the **No permission** check box. A DevTest administrator specifies permissions for types of components in ACL using roles.

If you do not have permission to view the health of a specific component, the component displays with the message, "Not authorized to monitor status of this resource." To remove those components from the display, clear the **Not authorized** check box. A DevTest administrator specifies permissions for specific components in ACL using resources and resource groups.

The following graphic shows the Server Health Monitor.

Server Health Monitor window

For each server, the following fields are displayed.

- **Type**
  The server type that is displayed is either Registry, Simulator, Coordinator, or VSE.

- **Name**
  Displays the name of the server.

- **Version**
  Displays the version of DevTest Server

- **Host**
  Displays the hostname for the server.

- **Port**
  Displays the port used by the server.

- **OS**
  Displays the operating system of the server.

- **Total Up Time**
  Displays the time, in an hours:minutes:seconds format, since the server was started.

- **Current CPU Usage (%) - JVM**
  Displays the percentage of total CPU usage for the Java virtual machine (DevTest Server).

- **Current CPU Usage (%) - System**
  Displays the percentage of total CPU usage for the system.

- **% of CPU Time - JVM GC**
  Displays the percentage of CPU that the JVM (DevTest Server) garbage collection is using.

- **JVM Heap Memory Consumption - Max (MB)**
  Displays the maximum consumption of heap memory that the JVM (DevTest Server) attempts to use, in megabytes.

- **JVM Heap Memory Consumption - Allocated (MB)**
  Displays the total amount of heap memory in the JVM (DevTest Server), in megabytes.

- **JVM Heap Memory Consumption - Current (MB)**
  Displays the current heap memory consumption in the JVM (DevTest Server), in megabytes.

- **JVM Heap Memory Consumption - Current (%)**
  Displays the current heap memory consumption in the JVM (DevTest Server), as a percentage of the total.

- **Server Info**
  Displays information about servers, depending on the server type. Information that can be displayed includes:

  - **Current User Count**
    (for registries) A snapshot of how many users are logged in to the registry at this moment.

  - **Used Capacity**
    (for coordinators and simulators) Displays roughly the number of running test cases and suites.

- **Available Capacity**
  (for coordinators and simulators) Displays roughly the number of how many more test cases and suites can be run on that component.

- **Max Capacity**
  (for coordinators and simulators) Displays the maximum number of test cases and suites that can be run on that component.
  Default: 128 for coordinators and 256 for simulators. This value can be adjusted with properties.

- **Performance Mode**
  (for VSEs) Displays either **Enabled** or **Disabled**, to indicate whether the VSE is in performance mode.

Use the ServiceManager

The ServiceManager command-line utility lets you perform various actions on a registry, coordinator, simulator, or VSE server.

- Service Manager Options
- Service Manager Examples

The ServiceManager command-line utility has the following format:

```
ServiceManager [--command]=service-name
```

The **service-name** is the name of the service to affect.

The command/name pair can be repeated.

To look up the name, wrap a lisa.properties key with double braces. For example: **{{lisa.registryName}**

Example service names:

- tcp://localhost:2010/Registry

- Simulator (resolves to tcp://localhost:2014/Simulator)

Service Manager Options

- **-h, --help**
  Displays help text.

- **-s service-name, --status=service-name**
  Displays a status message about the service. Entering *all* for the service-name returns status messages about all registered services.

- **-r service-name, --reset=service-name**
  Keeps the service in memory but refreshes its state.

- **-o service-name, --stop=service-name**
  Instructs the service to end.

- **-i valid-remote-init-service-name, --initialize=valid-remote-init-service-name**
  Remotely initializes a service.

- **-t service-name, --threaddump=service-name**
  Instructs the service to generate a thread dump (stack trace) for diagnostics.

- **-e service-name, --heapdump=service-name**
  Instructs the service to create an .hprof file for memory diagnostics.

- **-g service-name, --gc=service-name**
  Instructs the service to force a Java garbage collection.

- **-d service-name, --diagnostic=service-name**
  Creates a zip file that contains diagnostic files for the service. If the service is a registry, then the zip file also contains diagnostic files for all connected coordinators, simulators, and VSE servers. Typically, you use this option when requested to do so by Support.
  loglevel
  This option also includes an optional, secondary parameter that is named **loglevel,** followed by one of the following loglevel keywords: error, warn, info, debug, trace.
  For example, **ServiceManager -d tcp://10.1.1.23:2010/Registry loglevel debug** sets the log level of all components, including agents, to the debug level. Service Manager then writes the following message:
  Log levels set to debug. Run your repro steps, then press return to restore log levels and capture diagnostic.
  The generated zip file contains debug log level logs for all connected components and thread dumps and license information and properties. This zip file also contains the logs for any agents that are connected to the registry broker.

  > **Note:** If the -d command is sent to a VSE, coordinator, or simulator server, only the logs and diagnostics for that component are included in the zip.Agents do not have a trace log level, but the dev log level is similar and treated as such.

  ServiceManager -d requires the full component name (-n tcp://:/) to gather debug information from remote components. If '-n ' is passed only, the utility tries to get the diagnostic information from the same system where the registry service resides.

- **-u username, --username=username**
  Use this command to specify your user name.

- **-p password, --password=password**
  Use this command to specify your password.

- **--version**
  Print the version number.

- **-m registry-name, --registry-name=registry-name**
  Used with *initialize* to specify a registry to which to attach.

- **-n component-name, --component-name=registry-name**
  Used with *initialize* to specify a component name.

- **-l lab-name, --lab-name=lab-name**
  Used with *initialize* to specify a lab name to create.

- **-a app-name, --app=app-name**
  Used with *initialize* to specify a server appID.

Example:

If you have a simulator that you want to name *MySim* and you want to attach it to *MyRegistry* and start a new lab that is named *MyDevLab*, enter:

```
./SimulatorService -n MySim -m tcp://1.2.3.4:2010/MyRegistry -l MyDevLab
```

To add a second simulator to that lab:

```
./SimulatorService  -- component-name=MySecondSim  -- registry-name=tcp://1.2.3.4:
2010/MyRegistry  -- lab-name=MyDevLab
```

To add a VSE to that registry but in a different lab:

```
./VirtualServiceEnvironment -n CoreServices -m tcp://1.2.3.4:2010/MyRegistry -l QA
```

Service Manager Examples

The following example checks the status of the registry.

```
ServiceManager -s Registry
Coordinator Servers: 1 Simulator Servers: 2 VSEs: 1 Running vusers: 0
Labs: 1 Memory used 76mb, allocated 155mb, max 253mb (30%)
labSims: 2 labVSEs: 1 labCoords: 1
```

The following example checks the status of all registered services.

```
Coordinator Server: tcp://bdert-mbp.local:2011/Coordinator
OK: 1 Coordinators running. Memory used 223mb, allocated 461mb, max 910mb (24%) Ou
r cpu usage 0%, system cpu used 8%
Simulator Server: tcp://bdert-mbp.local:2014/Simulator
OK: 1 Simulators running. Memory used 301mb, allocated 437mb, max 910mb (33%) Our
cpu usage 0%, system cpu used 8%
```

The following example stops the registry.

```
ServiceManager -o Registry
Sending stop request to Registry.
```

The following example generates a thread dump for the VSE server.

```
ServiceManager --threaddump=tcp://remote.host.com:2013/VSE
< a bunch of stack traces >
```

The following example forces a Java garbage collection for the VSE server.

```
ServiceManager --gc=VSE
After GC: Memory used 55mb, allocated 225mb, max 246mb (22%)
```

The following example generates a zip file that contains trace level logs for all components that are connected to the registry.

```
ServiceManager --diagnostic=Registry loglevel TRACE
```

The following example generates a zip file that contains debug level logs for the simulator.

```
ServiceManager -d Simulator loglevel debug
```

Use the Registry Monitor

The Registry Monitor lets you monitor the test cases, simulators, coordinators, and virtual environments for a test suite.

- Registry Monitor - Test Tab
- Registry Monitor - Simulators Tab
- Registry Monitor - Coordinator Servers Tab
- Registry Monitor - Virtual Environments Tab

To open the **Registry Monitor**, click **Toggle Registry**

on the main toolbar of DevTest Workstation.

Image of the Registry Monitor

The **Registry Monitor** has the following tabs:

- Tests Tab
- Simulators Tab
- Coordinator Servers Tab
- Virtual Environments Tab

Registry Monitor - Test Tab

The **Tests** tab lists information about all test cases that are currently running in this test suite.

For each test case, you can view the test name, status, start time, instances, average time, and time left.

You can perform the following actions on a selected test:

- To stop a test, click **Stop** .
- To kill a test (stop immediately), click **Kill** .

- To optimize a test, click **Optimize Test** .
  For more information, see Using the Load Test Optimizer.

- To view a test, click **View Test** .
  The test information is displayed in the test monitor.

Registry Monitor - Simulators Tab

The **Simulators** tab lets you add virtual users in real time to the selected simulator server.

This tab lists the simulator server and its available instances.

Registry Monitor - Coordinator Servers Tab

The **Coordinator Servers** tab lists information about coordinator servers that are running.

You can perform the following actions on a selected coordinator server:

- To shut down this service, click **Stop** .

- To reset this service (stop and clear current activities), click **Reset** .

- To view a status message, click **View Status Message** .

Registry Monitor - Virtual Environments Tab

The **Virtual Environments** tab lists information about the virtual environments (if any) that are running.

Use the Enterprise Dashboard

This section contains the following pages describing the functionality of the Enterprise Dashboard:

- Features and Functionality of the Enterprise Dashboard
- Reactivate a Registry or Enterprise Dashboard
- Maintain Registries
- Export Dashboard Data
- Purge Dashboard Data
- Change Dashboard Language
- Return On Investment Calculator

Features and Functionality of the Enterprise Dashboard

This page contains the following information about the features and functionality of the Enterprise Dashboard:

- Open the Enterprise Dashboard
- Peak Usage Data
- SV Power Users Tile
- Performance VSEs Tile
- Dashboard Graphs
  - Transaction and Virtual Service Consumption Graphs
  - Advanced Metrics
- Running Registries
- Enterprise Dashboard Registry Details Window
- Email Notifications
- Enterprise Dashboard Return on Investment Window

Open the Enterprise Dashboard

> To access the Enterprise Dashboard, the Identity and Access Manager component of DevTest must be installed, configured, and running.

To open the Enterprise Dashboard from a web browser:

1. Ensure that Identity and Access Manager is running.
   You cannot login to Enterprise Dashboard if IAM is not up and running.

2. Navigate to the directory where you installed Enterprise Dashboard.

3. The Enterprise Dashboard server must be started if it is not already running. Navigate to the **bin** directory, and run **EnterpriseDashboard.exe**, or select **Enterprise Dashboard Server** from the **Start**, **Programs** menu.

4. Enter http://localhost:1506/ in a web browser or select **DevTest Enterprise Dashboard UI** from the **Start**, **Programs** menu.
   If the **Enterprise Dashboard** is running on a remote computer, replace **localhost** with the name or IP address of that computer.

5. At the login screen, enter your user name and password and click **Log in**.
   The **Enterprise Dashboard** opens.

The main window in the **Enterprise Dashboard** lets you view performance statistics about peak counts for Power Users, Functional Virtual Service Environments, Performance Virtual Service Environments, and Workstations.

The Functional VSE, Performance VSE, and workstation data are displayed in real time.

The SV Power User data is displayed according to session data and shows peak counts based on sessions that have finished (when the user logs out). It does not display counts according to users who may still be logged in as power users.

The Enterprise Dashboard also displays transaction and virtual service history for the current month and running registries.

Peak Usage Data

The peak usage data that is displayed at the top of the dashboard covers four usage areas: Completed Power User Sessions, Functional VSEs, Performance VSEs, and Workstations.

SV Power Users Tile

Click the SV Power Users tile to see the following details: start time, end time, total number users. This data counts only power users whose sessions have already ended. The data does not show a real-time report of currently logged users. Data is updated every hour.

Performance VSEs Tile

Click the Performance VSEs tile to see more details: start time, end time, total VSEs. This data is updated in real-time and shows current peak data for VSEs in performance mode.

Dashboard Graphs

Dashboard graphs display transaction and virtual service consumption.

Transaction and Virtual Service Consumption Graphs

Starting in DevTest 10.3, the Enterprise Dashboard shows transaction and virtual usage details. In our consumption-based licensing, the graphs will display the virtual services used and current transaction used and remaining.

- If the contractual transaction amount has been exceeded, this is displayed in the graph.
- The default view is the current month. You can change the date range to meet individual reporting needs.

For previous licensing models or with unlimited consumption, the graphs show the amount that is consumed in the given period.

Advanced Metrics

See more detail by clicking the **advanced metrics** links in the graph window under the transactions and virtual services graphs.

This opens a new page where four more graphs are available.

Transactions Advanced Metrics

On the **transactions advanced metrics** page, you see the following extra level of detail:

1. A graph showing cumulative transaction use (default current month view), projected use until the end of the month, and the monthly limit.
2. A graph showing daily transaction use and the daily limit.

Performance VSE view

Click the switch to turn on or off VSE performance mode data.

Virtual Services Advanced Metrics

On the **virtual services advanced metrics** page, you see the following:

1. A graph showing virtual service daily use.
2. A table showing:

- Current Virtual Service Environments running
- Number of virtual services that are running on each VSE
- How many transactions are running on each VSE.
- The host and port that is associated with each VSE.

Click the link inside the VSE name column to see details of each virtual service running on a particular VSE. This includes VS name and transactions running on that VS.

> **Note:** After you click the VSE name, the data may not match exactly with the total transaction count for that VSE on the VSE Detail table. This is due to a slight synchronization issue that can occur. The total transaction count on the VSE Detail table is the most accurate.

Performance VSE View

Click the Performance VSE view switch to turn on or off VSE performance mode data.

Running Registries

- **Display Name**
  The Display Name is a name that you assign when you configure the registry.

- **Name**
  The URL of the registry.

- **Last Update**
  The last date and time that Enterprise Dashboard received data from this registry.

- **Version**
  The DevTest version for that registry.

- **Coordinators, Simulators, VSEs, Workstations, Agents, and Labs**
  The number of each resource that is associated with the registry.

  > **Note:** To view agents, you must start broker.exe.

- **DevTest Console URL**
  The address of the landing page for DevTest Console, which provides links to DevTest Portal.

To display information about the database that the Enterprise Dashboard uses, hover over the database icon in the upper right corner of the panel.

The popup window displays the database URL, database type, database version, driver name, driver version, and user.

Enterprise Dashboard Registry Details Window

The Registry Details window in the Enterprise Dashboard lets you view details about registries. Click on a registry display name on the main page to bring up the registry details window.

The top part of the window lists all this registry's coordinators, simulators, VSE servers, labs, Workstations, and agents that were running during the timeframe of the supplied filter.

To designate the timeframe to use in displaying metrics, use the drop-down fields at the upper left of the panel. The first drop-down field provides a list of standard timeframes. To be more specific, specify a start date and start time and end date and end time in the other drop-down fields. Click **Refresh** to refresh the display.

If the **Enterprise Dashboard** has been idle for some time, click **Refresh** to refresh the data for each panel.

The bottom part of the window shows metrics for registries, VSE servers, coordinators, and simulators.

The **Registry Metrics** area of the panel shows the maximum number of workstations, simulators, coordinators, agents, and VSEs concurrently running in the designated time period.

The **VSE Metrics** area of the panel shows the number of transactions and the number of labs that were deployed for each VSE active during the designated time period.

The **Coordinator Metrics** area of the panel shows the number of tests that were started for each coordinator during the designated time period.

If you run a test suite and the **Enterprise Dashboard** "tests started" count is not what you expect, see the suite results section of DevTest Workstation. Check the results for each failed test. If any tests failed to stage, the tests started count does not include them.

The **Simulator Metrics** area of the panel shows the maximum number of virtual users active during the designated time period.

To return to the main window, click **Home** in the breadcrumbs in the top left corner of the page.

> Enterprise Dashboard may sporadically display inadvertent stop and restart of components as network connectivity is lost between the registry and its underlying components. When a registry loses connectivity to an underlying component for a period of time, it is assumed that the component has shut down, so an “End Event” is generated and submitted to Enterprise Dashboard for that component.

Email Notifications

If you are defined as an administrative user in Identity and Access Manager, you can configure email notifications if you want to receive notifications for certain usage milestones. Go to Settings and select **Set Email Notifications.**

A window opens with default notification levels set at 65% and 85% of the monthly consumption limit according to the license. Modify the notification trigger thresholds by changing the percent in the box.

You can turn off the notifications by clicking the check mark in the box to the left of the percent figure or by moving the percent figure to 0.

> **Note:** The email notification option provides valuable information and notifications on transaction usage for customers on the consumption-based licensing model. These notifications provide data on transaction consumption in relation to contractual limits.

Customers on the transaction consumption licensing plan will receive a notification when 100% of the allotted monthly limit is reached.

Enterprise Dashboard Return on Investment Window

This can be found under the tools section of the dashboard (found on the top right section of the main toolbar). This calculator provides a useful way to measure the potential financial benefits of using Service Virtualization within your organization.

For more information, see: Return On Investment Calculator

The email notification option provides valuable information and notifications on transaction usage for customers on the consumption-based licensing model. These notifications provide data on transaction consumption in relation to contractual limits.

Reactivate a Registry or Enterprise Dashboard

The DevTest Installation Setup Wizard configures new registries. When you start the Enterprise Dashboard and the registries, the registries are automatically activated. However, if there are subsequent changes to the Enterprise Dashboard URL, reactivation is needed. This procedure is valid for DevTest 9.1 and later. For reactivating registries for earlier versions, see Maintain Registries.

Follow these steps:

1. If the host name or port of the Enterprise Dashboard changes:

a. Log on to the computer with the registry.

b. Navigate to LISA_HOME and open **site.properties**.

c. Update the following lines if the host name or port of the Enterprise Dashboard changes.

```
devtest.enterprisedashboard.host=localhost
devtest.enterprisedashboard.port=1506
```

d. Update the following line if the URL changed from HTTP to HTTPS or from HTTPS to HTTP.

```
devtest.enterprisedashboard.https.enabled=false|true
```

e. Save the file.

f. Restart the registry.

2. Verify that the Enterprise Dashboard is running.

3. If the host name of the server where the registry is installed changes, restart the registry to reactivate it.

4. If the registry name, or registry port of any registry changes:

   a. Log on to the computer where there has been a change to the registry.

   b. Open a command prompt or terminal window and navigate to LISA_HOME.

   c. Start the registry with a new name or port. For example:

   ```
   ./bin/Registry.exe -n “tcp://localhost:2093/MyRegistry”
   ```

5. Verify registry reconfiguration.

Maintain Registries

The registry gets a license from the Enterprise Dashboard and stores and forwards its use counts to the dashboard.

Maintaining registries involves the following procedure:

- Add the registry of a DevTest Server and then validate the registry.

To add a registry:

1. Navigate to the LISA_HOME directory of a newly installed DevTest Server.

2. Copy **_site.properties** and rename the copy to **site.properties**.

3. Open **site.properties** for edit. Locate Section 1 - Enterprise Dashboard.

4. Uncomment the following line and substitute *localhost* or the valid hostname for *somehost*. Use the hostname for the CIC Enterprise Dashboard bridge.

   ```
   lisa.enterprisedashboard.service.url=tcp://somehost:2003/EnterpriseDashboard
   ```

5. Save **site.properties** and exit.

6. Restart the registry.

To activate CIC for registries older than Release 9.1:

DevTest uses CIC to support pre-9.1 registries with Enterprise Dashboard 9.1 or later. If your registries are Release 9.1 or later, you do not need to activate CIC.

1. Start Enterprise Dashboard
2. Start CIC by executing **LISA_HOME/bin/EnterpriseDashboardCIC.ex**e.
3. Modify the **lisa.enterprisedashboard.service.url** property for every legacy registry to point to the host where CIC is running: lisa.enterprisedashboard.service.url=tcp://<host>:2003/EnterpriseDashboard
4. Restart the legacy registry.
5. Start the DevTest Enterprise Dashboard UI if it is not running.
6. Confirm that the legacy registry appears on the home page display.

Export Dashboard Data

The Enterprise Dashboard lets you export current and historical dashboard data in an Excel format. You can export data from either the main window or the Registry Details window.

The following metrics are exported:

- **Registry Metrics**
  - Number of Workstations Running (Minimum/Maximum)
  - Number of Simulators Running (Minimum/Maximum)
  - Number of Agents Running (Minimum/Maximum)
  - Number of Coordinators Running (Minimum/Maximum)
  - Number of Virtual Service Environments (VSEs) Running (Minimum/Maximum)
- **VSE Metrics**
  - Maximum Number of Models Deployed
- **Coordinator Metrics**
  - Number of Tests Started
- **Simulator Metrics**
  - Maximum Number of VUs Allocated

Follow these steps:

1. Click **Options** in the upper right corner of the window and select **Export To Excel**.
   If you are exporting from the main window, the Export Registry Data window opens.

2. Complete the following fields:

   - **Export live data only**
     Selecting this check box limits your export to live data.
     Clearing this check box exports historical data.

   - **Ping registries**
     Selecting this check box pings each registry to determine the status of the registry to display in the exported file.
     Clearing this check box results in a status of Unknown for each registry in the exported file.

     > **Note:** This option only applies to historical data. For a live data export, the status of each registry is automatically determined.

3. You are prompted to either save or open the exported Excel file.

4. Click one of the following:

   - Click **Save** to save your exported file to a specified directory.

   - Click **Open** to view the exported file.

5. Click **OK**.

Export Usage Audit Data

The DevTest Solutions Usage Audit Report provides details on compliance with your license agreement, which is based on maximum concurrent usage by the following user types:

- Continuous Application Insight Power User
- Service Virtualization Power User
- Application Test Power User
- DevTest Runtime User

You can generate the Usage Audit Report on demand.

Follow these steps:

1. Browse to the Enterprise Dashboard.

   `http://hostname:1506`

2. Click the gear icon and select **Export Usage Audit Data**.
   The Export Usage Audit Data dialog opens.

3. To select a start date and and end date for the date range on which to report, click the calendar icons, then click **OK**.
   The generated report from the specified date range is downloaded to the bottom left of the window.

4. Click the DevTestSolutionsUsageAuditReport.xlsx to open the Excel book containing your report.
   See DevTest Solutions Usage Audit Report for details on each tab.

Purge Dashboard Data

If you are defined as an administrative user in Identity and Access Manager, the Enterprise Dashboard lets you purge historical event logs and metrics that are older than a specified date and time.

Follow these steps:

1. Click **Options** in the upper right corner of the window and select **Purge Data**.
   The Purge Data dialog opens.

2. Select a date and time for the data that you want to purge.
   The purge feature permanently deletes all event logs and metrics from the database that are older than the specified date and time.
   The default values delete event logs and metrics that are 100 days older than the current date and time.

3. Click **OK**.
   A confirmation dialog opens.

4. Click **Yes**.
   The selected data is purged from the Enterprise Dashboard.

Change Dashboard Language

The Enterprise Dashboard lets you change the language that is used on the Dashboard user interface.

Follow these steps:

1. Click **Options** in the upper right corner of the window and select **Language**.
   The available languages are listed.

2. Select a language.

Return On Investment Calculator

The ROI (Return on Investment) calculator allows you to set cost criteria and calculate potential savings from using Service Virtualization Enterprise Edition.

The link to the ROI calculator can be found under the tools section of the dashboard.

Set a specified cost and rate data in each of the fields and which criteria is to be measured and the ROI calculator pulls up to date Transaction and Virtual Service data to calculate savings based on those two real-time criteria.

Various output measurements are available in addition to US Dollars.

The new output measurements are:

- **FTE hours saved**
- **Defects avoided**
- **HW/SW purchases avoided.**

Each criteria comes with a tool tip that explains the underlying logic we use to calculate the savings.

Note: There is an additional option to export data to Excel from the ROI calculator for any reporting needs.

Telemetry

Starting with DevTest Solutions 10.2, the Telemetry feature lets you share product-specific metrics with CA on a daily basis through automated API calls.

Telemetry runs automatically every 24 hours, based on your latest system restart. The data that is shared helps CA Technologies better understand how our clients use CA Service Virtualization and CA Application Test. No actual test data or virtual responses are shared. The application collects usage statistics, CA product version, and platform information.

While not recommended, you can disable the Telemetry feature at any time if you decide to stop sending telemetry data to CA for any reason.

> **Note:** If you would like to enable telemetry for DevTest 10.0 or 10.1, please contact CA Support or your account representative.

- Benefits of Telemetry
- Enabling or Disabling Telemetry

Benefits of Telemetry

The Telemetry feature in CA Service Virtualization and CA Application Test 10.2 and later provides an easy way to collect and share basic system information and provides the following benefits.

- **Centralized System Profile**
  Telemetry provides users and CA with a constantly updated and centralized location for the most current system information.

- **Expedited Support Contact**
  Telemetry data helps to expedite any interactions with CA Support by ensuring the support team is always looking at up-to-date basic system information. This allows for quicker initial assessment of any issues that you might be having.

- **Better Planned Upgrades**
  By having current system versions and other pertinent information, CA can keep you notified of service patches and end of service dates. We can also make sure that you are kept in the loop when newer versions of Service Virtualization are available.

- **Accurate Usage Visibility**
  Telemetry provides accurate product usage metrics to ensure license compliance and improve system expansion when limits are about to be reached. This helps ensure that your organization is making the most efficient use of CA Service Virtualization and CA Application Test.

For more information about the specific telemetry data that is collected, see Telemetry Data.

Enabling or Disabling Telemetry

Telemetry is typically enabled during the installation process. The DevTest Solutions Setup Wizard prompts you to select whether you want to send diagnostic and usage data to CA Technologies.

If at any time you want to change your election, you can do so through the dradis.properties file.

Follow these steps:

1. Open the **dradis.properties** file in the top-level folder of your installation directory.

2. Locate the following property:
   com.ca.prism.enabled=

3. Set this property to **true** or **false** and save your changes.

4. Restart Enterprise Dashboard.

The following video illustrates the steps for enabling and disabling the telemetry feature that are described above.

Telemetry Data

The following metrics are read from the existing Enterprise Dashboard database and are collected for each hour.

> **Note:** A VSE is considered to have been in performance mode for a given hour if the VSE was in performance mode at any time during that hour.

Metric	Description
Operating System	The Operating System of the each registry in your DevTest installation.
Product Version	The version of DevTest Solutions that is running in your environment.
Test Execution Count	Number of tests started during the hour.
Functional VSE Peak Count	Maximum number of VSEs that were in functional VSE status for the entire hour.
Performance VSE Peak Count	Maximum number of VSEs that were run in performance mode at any point during the hour.
Workstation Peak Count	Maximum number of DevTest Workstations that were running during the hour.
Performance VSE Transaction Count	Total number of transactions run by all VSEs that were run in performance mode at any point during the hour.
Functional VSE Transaction Count	Total number of transactions run by all VSEs that were in functional VSE status for the entire hour.
Performance VSE Virtual Service Count	Total number of virtual services deployed on all VSEs that were run in performance mode at any point during the hour.
Functional VSE Virtual Service Count	Total number of virtual services deployed on all VSEs that were in functional VSE status for the entire hour.
App Test Power User - Maximum Concurrent Use Count	Concurrent usage count of two or more App Test power users. This is calculated by the number of overlapping sessions accessing the DevTest user interface or command-line interface simultaneously.
App Test Power User - Instances of Maximum Concurrent Use Count	App Test power users group counts, where the maximum concurrency count is reached.
SV Power User - Maximum Concurrent Use Count	Concurrent usage count of two or more SV Power User/ADMIN users. This is calculated by the number of overlapping sessions accessing the DevTest user interface or command-line interface simultaneouly.
SV Power User - Instances of Maximum Concurrent Use Count	SV power users/Admin group counts, where the maximum concurrency count is reached.

Metric	Description
CAI Power User - Maximum Concurrent Use Count	Concurrent usage count of two or more CAI power users. This is calculated by the number of overlapping sessions accessing the DevTest user interface or command-line interface simultaneouly.
CAI Power User - Instances of Maximum Concurrent Use Count	CAI power user group counts, where the maximum concurrency count is reached.

Note: The Power User counts that are listed above are only available in DevTest 10.3 or later.

Reference

This section provides detailed descriptions of elements that are commonly used in DevTest Solutions.

This section also provides an overview of the REST Invoke API. The overview includes a link to the full API documentation.

- Assertion Descriptions
- Asset Descriptions
- Companion Descriptions
- Correlation Schemes
- Data Set Descriptions
- Event Descriptions
- Filter Descriptions
- Metrics Descriptions
- Property Descriptions
- Test Step Descriptions

Assertion Descriptions

An *assertion* is an element that runs after a step and all its filters have run.

This section describes the following assertions:

- HTTP Assertions
- Database Assertions
- XML Assertions
- JSON Assertions
- Virtual Service Environment Assertion
- Other Assertions

Regular expressions are used for comparison purposes in many assertions. For more information about regular expressions, see Regular Expressions.

HTTP Assertions

The following assertions are available in the HTTP assertions list for any test step:

- 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

Highlight HTML Content for Comparison

The Highlight HTML Content for Comparison assertion lets you base a comparison on the contents of an HTML page. This assertion uses the "paint the screen" technique that is designed to work with HTML pages. For example, if there is a large HTML document, then you can identify the data before and after the "content of interest". Then, you simply identify what to compare the "content of interest" against (typically an expected value that is supplied in a data set).

The text is marked using the icons at the bottom of the editor:

This technique is best explained with an example.

In the following example, we want to ensure that the company name, currently ITKO, that appears in the phrase "Welcome to ITKO examples" matches the value in a specified property. We have marked the text, using the buttons that were shown previously, by selecting text and then clicking the appropriate icon.

- Yellow background indicates text that must appear exactly as shown.
- White background indicates text that does not need to be present, or can change.
- Red background identifies the text that must match the property that was entered into the dialog.

HTML Content for Comparison assertion, Select Property dialog

This screen shows the HTML that is 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 those phrases, 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. This property is compared to the text that appears between the two bounding phrases. The company name text has been replaced with the name of the property.

To execute an assertion, click **Run Assertion**.

When this assertion is run, the value of the property correctCompany is inserted between the phrases "Welcome to" and "examples". The resulting phrase is compared to the corresponding phrase in the HTML response. The phrase "Welcome to correctCompany examples" can change its location in the HTML and it is still found.

Check HTML for Properties in Page

Use the Check HTML for Properties in Page assertion when the web page contains property data that might be used the assertion can use. The property data is made available for assertion by parsing the web page for the following items:

- Meta tags
- Title tags

- Hidden form fields
- Other tags that the product can automatically parse, including <lisaprop> tags and the DevTest Integration API.

Here is a sample of the available properties table.

Check HTML for Properties in Page window listing some properties referenced in page

Complete the following fields:

- **Name**
  Defines the name of the assertion.
- **If**
  Specifies the behavior of the assertion from the drop-down list.
- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

Click **Run Assertion** to execute the assertion.

> **Note:** You may be prompted to install the Parse HTML for Tag filter.

Ensure HTTP Header Contains Expression

The Ensure HTTP Header Contains Expression assertion lets you check that a specific HTTP result header contains a field that matches a specified regular expression.

Complete the following fields:

- **Name**
  Defines the name of the assertion.
- **If**
  Specifies the behavior of the assertion from the drop-down list.
- **then**
  Specifies the step to which to redirect if the assertion fires.
- **Log**
  Identifies event text to print if the assertion fires.
- **Header Field**
  The name of the header field.
- **RegExpression**
  The regular expression that must appear in the header field.

Click **Run Assertion** to execute the assertion.

Check HTTP Response Code

The Check HTTP Response Code assertion lets you verify that the HTTP response code matches a specified regular expression.

Complete the following fields:

- **Name**
  Defines the name of the assertion.
- **If**
  Specifies the behavior of the assertion from the drop-down list.
- **then**
  Specifies the step to which to redirect if the assertion fires.
- **Log**
  Identifies event text to print if the assertion fires.

- **RegExpression**
  The regular expression that must appear in the response code. For example, to verify that the HTTP response code is in the 400-499 range, set the **RegExpression** to 4\d\d.

Click **Run Assertion** to execute the assertion.

Simple Web Assertion

The Simple Web 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, the assertion returns true.

The **multi-tier-combo** test case in the **examples** project has this type of assertion.

Complete the following fields:

- **Name**
  Defines the name of the assertion.

- **If**
  Specifies the behavior of the assertion from the drop-down list.

- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

Click **Run Assertion** to execute the assertion.

Check Links on Web Responses

The Check Links on Web Responses assertion verifies that every link on the returned web page is valid and does not return an HTTP error like a 404 error, or others. This assertion is commonly used to ensure that the links are working properly across the application and there are no inactive links on the page.

Complete the following fields:

- **Name**
  Defines the name of the assertion.

- **If**
  Specifies the behavior of the assertion from the drop-down list.

- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

The following criteria can be checked for the links:

- **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 they 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>)**
  The current web page is checked for script and links.

- **Skip Links Matching RegEx**
  Enter a RegEx expression for any links you want to skip.

Database Assertions

The following assertions are available in the Database assertions list for any test step:

- Ensure Result Set Size
- Ensure Result Set Contains Expression

Ensure Result Set Size

The Ensure Result Set Size assertion counts the rows in a result set and verifies that the size falls between an upper and lower value.

An example of this assertion could be verifying that the number of rows in an HTML table matches a specified expected value from a data set.

Complete the following fields:

- **Name**
  Defines the name of the assertion.

- **If**
  Specifies the behavior of the assertion from the drop-down list.

- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

- **Result set has warnings**
  If selected, the database could return warnings in the result set. To determine whether your database supports warnings in the result set, consult your system administrator.

- **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 **Run Assertion** to execute the assertion.

For example, to ensure that a Database Assertion step returns one, and only one, row, set the **Row Count >=** field to "1" and the **Row Count <=** field to "1".

Ensure Result Set Contains Expression

The Ensure Result Set Contains Expression assertion verifies that a supplied expression matches at least one value in a specific column in a result set.

Complete the following fields:

- **Name**
  Defines the name of the assertion.
- **If**
  Specifies the behavior of the assertion from the drop-down list.
- **then**
  Specifies the step to which to redirect if the assertion fires.
- **Log**
  Identifies event text to print if the assertion fires.
- **Column**
  Defines a column that contains the text to verify. This value can be a column name or an index.
- **Regular Expression**
  The regular expression to match in the column.

Click **Run Assertion** to execute the assertion

For example, to verify that at least one row that a query returned has a login value that starts with "wp", set the **Column** field to "login" and the **Regular Expression** field to "wp.*".

XML Assertions

The following assertions are available in the XML assertions list for any test step:

- Highlight Text for Comparison
- Ensure Result Contains String
- Ensure Step Response Time
- Graphical XML Side-by-Side Comparison
- XML XPath Assertion
- Ensure XML Validation

Highlight Text for Comparison

The Highlight Text Content for Comparison assertion uses the "paint the screen" technique that is designed to work with HTML pages. For example, if there is a large HTML document, then you identify the data before and after the content of interest. Then, you identify what to compare the content of interest against (usually this content is an expected value that is supplied in a data set).

Mark the text with the icons at the bottom of the editor:

In the following example, we want to:

- Ensure that the buffer contains specific files
- Compare the size of one of the files to the value of a property

We have marked the text using the three icons that were shown in the previous graphic, by selecting text, and then clicking the appropriate icon.

- Yellow background indicates text that must appear exactly as shown.
- White background indicates text that does not need to be present, or can change.
- Red background identifies the text that must match the property that is entered into the dialog.

Highlight Text Content for Comparison assertion window with Select Property dialog

The set of tokens that is shown in the previous graphic can be read this way:

- The buffer must start with the phrase in yellow: "Snapshot of: C:\Lisa\".
- A number of files may or may not be in the buffer in the next token. Because the next token is an "Any" token, the variance is immaterial.
- The file "i4jinst.dll" and "rw" attributes must appear.
  The red filesize means that the value associated with the property key filesize is swapped into the expression, and then the comparison made.
- The text "06/08/2011" must appear.
- The file "install.prop" must appear.
- The buffer can have any amount of content afterward.

After you have finished the markup, enter the following parameters:

- **Name**
  Defines the name of the assertion.

- **If**
  Specifies the behavior of the assertion from the drop-down list.

- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

Click **Run Assertion** to execute the assertion.

> **Note:** "Must" blocks must always appear on both sides of "Property" blocks.

Ensure Result Contains String

The Ensure Result Contains String assertion lets you search the response (as text) for a string.

Complete the following fields:

- **Name**
  Defines the name of the assertion.

- **If**
  Specifies the behavior of the assertion from the drop-down list.

- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

- **Contains String**
  The string to search for in the step result - this string can contain a property. To use a property, put the property name in double braces. An example is {{color}}, where the color property is defined and has a value.

Ensure Step Response Time

The Ensure Step Response Time assertion lets you define the minimum and maximum bounds on the response time and assert that the response time is in those bounds.

Complete the following fields:

- **Name**
  Defines the name of the assertion.

- **If**
  Specifies the behavior of the assertion from the drop-down list.

- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

- **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 value is ignored if set to -1.

> **Note:** Parameters can contain properties.

Graphical XML Side-by-Side Comparison

The Graphical XML Side-by-Side Comparison assertion lets you compare a test XML value received from a test with a control XML value. If the responses are the same or different, the assertion can return true. This assertion provides the flexibility to compare XML documents at various steps in a business process to ensure they match expected criteria. This approach is known as "exclusive" testing, where an entire response is compared except for a few values that are known to change.

The assertion editor works by comparing a left and right side of XML to each other. The left side is known as Control Content. The right side is known as Test Content. For example, Control Content is the expected XML returned from a web service in the application under test, while the Test Content is actual content. By default, Test Content is loaded from the last response of the test step that is associated with the assertion, signified by the empty property key. Otherwise, any valid property key can be used and the Test Content is loaded from it.

Alternatively, you can complete one of the following actions in test case authoring mode so that a quick graphical diff can be performed:

- Load XML from a file

- Enter XML manually for Control and Test Content

To perform the diff, click the green arrow.

Graphical XML Side-by-Side Comparison assertion window before diff is performed

After a diff is executed, the results appear in the visualizer in the Diff Viewer tab in the assertion editor.

Graphical XML Side-by-Side Comparison assertion window after diff is performed

Output During Execution

When an assertion is executed, DevTest logs the diff results as test events.

An Info message EventID containing the XML diff results is always logged.

If the assertion fires, an Assertion fired EventID containing the XML diff results is logged.

The diff results are reported in a format resembling the original UNIX diff utility. An example of a text diff report is:

```
Assert [Assert1] fired false of type Graphical XML Diff Assertion
XML is [Different]
=====
1,2[ELEMENT_NAME_CHANGED]1,2
<! <test2>
<! </test2>
---
>! <test>
>! </test>
```

Each difference is displayed with a heading of the format:

```
<First Start Line>, <First End Line>'['<Diff Type>']'<Second Start Line>,
<Second End Line>
```

Then the difference in the first content is displayed, followed by the separator '---', followed by the difference in the second content.

The + character indicates addition, the - character indicates a deletion, and the ! character indicates a change. When these characters are present, they indicate that an actual change occurred on the line of content (instead of to a context line).

XML Compare Options

The following comparison options are available for use by the diff engine:

- **General**

  - **Case sensitive**
    Whether case sensitivity is used during the comparison (enabled by default).

- **Whitespace**

  - **Trim whitespace**
    During a comparison, all leading and trailing whitespace is removed from element text and attribute values (enabled by default).

  - **Normalize whitespace**
    Convert all white space characters to single space characters.

  - **Ignore all whitespace**
    Remove all whitespace before the comparison.

  - **Compare all whitespace**
    Do nothing to the whitespace and then compare.

  - **Collapse whitespace**
    Along with trimming whitespace, any sequence of one or more whitespace characters inside text is converted to a single space character.

- **Namespaces**

  - **Ignore namespaces**
    The namespace value of an element or attribute is ignored.

  - **Ignore namespace prefixes**
    The namespace prefix of an element or attribute is ignored (enabled by default).

- **Ordering**

  - **Ignore child element ordering**
    Ignore the order of child elements in the XML document.

  - **Ignore attribute ordering**
    Ignore the order of attributes in the XML document (enabled by default).

- **Node Types**

  - **Ignore element text**
    Ignore all element text.

  - **Ignore attribute values**
    Compare attribute names but ignore attribute values.

  - **Ignore attributes**
    Ignore attribute names and values.

  - **Ignored Nodes**
    Ignored nodes are created from a list of XPaths that are run against the left and right documents. Each evaluated XPath that returns a node set is aggregated. When the diff occurs, any node that is found in the aggregate set is ignored.

Ignored node XPaths can be any arbitrary query that returns a node set. For example, the XPath **//*** excludes all nodes in an XML document. **/example/text()** excludes the first text node child of the **example** element in an XML document. **/example/@myattr** is the XPath to ignore the **myattr** attribute, including the attribute text value, belonging to the **example** element in an XML document.

Graphical XML Side-by-Side Comparison assertion window Ignored Nodes tab

A right-click menu item also lets a node be selected directly inside an XML document and its XPath is added to the **Ignored Nodes** list.

Graphical XML Side-by-Side Comparison assertion window Ignored Nodes tab,

XML XPath Assertion

The XML XPath assertion lets you use an XPath query that runs on the response. When this assertion is selected, the last response is loaded into the content panel.

Think of XPath as the "SQL for XML." XPath is a powerful query language that makes parsing the XML simple. XPath assertions are useful for validating a web service response in a more sophisticated way than simply parsing the entire result for a specific string. For example, you can ensure the second and third order item contains **price** and the value of those line items is greater than ten.

You can view the response as an XML document or as a DOM Tree. However, you can only make the XPath selection from the DOM Tree view.

You can construct the XPath query in the following ways:

- Manually enter the XPath expression in the XPath Query text box.
- Select an element from the DOM tree and let DevTest construct the XPath expression.
- Select an element from the DOM tree, and then edit the XPath that DevTest constructs. For example, you can modify it to use a property, or a counter data set.

Complete the following fields:

- **Name**
  Defines the name of the assertion.
- **If**
  Specifies the behavior of the assertion from the drop-down list.
- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

Now construct the XPath query using one of the methods that was described previously.

After an XPath query has been constructed, test it by clicking the **Run Assertion** button on the top of the panel. The results of the query are displayed in the **Query Results** panel.

The previous example uses the fourth occurrence of the <wsdl:part> tag.

A common use case is to select an XPath node in a web service result and compare the node to a text value. You can then assert that a response contains the expected value. For this common use case, you can add an equality operator to the end of the initial expression that is provided. For example, if we select the new password that is returned in the response (BobPass):

DevTest builds the following XPath expression:

```
string(/env:Envelope/env:Body/ns2:updatePasswordResponse/[name()='return']/[name()
='pwd'])=
```

Adding ='NewPassword' (as in the following example) compares the string of the result to the value of the property that is used to set the new password. If the equality test does not match, the assertion fails.

```
string(/env:Envelope/env:Body/ns2:updatePasswordResponse/[name()='return']/[name()
='pwd'])='NewPassword'
```

This expression checks the web service response, looking for the new password and making sure it matches as expected.

Ensure XML Validation

The Ensure XML Validation assertion lets you validate an XML document. You can verify whether the XML document is well-formed, you can validate against a Document Type Definition (DTD), or you can validate against one or more schemas. If you have an XML fragment, you can specify to have DevTest add the XML declaration tag. You can also specify that DevTest reports warnings as errors. You enter the XML to validate as a property.

Complete the following fields:

- **Name**
  Defines the name of the assertion.

- **If**
  Specifies the behavior of the assertion from the drop-down list.

- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

- **Source**
  The property that contains the XML. If this field is left blank, the last response is used.

- **Validate**
  Multiple validation options 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, an XML declaration is added to the top of the XML fragment.

- **Treat Warnings As Errors**
  Warnings are reported as errors.

- **Honor All Schema Locations**
  If you have multiple imports for the same namespace, this option opens each schema location instead of only the first one.

Click **Run Assertion** to execute the 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**. You can use the **Validation Type** option buttons to select 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. If you select this option, you can select the errors to ignore in the **Validation Error List**.

Schemas Tab

Ensure XML Validation Schemas tab

Enter the information for each schema you want to use in the validation. You can also specify the default schema:

- **Default Schema URL**
  Optionally, specify the default URL of the schema.

- **WSDL URL**
  Optionally, specify a URL of a WSDL.

JSON Assertions

The following assertions are available in the JSON assertions list for any test step:

- Ensure Result Equals
- Ensure Result Contains
- Ensure JSON Schema

Ensure Result Equals

The Ensure Result Equals assertion lets you ensure that a JSON Path result is equal to an expected value.

The following graphic shows the editor:

Screen capture of Ensure Result Equals assertion.

Complete the following fields:

- **Name**
  Defines the name of the assertion.

- **If**
  Specifies the behavior of the assertion from the drop-down list.

- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

- **JSON Path**
  Designates an expression that consists of a sequence of JSON properties in a JSON document. The **JSON Path** represents a path to a destination JSON property.

- **Expected value**
  Defines the expected value of the **JSON Path** result. Two arrays are considered equal when the order of the elements in both arrays is equal, because an array is an ordered list.

- **Run Assertion**
  To run and execute the assertion, click **Run Assertion**.

- **Ignore order of array elements**
  Two JSON arrays with the same set of elements but different element order are considered equal in comparison.

If you click **Run Assertion** and the assertion result is false, the **Run Assertion Result** area displays information about the failure.

Determining the JSON Path and Expected Value

The right portion of the assertion editor can help you determine the values of the **JSON Path** and **Expected value** fields. First, use one of the following approaches to load the expected JSON:

- **From Content**
  Select a file that contains the expected JSON.

- **From URL**
  Enter a URL path to the expected JSON.

- **From Property**
  Enter a property that defines the expected JSON. The default value is the response property for the test case.

After you load the expected JSON, it appears in the **Name**, **Type**, and **Value** columns.

Select a node in the **Name** column. The **JSON Path** and **Expected value** fields are populated.

> **Note:** The expected JSON in the right panel is not persistent. If you save, close, and reopen the test case, the expected JSON no longer appears in the right panel.

Secure Prefixes

JSON has a security issue that can be addressed by adding a prefix. The prefix prevents an attacker from hijacking the JSON.

Before this assertion performs the comparison, it removes any prefixes from the JSON content.

Ensure Result Contains

The Ensure Result Contains assertion lets you ensure that a JSON Path result that is a JSON object or a JSON array contains any or all of the values in a specified list.

Complete the following fields:

- **Name**
  Defines the name of the assertion.

- **If**
  Specifies the behavior of the assertion from the drop-down list.

- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

- **JSON Path**
  Designates an expression that consists of a sequence of JSON properties in a JSON document. The **JSON Path** represents a path to a destination JSON property.

- **Contains all expected values or Contains any expected values**
  Defines the parameters with which to match the **JSON Path** and the **Expected value**. If you select the **Contains all expected values** check box, the **JSON Path** result must include all values in the **Expected value** list to pass the assertion. If you select the **Contains any expected values** check box, the **JSON Path** result must include at least one value in the **Expected value** list to pass the assertion.

- **Expected value**
  Defines a list of values that the **JSON Path** result must include. To stop editing values in the **Expected value** field, hold down **Cntl** and click **Enter**. For an OS X system, hold down **Command** and click **Enter**.

- **Ignore order of array elements**
  Two JSON arrays with the same set of elements but different element order are considered equal in comparison.

- **Run Assertion**
  To run and execute the filter, click **Run Assertion**.

Secure Prefixes

JSON has a security issue that can be addressed by adding a prefix. The prefix prevents an attacker from hijacking the JSON.

Before this assertion performs the comparison, it removes any prefixes from the JSON content.

Ensure JSON Schema

The Ensure JSON Schema assertion lets you ensure that a JSON schema is valid and that the payload of a JSON response is valid for that schema.

Complete the following fields:

- **Name**
  Defines the name of the assertion.

- **If**
  Specifies the behavior of the assertion from the drop-down list.

- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

- **Validate Payload Against Schema**
  Indicates whether to validate both the JSON schema and the payload (the response). When cleared, only the schema is validated.
  Default: selected

Select Contents Tab

The **Select Contents** tab lets you select the origin of the JSON schema and the payload, and inspect their contents.

Complete the following fields:

- **Schema**
  Indicates the source of the JSON schema for this assertion. From the drop-down list, select from:

  - **From Content**: Loads the JSON schema from a flat file on your file system. To select the file, click **Browse File Selection**.

  - **From URL**: Loads the JSON schema from a URL. Enter the URL in the **URL Path** field. To browse the file system, click Browse . To reload the URL, click Refresh .

  - **From Property**: Loads the JSON schema from a property. To refresh the property value, click Refresh .

- **Payload**
  Indicates the source of the payload, or JSON text. From the drop-down list, select from:

  - **From Content**: Loads the JSON schema from a flat file on your file system. To select the file, click **Browse File Selection**.

  - **From URL**: Loads the JSON schema from a URL. Enter the URL in the **URL Path** field. To browse the file system, click Browse . To reload the URL, click Refresh .

  - **From Property**: Loads the JSON schema from a property. To refresh the property value, click Refresh .

Settings Tab

The Settings tab lets you set advanced options for the Ensure JSON Schema assertion.

Complete the following fields:

- **Use Subschema**
  Instructs the application to look for a specific subschema. Enter the subschema in the **Subschema Filter** field.

- **Referencing Mode**
  Indicates the dereferencing mode to use.
  To dereference all resolved URIs, select **Canonical**. To dereference URIs within the schema, select **Inline**.
  Default: Canonical

- **Run Assertion**
  To run and execute the filter, click **Run Assertion**.

Virtual Service Environment Assertion

The following assertion is available in the Virtual Service Environment assertions list for any test step:

- Assert on Execution Mode

Assert on Execution Mode

The Assert on Execution Mode assertion checks the current execution mode to its reference and fires if they match. This assertion is primarily used to control step flow for virtual service models.

Complete the following fields:

- **Name**
  Defines the name of the assertion.

- **If**
  Specifies the behavior of the assertion from the drop-down list.

- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

- **Execution Mode**
  Select the execution mode from the available options in the drop-down.

Click **Run Assertion** to execute the assertion.

Other Assertions

The following assertions are available in the Other assertions list for any test step:

- Highlight Text Content for Comparison
- Ensure Non-Empty Result Assertion
- Ensure Result Contains String Assertion
- Ensure Result Contains Expression Assertion
- Ensure Property Matches Expression Assertion
- Ensure Step Response Time Assertion
- Scripted Assertion
- Ensure Properties Are Equal Assertion
- Assert on Invocation Exception Assertion

- File Watcher Assertion
- Check Content of Collection Object Assertion
- WS-I Basic Profile 1.1 Assertion
- Messaging VSE Workflow Assertion
- Validate SWIFT Message Assertion

Highlight Text Content for Comparison

The Highlight Text Content for Comparison assertion uses the "paint the screen" technique that was designed to work with HTML pages. For example, if there is a large HTML document, you identify the data before and after the content of interest. Then, you simply identify what to compare the "content of interest" against (typically an expected value that is supplied in a data set).

The text is marked using the icons at the bottom of the editor:

This technique is best explained by using an example.

In the following example, we want to complete the following actions:

- Ensure that specific files appear in the buffer
- Compare one of the file sizes to the value of a property

The text is marked using the three icons that are shown in the previous graphic, by selecting text and clicking the appropriate icon.

- Yellow background indicates text that must appear exactly as shown.
- White background indicates text that does not need to be present, or can change.
- Red background identifies the text that must match the property that was entered into the dialog.

Highlight Text Content for Comparison assertion window with Select Property dialog

The set of tokens that is shown here can be read this way:

- The buffer must start with the phrase in yellow: "Snapshot of: C:\Lisa\".
- The number of files in the buffer in the next token can vary. Because the token is an "Any" token, the variance is immaterial.
- The file "i4jinst.dll" and "rw" attributes must appear.
- The red filesize means that the value associated with the property key filesize are swapped into the expression, then the comparison made.
- The text "06/08/2011" must appear.
- The file "install.prop" must appear.
- The buffer can have any amount of content afterward.

After you have finished the markup, enter the following parameters:

- **Name**
  Defines the name of the assertion.

- **If**
  Specifies the behavior of the assertion from the drop-down list.

- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

Click **Run Assertion** to execute the assertion.

> **Note:** "Must" blocks must always appear on both sides of "Property" blocks.

Ensure Non-Empty Result Assertion

The Ensure Non-Empty Result assertion checks the return value from the step to verify that some value has been returned. If there is no response (timeout) or the return value has a length of 0, the return value is considered empty. When the If field is set to True, this assertion fails the test unless it receives an empty response. When the If field is set to False, this assertion fails the test when it receives an empty response.

If the result is NULL, a Fail event is raised and the assertion's evaluate() method returns. Therefore, the If/then logic of the assertion is never reached.

Complete the following fields:

- **Name**
  Defines the name of the assertion.

- **If**
  Specifies the behavior of the assertion from the drop-down list.

- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

Click **Run Assertion** to execute the assertion.

No other attributes are required.

> **Note:** Use this assertion carefully, because it does not have any content validation.

Ensure Result Contains String Assertion

If the Ensure Result Contains String assertion finds the value being searched anywhere in the response, the assertion returns true. This assertion is typically used to ensure that the response contains a required value such as a unique ID that was supplied during the request.

For more information, see Ensure Result Contains String.

Ensure Result Contains Expression Assertion

The Ensure Result Contains Expression assertion lets you verify that a specified regular expression occurs somewhere in the result, as text.

Complete the following fields:

- **Name**
  Defines the name of the assertion.
- **If**
  Specifies the behavior of the assertion from the drop-down list.
- **then**
  Specifies the step to which to redirect if the assertion fires.
- **Log**
  Identifies event text to print if the assertion fires.
- **Regular Expression**
  Defines the regular expression for which to search in the step result. For example, to verify that the result contains a number in the 400s, set this parameter to "4/d/d".

Click **Run Assertion** to execute the assertion.

Ensure Property Matches Expression Assertion

The Ensure Property Matches Expression assertion lets you verify that the current value of a property matches a specified regular expression.

Complete the following fields:

- **Name**
  Defines the name of the assertion.
- **If**
  Specifies the behavior of the assertion from the drop-down list.
- **then**
  Specifies the step to which to redirect if the assertion fires.
- **Log**
  Identifies event text to print if the assertion fires.
- **Property Key**
  The name of the property to be checked. Enter the property name, select from the drop-down list of properties, select an existing string pattern, or create a string pattern.

- **RegExpression**
  The regular expression that must appear in the current value of the property.

Click **Run Assertion** to execute the assertion.

Ensure Step Response Time Assertion

The Ensure Step Response Time assertion lets you 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 quickly can be a sign that the transaction was not properly processed.

For more information, see Ensure Step Response Time.

Scripted Assertion

The Scripted Assertion assertion lets you write and run scripts. The result must be a Boolean, or false is returned.

Complete the following fields:

- **Name**
  Defines the name of the assertion.

- **If**
  Specifies the behavior of the assertion from the drop-down list.

- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

Click **Run Assertion** to execute the assertion.

- **Language**
  Designates the scripting language to use.
  Values:

  - Applescript (for OS X)

  - Beanshell

  - Freemarker

  - Groovy

  - JavaScript

  - Velocity

  Default: Beanshell
  To use additional scripting languages, see Enabling Additional Scripting Languages.

- **Copy properties into scope**
  Allows you to specify which properties to download for use in the step.
  Values:
  - Test state and system properties: all properties for the test case and system
  - Test state properties: properties that provide information about the test case
  - TestExec and logger only: only the TestExec and logger properties

  Default: Test state properties

Enter your script into the script editor on the left.

All the objects available for use in the script editor are listed in the **Available Objects** panel on the right. The list includes primitive types of data (strings and numbers) and objects such as EJB response objects that were 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.

When you save the test case, the assertion is checked for syntax errors.

Some things to remember:

- If you use a property - **{{someprop}}** - in a script, it is substituted for the property value at run time before the script is executed.
- If you need access to a property with a "." in the name, such properties are imported into the script environment replacing "." with "_". So **{{foo.bar}}** in a script is the same as **foo_bar**.
- You can produce a DevTest log event inside a script step or assertion by using the **testExec** object. To produce a DevTest log event, code the following line, as opposed to using the log4j logger. The **testExec.log()** method causes an actual DevTest event to be raised. You can see the event in the ITR.

  ```
  testExec.log("Got here");
  ```

Ensure Properties Are Equal Assertion

The Ensure Properties Are Equal assertion lets you compare the values of two properties to ensure that they are same. Typically this assertion is used with a data set and supplied "expected value" to ensure that the application functionality is correct.

Complete the following fields:

- **Name**
  Defines the name of the assertion.
- **If**
  Specifies the behavior of the assertion from the drop-down list.
- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

- **First Property**
  The first property in the comparison. Enter the property name, select from the drop-down list of properties, select an existing string pattern, or create a string pattern.

- **Second Property**
  The second property in the comparison. Enter the property name, select from the drop-down list of properties, select an existing string pattern, or create a string pattern.

Click **Run Assertion** to execute the assertion.

Assert on Invocation Exception Assertion

The Assert on Invocation Exception assertion lets you alter the test flow, in regard to the occurrence of a Java exception. The assertion asserts true if a specific Java exception is returned in the response.

Complete the following fields:

- **Name**
  Defines the name of the assertion.

- **Log**
  Identifies event text to print if the assertion fires.

- **Assert**
  Select the behavior of the assertion using the option buttons.

- **Execute**
  Select the step to redirect to if the assertion fires.

- **Expression**
  The expression to search for in the invocation exception. The expression can be a regular expression. The expression '.*' is common.

File Watcher Assertion

The File Watcher assertion lets you monitor a file for specific content, and react to the presence (or absence) of a specific expression. This assertion runs in the background while your test case is executing.

Complete the following fields:

- **Name**
  Defines the name of the assertion.

- **Log**
  Identifies event text to print if the assertion fires.

- **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 File Watcher will give up watching for the expression**
  The total number of seconds this assertion checks for the expression.

- **The url of the file to watch**
  The URL or path to the file being watched.

- **File encoding**
  Accept the default encoding of UTF-8 or select an alternate encoding from the drop-down list. If you select **Platform Default**, the JVM chooses an encoding, which can change across machines and even across JVMs on the same machine.
  We recommend that you select a specific character encoding, such as **UTF-8**.

- **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. The times default to 0.

Check Content of Collection Object Assertion

The Check Content of Collection Object assertion lets you make simple assertions on the contents of a collection. This assertion is a useful way to find out if specific tokens are in the collection, with the option of adding some simple constraints. For example, if data that a bank web service returns includes an account list, this assertion can verify whether the account IDs match expected values.

Complete the following fields:

- **Name**
  Defines the name of the assertion.

- **If**
  Specifies the behavior of the assertion from the drop-down list.

- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

- **Property to Check (blank for whole response)**
  The name of the property holding the object that you want to use in the assertion. Leave this field blank to use the last response. The object must be of type Java Collection or Java Array.

- **Field to Check (blank for "toString")**
  Enter the name of a field and DevTest calls its get method.

- **Tokens To Find (value1, value2)**
  A comma-delimited string of tokens for which to check.

- **Exact Match Only**
  The token names much match exactly.

- **Must be in this order**
  Select this check box if the tokens must be found in the same order as 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.

Click **Run Assertion** to execute the assertion.

WS-I Basic Profile 1.1 Assertion

The WS-I Basic Profile 1.1 assertion lets you get a WS-I Basic Profile compliance report for a specific web service. This report is delivered in the standard format that the WS-I Basic Profile specifies.

WS-I Basic Profile compliance report assertion

Complete the following fields:

- **Name**
  Defines the name of the assertion.

- **Log**
  Identifies event text to print if the assertion fires.

- **Report Type**
  Select one of the following levels of (WS-I) assertions to include in the report:

  - Display All Assertions

  - Display All But Info Assertions

  - Display Only Failed Assertions

    - Display Only Not Passed Assertions

- **Auto-Select Port**
  Determine how the port is selected - a specific port or "Don't Auto-select".

- **Service Name**
  Select a service name from the list. This name can auto-populate from the step.

- **Service Namespace**
  Is automatically populated based on the service name.

- **Port Name**
  Is automatically populated based on the service name.

Click **Test** to run the assertion.

Messaging VSE Workflow Assertion

The Messaging VSE Workflow Assertion assertion is automatically added from the VSE recorder. The assertion serves a specific purpose that makes VSE recordings work properly. Use it with care. If the assertion was added to a step in a VSE model, do not remove or edit it.

Click **Run Assertion** to execute the assertion.

Validate SWIFT Message Assertion

The Validate SWIFT Message assertion lets you validate the syntax and semantics of SWIFT messages.

The assertion includes the following fields:

- **Name**
  Defines the name of the assertion.

- **If**
  Specifies the behavior of the assertion from the drop-down list.

- **then**
  Specifies the step to which to redirect if the assertion fires.

- **Log**
  Identifies event text to print if the assertion fires.

The assertion also includes the following SWIFT-specific fields:

- **SWIFT Message Type**
  Specifies the message type with which to validate the message. The available message types are MT, MX, and SEPA. The default message type is MT.
  For the message type MT, DevTest supports the following categories:

    - **MT1nn**
      Customer Payments

    - **MT2nn**
      Financial Institution Transfers

- **MT3nn**
  FX, Money Market, and Derivatives

- **MT4nn**
  Collections and cash letters

- **MT5nn**
  Securities Markets

- **MT7nn**
  Documentary Credits and Guarantees

- **MT9nn**
  Cash Management and Customer Status

For the message type SEPA, DevTest supports the following categories:

- Resolution of Investigation (camt.029.001.03)

- Payment Cancelation Request (camt.056.001.01)

- Customer Credit Transfer Initiation(pain.001.001.03)

- Customer Payment Status Report (pain.002.001.03)

- Financial Institution Payment Status Report (pacs.002.001.03S2)

- Payment Return (pacs.004.001.02)

- Financial Institution Customer Credit Transfer (pacs.008.001.02)

For the message type MX, DevTest supports the latest versions (as of February 2014) of the full catalog of ISO20022 messages, which are listed at http://www.iso20022.org /full_catalogue.page.

- **Validation Syntax only**
  Specifies whether the assertion validates syntax and semantics.
  Values:

  - **Selected:** The assertion validates only the syntax.

  - **Cleared:** The assertion validates both syntax and semantics.

  Default: Cleared

If you click **Run Assertion**, any validation errors appear in the **System Messages** window.

Notes:

- Ensure that lines in MT messages end with the required Carriage Return/Line Feed characters (0D0A in ASCII hex; 0D25 in EBCDIC hex). Otherwise, the following error is reported.

  ```
  "The input Swift message cannot be parsed because of invalid syntax. P
  lease check the message structure. Take notice of block separators,
  carriage-return line-
  feed characters and the presence of mandatory blocks."
  ```

- Ensure that an MT message has no invalid trailing spaces before the end of a line. The reported errors are sometimes unclear.
  For example, if the following line

  ```
  :32A:071119EUR50000,
  ```

  contains an invalid trailing space before the end of the line, DevTest reports the error message:

  ```
  "T43 - The integer part of Rate must contain at least one digit. A dec
  imal comma is mandatory and is included in the maximum length tag:32A."
  ```

- Ensure that the MX messages conform to the supported versions.
  For example, **camt.052.001.04** is supported but the older **camt.052.001.01** is not.

Asset Descriptions

An *asset* is a set of configuration properties that are grouped into a logical unit.

This section describes the following assets:

- Apache Kafka Assets
- Email Connection Asset
- IBM MQ Native Assets
- JDBC Connection Assets
- JMS Client Assets
- JNDI Initial Context Asset
- RabbitMQ Assets
- SAP JCo Destination Assets
- SSL Assets

Apache Kafka Assets

You can create the following types of assets for Apache Kafka:

- Kafka Connection
- Kafka Topic
- Kafka Consumer and Producer

In the editor for each asset, each parameter has a tooltip that describes the purpose of the parameter.

Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Kafka Connection

Apache Kafka does not have a common connection object. The actual connection is encapsulated within a producer or consumer. The Kafka Connection asset lets you define the configuration elements that are common to both producers and consumers. The only required configuration is the Bootstrap Servers section, which lists the host and port for each Kafka server in the cluster.

Click **Add** to add an additional host and port pair.

> **Note:** If you list multiple bootstrap servers, any connection that you make using this connection asset must be able to access *all* of the servers in the list. This is not a fail over list where if one connection fails, the system moves to the next connection.

Screen capture of the Kafka Connection asset

To display the advanced mode, click **PRO** in the upper right corner of the asset editor. The following additional configuration options display:

- **Common Config**
  This section contains the configuration options that are common to both consumers and producers.

- **Default Producer Config**
  This section contains the configuration options that are specific to producers. The options that you define here become the default configuration for each producer that is created through this connection asset. However, you can override these values for an individual producer through the Kafka Producer asset.

- **Default Consumer Config**
  This section contains the configuration options that are specific to consumers. The options that you define here become the default configuration for each consumer that is created through this connection asset. However, you can override these values for an individual consumer through the Kafka Consumer asset.

Kafka Topic

The Kafka Topic asset defines the name of a topic and references the Kafka connection that it belongs to.

Screen capture of the Kafka Topic asset

To display the advanced mode, click **PRO** in the upper right corner of the asset editor. The following additional configuration options display:

- **Partition Assignment**
  Defines the partition that is used when producing or consuming messages.

  - **Default**
    If a topic has multiple partitions, Kafka selects the partition to use, based on load balancing considerations.

  - **Specific**
    Allows you to specify the specific partition to use in the **Partition** field. Any send operation that uses this topic always sends its messages to the specified partition. Any receive operation that uses this topic will always listen on the specified partition.

  - **All**
    Any receive operation that uses this topic listens to all partitions. The consumer queries Kafka to get a listing of all partitions for this topic and subscribes to all of them.

  > **Note:** If you want to listen to more than one partition of the same topic but not all partitions, you must create a separate Kafka Topic asset for each partition. If you have a consumer with multiple topics, and some are using the Specific or All partition assignment, then all topics for that consumer must be defined as Specific or All. For any partition left as Default, DevTest selects partition 0.

- **Consume Offset Type**
  Defines where in the topic log that a consumer starts consuming messages.

  - **Default**
    Kafka automatically selects the point at which to start consuming messages.

  - **Absolute**
    Allows you to select the specific offset in the **Consume Offset** field. This offset is measured from the beginning of a topic's lifetime.

- **Earliest**
  Starts from the earliest available message on the topic and then adds the value in the **Consume Offset** field to define the point at which to start consuming messages. Leaving **Consume Offset** as **0** starts the consumer at the earliest available message.

- **Latest**
  Starts from the last available message on the topic and then subtracts the value in the Consume Offset field to define the point at which to start consuming messages. If you leave **Consume Offset** as **0,** the consumer only sees the last available message. If you set **Consume Offset** to **9**, the consumer sees the last 10 messages on the topic.

Kafka Consumer and Producer

For most basic use cases, you do not need to create your own producers or consumers.

For more information about configuring the Kafka Consumer and Producer for advanced use cases, see Kafka Advanced Use Cases.

Email Connection Asset

To define the connection information to the SMTP mail server, use a connection asset.

If you have predefined connection assets, you can select one from the **Connection** drop-down list in the step editor. To create an asset from the step editor, click **Add Asset** . To edit an asset from the step editor, click **Edit Asset** .

To create an asset:

1. Define the following fields for this asset. You can use properties for SMTP connection parameters.

   - **Name**
     Defines the name of the asset. This name appears in the **Connection** field in the step editor. Use a name that is meaningful for your SMTP mail system.

   - **Description**
     Specifies information that provides more details about the system that the asset targets.

   - **Server**
     Specifies the name of the SMTP mail server.

   - **Security**
     Specifies which type of encryption to use to secure a communication channel.
     Values:

     - **None:** The communication uses no encryption.

     - **SSL/TLS:** The communication uses SSL/TLS encryption.

- **Port**
  (Optional) Specifies the port on which the SMTP mail server connects.
  Defaults:
  - **25:** The default when **None** is specified for **Security**.
  - **465**: The default when **SSL/TLS** is specified for **Security**.
- **Authentication**
  Specifies whether to use authentication to connect to an email serverl.
  Values:
  - **Off:** The email server requires no authentication.
  - **Password Authentication:** The email server requires user and password authentication.
- **User**
  (Optional) Specifies the user ID that the SMTP mail server uses to authenticate the connection. This field is disabled when **Authentication** is **Off**.
- **Password**
  (Optional) Specifies the password that the SMTP mail server validates. This field is disabled when **Authentication** is **Off**.

2. To display the advanced mode, click **PRO** in the upper right corner of the asset editor. The advanced mode allows you to specify the runtime scope of the asset.

IBM MQ Native Assets

You can create the following types of assets for WebSphere MQ:

- **IBM MQ Native Queue Manager**
  A queue manager provides WebSphere MQ services to applications.
- **IBM MQ Native Queue**
  A queue is where messages are placed in WebSphere MQ.
- **IBM MQ Native Temporary Queue**
  A type of dynamic queue that is deleted when the client that created it closes the queue.

> **Note:** These assets are for WebSphere MQ in native mode. If you are using WebSphere MQ in JMS mode, we recommend that you use the **IBM MQ JMS** assets.

In the editor for each asset, each parameter has a tooltip that describes the purpose of the parameter.

The queue and temporary queue assets require you to specify a queue manager asset.

The following graphic shows a queue manager asset.

Screen capture of queue manager asset.

To add authentication information, click the plus sign and select the **User ID** and **Password** fields.

The following graphic shows a queue asset. The parameter where you specify the queue manager asset is highlighted.

Screen capture of queue asset.

For information about testing the assets, see Verify Assets.

JDBC Connection Assets

To define the connection information to your JDBC system, use a destination asset. The asset class for JDBC connection assets is named **JDBC Connection Assets**.

Prerequisites: The JDBC driver appropriate for your database must be on the DevTest classpath. You can place the driver JAR file in the hot deploy directory. The Derby client driver is included in the DevTest classpath, so you do not need to add it again.

Parameter Requirements: Have the name of the JDBC driver class, the JDBC URL for your database, and a user ID and password for the database. You also must know the schemas for the tables in the database to construct your SQL queries.

If you have predefined destination assets, you can select one from the **Destination** drop-down field in the step editor. To create an asset from the step editor, select the **Add Asset** icon. To edit an asset from the step editor, select the **Edit Asset** icon.

To create an asset:

1. Define the following fields for this asset. You can use properties for connection parameters.

   - **Name**
     The name of the asset. This name appears in the **Destination** field in the step editor. Use a name that is meaningful for your JDBC system.

   - **Description**
     Information that provides more details about the system that the asset targets.

   - **JDBC Driver**
     Enter or select the full package name of the appropriate driver class. Standard driver classes are available in the drop-down list. You can also use the **Browse** button to browse the DevTest class path for the driver class.

   - **Connect String**
     The connect string is the standard JDBC URL for your database. Enter or select the URL. The drop-down list contains JDBC URL templates for the common database managers.

   - **User ID**
     Enter a user ID (if the database requires it).

   - **Password**
     Enter a password (if the database requires it).

   - **Use Connection Pool**
     When you select **Use Connection Pool**, you can configure the connection pool size can be configured with the **lisa.jdbc.asset.pool.size** property in the lisa.properties file.

JDBC Connection assets panel

When **Use Connection Pool** is selected, the simulator that is executing the step checks for the existence of a connection pool based on the following items:

- Class Driver
- JDBC Connect String
- Username
- Password

If a connection pool is not available, CA Service Virtualization creates a new connection pool using the same items that were defined in the step.

2. To display the advanced mode, select the **PRO** icon in the upper right corner of the asset editor. The advanced mode allows you to specify the run-time scope of the asset.

JMS Client Assets

The Java Message Service (JMS) is a specification that allows Java programs to interact with enterprise messaging systems. The original version is 1.0. The latest version is 1.1.

JMS includes the following terminology:

- **JMS client**
  A Java program that uses JMS to communicate with a messaging system.
- **JMS provider**
  A messaging system that implements JMS.

You can create assets for the following types of objects that the JMS clients use:

- Connection factory
- Connection
- Session
- Producer
- Consumer
- Destination

In the editor for each JMS client asset, each parameter has a tooltip that describes the purpose of the parameter.

The following graphic shows how a JMS client handles request messages and response messages.

Connection Factories

A connection factory is used to create connections.

The connection factories can be characterized in terms of how they are initialized:

- **Generic connection factory**
  Initialized by using the Java Naming and Directory Interface (JNDI).

- **Direct connection factory**
  Initialized in a way that is specific to a JMS provider. This type of connection factory has its own, often large, set of parameters.

The connection factories can also be characterized in terms of the destinations that they support:

- **Queue connection factory**
  Supports queues only. This type is from JMS version 1.0.

- **Topic connection factory**
  Supports topics only. This type is from JMS version 1.0.

- **Connection factory**
  If a connection factory is not specified as either of the preceding types, then it supports both queues and topics.

Most connection factory assets are a combination of these two categories. For example, the Direct JMS 1.0 Topic Connection Factory for TIBCO EMS asset is a direct connection factory that supports topics only.

The following connection factory assets are available:

- JMS Generic Connection Factory
- JMS 1.0 Generic Queue Connection Factory
- JMS 1.0 Generic Topic Connection Factory
- Direct JMS Connection Factory for ActiveMQ
- Direct JMS Connection Factory for IBM MQ
- Direct JMS 1.0 Queue Connection Factory for IBM MQ
- Direct JMS 1.0 Topic Connection Factory for IBM MQ
- Direct JMS Connection Factory for JCAPS
- Direct JMS 1.0 Queue Connection Factory for JCAPS
- Direct JMS 1.0 Topic Connection Factory for JCAPS
- Direct JMS Connection Factory for SonicMQ
- Direct JMS 1.0 Queue Connection Factory for SonicMQ
- Direct JMS 1.0 Topic Connection Factory for SonicMQ
- Direct JMS Connection Factory for TIBCO EMS
- Direct JMS 1.0 Queue Connection Factory for TIBCO EMS
- Direct JMS 1.0 Topic Connection Factory for TIBCO EMS

- Direct JMS Connection Factory for Oracle AQ

Notice that ActiveMQ has one version of the connection factory asset, instead of three versions.

Connections

A connection represents an active connection with a JMS provider. Connections are used to create sessions.

Connections can be characterized in terms of the destinations that they support:

- **Queue connection**
  Supports queues only. This type is from JMS version 1.0.

- **Topic connection**
  Supports topics only. This type is from JMS version 1.0.

- **Connection**
  If a connection is not specified as either of the preceding types, then it supports both queues and topics.

A connection must remain open while its sessions are open.

The following connection assets are available:

- JMS Connection

- JMS 1.0 Queue Connection

- JMS 1.0 Topic Connection

- TIBCO Admin Connection

Sessions

A session is used to create producers and consumers.

Sessions can be characterized in terms of the destinations that they support:

- **Queue session**
  Supports queues only. This type is from JMS version 1.0.

- **Topic session**
  Supports topics only. This type is from JMS version 1.0.

- **Session**
  If a session is not specified as either of the preceding types, then it supports both queues and topics.

A session must remain open while its producers and consumers are active.

The following session assets are available:

- JMS Session

- JMS 1.0 Queue Session

- JMS 1.0 Topic Session

Producers

A producer is used to send a message to a destination.

One producer asset is available: JMS Producer.

Consumers

A consumer is used to receive a message from a destination.

One consumer asset is available: JMS Consumer. However, you can use the asset in either of two ways:

- **Synchronously**
  While a client is waiting to receive a message, the client cannot do anything else.

- **Asynchronously**
  While a client is waiting to receive a message, the client can perform other tasks.

Destinations

A destination represents a location on the JMS platform where messages are placed.

Destinations are divided into queues and topics:

- **Queue**
  A destination that supports the point-to-point messaging model. When a message is sent to a queue, only one receiver can receive the message.

- **Topic**
  A destination that supports the publish/subscribe messaging model. When a message is sent to a topic, multiple receivers can receive the message.

Destinations can also be characterized in terms of how they are created:

- **JNDI destination**
  Initialized by using the Java Naming and Directory Interface (JNDI).

- **Temp JNDI destination**
  A JMS session can be used to create a destination temporarily. This destination exists as long as the session is open, and is deleted when the session is closed.

- **Destination**
  A static destination is not specified as either of the preceding types, and is obtained through the JMS session.

Destination assets are a combination of these two categories. For example, the JMS JNDI Queue asset is a point-to-point destination that is initialized through JNDI.

The following destination assets are available:

- JMS Queue

- JMS JNDI Queue

- JMS Temporary Queue
- JMS Topic
- JMS JNDI Topic
- JMS Temporary Topic

JNDI Initial Context Asset

The Java Naming and Directory Interface (JNDI) is a specification that allows Java programs to interact with naming and directory services.

DevTest includes a **JNDI Initial Context** asset. The JMS client assets use the **JNDI Initial Context** asset to locate JMS connection factories and destinations.

In the editor for the **JNDI Initial Context** asset, each parameter has a tooltip that describes the purpose of the parameter.

RabbitMQ Assets

You can create the following types of assets for RabbitMQ:

- RabbitMQ Connection
- RabbitMQ Queue
- RabbitMQ Exchange
- RabbitMQ Temp Queue
- RabbitMQ Channel

In the editor for each asset, each parameter has a tooltip that describes the purpose of the parameter.

Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

- Connection
- Queue
- Exchange
- Temp Queue
- Channel

Connection

The connection asset lets you establish a connection to the RabbitMQ server.

Specify the following information:

- Host name and port number of the RabbitMQ server
- User name and password for accessing the RabbitMQ server

When you configure the connection asset, you can use the green **Verify** button to confirm that the server can be reached.

The following graphic shows a connection asset.

Screen capture of connection asset.

Queue

A queue represents a location where a message can wait to be picked up by a consumer.

Like JMS and WebSphere MQ, more than one client can listen on the same queue. Each message is delivered to only one consumer.

Like JMS and WebSphere MQ, a client can create a temporary queue on demand.

Like WebSphere MQ, a client can specify the name of the temporary queue or let RabbitMQ generate a name.

Unlike JMS, WebSphere MQ, and most other messaging providers, you cannot publish a message directly to a queue. Instead, you publish to an exchange.

The following graphic shows a queue asset.

Screen capture of queue asset.

The **Channel** field must be set to a channel asset or a connection asset.

Exchange

An *exchange* is an additional routing layer that exists between publishers and queues.

A consumer binds the queue it is listening on to an exchange using a routing key. This routing key tells the exchange which messages should be routed to the queue. The exact interpretation of the routing key depends on the type of exchange.

A publisher sends a message to an exchange along with its own routing key. This routing key tells the exchange how to route the message to one or more destination queues. Again, the exact interpretation of the routing key depends on the type of exchange.

The routing key used by the publisher to send a message to the exchange and the routing key used by the consumer that receives the message on its queue are usually matched in some way, but they are not necessarily equal.

If a message is published to an exchange with a routing key that does not match any currently bound queues, the message is discarded. So while the queue side is like a JMS queue, where a message waits to be consumed, the exchange side is like a JMS topic, where a queue has to be bound to the exchange with a matching routing key at the time the message is published in order to receive it. Also like a topic, if more than one queue has binding parameters that match a given message according to the exchange's routing rules, the message is forwarded to all of those queues at the same time.

The exchange types have different routing strategies:

- **Default**
  If you do not specify the name of an exchange when publishing a message, the default exchange is used. Each queue created on a RabbitMQ server, including temp queues, is automatically bound to the default exchange with a routing key equal to the queue name. Therefore, publishing with an empty exchange name and the queue name as the routing key is the closest you can get to publishing directly to a queue.

- **Direct**
  With a direct exchange, each consumer's queue is bound to the exchange with a unique string as its routing key. The routing key might be the name of the queue or something completely different. Publishers send a message to the exchange with a routing key. The message is routed to every queue that is bound with the same routing key.

- **Topic**
  Topic exchanges are almost the same as direct exchanges. However, with a topic exchange the routing key used to bind a queue can be a routing pattern that can match more than one message routing key. Basically, routing patterns are dot-delimited strings where the text between each dot can be a literal string or one of two wildcards. The star wildcard (*) matches one string. The hash wildcard (#) matches zero or more dot-delimited strings.

- **Headers**
  Rather than routing based on the routing key, a headers exchange routes based on one or more custom message properties. A queue is bound to the headers exchange with one or more custom bind properties that must match the custom properties on a message sent to that exchange. By default, to be routed to a queue a message must have properties that match all the properties that the queue was bound with. An additional bind property can change this behavior so that only one of the message's properties must match.

- **Fanout**
  A fanout exchange routes every message to every queue bound to it. It does not use the routing key or any other routing logic.

The **Channel** field must be set to a channel asset or a connection asset.

Temp Queue

Temp queues are probably much more prevalent with RabbitMQ RPC-style services than they are with other messaging providers. The reason is that message filtering is not done at the queue level. All the filtering is done at the exchange level with routing rules. Once a message is passed on to a particular queue, then there it is going to be delivered to one of the consumers on that queue. Basically, a one-to-one correspondence exists between consumers and queues. The only reason to have more than one consumer on the same queue is to load balance or parallelize the processing of messages from that queue; all of those consumers must necessarily do the same processing.

Because each consumer needs its own queue and temp queues are so easy to create, it makes sense to use temp queues for the response side of most RPC applications as a standard practice. Temp queues also work well with the exchange-based correlation schemes. An exchange does not care whether the queue that it is routing messages to is a temp queue or not.

The following graphic shows a temp queue asset.

Screen capture of temp queue asset.

The **Channel** field must be set to a channel asset or a connection asset.

Channel

The channel asset represents a RabbitMQ channel. Typically, you do not need to create this type of asset.

SAP JCo Destination Assets

To define the connection information to your SAP system, use a destination asset. One asset class is used for all three SAP steps. The class is named **SAP JCo Destination Assets**.

If you have predefined destination assets, you can select one from the **Destination** drop-down field in the step editor. To create an asset from the step editor, click **Add Asset** . To edit an asset from the step editor, click **Edit Asset** .

To create an asset:

1. Define the following fields for this asset. You can use properties for SAP connection parameters.

   - **Name**
     The name of the asset. This name appears in the **Destination** field in the step editor. Use a name that is meaningful for your SAP system.

   - **Description**
     Information that provides more details about the system that the asset targets.

   - **Server Type**
     Select **Application Server** or **Message Server**.

- **Host**
  Hostname or IP address of SAP system
- **System Number**
  SAP system number (for Application Server)
- **R/3 Name**
  R/3 name (for Message Server)
- **Group**
  Group of SAP application servers (for Message Server)
- **User**
  SAP username
- **Password**
  SAP user password
- **Client**
  SAP client

The following graphics illustrate destination asset definitions for both application and message servers.

SAP JCo Destination Assets message server

SAP JCo assets - application server

2. To display the advanced mode, select **PRO** in the upper right corner of the asset editor. The advanced mode allows you to specify the runtime scope of the asset.

3. To select advanced SAP connection properties, click **Advanced Connection Settings**. You can override the default value of a setting by entering a value for the setting. Select the settings that you want, and click **OK**.

SSL Assets

Historically, the only way to configure most SSL operations was to define global properties in the **local.properties** file:

```
javax.net.ssl.keyStore=keystore.jks
javax.net.ssl.keyStorePassword=12345
javax.net.ssl.trustStore=truststore.jks
javax.net.ssl.trustStorePassword=12345
```

The problem with this approach is that it reconfigures SSL globally, affecting every SSL connection that is made.

As of release 8.1, the following assets let you define an *SSL context*:

- IBM MQ Native Queue Manager

- Direct JMS Connection Factory for IBM MQ
- Direct JMS 1.0 Queue Connection Factory for IBM MQ
- Direct JMS 1.0 Topic Connection Factory for IBM MQ

As of release 9.0, the RabbitMQ Connection asset lets you define an SSL context.

As of release 9.5, the Direct JMS Connection Factory for ActiveMQ asset lets you define an SSL context.

An SSL context is a way to encapsulate the key store and trust store information in a way that applies to a single operation, not globally.

You can set the SSL context to either of the following SSL assets:

- Key Store File
- SSL Context

A full setup is complicated, involving at least five assets, so there are shortcuts that cover common scenarios. This page describes the scenarios in order from the simplest to the most complex.

- Basic Scenarios
  - Just a trust store: The Key Store File Asset
  - A trust store and key store using the default key: Two Key Store File Assets
- Advanced Scenarios
  - Forgoing the trust store: The "Trust All" Trust Manager Asset
  - Using a specific client key: The Key Manager Asset
  - Full configuration: The "Generic" Trust Manager Asset
- SSL Context Parameters

Basic Scenarios

The first two scenarios mirror what is possible to do through the **local.properties** file.

Just a trust store: The Key Store File Asset

This is the one-way SSL situation, in which you add a server-side certificate to a trust store.

Historically, this scenario would have required the following global configuration in the **local.properties** file:

```
javax.net.ssl.trustStore=truststore.jks
javax.net.ssl.trustStorePassword=12345
```

To configure the scenario with SSL assets, follow these steps:

1. Create a Key Store File asset that references the trust store.
2. Select this asset in the **SSL Context** field.

The following graphic shows this configuration.

Screen capture of Key Store File asset scenario.

This configuration is equivalent to the **local.properties** lines, but applies only to the IBM MQ connection.

> **Note:** Key store files and trust store files use the same file format. It is still called a "key store" in most places, even if it is being used as a trust store.

A trust store and key store using the default key: Two Key Store File Assets

This is the standard two-way SSL situation, where we need a trust store for the server certificate and a key store containing the client private key.

Historically, this scenario would have required the following global configuration in the **local.properties** file:

```
javax.net.ssl.keyStore=keystore.jks
javax.net.ssl.keyStorePassword=12345
javax.net.ssl.trustStore=truststore.jks
javax.net.ssl.trustStorePassword=12345
```

To configure the scenario with SSL assets, follow these steps:

1. Create a Key Store File asset that references the trust store.
2. Create a Key Store File asset that references the client key store.
3. Create an SSL Context asset, and select the trust store and key store assets.
4. Select the SSL Context asset in the **SSL Context** field.

The following graphic shows this configuration.

Screen capture of two Key Store File assets scenario.

Again, this configuration is equivalent to the **local.properties** lines, but applies only to the IBM MQ connection.

Advanced Scenarios

Because you can manipulate the SSL context directly, there are a few things that you can do that are not possible through the **local.properties** file.

Forgoing the trust store: The "Trust All" Trust Manager Asset

If the server is using a self-signed or otherwise untrusted certificate, but you do not care about validating it against a known certificate, you can use the "Trust All" Trust Manager asset.

Follow the above steps, but rather than selecting a specific Key Store asset as the trust store under an SSL Context, create a new Trust All Manager asset.

This is a Trust Manager rather than a Key Store. It basically implements an "acceptance rule" that always passes for any server certificate.

> **Note:** If you want to use the Trust All Manager without a client-side key store, you can leave the Key Store field in the SSL Context blank. However, you cannot select a Trust All Manager directly from the SSL Context field in the IBM MQ Native Queue Manager asset.

Using a specific client key: The Key Manager Asset

Occasionally you run into a key store that contains multiple client-side private keys. When configuring the key store through **local.properties**, there is no way to select which key to use; which key that gets used is undefined.

However, because we are controlling the SSL Context directly, we have the option of telling it which key to use. Instead of adding a Key Store asset directly to the SSL Context under the Key Store field, instead add a Key Manager asset. This allows you to specify the alias of the client-side private key to use. The Key Store asset is now selected under the Key Manager instead of directly under the SSL Context.

Screen capture of Key Manager asset scenario.

This also allows you to specify a password for the private key. This is necessary if the private key's password is different from the key store's password, in fact, this is the only way to support that scenario.

Full configuration: The "Generic" Trust Manager Asset

Similar to the Key Manager asset, there is also a generic Trust Manager asset that wraps a Key Store asset specifying the trust store file. There are no parameters on this Trust Manager asset besides the key store, but it serves two purposes.

- When you select a Key Store asset directly as the Trust Store under an SSL Context asset, what happens internally is that a Trust Manager asset is automatically configured so we conform to Java's SSL API. The same thing applies to the Key Manager if just a Key Store is selected for that. In other words, if you followed the above scenario labeled "Two Key Manager Assets", where just an SSL Context and two Key Stores are provided, what actually gets auto-configured under the covers looks like this:

Screen capture of Generic Trust Manager asset scenario

- The generic Trust Manager asset and the Trust All Manager asset are the same type. As a result, you can override one with the other in child configs if you want to turn certificate checking on or off under certain environments.

SSL Context Parameters

The following SSL Context parameters are available in advanced mode:

- **Protocol**
  Allows you to force it to use a particular version of SSL. The list of options is built dynamically by querying the Java SSL API for supported protocols.

- **Provider**
  Allows you to select a different SSL implementation. The list of options is built dynamically by querying the Java SSL API for installed providers.

- **Verify Hostname**
  Specifies whether to verify the server host name used to establish a TCP connection against the host name provided in the SSL certificate.

- **Secure Random**
  This parameter is reserved for future use.

Companion Descriptions

A *companion* is an element that runs before and after every test case execution.

This section describes the following companions:

- Web Browser Simulation Companion
- Browser Bandwidth Simulation Companion
- HTTP Connection Pool Companion
- Configure DevTest to Use a Web Proxy Companion
- Set Up a Synchronization Point Companion
- Set Up an Aggregate Step Companion

- Execute Event-driven Subprocess Companion
- Observed System VSE Companion
- VSE Think Scale Companion
- Batch Response Think Time Companion
- Recurring Period Think Time Companion
- Messaging Retry Policy Companion
- Create a Sandbox Class Loader for Each Test Companion
- Set Final Step to Execute Companion
- Negative Testing Companion
- Fail Test Case Companion
- XML Diff Ignored Nodes Companion
- JSON Ignored Nodes Companion

Web Browser Simulation Companion

The Web Browser Simulation companion lets you simulate various web browsers. The web browsers identify themselves to a web server using the User-Agent HTTP header. You configure DevTest 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.

To specify the weights, use the default Browser Selection Companion Editor.

To configure the Web Browser Selection companion, enter or edit the list of browser agents and the weights:

Complete the following fields:

- **User-Agent**
  The browser to simulate.

- **Weight**
  The weight for this browser. For example, assume you want to assign weights of 25 percent, 25 percent, and 50 percent for three browsers. Enter the weights as 1, 1, and 2 for the three rows, and 0 for the others (or delete the extra rows).

When the test case is run, DevTest selects one of the browsers to emulate: all HTTP transactions sent from DevTest include the **User-Agent** string for that browser. The selection criteria is random, with each browser weight representing one “chance” that the browser will be selected.

DevTest only selects a browser to emulate when the test case initializes. The browser agent string that DevTest selects remains in effect during the execution of the test case. To simulate a distribution of browsers, run the test case multiple times inside a suite.

To add another user-agent, click **Add** .

To delete a line, click **Delete** .

Browser Bandwidth Simulation Companion

The Browser Bandwidth Simulation companion lets you simulate varied bandwidths for the virtual users. Some testing scenarios call for the simulations of different types of internet connections.

To configure the Browser Bandwidth Simulation companion:

Complete the following fields:

- **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).

- **Weight**
  The weight that is given to this row. For example, to assign weights of 25 percent, 25 percent, and 50percent to three rows, set the Weight columns of the rows to 1, 1, and 2. In the example that is shown, half the virtual users connect at 6000 bytes per second and half at 100,000 bytes per second.

To add a line, use the **Add** button.

To delete a line, use the **Delete** button.

HTTP Connection Pool Companion

The HTTP Connection Pool companion lets you limit the number of HTTP connections for each target server. This companion applies only to the HTTP/HTML Request and REST test steps.

DevTest typically uses one HTTP connection for each virtual user. For example, if you run a test with 100 virtual users, the client and the server each have 100 sockets open.

If you run a load test with thousands of virtual users for each simulator, the underlying operating system can run out of available sockets. In this scenario, use the HTTP Connection Pool companion.

The **Connections Per Target Host** parameter specifies the number of connections to allocate to each unique endpoint.

Assume that a test case has two steps:

- An HTTP step to go to web server 1

- An HTTP step to go to web server 2

The test case has the HTTP Connection Pool companion with a setting of five connections for each target host. The staging document is configured to run 100 virtual users. Because there are two simulator servers, they get 50 virtual users each by default.

Simulator 1 creates five connections to web server 1, and five connections to web server 2. Simulator 2 does the same thing. Each web server now has 10 client connections. At the first HTTP step, a virtual user must wait for one of the five connections to web server 1 to become available. The virtual user uses the connection to make the HTTP call, and the connection goes back into the pool.

The following graphic illustrates this scenario. Simulator 1 has five connections to web server 1 and five connections to web server 2. Simulator 2 has five connections to web server 1 and five connections to web server 2.

Configure DevTest to Use a Web Proxy Companion

The Configure DevTest to use a Web Proxy companion lets you set up a proxy for all web testing steps. If your environment dictates the use of a proxy, use this companion. Proxy information is specific to your organization. Consult your operations team for the proxy settings for your company.

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.

- **Bypass web proxy for these Hosts & Domains**
  Names of the domains and hosts for which you want proxy to be bypassed.

- **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**
  Names of the domains and hosts for which you want the secure proxy to be bypassed.

- **Exclude Simple Hostnames**
  Select to exclude hostnames like localhost or servername.company.com.

- **Proxy Server Authentication**
  The domain name with the user name and password, if necessary, for authenticating to the proxy server.

- **Send Preemptively**
  Select **Wait for Challenge**, **Send BASIC**, or **Send NTLM**.

DevTest can also use its local.properties file to assign a web proxy for all test cases. This file is in the DevTest home directory. Update the **lisa.http.webProxy.host** and **lisa.http.webProxy.port** properties as appropriate and restart DevTest. If you do not already have a local.properties file in the DevTest home directory, rename the existing _local.properties to local.properties, and use it.

Set Up a Synchronization Point Companion

The Set up a Synchronization Point companion lets you select a test step to use as a synchronization point in a test case or a test suite. At a synchronization point, virtual users pause and wait until each one has reached this step. Then all virtual users are released to execute the step simultaneously. This capability is useful when setting up load tests for concurrent testing or peak resource utilization.

For example, you could set up a 100-user test so all users log in to your application and order the same seat in a theater simultaneously.

Synchronization points apply to a single test or you can apply them to all tests in 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, because serial tests cannot access the synchronization point simultaneously by definition. Multiple synchronization points can be defined in a test case or test suite.

To configure the Set up a Synchronization Point companion, enter the following parameters:

- **Sync Point Name**
  The name that you specify for the synchronization point.

- **At Step**
  Select the step for the synchronization point from the drop-down list. The virtual users pause before executing this step.

- **Time out secs (0 for none)**
  The number of seconds to wait for the synchronization to occur. All virtual users must reach the **At Step** before the timeout period elapses.

Set Up an Aggregate Step Companion

The Set up an Aggregate Step companion lets you aggregate and report several physical test steps as one logical step for metrics collection and reporting. The companion automatically sets all included steps to **Quiet**.

Enter the following parameters:

- **Aggregate name**
  Defines the name for the aggregation step. Spaces are allowed.

- **Starting step**
  Specifies the start (first) step of the aggregation.

- **Participants**
  Select the steps to include in the aggregate (exclude the Starting and Ending steps).

- **Ending step**
  Specifies the end (last) step of the aggregation from the pull-down menu.

All reports show the aggregate step.

A test case can have only one Aggregate Step companion.

Execute Event-driven Subprocess Companion

The Execute Event-driven Subprocess companion lets you run a subprocess test case when one or more specified events happen.

Enter the following parameters:

- **Identify Event Types**
  Click **Add** to display a drop-down list of event types. Select any number of events from the list.

  Click **Delete** to delete an event type from the list.

- **Define Properties**
  Select a component name and associated property from the drop-down list. You can add components by clicking **Add** and delete components by clicking **Delete** .

- **Configure Subprocess Call**
  Enter the parameters for the subprocess, as documented in Execute Subprocess.

Observed System VSE Companion

Note: This companion has been deprecated and is no longer supported.

The Observed System VSE companion is used to support modeling response times based on an observed set of real response times.

Before LISA 6.0.6, the response time for an inbound request came from the think time specification on the specific response that was determined for that request. Sometimes, primarily during load and performance testing scenarios, the response times should be modeled after the times from a live system. For example, a live system can show a drop in performance equivalent to twice the nominal response times during peak load. In this case, it is helpful to have VSE emulate this response time curve. It is also helpful to let VSE cover (or play back) this curve over an arbitrary interval to allow, for example, a 12-hour period of observed response time metrics to be played out over a 3-hour test.

The Observed System VSE companion supports these requirements. If a virtual service will provide this behavior, add and configure this companion.

- Configuration Information
- Data Set Source Example

- Observed System Data Provider (Wily) Source
- Run Time Priorities

Configuration Information

The Observed System companion requires the following parameters:

- **Start date/time and End date/time**
  Defines a time "window" in which to read observed response times. These timestamps are inclusive. Timestamp data from your data set or data provider that is outside this window is ignored.

- **Assumed run length**
  Defines a duration that represents the interval over which the virtual service "fits" the response time curve. For the previous example, this value would be set to three hours.
  For example, with the following values:

  - The assumed run length is 30 minutes
  - The Start date/time window is 30 minutes
  - The End date/time window is 30 minutes
  - The buffer size is 10 minutes

  we get the buffer for the first 10 minutes, then for 10-20 minutes, then for 20-30 minutes, then 45-60 minutes.
  With the following values:

  - The assumed run length is 15 minutes
  - The Start date/time window is 30 minutes
  - The End date/time window is 30 minutes
  - The buffer size is 10 minutes

  The first 10 minutes plays back in the first 5 minutes, the second 10 minutes plays back in the second 5 minutes, and the third 10 minutes plays back in the third 5 minutes.

- **Buffer size**
  Defines a duration in minutes and can be used to control how much of the response time data is acquired at once. The parameter defaults to one hour of data at a time.
  For example, an assumed run length of 1 hour and a buffer size of 15 minutes provides the following results:

  - We get the buffer for the first 15 minutes
  - Then for 15-30 minutes
  - Then for 30-45 minutes
  - Then 45-60 minutes
  - Then refetch the data from the first 15 minutes

- **Observed System Data Provider**
  This parameter tells the companion where to get the data. We currently provide a DevTest data set data provider and a CA Application Performance Management (Wily) data provider.

Any data provider must provide the following pieces of information:

- An ID (which is a string)
- A timestamp
- The response time at the timestamp

The DevTest data set data provider requires that the data set provide this information in fields labeled "id", "timestamp" and "responseTime", respectively. The timestamp value, if not an actual Date object, must be a string in the form "yyyy-MM-dd HH:mm:ss.SSS". How the ID maps to any specified inbound request is data-provider specific. For the data set provider, it must match the operation of the request. The CA Application Performance Management (Wily) provider uses a regular expression-based approach.

Note: For DevTest 8.2 and later, the **metric-data-service.jar** from CA APM must be in the lib/core folder in the DevTest directory.

Data Set Source Example

In this example, a data set provides the input for the Observed System VSE companion. The definition of the data that this companion provides lets you change the response time for a transaction, or for multiple transactions over time.

The service image that is shown contains one transaction, and the **Think time spec** is set to 15 milliseconds.

Observed System Companion service image-SCR

The virtual service model that is associated with this service image has a few simple steps.

1. To add a companion, click **Add** under the **Companions** panel.
2. Select VSE Companions, Observed System VSE Companion.

Observed System Companion - adding a companion to a test case

The top part of the panel provides general information about the parameters for the companion.

In this example, the Start date/time and End date/time define a two-hour window, but the Assumed run length is set to one hour. Therefore, for every one hour of VSE run time, the companion goes through two hours of data from the data set data provider. The buffer size of 30 minutes means that VSE accesses the data set every 30 minutes to retrieve data. The buffer size is before scaling. Therefore, with a 30-minute buffer:

- VSE retrieves data from the data set between 10:22 and 10:52.
- VSE scales (in this case) by half so it can calculate correctly.

To disable the companion temporarily, clear the **Enabled** check box.

Any data providers, including DevTest data sets and the Wily Observed System Data Provider, can back the Observed System Companion. Click **(click here to select)**, select **Data Set Source**, and the **Select Data Set Type** button appears. Click **Select Data Set Type**, select **Common DataSets**, then select **Create your own Data Sheet**.

Observed System Companion - Select Data Set Type right-click menu enabled

When the table for Create your own Data Sheet opens, the columns are prepopulated with:

- **id**
  Matched against the Operation Name when the request is being processed.

- **timestamp:**

- **responseTime**
  If the **standardDeviation** value is 0, the **responseTime** defines the interval to use during playback. Use the mathematical formula that is described for the **standardDeviation** column for non-zero values. This response time calculation overrides the think time spec of 15 milliseconds that was set in the service image.

- **standardDeviation**
  Defines the statistical data that represents the standard deviation from the mean value of the response time. The response time that is used during playback is within +/- 3sigma of the average response time. If **x** is the **responseTime** and **y** is the **standardDeviation**, the response time during playback is a random value in the range **(x** -3 **y**) and (**x** +3 **y**).

These columns are required for any type of data set provider that you use to provide information for this companion.

In this example, copy the Operation Name of GET / dsdpTest from the service image and enter it in the **id** field. In the **responseTime** field, enter 150.

Observed System Companion - Select Data Set Type right-click menu enabled

In the **timestamp** field, enter a date and time that falls into the time window that **Start date/time** and **End date/time** define.

To continue the example, we use an existing project with a prepopulated data set.

Observed System Companion example using data set as source

The data set response times define an increase of 100 milliseconds every 30 seconds. Therefore, this companion should always override the Think time spec of 15 milliseconds that is set in the virtual service model.

Note: When you deploy the virtual service, do not enter a **Think time scale** of 0 percent.

Observed System Companion example Deploy Virtual Service dialog

Observed System Data Provider (Wily) Source

The Observed System VSE Companion can also receive input from the CA Application Performance Management (Wily) application. To configure the companion to work with Wily, enter the configuration information in the top part of the panel. Then click **(click here to select)**, and select **Data Set Source** of Wily Source.

Parameters for the Wily Observed System companion are:

- **Web Service URL**
  Specifies the URL for the Wily web service.

- **Agent Regex**
  Specifies a regular expression that identifies the agent.

- **Metric Regex**
  Specifies a regular expression that identifies the metric. This metric RegEx is wrapped as the middle part of the metric RegEx that is sent to the Wily server. The metric RegEx sent to the server is:

  ```
  ".*WebServices\|Server\|<user entered pattern>:Average Response Time.*"
  ```

  For example, if you want to query the metric:

  ```
  "WebServices\|Server\|http_//ejb3\.examples\.itko\.com/\|(.*)
  Average Response Time \(ms\)"
  ```

  you can input

  ```
  "http_//ejb3\.examples\.itko\.com/"
  ```

  This example fetches all the Average Response Time metrics for the web service. To retrieve the metrics for specific operations, use a custom RegEx.
  Default: ".*"

- **Service Username**
  Specifies the user name used to access the Wily service.

- **Service Password**
  Specifies the password that is associated with the **Service Username**, if necessary.

To test the behavior of the companion, stage a quick test.

The test case in this example had one step, an output log message step. You can see the output in the **Test Events** tab of the **Quick Stage Run** window.

Observed System Companion example Quick Stage results

In the config file that is used for this test case, setting the property debug to true means that the log message appears in the output.

Observed System Companion example Properties Editor

Run Time Priorities

The process that is followed at run time is:

- The think time specification from the VSE response is examined when the delay factor is being determined.

- If the think time specification contains state (that is, a double-brace expression), it is evaluated directly and the result is used as the response time for the response.
- If the think time specification does not contain state and the virtual service contains the Observed System companion, the companion is asked to determine the response time for the response.
- If the companion is not present or cannot determine a response time, the think time specification is used as the response time.

For example, an assumed run length of 1 hour and a buffer size of 15 provides the following results:

- We get the buffer for the first 15 minutes
- Then for 15-30 minutes
- Then for 30-45 minutes
- Then 45-60 minutes
- Then refetch the data from the first 15 minutes

VSE Think Scale Companion

The VSE Think Scale companion can be added to a virtual service model. This companion allows the think scale percent for the service to change over time by specifying the graph of think scale changes over time.

VSE Think Scale Companion graph

Click **Add**, **Delete**, and **Move** to add, delete, and reorder Transition Points. You can directly edit the **Delay** and **Think Scale** entries. You can also click and drag the lines in the Timeline display to indicate the delay and scale that you want. The Transition Points table is updated to correspond.

- **Relative TS**
  Calculated based on the delay you enter, in format hh:mm:ss.ms.

- **Delay**
  The interval, in minutes and seconds, to wait before the specified think scale is applied.

- **Think Scale**
  The think scale is a percentage that is applied to the think times in the responses.

- **Start over when timeline runs out**
  If the end of the timeline is reached, start from the first Relative TS.

If you click the label for the right-most tick mark on the horizontal axis, you can adjust the total timeline of the companion. This label is the "1h" label in the previous window. The **Update Timeline Duration** dialog opens. If you update the timeline duration to be shorter than your transition points, you receive a warning that some transition points are removed.

VSE Think Scale Companion Update Timeline Duration dialog

As you move your cursor over the graph, tooltips display that show time on vertical lines and think scale percentages on horizontal lines.

Batch Response Think Time Companion

The Batch Response Think Time companion varies the think scale so that a virtual service model sends a defined number of responses at specified times.

This companion lets you batch responses for a virtual service model. You can specify a schedule for sending responses. For example, you can send 100 responses every day at 8:15. The requests can arrive at any time, but VSE only responds at 8:15. If there are fewer than 100 pending responses, VSE sends all the responses. If there are more than 100 pending responses, VSE sends 100 responses and saves the rest until the next batch.

VSE only responds to incoming requests; it does not generate unsolicited responses. If VSE receives 12 requests, it sends 12 responses, even if the **Quantity** is greater than 12.

This companion is useful only for the SAP transport protocol.

Enter the following parameters in the companion editor:

- **Enabled**
  Specifies whether to enable the companion.

- **Time**
  Defines when to send the responses.
  Values: 00:00 to 23:59.

- **Quantity**
  Defines the maximum number of responses to send.

Click **Add**, **Up**, **Down**, and **Delete** to add, move, and delete rows in the **Response Schedule** table.

- **Repeat daily**
  Specifies whether to send responses each day.

- **Send all at once**
  Specifies how to distribute the responses.
  Values:

  - **Selected:** Sends the responses all at once or as the requests arrive for the specified interval until the size limit is reached.

  - **Cleared:** Spreads the responses evenly between the time for the current batch until the time for the next batch or the scheduled end time.

- **Schedule End Time**
  If the **Send all at once** check box is cleared, this field defines the end time for the last-scheduled time in the **Response Schedule** table.
  Values: 00:00 to 23:59.
  Limits: The value must be later than the last-scheduled time in the **Response Schedule** table.

Example:

Assume that the **Response Schedule** table contains two entries:

- The first entry has the values 08:00 and 10.

- The second entry has the values 08:10 and 10.

Also assume that the **Send all at once** check box is cleared and the **Schedule End Time** field is set to 08:15.

When you deploy the virtual service model, the results of this configuration are:

- One response each minute between 08:00 and 08:10

- One response every 30 seconds between 08:10 and 08:15.

These results assume that the service receives enough requests with matching responses during those periods.

Recurring Period Think Time Companion

The **Recurring Period Think Time** companion adjusts response think time to send a batch of responses for every time period that you specify.

This companion lets you batch responses for a virtual service model. You can specify a schedule for sending responses. For example, you can send 100 responses every hour. The requests can arrive at any time, but VSE only responds on the hour. If there are fewer than 100 pending responses, VSE sends all the responses. If there are more than 100 pending responses, VSE sends 100 responses and saves the rest until the next batch.

VSE only responds to incoming requests; it does not generate unsolicited responses. If VSE receives 12 requests, it sends 12 responses, even if the **Quantity** is greater than 12.

This companion is useful only for the SAP transport protocol.

Enter the following parameters in the companion editor:

- **Enabled**
  Specifies whether to enable the companion.

- **Start**
  Defines when the timer starts.

- Values:
  - Immediately
  - On the Quarter Hour
  - On the Half Hour
  - On the Hour

- **Every**
  Defines the number of minutes or hours to wait between sending responses.

- **Quantity**
  Defines the maximum number of responses to send.

- **Repeat daily**
  Specifies whether to send responses each day.

- **Send all at once**
  Specifies how to distribute the responses.
  Values:

  - **Selected:** Sends the responses all at once or as the requests arrive until the size limit is reached.

  - **Cleared:** Spreads the responses evenly over the configured period.

Example:

Assume that the **Start** field is set to Immediately and the **Every** field is set to 5 minutes. Also assume that the **Quantity** field is set to 10 and the **Send all at once** check box is cleared.

When you deploy the virtual service model, the result of this configuration is one response every 30 seconds. This result assumes that the service receives enough requests that have matching responses during those periods.

Messaging Retry Policy Companion

The Messaging Retry Policy companion lets you configure a retry policy for operations that involve messaging middleware.

When you add the companion to a model, the retry policy applies to each send and receive operation in the model.

If a VSE transaction has two responses and sending the second response fails, only the second response is retried.

If a VSE service has more than one request queue and the connection to one of the queues goes down, only the one failed receive connection is retried.

The following retry policies are available:

- Counting Retry
- Simple Delay

- Exponential Backoff

Counting Retry

The Counting Retry policy retries a set number of times, with a delay between each one. This policy is appropriate when you want retrying to be performed, but limiting the number of failures is more important than successfully retrying.

The following graphic shows the editor for the policy.

Screen capture of Counting Retry policy.

The **Delay** parameter takes a think time expression, such as **30s** or **1m-2m**. If you specify a range, the delay between each retry is randomly chosen within that range, with successive retries having different randomly chosen delays. Supporting a range is useful if the same platform outage affects many VSE services. The retry attempts are spread out, thus reducing the peak polling and reconnection load.

The **Retries** parameter specifies the maximum number of retries to perform. The first failure does not count as a retry. If you set the value to 15, a send or receive operation can run and fail up to 16 times in a row before the execution fails.

Simple Delay

The Simple Delay policy retries until a timeout is exceeded. This policy is appropriate when there is a known downtime for the messaging platform, such as a nightly restart.

The following graphic shows the editor for the policy.

Screen capture of Simple Delay policy.

The **Delay** parameter takes a think time expression and supports a range. If you specify a range, the delay between each retry is randomly chosen within that range, with successive retries having different randomly chosen delays.

The **Timeout** parameter takes a think time expression but does not support a range. The parameter specifies the maximum amount of time, after which retries are abandoned and the execution fails. The timeout starts with the first retry, not with the start of the original failed operation.

Exponential Backoff

The Exponential Backoff policy implements an increasing delay between retries. This policy is appropriate when you have a large range of downtime with potential outages, such as both quick restarts and hours-long outages that can happen with the same platform.

The following graphic shows the editor for the policy.

Screen capture of Exponential Backoff policy.

The **Initial Delay** parameter is in think time format and supports a range. Each successive retry doubles this delay before reaching the **Maximum Delay** value, after which all delays are the maximum delay.

The **Timeout** parameter takes a think time expression but does not support a range. The parameter specifies the maximum amount of time, after which retries are abandoned and the execution fails. The timeout starts with the first retry, not with the start of the original failed operation.

Create a Sandbox Class Loader for Each Test Companion

The Create a Sandbox Class Loader for Each Test companion lets you verify that every test run executes in its own Java Class loader (JVM). This companion is valuable when testing local Java objects that are not designed for multithreaded or multiuser access. Most testing does not require this companion. It is usually only necessary when testing local Java objects with the Dynamic Java Execution step.

To configure the Class Loader Sandbox companion, use the Class Path Sandbox companion editor.

If the **hotDeploy** directory contains the class you want to run, select the **Add Hot Deploy Path Entries** check box.

If the **hotDeploy** directory does not contain the class you want to run, click **Add** to add a line in the **Class Path Directories** list, then add the appropriate class path.

> **Note:** Any Java objects that you want to edit or run must be in the class path or the **hotDeploy** directory. They must not be in the DevTest **lib** or **bin** directories. The Class Loader Sandbox will not work because of the way the Java VM loads classes.

Set Final Step to Execute Companion

The Set Final Step to Execute companion lets you verify that the system under test is left in a consistent state regardless of the test result. You specify which step is always run last when the normal test flow is circumvented.

A common use is to ensure 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.

The final step is executed 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. The results can be seen in the Events tab for the last step that was executed.

The Set Final Step companion Editor is used to set the final step.

To configure the Set Final Step companion, enter the following parameter:

- **Final Step**
  Select the final step to execute from the drop-down list.

A step that is designated as the final step to execute is noted with a green flag icon.

Negative Testing Companion

The Negative Testing companion is useful when you want all your steps to fail. To cause a normal-ending test case to fail if any contained test step passes, use this companion.

This companion has no configuration parameters.

Fail Test Case Companion

If a step in a test case has an error, the Fail Test Case companion marks a test case as failed, even if it ends successfully. The companion registers an event listener for the EVENT_TRANSFAILED event.

Example: A WSDL validation assertion on a web service step has an error. You want to be informed that the validation failed, but you also want the test case to continue.

This companion has no configuration parameters.

XML Diff Ignored Nodes Companion

The XML Diff Ignored Nodes companion lets you enter XPath expressions that return one or more nodes in the following table. The companion uses an OR operation to collect these nodes, then ignores them during all XML diff comparisons for this test case.

JSON Ignored Nodes Companion

The JSON Ignored Nodes companion works with the JSON Ensure Result Equals assertion. Without this companion, the result of applying the JSON Path to a JSON response must be equivalent to the **Expected value** for the assertion to pass. However, some JSON properties (for example, a timestamp) that are returned as part of the response are different each time a response is returned. To exclude these properties from the baseline comparison, enter the **JSON Name** or **JSON Value** into the JSON Ignored Nodes Companion.

JSON Ignored Nodes Companion

JSON Ensure Result Equals assertion

Correlation Schemes

You can enable correlation in the following test steps:

- JMS Send Receive Step
- IBM MQ Native Send Receive Step
- RabbitMQ Send Receive Step

Overview of Correlation Schemes

Consider a messaging service with two clients that are running simultaneously. Each client can send a request message to the same request queue. The service can process the requests in any order and can send the response messages to a response queue. How does each client receive its own response and not receive the response that is meant for the other client?

The most common solution is to incorporate some type of identifier into the request, and copy the identifier into the response. This identifier is known as a *correlation ID*. Each client uses a unique correlation ID. Some mechanism is used to ensure that each client can only receive a response containing the correlation ID for that client.

The following graphic shows an example of correlation IDs being used with two clients.

Correlation schemes differ based on:

- How they route responses to the correct client by correlation ID
- Where the correlation ID is located in the request and response messages

Correlation ID

Most messaging platforms have a correlation ID field. In this scheme, the client generates a unique ID and sends a request message containing that ID in the correlation ID field. Before the service sends the response message, the service copies the correlation ID from the request message to the response message. The client listens for a response message that contains the same correlation ID as the original request.

By default, the correlation ID is automatically generated. To specify the value manually, use the **Manual Value** field.

The **Reuse ID** list indicates whether to generate a new correlation ID for each new transaction or use the same correlation ID throughout the test.

Message ID to Correlation ID

This scheme is similar to the Correlation ID scheme.

Most messaging platforms also have a message ID field, which contains a unique identifier for the message. The message ID is automatically generated.

Before the service sends the response message, the service copies the message ID of the request message to the correlation ID of the response message. The client listens for a response message that contains a correlation ID that matches the message ID of the original request.

You cannot specify the message ID manually. You cannot reuse message IDs.

Payload

This scheme is based on a correlation ID that is embedded in the payload of the request and response messages. The scheme has an associated *payload scheme* that controls how the correlation ID is embedded. The default payload scheme lets you define XPath expressions for obtaining the correlation ID from the messages.

The following graphic shows an example of the JMS Payload correlation scheme with the XPath payload scheme.

Screen capture of JMS Payload correlation scheme.

The **Reuse ID** list indicates whether to generate a new correlation ID for each new transaction or use the same correlation ID for the duration of the test.

You cannot mix the Payload correlation scheme with the other two correlation schemes on the same queue. If one listener is listening on a queue with the Payload correlation scheme, then all listeners on that queue must be using the Payload correlation scheme. However, you can use Correlation ID and Message ID to Correlation ID as payload schemes for the Payload correlation scheme.

ReplyTo and Temporary Response Queues

This approach is not a correlation scheme that you can select in a test step, but it achieves the same goal.

In some messaging applications, rather than having a set response queue, the client specifies its own response queue when sending the request. This queue is called the ReplyTo queue. The service sends the response to that queue.

Usually, this approach is used with temporary queues. The client creates its own temporary queue, sends that in its request as the ReplyTo queue, receives a response on that temporary queue, and deletes the temporary queue when finished.

Data Set Descriptions

A *data set* is a collection of values that can be used to set properties in a test case or virtual service model at run time.

This section describes the following data sets:

- Read Rows from a Delimited Data File Data Set
- Create Your Own Data Sheet Data Set
- Create Your Own Set of Large Data Data Set
- Read Rows from a JDBC Table Data Set
- Create a Numeric Counting Data Set
- Read Rows from Excel File Data Set
- Read DTOs from Excel File Data Set
- Unique Code Generator Data Set
- Random Code Generator Data Set
- Message-Correlation ID Generator Data Set
- Load a Set of File Names Data Set
- XML Data Set
- Scripted DataSet

Read Rows from a Delimited Data File Data Set

The Read Rows from a Delimited Data File data set assigns values to properties based on the contents of a text file. It is the most commonly used type of data set in DevTest Solutions. The first line of the text file specifies the names of the properties into which the data values are stored. The subsequent lines list the data values to be used for these properties. The text file is created using a simple text editor.

The following example shows a comma-delimited data file. The first row shows the property names in this data set.

Notepad file with month,day,year - example of a Delimited Data File data set

The Data Set Editor is used to define the data set.

Complete the following fields:

- **Name**
  Enter the name of the data set.

- **Local**
  Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

- **Random**
  Whether the record after the current record (sequential access) is read, or a random record is 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**
  Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

- **File Location**
  The full path name of the text file, or browse to file with the Browse button.

- **File Encoding**
  Accept the default encoding of UTF-8 or select an alternate encoding from the drop-down list. You can also select **Auto-detect** and click **Detect** to have an encoding type selected for you.

- **Delimiter**
  The delimiter being used. Any character can be a delimiter. The drop-down list contains common delimiters.

Click **Test and Keep** to test and load the data. You get a message that confirms that the data set can be read, and shows the first set of data.

Test successful message dialog

Create Your Own Data Sheet Data Set

The Create Your Own Data Sheet data set lets you generate your data set data in DevTest, 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). The default name is **Create your own Data Sheet**. Subsequent data sheets that are created have "~number" appended to the data sheet name.

Complete the following fields:

- **Name**
  Enter the name of the data set.

- **Local**
  Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

- **Random**
  Whether the record after the current record (sequential access) is read, or a random record is 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**
  Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

- **Number of Rows**
  The initial estimate of the number of rows. This value can be modified later.

- **Column Names**
  Comma delimited list of column names. These names are also the property names.

Click **Create Data Sheet Skeleton** and enter your data into the table.

To sort ascending or descending on a column, click the column labels. Right-click the column label for a column menu.

- **Encrypt Column**
  To encrypt all values in the column, select this option. An encrypted column has a lock icon and all values appear as a row of asterisks. Exporting shows that the column is <name>_enc and the data is encrypted.

- **Change column name:**
  To rename the column, select this option.

- **Sort Ascending**
  To sort the column values ascending, select this option. To change the sort to descending, click again.

- **Reset Sort**
  To sort the column values as they were originally, select this option.

- **Column Resize Mode:**
  Select **Automatic**, **Subsequent**, **Next**, **Last**, or **Manual**.

- **Maximize All Columns**
  Selecting this option changes the **Column Resize Mode** to **Manual**. The columns are all displayed, and scroll bars are available.

To create and modify the table, use the functions in the bottom toolbar.

- **Add**
  Add rows to the table.

- **Up and Down**
  Select a row and move it up or down in the table.

- **Delete**
  Delete the selected row.

- **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 selected for editing, and click this button.

Click **Test and Keep** to test and load the data. You get a message that confirms that the data set can be read, and shows the first set of data.

Test successful message dialog

You can also complete the following actions:

- To sort the rows, double-click the column header.

- To change the column name, right-click on the column name

- To select the toolbar functions and **Launch Extended View** to edit the cell, select and then right-click a cell.

- To convert your data sheet to a data set or a file, click **Convert to Data Set** at the bottom of the panel.

DataSheet Converter dialog

- **Data Sheet to Convert**
  By default, the data sheet you created, or select from the pull-down list.

- **External File Name**
  The name and path of the file you want to create from the data sheet.

Note: Moving the rows up or down in the table can affect the outcome of a test. The order of columns does not affect the outcome of the test.

Select the steps that use the data sheet.

Create Your Own Set of Large Data Data Set

The Create Your Own Set of Large Data data set lets you define a custom data table that can be arbitrarily large. The data can have any number of rows and columns.

The following graphic shows the editor:

Create Large Data Set window with data and all parameters specified

Complete the following fields:

- **Name**
  Enter the name of the data set.

- **Local**
  Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

- **Random**
  Whether the record after the current record (sequential access) is read, or a random record is 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**
  Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

- **Backing File Name**
  The name of the file in which the data is stored. This file is created automatically.

The **Create** button opens a dialog that lets you specify more parameters for file creation.

Create Large Data File dialog for specifying number of rows and column names

The dialog contains the following fields:

- **Initial # of Rows**
  Defines the initial number of rows to create. You can use the editor to add more rows.

- **Column Names**
  Defines comma-separated column names that go in the data set.

Click **Test and Keep** after entering data in the columns and data is copied in the backing file created.

Convert to Comma-Separated Value (CSV) File

You can convert the data set to a comma-separated value file. This feature has the following limits:

- The maximum number of rows in a worksheet is 1048576.

- The maximum number of columns in a worksheet is 16384.

- The maximum number of characters is a cell is 32767.

Follow these steps:

1. Click **Convert Large Data Set to Delimited File Data Set**.

2. (Optional) Change the output directory and file name.

3. Click **Convert**.

Read Rows from a JDBC Table Data Set

The Read Rows from a JDBC Table data set 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 is represented as a property. The data set then loops across the rows that were returned from the SQL query.

Complete the following fields:

- **Name**
  Enter the name of the data set.

- **Local**
  Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

- **Random**
  Whether the record after the current record (sequential access) is read, or a random record is 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**
  Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

- **Driver Class**
  Enter or select the full package name of the appropriate driver class. Standard driver classes are available in the pull-down menu.

- **Connect String**
  The connect string is the standard JDBC URL for your database. Enter or select the URL. The pull-down menu contains JDBC URL templates for common database managers.

- **User ID**
  Enter a user ID (if the database requires it).

- **Password**
  Enter a password (if the database requires it).

- **SQL Query**
  The SQL query that is used to create the data set.

Click **Test and Keep** to test and load the data. You get a dialog window that confirms that the data set can be read, and shows the first set of data.

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 specific value and changes by a fixed amount 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 occurs. The following example shows how to use the Create a Numeric Counting data set to make 100 calls to the same step.

Complete the following fields:

- **Name**
  Enter the name of the data set.

- **Local**
  Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

- **Random**
  Whether the record after the current record (sequential access) is read, or a random record is 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**
  Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

- **Property Key**
  The name of the property into which the counter value is stored.

- **From**
  The initial counter value.

- **To**
  The final counter value.

- **Increment**
  The counter step amount. To use counter data set to count backwards, set a negative Increment value.

Click **Test and Keep** to test and load the data. A message confirms that the data set can be read, and shows the first set of data.

Read Rows from Excel File Data Set

The Read Rows from Excel File data set assigns values to properties based on the contents of an Excel spreadsheet.

The first nonblank row of the Excel spreadsheet specifies the names of the properties to which the data values are assigned. The 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 can test a set of first name, last name, user ID, and password combinations.

Complete the following fields:

- **Name**
  Enter the name of the data set.

- **Local**
  Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

- **Random**
  Whether the record after the current record (sequential access) is read, or a random record is 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**
  Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

- **File Location**
  Enter 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 example, LISA_HOME).

- **Sheet Name**
  Enter the name of the sheet in the Excel spreadsheet.

Click **Test and Keep** to test and load the data. You get a message that confirms that the data set can be read, and shows the first set of data.

Select the steps that use the data set.

You can click **Open XLS File** to open and edit the Excel spreadsheet.

Tip: The Excel spreadsheet must not contain formulas.

Note: Excel File data sets and Excel DTO data sets can use Excel 2007 or later spreadsheets. Excel DTO data sets are still created using the XLS format.

Read DTOs from Excel File Data Set

The Read DTOs from Excel File data set lets you parameterize Java data transfer objects (DTOs) in your test steps. The data set gives you an easy way, through Excel spreadsheets, to provide data values for those parameters.

The Read DTOs from Excel file data set assigns values to the properties of a DTO and stores the object in a 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. The data that is represented in Excel is converted into the proper data types automatically when needed. The only complex part of using this data set is the initial creation of the Excel spreadsheet. Fortunately, the creation is done for you. Given the package name of the DTO, a template is created, 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, require more Excel sheets to represent the full DTO.

Often, a web service endpoint expects complex DTOs. The Excel data set simplifies creating objects to use as parameters to the web service. When the web service is first referenced, it is given a name and a URL for the WSDL. Java DTO classes in the form **com.lisa.wsgen. SERVICENAME.OBJECTNAME** are automatically generated and made available on the classpath. You can browse to that generated class in the DTO class browser, generate an Excel file, and simply fill in the template. See the following example.

Building the data set is a two-step process. First, let DevTest build the template in Excel. Then open the Excel spreadsheet and fill in the data fields in all the sheets that are produced.

To build the template:

1. Enter the following parameters in the Data Set Editor:

   - **Name**
     The name of the data set. This name becomes the property that is used to store the current DTO object.

   - **Local**
     Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

   - **Random**
     Whether the record after the current record (sequential access) is read, or a random record is 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**
  Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

- **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 Test Manager. Your class can be copied to the hotDeploy directory to put it on the Test Manager.
  Advanced Settings let you specify:

- **Use flattened child property notation during generation**
  Select to override child property flattening during Excel DTO data set generation. If cleared, child properties are generated as references with their own worksheets.

- **Use new empty cell semantics for flattened properties**
  Empty cell semantics for flattened properties means how empty cells are interpreted in a DTO spreadsheet. For example, given the following flattened properties:

  ```
  { "prop1.subprop1", "prop1.subprop2" }
  ```

  if both subprop1 and subprop2 have empty cell values, then under the new semantics the reference to "prop1" is set to null. Under the old semantics, prop1 would be not null, but references to subprop1 and subprop2 would both be null. The new semantics should be used by default, especially by web services with WSDLs that use nonnillable types. If not used in the case of nonnillable types, the intermediate not null references for containing properties that are automatically created when reading a DTO spreadsheet could result in the generation of invalid XML according to the schema (because cell values are empty).

2. Click **Generate Template**. DevTest builds the template and the system messages report when the file is built.

3. Click **Open XLS File**.
   The spreadsheet contains everything that is necessary to construct the object. We will show how to add data in the next section.

4. Close the XLS file.

5. Click **Test and Keep** to test and load the data. You see a 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 use an actual DTO object: com.itko.example.dto.Customer. This class is included with the DevTest examples and can be found by browsing the Test Manager by clicking **Browse**.

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, DevTest lists the DTO spec (Customer) and the current DTO object (Customer). The spreadsheet would also list the Java doc location, if available. The following graphic shows the data sheet, 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 Methods:	No methods available									
6		Static Constructor Fields:	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												
11												
12												

Excel file with with a row specifying property names, followed by a row specifying the data types

Looking at the data sheet we can see:

- 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 (that is, poAddr.city).
- The since property, of type date, is prefilled with the current date. This entry 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 that can contain the array elements as a comma-separated list. This list is only possible for arrays of primitives or strings.

The location property, of type Address[], does not appear on this sheet. Because it is an array of objects, the location property is on the second sheet in the Excel file. This sheet contains the data for an Address object in each row. There are wo 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.

	A	B	C	D	E	F	G	H	I	J
1										
2		Spec:	com.itko.examples.dto.Customer.locations							
3		DTO:	com.itko.examples.dto.Address							
4		Docs:	No docs available							
5		Static Con	No methods available							
6		Static Con	No fields available							
7										
8		Primary Ke	com.itko.e	city	line1	line2	state	zip		
9		(unique per row)	(reference the containing DTO)	String	String	String	String	String		
10										
11										
12										
13										

com.itko.examples.dto.Customer | xamples.dto.Customer.locations

Excel file his sheet contains the data for an Address object in each row

Because each Customer object can have several locations, several rows in the locations sheet belong to an object specified in a single row of the Customer sheet. The second sheet manifests this by listing the primary key of the parent Customer object in the "reference the containing DTO" field of each location that belongs to the Customer. This is similar to primary/foreign key relationships in databases.

	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 Methods:	No methods available									
6		Static Constructor Fields:	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	75	101	John An	Dallas	750 W. John Ca	Suite 250	TX	75024	8/11/2007	1,5,10
11		2	250	102	Dirk Mar	Plano	5800 Granite Pa	Apt. 3455	TX	75031	6/29/2010	2,4,9
12		3	800	103	Francis I	Southlake	1376 Lakeview D	Suite 809	TX	76092	5/10/2011	3,7,9
13												

Excel spread sheet, second sheet example

	A	B	C	D	E	F	G	H
1								
2		Spec:	com.itko.examples.dto.Customer.locations					
3		DTO:	com.itko.examples.dto.Address					
4		Docs:	No docs available					
5		Static Con	No methods available					
6		Static Con	No fields available					
7								
8		Primary K	com.itko.e	city	line1	line2	state	zip
9		(unique per row)	(reference the containing DTO)	String	String	String	String	String
10		1	1	Dallas	12 Main St	#45	TX	75201
11		2	1	Plano	3354 Bilglade Ave.		TX	76133
12		3	1	Frisco	2109 Tanglewood Driv		TX	75061
13		4	2	Dallas	6788 Woodbrook Driv		TX	76092
14		5	2	Fort Worth	443 Churcl	Suite 87	TX	75925
15		6	3	Dallas	7789 17th Blvd.		TX	74552
16		7	3	Dallas	122 Hwy. 26		TX	71655
17								

Excel file for Read DTOs from Excel File Data Set

When you save the spreadsheet and click **Test and Keep** in DevTest, you see the first Customer DTO in the message.

Message - Test successful

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 in the previous example.

To see how you could use this Customer DTO as a property, see the sections on testing Java objects in Test Steps.

Unique Code Generator Data Set

The Unique Code Generator data set provides a unique token (or code) each time DevTest calls it. The token can be numeric or alphanumeric, and can have a user-defined prefix prepended to it. The Unique Code Generator is commonly used in testing to create new users, accounts, and so on. By using the generator, you can ensure that the generated token or code does not use a value that is already inside a system.

Complete the following fields:

- **Name**
  Enter the name of the data set.

- **Local**
  Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

- **Random**
  Whether the record after the current record (sequential access) is read, or a random record is 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**
  Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

- **Type**
  The type of token to return. The choices are number or alphanumeric.

- **Prefix (opt)**
  Prefix to be prepended (optional).

Click **Test and Keep** to test and load the data. A dialog opens to verify that the data set is readable. This dialog shows the first set of data.

This data set always returns a token. The **At End of Data** section of the Data Set Editor has no meaning for this data set type.

Random Code Generator Data Set

The Random Code Generator data set generates numeric or alphanumeric data randomly for use in a test case. This data set is similar to the Unique Code Generator data set, but it lets you set a length for the result. This data set can be used to create a specific type of unique value, such as a telephone number or Social Security number.

Complete the following fields:

- **Name**
  Enter the name of the data set.

- **Local**
  Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

- **Random**
  Whether the record after the current record (sequential access) is read, or a random record is 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**
  Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

- **Prefix (opt)**
  Adds a 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 that is generated to the value set here.

Click **Test and Keep** to test and load the data. You get a message that confirms that the data set can be read, and shows the first set of data.

Message-Correlation ID Generator Data Set

The Message/Correlation ID Generator data set is a specialized unique code generator useful for messaging. The data set generates a 24-byte unique code. The data set is designed specifically for an IBM MQ Series correlation ID, but can also be used for any JMS provider. This data set creates or updates two special properties: **lisa.jms.correlation.id** and **lisa.mq.correlation.id**. The messaging steps recognize these properties and set the message correlation ID appropriately.

Complete the following fields:

- **Name**
  Enter the name of the data set.

- **Local**
  Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

- **Random**
  Whether the record after the current record (sequential access) is read, or a random record is 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**
  Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

- **Prop to set**
  DevTest property to set. The default is **lisa.jms.correlation.id**.

Load a Set of File Names Data Set

The Load a Set of File Names data set assigns a value to a property based on a filtered set of file names from the file system.

The set of file names can include all the files in a specific directory, or a set that a "file pattern" filters. Another option lets you include subdirectories in the set, recursively. The files are returned in a case-sensitive, alphabetical order, top directories first, followed by subdirectories (using depth first ordering). Each time the data set is used, it returns the next file name (full path name) in the data set. This file name is stored in a property with the same name as the data set.

Complete the following fields:

- **Name**
  Enter the name of the data set.

- **Local**
  Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

- **Random**
  Whether the record after the current record (sequential access) is read, or a random record is 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**
  Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

- **Directory Location**
  The full path name of the directory to scan, or you can browse using the **Browse** button.

- **File Pattern**
  A filter pattern string, using an asterisk (*) as a wild card if appropriate.

- **Include Files from Subdirectories**
  If the files in subdirectories are listed, select this option.

Click **Test and Keep** to test and load the data. You get a dialog that confirms that the data set can be read, and shows the first set of data.

> **Note:** A large amount of data could take longer than you want; use the **Cancel** button to stop the operation.

XML Data Set

Like other data sets in DevTest, 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 window

At design-time, the first record of an XML Data Set populates the value for a property in test state by clicking **Test and Keep**. The property name is the same as the name of the data set. For example, if the data set has the name **DataSet1**, then the property that is populated in the 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

- **Name**
  The name of this data set. This value is used as the name of the property in test step. For example, an XML Data Set with the name **DataSet1** populates the property {{DataSet1}}.

- **Local**
  Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

- **Random**
  Whether the record after the current record (sequential access) is read, or a random record is 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**
  Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

- **Test and Keep**
  At design-time, populate the property that is associated with the data set in test state with the value of the first record.

Advanced Settings Tab

XML Data Set window - Advanced Settings tab

- **Directory path**
  This read-only value represents the directory on the file system where records are saved. The mapping between a record and a file in the directory path is a one-to-one mapping. If an XML data set is created within a project, the directory path starts with "{{LISA_PROJ_ROOT}}/Data/datasets/xml/" or "{{LISA_RELATIVE_PROJ_ROOT}}/Data/datasets/xml/". If no project is used, the directory path starts with "{{LISA_HOME}}/datasets/xml/".

Record Editing Panel

XML Data Set window - Record Editing panel

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 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**
  To jump to a specific record, enter the record number.

Visual XML Tab

XML Data Set window - Visual XML tab

Visual XML Editor

Moving the mouse over the **Node** and **Value** columns displays a tooltip. The tooltip can help if the value in the table is too long to be fully displayed.

Raw XML Tab

XML Data Set window - Raw XML tab

This tab contains a raw text view of the XML contained in a record.

Double-clicking the left border toggles the visibility of a top toolbar, line numbers bar, and bottom editing info bar in the editor.

Scripted DataSet

The script in this data set is invoked each time the data set is queried for a new record (for example, each time the test case loops). The script must return a map, representing one “row” of data.

Returning null indicates the data set has finished.

You can reference all the TestExec properties from within the script.

Complete the following fields:

- **Name**
  Enter the name of the data set.

- **Local**
  Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

- **Random**
  Whether the record after the current record (sequential access) is read, or a random record is 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**
  Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

- **Language**
  Choose from:

  - Beanshell
  - Groovy
  - JavaScript
  - Velocity

- **Copy properties into scope**
  Allows you to specify which properties to download for use in the data set.
  Values:

  - **Test and system properties**: all properties for the test case and system
  - **Test properties**: properties that provide information about the test case
  - **Helpers only**: only the TestExec and logger properties

  Default: Test and system properties

To use additional scripting languages, see Enabling Additional Scripting Languages.

Enter your script into the script editor on the left.

All the objects available for use in the script editor are listed in the **Available Objects** panel on the right. The list includes primitive types of data (strings and numbers) and objects such as EJB response objects that were 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 and Keep** to open a window with the result of the script execution or a description of the errors that occurred.

When you save the test case, the script is checked for syntax errors.

To see an example of a script, see the examples project in DevTest Workstation.

Event Descriptions

An *event* is a message about an action that has occurred.

The following table describes the standard events.

Event Name	Description	Short Info	Long Info
Abort	This event ends a test in a "cannot finish" state. It is a failing type of end event.	Event Aborted	
Assert evaluated	A non-embedded assertion will generate this event if it does not fire. These events are the assertions that did not get to execute their consequence.	The name of the assertion	
Assertion fired	A non-embedded assertion will generate this event if it does fire. Firing means it is true and its consequence will be followed.	The name of the assertion	The log message of the assertion, or a DevTest-generated message if there is no log message set
Call made	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, for example, **void setName(java.lang. String name[Basic Checking])**
Call result	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, then an XML view of the object is shown.
Configure for load test		Configure for load test	The test run has been configured as a load test. Selecting individual events has been disabled.
Coordinator ended	A coordinator has been removed.	The name of the coordinator server	
Coordinator server ended	The coordinator server was ended.	The name of the coordinator server	
Coordinator server started	The coordinator server was created.	The name of the coordinator server	
Coordinator started	A new coordinator was created.	The name of the coordinator server	
Cycle ended normally	The test execution completed in a successful state.	The name of the test case	
Cycle ending	The instances have been instructed to stop testing.		
Cycle failed	The test execution failed. 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	
Cycle history		Cycle History	

	Every model that runs will generate one of these events. It is the final trace of all the details of its run.		
Cycle initialized	The test has been initialized (loaded).	The name of the test case	
Cycle runtime error	An abnormal DevTest 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 (step, data set, or filter) that has the error	Usually a message that explains the error
Cycle started	This test instance and cycle have just started.	The name of the test case	
Data set read	Data sets generate an event that makes it clear what values are about to be used.	Data set read	
HTTP performance	This event is generated for every HTTP transaction that DevTest executes to capture the performance statistics.	HTTP performance statistics	
Info message	Basic logging data. For example, the HTTP/HTML request step sends 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 DevTest generated
Instance ended	Sent when a simulator instance has finished.	The name of the simulator	
Instance started	Sent when a simulator creates an instance.	The name of the simulator	
Log message	Basic logging data. Can be turned off to minimize overhead when filtering events.	The message sent to the log	
Metric alert	A metric has been collected and is reporting its value.	The short name of the metric, for example, **DevTest: Avg Response Time**	The value of the metric collected
Metric started	Metrics that are collected generate real-time events of their values.	Metric Started	
Metric value	Metrics that are collected generate real-time events of their values.	Metric value	
Model definition error	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 (step, data set, or filter) that has the error	Usually a message that explains the error
Pathfinder	The system being tested has DevTest integration enabled. This event contains the DevTest integration XML data that was captured.	The name of the step	The XML representation of the DevTest Integration data captured
	A property was removed.	The property key	

Property removed			The word "<removed>"
Property set	A property was set.	The property key	The property value
Simulator ended	Sent when a simulator has ended.	The name of the simulator	
Simulator started	Sent when a simulator has started.	The name of the simulator	
Step bandwidth consumed	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
Step error	An error has occurred in the system under test. For example, there was no response from a web server. This event is for a step. 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
Step history	Every step has a history event that fires of type **com.itko.lisa.test. NodeExecHistory** in its long info.	Step history	
Step 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
Step request bandwidth			
Step response	A step was completed against the system under test.	The name of the step	The response data as a string
Step response bandwidth			
Step response time	The amount of time a step took to execute against the system under test.	The name of the step	The number of milliseconds to execute the step
Step started	A step is being executed.	The name of the step	
Step target	Every step has a target, such as the URL for a web request or the JNDI name of an EJB.		
Step warning	A warning was recorded. For example, a filter took the default value because it could not find the current value.	Step warning	
Subprocess finished	A subprocess has finished executing.		
Subprocess ran	A subprocess has started.		
Suite aborted	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 indicates that this happened.	The name of the suite	
Suite ended	All the tests running as part of a suite have finished.	The name of the suite	

Suite history	Every suite that runs will generate one of these events. It is the final trace of all the details of its run.	Suite History	
Suite setup /teardown	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
Suite started	A suite is starting to run.	The name of the suite	
Suite test failed	A test running as part of a suite ended in failure.	The name of the suite	
Suite test passed	A test running as part of a suite ended successfully.	The name of the suite	
Suite test staged	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
Test ended	Sent when the coordinator stops the test.	The name of the test case	
Test not active	A test in a suite is marked as inactive.	Event Test Not Active	
Test started	Sent when the coordinator starts the test.	The name of the test case	
VS log message	VSE internal logging	VS log message	
VS no transaction match	A virtual service did not match a transaction request to at least one recorded response.	VS transaction no match	
VS service ended	A virtual service was stopped.	VS service stopped	
VS service started	A virtual service was started.	VS service started	
VS transaction finished	A virtual service finished processing a transaction request.		
VS transaction match	A virtual service matched a transaction request to at least one recorded response.	VS transaction match	
VSE server reset	A service reset request was made of a server.	VSE server resets	
VSE server shutdown	A virtual service environment was asked to shut down.	VSE server shutdown	
VSE server stop	A service stop request was made of a server.	VSE server stops	

Filter Descriptions

A *filter* is an element that runs before and after a step.

This section describes the following filters:

- Utility Filters
- Database Filters
- Messaging/ESB Filters
- HTTP/HTML Filters
- XML Filters
- JSON Filters
- Java Filters
- VSE Filters
- CAI Filters
- Copybook Filters

Regular expressions are used for comparisons in several filters. For more information about regular expressions, see Regular Expressions.

Utility Filters

The following filters are available in the Utility Filters list for any test step:

- 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

Create Property Based on Surrounding Values

The Create Property Based on Surrounding Values filter lets you read textual content and filter it for information to be stored in DevTest properties. The filter can be used on text, and on XML and HTML treated as text. The filter uses a "paint the screen" technique.

"Paint the screen" gives you great flexibility to define what in the HTML you want to parse as properties. Mark the text in one of the following ways:

- Text that must appear in the response exactly as shown: a Must block.
- Text that is not required to appear in the response, or can change: an Any block.
- Text that is stored in a property: a Property block.

The text is marked using the icons at the bottom of the editor.

Filter editor toolbar with icons identified

In the following example, the goal is to store the size of a specific file in a property.

The text is marked using the editor icons, by selecting the text, and then clicking 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 Must icon.
- Red background identifies the text that is stored in the property that is entered in the dialog (indicated by a single arrow). This is a Property block and is marked using the Property icon.

Note: Property blocks must always be bounded by Must blocks.

Create Property Based on Surrounding Values window

This screen shows the contents of the text buffer. The goal is to parse the file size of the _misc-reporting.jar file. The file size is the number that appears after **_misc-reporting.jar**.

The boundaries are set around the file size, and **Must** has been clicked. The actual file size text inside the selected content was selected, and **Property** was clicked.

The property name was then entered into the dialog. The actual value of the file size has been replaced with the name of the property.

When this filter is run, the property **filesize** is assigned the size of the _misc-reporting.jar file.

You can repeat this process on this text buffer to define as many properties as necessary.

Handling Nonunique Tokens

If you see the following error message, your selected token is not unique; the selection you made is repeated in the token before it.

Parse Web Page for Properties error message

To solve this issue in most cases, simply create another token to make the prior token a Must token also. In other cases, when this technique does not work, a judicious placement of another Must block between the two duplicate tokens avoids the error.

This solution works because DevTest can distinguish between the two duplicate tokens, which are based on their relative location.

Store Step Response

The Store Step Response filter lets you save the last response as a property for future use.

Complete the following fields:

- **Filter in**
  Where to apply the filter. 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 that result from running the filter.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

Override Last Response Property

The Override "Last Response" Property filter lets you replace the current value of the last response with the value of an existing property. For example, assume that you execute an EJB, but you eventually receive a value after making some method calls on the EJB. Using the result of a method call (instead of the EJB object) as the last response could improve the test case. You can use a filter to save the return value from that call as a property, and then use that property in this filter.

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **Convert to XML**
  Select this option if you want the response to be converted to valid XML.

- **Filter Run Results**

  Displays the property and values that result from running the filter.

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

Save Property Value to File

The Save Property Value to File filter lets you save the value of an existing property to a file in your file system.

Complete the following fields:

- **Filter in**
  The name of the property whose value is written 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**
  To append the information to an existing file, select this check box.

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

Parse Property Value As Argument String

The Parse Property Value as Argument String filter lets you store the text of a specific attribute in a property. This filter is useful as a second filter, where you parse a filtered value for information.

Complete the following fields:

- **Filter in**
  The name of the existing property to parse. For example, to parse the **lisa.deleteUser.cookies.rsp** property to return the value of the SESSIONID attribute, enter **lisa.deleteUser.cookies.rsp**
  .

- **IsURL**
  Select this check box if the property value is a URL.

- **Attribute**
  The attribute to retrieve.

- **Property**
  The name of the property to store the text of the attribute.

- **Default (if not found)**
  If the attribute is not found, the default value to use.

- **Filter Run Results**

  Displays the property and values that result from running the filter.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

Save Property from One Key to Another

The Save Property from One Key to Another 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.

Complete the following fields:

- **Filter in**
  This field accepts the property content of which is copied in some other property. This field is the input source property.

- **To Property**
  This field is the property name where input property content is copied.

- **Run Filter**
  Lets you test the filter immediately while developing test steps rather than waiting until the test case is completed.

- **Pre Process**
  Run the filter before the step.

- **Post Process**
  Run the filter after the step.

Time Stamp Filter

Use the Time Stamp filter to assign the current time and date to the property so that you can use it in the following steps.

Complete the following fields:

- **Filter in**
  The name of the existing step.

- **Date Pattern**
  Select the date pattern to display.

- **Offset**
  Used to offset the date to an appropriate (future or past) date that is based on the current date.

- **Pre Process**
  When selected, generates a timestamp before the step runs.

- **Property for Pre Process**
  The property to store the preprocess timestamp.

- **Post Process**
  When selected, generates a timestamp after the step runs.

- **Property for Post Process**
  The property to store the post timestamp.

- **Filter Run Results**

  Displays the property and values that result from running the filter

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

Database Filters

The following filters are available in the Database Filters list for any test step:

- Extract Value from JDBC Result Set
- Size of JDBC Result Set
- Set Size of a Result Set to a Property
- Get Value For Another Value in a Result Set Row

Extract Value from JDBC Result Set

The Extract Value from JDBC Result Set filter lets you store the text of a specific JDBC result set value in a property. You can create this filter either manually from the filter list or by using the embedded filter commands on a result set response.

To create the filter manually:

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **Column (1-based or name)**
  The index or name of the column (field).

- **Row (0-based)**
  The row to retrieve the value. This field is a zero-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 that result from running the filter.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

To create a filter from a result set response:

Follow these steps:

1. Display the step response that contains the result set.

2. From the result set, select the cell of the value to store in the filter.

LOGIN	PWD	NEWFLAG	FNAME	LNAME	EMAIL	PHONE	ROLEKEY	SSN
dmxxx-009	tIDAdNa3...	1	first-9	last-9	test@test...		1	
Testuser	IGyAQTu...	0	Test	User	anne@itk...	817-433-...	1	433-87-3...
TEST1	nU4eI71b...	1					1	
First1	8FePHnF0...	1	Firstname1	Lastname1	first1@itk...	555-555-...	1	555-55-8...
Last1	i+UhJqb9...	1	Firstname2	Lastname2	last1@itko...	333-448-...	1	533-88-9...
test1	nU4eI71b...	1	First1	Last1	test1@itk...	214-111-...	1	111-11-1...
test2	nU4eI71b...	1	First2	Last2	test2@itk...	215-222-...	1	222-22-2...
demo	89yJPVNn...	0	DEMOFIRST	DEMOLAST	demo@itk...	817-433-...	1	677234567
admin	0DPiKuNIr...	1	iTKO	Admin	lisabank-a...	123-4567	2	434-47-5...
sbellum	26yJsXNp...	1	Sara	Bellum	sbellum@...	232-4345	1	614-40-1...
wpiece	/UuJ0Me...	1	Warren	Piece	wpiece@...	455-3232	1	546-71-4...
areck	AHDRRjD...	1	Amanda	Reckonwith	areck@my...	555-2244	1	350-02-1...
boaty	RQIil0Ldp...	1	Boaty	Rabbit	boaty@ra...	333-4521	1	616-51-0...
itko	qUqP5cyx...	1	itko	test	itko.test@...	650-234-...	1	140-72-2...
lisa_simpson	60fAFoq+...	1	lisa	simpson	lisa.simps...	123-456-...	1	295-20-0...
virtuser	nU4eI71b...	1	Virtual	User	virtuser@i...	123-456-...	1	297-55-9...

Creating a filter from a result set response example

3. Click **Generate Filter** , as the arrow in the previous graphic shows.

4. In the dialog, enter the property key:

Enter the property key dialog

5. Click **OK**.

The filter to store the value **sbellum** in the property **theLogin** is created.

Size of JDBC Result Set

The Size of JDBC Result Set filter lets you check that the result set returned in each JDBC-based step matches the criteria that are specified. The filter is a simple filter to handle most common database errors automatically.

This filter does not affect non-JDBC steps, and is often used as a global filter in a test case.

Complete the following fields:

- **Result Set Has Warnings**
  Some databases return warnings in the result set. If your database supports this feature and you want a warning to fire the **On error** step for this filter, select this check box.

- **Row Count >=**
  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 <_=**
  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 conditions for this filter are not met.

> **Note:** This filter can serve the purpose of a general global assertion because you can select a next step based on the presence of an error.

Set Size of a Result Set to a Property

The Set Size of a Result Set to a Property filter lets you store the count of a result set to a property provided.

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **Property to Store Row Count**
  User-provided property name to store the row count. The default property name is PROP.

- **Filter Run Results**

  Displays the property and values that result from running the filter.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

Get Value For Another Value in a Result Set Row

The Get Value For Another Value in a Result Set Row filter lets you search a column (field) in a result set for a specific value. If the value is found the value in another column (field) and the same row is placed in a property.

You can create this filter either manually from the filter list or by using the embedded filter commands on a result set response.

To create the filter manually:

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **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 that result from running the filter.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

To create the filter from a result set response:

1. Display the step response that contains the result set.

2. From the result set select the two values in different columns, using the Ctrl key.

LOGIN	PWD	NEWFLAG	FNAME	LNAME	EMAIL	PHONE	ROLEKEY	SSN
dmxxx-009	tIDAdNa3...	1	first-9	last-9	test@test...		1	
Testuser	IGyAQTu...	0	Test	User	anne@itk...	817-433-...	1	433-87-3...
TEST1	nU4eI71b...	1					1	
First1	8FePHnF0...	1	Firstname1	Lastname1	first1@itk...	555-555-...	1	555-55-8...
Last1	i+UhJqb9...	1	Firstname2	Lastname2	last1@itko...	333-448-...	1	533-88-9...
test1	nU4eI71b...	1	First1	Last1	test1@itk...	214-111-...	1	111-11-1...
test2	nU4eI71b...	1	First2	Last2	test2@itk...	215-222-...	1	222-22-2...
demo	89yJPVNn...	0	DEMOFIRST	DEMOLAST	demo@itk...	817-433-...	1	677234567
admin	0DPiKuNIr...	1	iTKO	Admin	lisabank-a...	123-4567	2	434-47-5...
sbellum	26yJsXNp...	1	Sara	Bellum	sbellum@...	232-4345	1	614-40-1...
wpiece	/UuJ0Me...	1	Warren	Piece	wpiece@...	455-3232	1	546-71-4...
areck	AHDRRjD...	1	Amanda	Reckonwith	areck@my...	555-2244	1	350-02-1...
boaty	RQIil0Ldp...	1	Boaty	Rabbit	boaty@ra...	333-4521	1	616-51-0...
itko	qUqP5cyx...	1	itko	test	itko.test@...	650-234-...	1	140-72-2...
lisa_simpson	60fAFoq+...	1	lisa	simpson	lisa.simps...	123-456-...	1	295-20-0...
virtuser	nU4eI71b...	1	Virtual	User	virtuser@i...	123-456-...	1	297-55-9...

Create the filter from a result set response

3. Select **Filter for a value** and then get another column value filter using the **Filter** icon.

4. In the the **Generate Value for Value Filter** dialog, select or reassign the columns for the search and the value, then enter the Property Key.

Generate Value for Value Filter dialog

5. Click **OK**.

The filter that is created in this example is the same as the filter you created manually in the previous example.

In the example, **sbellum** is searched for, and if found, the value in the EMAIL column of that row is placed in the property **theEmail**.

Messaging/ESB Filters

The following filters are available in the Messaging/ESB Filters list for any test step:

- Extract Payload and Properties from Messages
- Convert an MQ Message to a VSE Request
- Convert a JMS Message to a VSE Request

Extract Payload and Properties from Messages

DevTest auto extracts several internal properties of messages into properties in the test step using the Extract Payload and Properties from Messages filter. You can also select to auto extract the payload into a property. This method is a fast way to get data from a message.

Different messaging platforms impose various restrictions and can be seen as warnings at run time.

The property names can default to **lisa.stepName.message**, or you can specify the prefix. You can specify an exact name for the payload.

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **Get Payload**
  Select this option if you require a payload.

- **Property key to store the Payload**
  Enter or select the property key to use as the payload.

- **Prefix for extracted details**
  Enter the prefix to be attached to the property name in the result.

- **Get Message ID**
  Select to get the message ID.

- **Get Correlation ID where appropriate**
  Select to get the correlation ID.

- **Additional extended properties**
  Select to get any additional extended properties.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

Convert an MQ Message to a VSE Request

The Convert an MQ Message to a VSE Request filter is automatically added from the VSE Recorder. This filter serves the specific purpose that enables proper functioning with recordings. Use it carefully. If this filter is added to a step in a VSE model, do not remove or edit it.

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **Object form**
  Select to get the Object Form.

- **Track Correlation ID**
  Select to track the correlation ID.

- **Track Message ID**
  Select to track the message ID.

- **Transaction Tracking Type**
  Select the appropriate tracking type from - Sequential, Correlation ID, Message ID, or Message ID to Correlation ID.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

Convert a JMS Message to a VSE Request

The Convert a JMS Message to a VSE Request filter is automatically added from the VSE Recorder. This filter serves the specific purpose that enables proper functioning with recordings. Use it carefully. If this filter is added to a step in a VSE model, do not remove or edit it.

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **Object form**
  Select to get the Object Form.

- **Track Correlation ID**
  Select to track the correlation ID.

- **Track Message ID**
  Select to track the message ID.

- **Transaction Tracking Type**
  Select appropriate tracking type from: Sequential, Correlation ID, Message ID, or Message ID to Correlation ID.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

HTTP/HTML Filters

The following filters are available in the HTTP/HTML Filters list for any test step:

- Create Resultset from HTML Table Rows
- Parse Web Page for Properties
- Parse HTML/XML Result for the Value of a Specific Tag or Attribute
- Parse HTML Result for Specific Value and Parse It
- Parse HTML Result for the Child Text of a Tag
- Parse HTML Result for HTTP Header Value
- Parse HTML Result for the Value of an Attribute
- Parse HTML Result for DevTest Tag
- Choose Random HTML Attribute
- Parse HTML into List of Attributes
- Parse HTTP Header Cookies
- Dynamic Form Filter
- Parse HTML Result by Searching Tag Attribute Values
- Inject HTTP Header

Create Resultset from HTML Table Rows

The Create Resultset from HTML Table Rows filter lets you create a result set (for example, a JDBC result set) from an HTML table that the response returns. You can select the columns and rows of an HTML table, then create a result set from them. You can then use the result set to generate assertions exactly as in a database step.

You can create this filter by selecting it from the filter list and defining the parameters. However, it is easier to create the filter directly from the HTTP/HTML Request step response using one of the filter commands for that step. This approach is used here. The parameters that are produced in the following procedure (the parameters you would calculate to create this filter manually) are shown later in this section.

To create a filter on a table:

1. Record a web page that contains the table.

2. Go to the appropriate HTML step, and view it from the DOM tree.

3. Select the values to place in the table, using the **Ctrl** key to select multiple fields. Select one example value from each column in the table that you want to use in the result set.

Create HTML Table Results Filter from an HTML step

4. When it is highlighted, select **Create HTML Table Results Filter**.

5. Enter the property name in the window.

Enter property key name dialog

The property is now available in the test case.

The property is added to the current step. The following graphic shows the parameters that were calculated for this step. These parameters are required when you create this filter manually.

Create Resultset from HTML Table Rows filter parameters

To show the filter results, we added a step of type Save Property as Last Response and added the property that the filter creates. The result set panel displays the results.

Result set panel showing results of Create Resultset from HTML Table Rows filter

If you are editing a 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. To do so, use the **Replay test case to a specific point** command. The **Replay test case to a specific point** command is activated by clicking **Replay** on the toolbar. You can now use the embedded filters and assertions that are available at the bottom of the result set window of this step.

Parse Web Page for Properties

The Parse Web Page for Properties filter lets you view a rendered web page to create properties from the HTML content. This filter uses the "paint the screen" technique.

"Paint the screen" gives you great flexibility to define what in the HTML you want to parse as properties. Mark the text in one of the following ways:

- Text that must appear in the response exactly as shown: a Must block.
- Text that is not required to appear in the response, or can change: an Any block.
- Text that is stored in a property: a Property block.

The text is marked using the icons at the bottom of the editor.

In the following example, assume that the website title "LisaBank - Home" changes from user to user, and therefore must be stored as a property.

This window shows the HTML that is rendered in a browser in the top panel, and the actual HTML text in the bottom panel.

To create properties from fields on a website:

1. Navigate to the field identified in the HTML text in the bottom panel.

2. Select the text and click **Must** .
   In this example, the selected field is the website title. The yellow highlights indicate text that must appear exactly as shown.

3. Select the name text inside the highlighted content, and click **Property** .
   In this example, the website name text, "LisaBank - Home," is selected.

4. Click **Property** .
   The **Select Property** dialog is displayed.

5. Enter the property name into the dialog.
   In the HTML text in the bottom panel, the name of the property replaces the website name text. The red background identifies the text that is stored in the property that is entered into the dialog.

```
        <meta http-equiv="expires" content="0">


  <title>{{WebsiteTitle}}</title>


  <link href="/lisabank/css/financial.css" rel="stylesheet" type="text/css">
</head>
```

Note: All **Property** blocks must always be bounded by **Must** blocks.

Frequently you can do this action purely from the web page view by selecting the content in the web browser. It can be easier to click the web browser in the area that you want to select, and then make your selection in the HTML panel.

The **Website Title** property is assigned the current value that appears on the HTML page when the filter runs. The website title can change its location in the text buffer and it is still found and parsed for the property.

To define more properties, repeat this process on this text buffer.

Handling Nonunique Tokens

If you see the following error message, your selected token is not unique; the selection you made is repeated in the token before it.

To solve this issue in most cases, simply create another token to make the prior token a Must token also. In other cases, when this technique does not work, a judicious placement of another Must block between the two duplicate tokens avoids the error.

This solution works because DevTest can distinguish between the two duplicate tokens, which are based on their relative location.

Parse HTML/XML Result for the Value of a Specific Tag or Attribute

The Parse HTML/XML Result for the value of a Specific Tag or Attribute filter lets you get specific attributes from an HTML tag.

You can create this filter either manually from the filter list or by using the embedded filter commands on a result set response.

To create the filter manually:

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **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 selected, property value is URLEncoded.

- **Filter Run Results**

  Displays the property and values that result from running the filter.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

To create the filter from the HTTP/HTML request step response page:

1. Display the step response that contains the HTML response.

Step response that contains HTML

2. From the **DOM Tree** view, select the attribute whose value you want to store in a property.

3. When it is highlighted, select **Parse Value Filter**.

4. Enter the property name in the window.

HTML/XML Filter Info dialog

You can also add assertions here.

Parse HTML Result for Specific Value and Parse It

The Parse HTML Result for the Value of a Specific Tag or Attribute Value and Parse It filter combines the following filters:

- Parse HTML Result for the Value of an Attribute
- Parse Property Value as Argument String

This filter is designed to find a specific attribute in a web page, and then further parse that attribute. If the attribute is a URL, and not only a name-value pair, there is a function for handling that information.

In this example, the filter finds the "href" attribute for the seventh anchor tag, which is a URL. The filter takes the "cmd" parameter and stores that value in the **cmdlistusers_KEY**.

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **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**
  Select the check box if the attribute value is a URL.

- **Argument to Parse**
  The name of the argument to parse for its value; in this example, "cmd".

- **Property**
  The property in which to store the value.

  In this example, "cmdlistusers_KEY".

- **URLEncode**
  When selected, 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 that result from running the filter.

- **Run Filter**

To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

Parse HTML Result for the Child Text of a Tag

The Parse HTML Result for the Child Text of a Tag filter lets you store the child text of a tag in a property.

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **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 property in which to store the value.

- **Filter Run Results**

  Displays the property and values that result from running the filter.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

Parse HTML Result for HTTP Header Value

The Parse HTML Result to HTTP Header Value filter is commonly used to save the HTTP header Server in a property with the name **SERVER_NAME**.

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **HTTP Header Key**
  The name of the HTTP header; for example, "Server".

- **Property**
  The property in which to store the value.

- **Filter Run Results**

  Displays the property and values that result from running the filter.

- **Run Filter**

To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

Parse HTML Result for the Value of an Attribute

The Parse HTML Result for the Value of an Attribute filter lets you store the text of a specific attribute in a property. The attribute can occur anywhere in the result, including scripting code.

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **Attribute**
  The type of attribute to retrieve. For example, for the URL of an anchor tag, enter "href".

- **Count**
  The occurrence of the tag. For example, for the URL of the third anchor tag on the page, enter "3".

  In this example, "anchor3".

- **Property**
  The property in which to store the value.

- **Filter Run Results**

  Displays the property and values that result from running the filter.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

Parse HTML Result for DevTest Tag

The Parse HTML Result for DevTest Tag filter provides a way for developers to test-enable their web applications. For an in-depth study on test-enabling, see Using the SDK.

This filter lets you 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 provide valuable information about your web page to a tester. An example of a LISAPROP is:

```
<LISAPROP name="FIRST_USER" value="sbellum">.
```

If a tester has installed this type of filter, the property **FIRST_USER** is automatically assigned the value **sbellum**. This filter removes any need for the tester to parse for this value. This type of filter helps a developer make the testing easier.

Frequently a web page does not contain the information that is necessary for proper validation, or that information is difficult to parse. Even when the information exists, subtle changes in the generated HTML can result in incorrect parsing. This filter can resolve many parsing issues for web testing.

No parameters are required.

Choose Random HTML Attribute

The Choose Random HTML Attribute filter lets you store the text of a random selection from a set in a property. The attribute can occur anywhere in the result, including scripting code.

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **Outer Tag**
  The outer element that contains the list from which to pick. For example, to select a drop-down list, enter the text "select".

- **Tag Count**
  The occurrence of the outer tag. For example, to select the second drop-down list, enter the text "2".

- **Inner Tag**
  The tag to pick the attribute from, randomly. To pick a random item in the drop-down list, enter the text "option".

- **Filter Attribute**
  An optional field to specify attribute names to exclude from the pick list.

- **Filter Value**
  An optional field to specify attribute values to exclude from the pick list.

- **Attribute**
  The attribute from which to retrieve text. If this field is blank, the child text of the inner tag is returned.

- **Property Key**
  The property to store the text of the attribute.

- **Filter Run Results**

  Displays the property and values that result from running the filter.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

Parse HTML into List of Attributes

The Parse HTML into List of Attributes filter lets you store the text of a set of attributes as a list in a property.

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **Outer Tag**
  The outer element that contains the list of tags to parse. For example, to store all 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 the anchor tags in the table enter "a".

- **Filter Attribute**
  An optional field to specify attribute names that must not appear in the pick list.

- **Filter Value**
  An optional field to specify attribute values that must not appear in the pick list.

- **Attribute**
  The attribute of the inner tag to retrieve the text from. If this field is blank, the child text of the inner tag is returned. To store all 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 that result from running the filter.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

Parse HTTP Header Cookies

The Parse HTTP Header Cookies filter lets you parse the HTTP header and store cookie values in a property that starts with a specific prefix.

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **Property Prefix**
  A text string that is 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 were 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 that result from running the filter.

- **Run Filter**

To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

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 behavior is easier to understand by examining the example that follows.

You could 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>
```

Dynamic Form Filter window

Using a property key of FormTest in the filter panel creates 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	

Parse HTML Result by Searching Tag Attribute Values

The Parse HTML Result by Searching Tag/Attribute Values filter lets you filter an attribute value by searching the tag for the name and value of another attribute. If multiple tags fit the criteria, you can specify which tag to use.

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **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 tags 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 the attribute value is not found.

- **URLEncode**
  When selected, property value is URLEncoded.

- **Filter Run Results**

  Displays the property and values that result from running the filter.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

Inject HTTP Header

The Inject HTTP Header filter lets you replace a header in an HTTP call with the header defined in the filter. If no such header exists, a new header is added.

> **Note:** For an example that uses this filter, see Test Templates.

This filter is a global filter.

The following test steps are supported:

- HTTP/HTML Request
- REST
- Web Service Execution (XML)

Complete the following fields:

- **Header Key**
  Specifies the key of the header.

- **Header Value**
  Specifies the value of the header.

- **Skip To Step**
  Specifies the last step to exclude from being updated. This step is optional. For example, assume that the test case has four steps: A, B, C, and D. If you set this field to B, then the filter applies only to steps C and D. If you set this field to C, then the filter applies only to step D.

The headers are case sensitive. For example, **Authorization** is different from **authorization**.

The following graphic shows an example of this filter. The **Header Key** field is set to **Authorization** . The **Header Value** field is set to **{{token_type}}{{space}}{{access_token}}**.

Screen capture of Inject HTTP Header filter.

XML Filters

The following filters are available in the XML Filters list for any test step:

- Parse Text from XML
- Read Attribute from XML Tag
- Parse XML Result for DevTest Tag
- Choose Random XML Attribute
- XML XPath Filter

Parse Text from XML

The Parse Text from XML filter stores the child text of a tag in a property. To define a Parse Text from XML filter, set the type of the filter and set the three attributes.

You can create this filter either manually from the filter list or by using the embedded filter commands on an XML response.

To create the filter manually:

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

  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". You can use a property in this field.

- **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". You cannot use a property in this field.

- **Property**
  The property in which to store the value.

- **Filter Run Results**

  Displays the property and values that result from running the filter.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

To create the filter directly from the response page:

Response window, DOM Tree tab

1. From the **DOM Tree** view, select the attribute whose value you want to store in a property.

2. After it is selected, select **Generate Filter for Attribute or Text**.

3. Enter the property name in the dialog.

HTML/XML Filter Info dialog

Assertions can also be added here.

Read Attribute from XML Tag

The Read Attribute from XML Tag filter lets you store the text of a specific attribute in a property. The attribute can occur anywhere in the result.

You can create this filter either manually from the filter list, or by using the embedded filter commands on an XML response.

To create the filter manually:

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **Tag**
  The name of the XML tag; for example, "target".

- **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.

- **Default (if not found)**
  The value to use if the attribute value is not found.

- **URLEncode**
  When selected, property value is URLEncoded.

- **Filter Run Results**

  Displays the property and values that result from running the filter.

- **Run Filter**

To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

To create the filter from the response page:

1. Display the step response that contains the XML.

Step response

2. From the **DOM Tree** view, select the attribute whose value you want to store in a property.

3. When it is highlighted, select **Generate Filter for Attribute or Text**.

4. Enter the property name in the window.

HTML/XML Filter Info

You can also add an assertion here. A Property Value Expression Assertion can be added to this step.

Parse XML Result for DevTest Tag

The Parse XML Result for DevTest Tag filter provides a way for developers to test-enable their XML applications. For an in-depth study on test-enabling, see Using the SDK.

This filter lets you insert LISAPROP tags into your XML page. The LISAPROP tag has two attributes: name and value. The LISAPROP tags function only to provide valuable information about your XML to a tester. An example of a LISAPROP is:

```
<LISAPROP name="FIRST_USER" value="sbellum">.
```

If a tester has installed this type of filter, the property "FIRST_USER" is automatically assigned the value **sbellum**. This filter removes any need on behalf of the tester to parse for this value. This type of filter helps a developer make the testing easier.

Sometimes the XML does not contain the information that is necessary to validate properly, or that information is difficult to parse. Even when the information exists, subtle changes in the generated XML can result in incorrect parsing. This LISAPROP filter can resolve many parsing issues.

No parameters are required.

Choose Random XML Attribute

The Choose Random XML Attribute filter lets you store the text of a random selection from a set in a property. The attribute can occur anywhere in the result. This filter works exactly like Choose Random HTML Attribute.

Example

Using a Parse Text as Response step to output some XML:

And setting up the filter:

Each time you click **Run Filter**, a random value is selected.

To see how Tag Count works, change the content as shown on the following graphic.

When Tag Count = 1, the first list of cars is used; when Tag Count = 2, the second list of cars (exoticCars) is used.

XML XPath Filter

The XML XPath filter lets you use an XPath query that runs on a property or the last response and stores it in a property. When this filter is selected, the last response is loaded into the content panel.

You can view the response as an XML document or as a DOM tree. However, the XPath selection can be made only from the DOM tree.

Construct the XPath query by using one of the following methods:

- Manually enter the XPath expression in the **XPath Query** text box.
- Select an element from the DOM tree and let DevTest construct the XPath expression.
- Select an element from the DOM tree, and then edit the XPath that is constructed. For example, you can modify it to use a property, or a counter data set.

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it in this field. The property must exist.

- **Save To Property**
  The property in which to store the result of the XPath query.

Now, you construct the XPath query using one of the methods described previously.

After an XPath query has been constructed, test it by clicking **Run Filter**. The results of the query appear in the **Filter Run Results** pane.

The **lisa.xml.xpath.computeXPath.alwaysUseLocalName** property controls whether the XPath local-name() function is always used during the XPath generation. The default value is false, which means that the local-name() function is used only when necessary. To generate an XPath that works regardless of an XML node namespace, set the value to true.

> **Note:** The DevTest XML XPath filter is based on the HTML DOM spec. That spec maintains that all DOM element names must be upper case, while attribute names must be lower case.

JSON Filters

The following filter is available in the JSON filters list for any test step:

- JSON Path Filter

JSON Path Filter

The JSON Path filter lets you obtain a property value or length from a JSON result and save it to a property.

The following graphic shows the editor:

Screen capture of JSON Path filter.

> **Note:** Web sites are available to check the validity of the JSON path expression. One example is https://jsonpath.curiousconcept.com/. Another resource is the editor for the Ensure Result Equals assertion.

Complete the following fields:

- **Filter in**
  Specifies the name of the property to filter for the step. This property specifies the source of the JSON that is used in this filter. If the property is not in the pull-down menu, you can enter it. The property must exist. You can edit this value for this filter.

- **JSON Path**
  Designates an expression that consists of a sequence of JSON properties in a JSON document. The JSON Path represents a path to a destination JSON property.
  The right portion of the filter editor can help you determine the value of the JSON Path field. The JSON document for the step response appears. When you select the icon for a node in Name column, the JSON Path field is populated.
  An array filter, denoted by a ?() expression, can be applied to an array as a select criteria to select certain array elements. The following table lists supported filter operators that are available for JSON data types.

Operator	String	Number	Boolean
== (equal)	supported	supported	supported
!= (not equal)	supported	supported	supported
>=	not supported	supported	not supported
<=	not supported	supported	not supported
>	not supported	supported	not supported

<	not supported	supported	not supported
in	supported	supported	not supported
not in	supported	supported	not supported

Sample JSON Path entries (based on the screen shot above):

- **$**

  Returns the entire JSON document.

- **$..name**

  Returns all "name" strings in the entire document.

- **$.[*].name**

  Returns all "name" strings in an unnamed array.

- **$.myarray[*].name**

  Returns all "name" strings in myarray.

- **$.myarray[0].name**

  Returns the "name" string for the first element in the myarray.

- **$.myarray[?(@.id < 5)]**

  Returns the entire element for each element in myarray where the ID is less than 5.

- **$.myarray[?(@.id < 5)].name**

  Returns the “name” string for each element in myarray where the ID is less than 5.

- **Save Value to Property**
  Defines the name of the property where the results of the JSON path filter are saved. If you specify an array in JSON Path field, the saved value is a comma-separated list.

- **Save Length to Property**
  Defines the name of the property that stores number of items in the results from the JSON path filter. If the property value is an array, the number of components is the number of elements in the array. If the property value is a JSON object, the number of components is the number of properties in the JSON object. For simple data types such as String or Number, the number of components is 1.

- **Filter Run Results**

  Displays the property and values that result from running the filter.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

Java Filters

The following filters are available in the Java Filters list for any test step:

- Java Override Last Response Property Filter
- Java Store Step Response Filter
- Java Save Property Value to File Filter

Java Override Last Response Property Filter

The Last Response property contains the response from the previous step. For example, if the previous response was an HTTP step, the last response is the web page that was returned. To make the last response be something other than the default value, use the Override "Last Response" Property filter. This filter lets you replace the current value of the last response with the value of an existing property.

Complete the following fields:

- **Filter in**

  The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

- **Convert to XML**
  If you want the response to be converted to valid XML, select this option.

- **Filter Run Results**

  Displays the property and values that result from running the filter.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

For more information, see Override Last Response Property.

Java Store Step Response Filter

The Store Step Response filter lets you save the last response as a property for future use.

Complete the following fields:

- **Filter in**
  Enter the response to which to apply the filter. The previous graphic shows **lisa.Add User.rsp**, which means that the filter is applied to the response of Add User. You cannot change this value for this filter.

- **Property**
  The property in which to store the value.

- **Filter Run Results**

  Displays the property and values that result from running the filter.

- **Run Filter**

To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

Java Save Property Value to File Filter

The Save Property Value to File filter lets you save the value of an existing property to a file in your file system.

Complete the following fields:

- **Filter in**
  The name of the property whose value you want written 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**
  To append the information to an existing file, select this check box.
- **Filter Run Results**

  Displays the property and values that result from running the filter.
- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

VSE Filters

The following filter is available in the VSE Filters list for any test step:

- Data Protocol Filter

Data Protocol Filter

The Data Protocol 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, which is the appropriate functionallity for the VSE run-time side.

To open its editor, click the filter.

Data Protocol Filter

Complete the following fields:

- **Filter in**
  Enter the response to which to apply the filter. The previous graphic shows **lisa.Get User.rsp**, which means that the filter is applied to the response of Get User. You cannot change this value for this filter.

- **Data Protocol**
  Select the appropriate data protocol to be used from the drop-down list.

- **Process Requests**
  Select to see the process request.

- **Process Responses**
  Select to see the process response.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

CAI Filters

The following filters are available in the Application Insight Filters list for any test step:

- DevTest Integration Support for CA CAI
- DevTest Integration Support for webMethods Integration Server

DevTest Integration Support for CA CAI

The DevTest Integration Support for CA Application Insight filter is a common filter to enable Application Insight for all the technologies that DevTest supports. This filter collects extra information from a Application Insight application.

DevTest supports integration with web services, JMS, servlets, EJB, and Java objects.

Complete the following fields:

- **Error if Max Build Time (millis) Exceeds**
  Enter build time in milliseconds. An error is generated if the build time exceeds the specified interval.

- **On Transaction Error Step**
  Select the step to redirect to on transaction error after filter is set to run.

- **On CAI Warnings Step**
  Select the step to redirect to on Application Insight warnings after filter is set to run.

- **Report Component Content**
  Generate a report of the component content.

- **Force a Garbage Collection on the server at the start & end of the request**
  Forces a garbage collection on the server at the start and end of the request.

- **Fail test if server-side exception is logged**
  Fail the test case if an exception is thrown at the server side.

- **Log4J Level to capture in the test events**
  Select the log4j level that is captured in the test events.

- **Log4J Logger to temporarily change (blank is Root Logger)**
  Enter the name of the logger.

DevTest Integration Support for webMethods Integration Server

The DevTest Integration Support for webMethods Integration Server filter collects additional information from a Application Insight-enabled webMethods Integration Server.

Complete the following fields:

- **Error if Max Build Time (millis) Exceeds**
  Enter build time in milliseconds. An error is generated if the build time exceeds the specified interval.

- **On Transaction Error Step**
  Select the step to redirect to on transaction error after filter is set to run.

- **On CAI Warnings Step**
  Select the step to redirect to on Application Insight warnings after filter is set to run.

Copybook Filters

The following filter is available in the Copybook Filters list for any test step:

- Copybook Filter

Copybook Filter

The Copybook Filter converts copybook payloads to XML at run time. You can review and maintain your data by displaying these payloads in XML.

To open its editor, click the filter.

Complete the following fields:

- **Filter in**
  Specifies the name of the property to filter for the step. If the property is not in the pull-down menu, you can enter it. The property must exist. You can edit this value for this filter.

- **Copybook definition file**
  Designates the location where you store your copybook file definition.

- **Encoding**
  Select a valid Java charset. This value is used to try to convert the bytes in the payload into text for use in the output XML.
  Default: UTF-8

- **Copybook parser column start**
  Copybooks frequently start each line with a line number. This parameter defines the column on which the parser starts when trying to parse a copybook file definition.
  Value: A zero-based inclusive index. However, you can think of it as a "normal" one-based

exclusive index.
Default: 6
Example: If you set this value to 6, the parser skips the first six characters in a line and starts with the seventh character.

- **Copybook parser column end**
  Occasionally, copybooks contain other reference data at the end of each line. When that happens, the parser must know on which column to stop. If there is no "extra" data at the end of the lines in the file, set this number to something greater than the length of the longest line in the file. If this number is greater than the length of a line, the parser stops at the end of the line.
  Value: A zero-based exclusive index. However, you can think of it as a "normal" one-based inclusive index.
  Example: If you set this value to 72, the parser reads the 72nd character in the line and then it stops.

- **Filter Run Results**

  Displays the property and values that result from running the filter.

- **Run Filter**

  To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

For performance purposes, the Copybook filter caches the copybook definition in memory for 86400 seconds. When the time expires, DevTest removes the converted copybook definition from the cache. If the file is needed again, DevTest reads and reconverts it.

Metrics Descriptions

Metrics let you apply quantitative methods and measurements to the performance and functional aspects of your tests, and the system under test.

This section describes the following metrics:

- DevTest Whole Test Metrics
- DevTest Test Event Metrics
- SNMP Metrics
- JMX Metrics
- TIBCO Hawk Metrics
- Windows Perfmon Metrics
- UNIX Metrics Via SSH

DevTest Whole Test Metrics

Whole Test Metrics collect all the basic information about the test case.

The following submetrics are collected:

- **Instances** (the number of virtual users)
- **Avg Resp Time**
- **Max Resp Time**
- **Min Resp Time**
- **Last Resp Time**
- **Steps Per second**

The response time metrics are reported in milliseconds.

> **Note:** The number of virtual users (Instances), average response times (Avg Resp Time), and Steps Per Second sub metrics are added by default to the metric list in a staging document.

DevTest Test Event Metrics

Test Event Metrics provide metrics in regard to DevTest events. These metrics include counters and gauges.

Event metrics let you filter events of a specific type by including a regular expression to match the short description field of the event.

The Test Event Metrics category has the following submetrics:

- **Coordinator server started**
- **Coordinator server ended**
- **Coordinator started**
- **Coordinator ended**
- **Test started**
- **Test ended**
- **Instance started**
- **Instance ended**

To add test event metrics:

In addition to selecting the metric, you can specify:

- **Key Expression Match:** Enter an expression to say to sample the chosen event only if it has this expression in its short description field. If you leave this field blank, or enter *, then every event of this type is reported.

- **Metric is a Counter:** If this box is selected, the value counts over time are recorded. If the box is cleared, the absolute value is recorded (metric is functioning as a gauge).

Test Event metrics are meant to track the presence of specific events in the workflow engine. When the **Counter** check box is selected, the metric collector tracks the number of times that the chosen event was observed. The count increases as the test runs; however, the graphs in DevTest Workstation and the reporting console shows the value that was collected during the sampling period. This is the default option for Test Event Metrics.

If the **Counter** check box is not selected, the metric collector attempts to parse the long description of the event as a number. During the sampling period the collector keeps a running total of these parsed values. When the metric is reported it is a simple average of the parsed numbers. The running total is reset for each sampling period. If there is not a number value that is contained in the long description of the event, the metric value is 0.

The default sampling period is 1 second and can be modified in the staging document.

- Click **OK** to add this metric to the list of metrics.

SNMP Metrics

The SNMP metrics use the Simple Network Management Protocol (SNMP) to monitor the system performance.

Setting up SNMP Support

Configure SNMP before you can collect the SNMP metrics. For more details about setting up SNMP on UNIX and Windows, see Install and Configuring SNMP.

To add SNMP Metrics:

1. Select **SNMP Metric** from the dialog and click **OK**.
   The **Add SNMP Metric** dialog opens.
   The **MIB Groups** tree displays all the SNMP metrics that come standard with DevTest Workstation.
   A MIB is a predefined database of a set of metrics on a specific domain.

   - **Host**
     The **Host** field provides system information about a server hosting a coordinator server. Some host metrics include **hrProcessorLoad** for CPU utilization and **hrSystemUptime** for the amount of time after this host was last initialized.

   - **Server O/S**
     Provides information that is related to the system, like up time, date, and number of users.

   - **BEA WebLogic**
     Provides JDBC, JMS, JVM, socket, servlet, and web application information about a Server running BEA WebLogic. Some 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. Some RDBMS metrics include **rdbmsSrvInfoPageReads** for the number of physical page reads that were completed after the RDBMS was last restarted, and **rdbmsSrvInfoPageWrites** for the number of single page writes that were completed after the RDBMS was last restarted.

- **Oracle**
  Provides information about a server running Oracle. Some 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. Some 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.

2. To select metrics, browse through these choices.

3. Click the target metric in the left panel.
   The required parameters for this target metric are completed in the **MIB Object** form in the right panel. A description of the metric is displayed in the text box. The other information can be ignored or accepted.

4. Enter the domain name or IP address of the host computer (Host) where you are collecting the metrics. To add this metric, click **OK**.

5. Repeat until you have added all the target SNMP metrics.
   To add SNMP metrics that are not in the **MIB Group** tree, manually enter the data (OID) into the **MIB Object** form. An OID is the unique identifier of a specific 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 start from there.
   The **Load From MIB** button lets you browse the file system for MIBs.
   The **SNMP Walk** button lets you browse an SNMP tree.

Note: All SNMP MIBs are supported. The set of MIBs displayed in the Add SNMP Metric dialog are only a sample of the MIBs that are supported. The **Add SNMP Metric** dialog makes it easier to understand the Object IDs (OIDs) in those MIBs. However, any valid OID works, not only those OIDs that are displayed in the **Add SNMP Metric** dialog. A set of all the standard MIBs from IETF and IANA are provided. Those MIBs are stored in the **LISA_HOME\snmp\ietf** and **LISA_HOME\snmp\iana** directories.

JMX Metrics

The JMX metrics use the Java Management Extension (JMX) API to provide metrics.

These JMX connectors are provided for an easy setup:

- 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 applications requires slightly different connection parameters. The values that you require for your server are available from your server administrator. Each provider provides slightly different metrics. To use other JMX features, you can invoke them as RMI steps.

Only numerical attributes are supported.

The following example uses JBoss.

Follow these steps:

1. Select **JMX Attribute Reader** from the dialog.
2. Click **OK**.
   The **Select and Configure JMX Agent** dialog opens.
3. Select **JBoss** from the list of JMX Connectors.
4. Enter the **Server Naming URL** and **Agent RMI name** for your installation.
5. The values that you enter in the remaining fields depend on whether your JBoss server requires authentication to access JMX data.
   - If your JBoss server does not require authentication for JMX access, use the default context factory "org.jnp.interfaces.NamingContextFactory".
   - If your JBoss server does require authentication:
     a. Change the context factory to "org.jboss.security.jndi.JndiLoginInitialContextFactory".
     b. Enter the username and password (principal and credentials) needed for the authentication.
     c. Enter the JAAS login module that will authenticate these credentials.
6. Click **OK**.
   The **JMX Object Attribute Viewer** opens.

JMX Object Attribute Viewer

On the left is the JMX Domain hierarchy for JBoss. JMX metrics use an object-attribute model. In this model, domain objects are published by the specific application server (for example, "system"), and attributes are name/value pairs inside the object.
When you select an object from this tree, then the base attributes of that object are also displayed in the tree. After you select a base attribute, the rest of the attribute name appears in the list in the top right **Object Attribute** panel.
In the previous example, 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.

7. To select one of these attributes (metrics), select the metric, and click **Add** .
   This metric is added to the list of selected metrics in the **Selected Attributes to Monitor** panel at the bottom of the window.
   To remove an attribute from this panel, click **Delete** .

8. Repeat this process until all of your chosen metrics appear in your list (bottom panel).

9. Click **OK** to return to the main metrics panel.
   Notice that the JMX metrics are now on the list of metrics.
   Depending on the application server, vendor-specific JARs could be required to enable the JMX communication with that application server.

10. Similarly, you can also select the **JSE 5 Connector** from the JMX Connector list.

11. Click **OK** to connect to the JMX Agent.

Enable JMX Metrics for Tomcat

Follow these steps:

1. Modify the **catalina.bat** file and add **CATALINA_OPTS**.
   You can use the embedded Jakarta-Tomcat that is packaged with DevTest Solutions.

2. Connect to Tomcat through JConsole.

3. Start DevTest, open a staging document, and go to the **Metrics** tab.

4. Click **Add** .

5. Select **JMX Attribute Reader** and click **OK**.

6. Select **Tomcat 5.0.28**.
   All parameters other than user name and password are prepopulated.

7. Click **OK** and connect to the JMX console.

8. Select a few objects specific to Catalina and add them in the list.

9. Click **OK**.
   These attributes appear in the metrics list.

TIBCO Hawk Metrics

TIBCO Hawk is a tool for monitoring and managing distributed applications and operating systems. Unlike other monitoring solutions, TIBCO Hawk software uses TIBCO Messaging software for communicating and inherits many of its benefits. These benefits include a flexible architecture, enterprise-wide scalability, and location-transparent product components that are simple to configure.

DevTest has out-of-the-box integration with TIBCO Hawk for monitoring distributed applications and operating systems metrics in the context of testing. TIBCO Hawk provides in-container metrics for TIBCO BusinessWorks process archives. By using TIBCO Hawk, DevTest can monitor metrics of all activities in any TIBCO BusinessWorks process that is deployed. This integration facilitates peering inside of a process to understand where bottlenecks are occurring.
Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Follow these steps:

1. Create a staging document.

2. On the **Metrics** tab of the staging document, click **Add** .

3. Select **TIBCO Hawk** from the drop-down list.
   The following parameters are available:

   - **Transport:** Select **Rendezvous** or **EMS**.

   - **Hawk Domain:** Enter the TIBCO Hawk domain.

   - **Rendezvous Service**

   - **Rendezvous Network**

   - **Rendezvous Daemon**

   - **EMS Url**

   - **EMS User**

   - **EMS Password**

4. Click **OK**.
   The **Hawk Object Attribute Viewer** opens.

5. Select the **Process Archive** and expand beneath it.

6. The **GetProcessDefinitions** method retrieves the process definitions that are defined in the Process Archive.

7. Use the Process Definition Name as parameter for the **GetActivities** method.
   TIBCO Hawk provides a number of metrics for Process Activities.

8. Provide a filter with the Process Definition Name and the Activity Name.

9. Select **Method Return Values** and click **OK**.
   Activity Metrics are now monitored for test cases staged with this staging document.

TIBCO Hawk can also be invoked in a test case, through TIBCO Hawk APIs.

> **Note:** It takes a few minutes for TIBCO Hawk to initialize itself and achieve a ready state where it can report metrics. TIBCO notifies DevTest when this ready state is achieved. It is important to set **lisa.net.timeout.ms** to a fairly high value (at least 2-3 minutes) to avoid timeouts when using staging documents that collect the TIBCO Hawk metrics.

Windows Perfmon Metrics

The Perfmon Metrics use Microsoft Windows Perfmon to provide metrics to monitor the system performance on a Windows operating system. These metrics are similar to the SNMP metrics.

Before you can collect the Perfmon metrics, configure Perfmon on your Windows computer.

For information about setting up Perfmon, see Install Performance Monitor (Perfmon).

> **Note:** If you are collecting Perfmon metrics for remote computers, ensure the Remote Registry service is running. This service does not run by default on Windows 7 or Windows 8. You must manually start the Remote Registry service from the Windows Services Manager.

Follow these steps:

1. Select **Windows Perfmon Metrics** from the Metrics dialog and click **OK**.
   The **Windows Perfmon Machine Name** dialog opens.

2. Complete the following fields:

   - **Machine Name**
     Enter the Machine Name of your Windows installation. You can find your machine name by right-clicking **My Computer**, selecting **Properties**, and going to the **Computer Name** tab.

- **User Name**
  Enter the user name of the remote computer.

- **Password**
  Enter the password of the remote computer.

- **Domain**
  Enter the domain name. A domain is optional.

3. Click **OK** to proceed.
   After you enter valid login credentials, a window appears showing all the available metric types. The **Windows Perfmon Metrics** window opens.
   The **Perfmon Metric** window has three panels.

   - The left panel displays the **Selected Performance Category**.

   - The top right panel displays the description of the selected category in the **Counter Description** section.

   - The bottom right panel lists the metric that is selected in the left panel.

4. Select a metric from the **Performance Category** list.
   This list shows all available categories that can be monitored:

   - .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

   After you select the category, it is added to the left panel.

5. Double-click the target metric in the left panel to add it to the **Selected Counters to Monitor** table on the right panel.

6. Repeat until you have added all the target Perfmon metrics.

7. Click **OK**.

UNIX Metrics Via SSH

UNIX Metrics Via SSH is used to collect command-line metrics. This metric gathers inputs such as authentication details, host name, and the selection of metrics to be collected.

The metric data is stored in an XML file that is located in the **LISA_HOME\umetrics** directory by default. You can define a new location by updating the key **stats.unix.xml.folder** in the **local.properties** file.

Each XML file has a unique file name that is based on the operating system:

- **Linux.xml**
- **osx.xml**
- **Unix.xml**
- **windows.xml**

The XML file is used to feed the information about the command and metrics to be collected on a specific operating system. For example, to make the command and metrics known on a Linux operating system, a **Linux.xml** file must be in the **LISA_HOME\umetrics** folder.

The following graphic is an example of an OS X 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>

    <MetricCollectorSet>
```

An OSX command parser for iostat that collects CPU and disk0 metrics

In the **Specify Remote Machine details** window, enter the operating system: Linux, OS X, or Solaris.

Enter the remote computer details:

- Host
- User

Select authentication type and enter information:

- Password
- Private Key
- Encrypted Private Key

When you use a private key or an encrypted private key, the private key file must be in PEM format and in the **Data** folder under the project. Use **Browse** to select the file.

Each operating system has a slightly different set of metrics from which to select. To select the metrics you would like to collect, use the check box in the left column.

Property Descriptions

This section describes each property in the following files:

- DevTest Property File (lisa.properties)
- Custom Property Files
  - Local Properties File
  - Site Properties File
  - logging.properties
  - Enterprise Dashboard Properties (dradis.properties)
  - Portal Properties File (phoenix.properties)

DevTest Property File (lisa.properties)

The **lisa.properties** file stores initialization and configuration information.

> **Important!** Do not modify the **lisa.properties** file.

Property files are stored in the DevTest install directory.

You can display the contents of this file in DevTest Workstation. Select **System**, **Edit Properties** from the main menu.

- Enterprise Dashboard Properties
- Comma-separated List of Paths for Javadoc and Source Code
- System Properties
- Server Properties
- OS X Properties
- Update Notifications
- Basic Defaults
- HTTP Header Keys Properties
- HTTP Field Editor Properties
- Test Case Execution Parameters
- TestEvent Handling Customizations

- Test Manager/Editor Properties
- J2EE Server Parameters
- Native Browser Information to Use for Internal Rendering
- Test Manager/Monitor Properties
- Built-In String-Generator Patterns
- JMX Information
- Test Manager/Interactive Test Run Properties
- External Command Shells
- Testing Parameters
- Properties for Use by StdSchedulerFactory to Create a Quartz Scheduler Instance
- VSE Properties
- Network Port Properties
- Application Insight Properties
- Database Properties
- Mainframe Properties
- Selenium Integration Properties
- VSEasy Properties
- Solr Properties
- Application Insight Stateful Baseline Parameterization
- Application Insight Shelve Frames
- CA Agile Central Integration Properties
- SAP Properties
- Community Edition Properties
- Identity and Access Manager Properties
- VS Recorder

Enterprise Dashboard Properties

- **devtest.enterprisedashboard.host**=localhost

- **devtest.enterprisedashboard.port**=1506
  The host and port to use for sending component, metric, and audit information to the DevTest Enterprise Dashboard.

- **lisa.activation.retry.count**=3

  Number of attempts a registry should make to perform activation with Enterprise Dashboard.

- **devtest.enterprisedashboard.https.enabled**=false
  Specifies that Enterprise Dashboard runs with schema HTTP or HTTP/S.

Comma-separated List of Paths for Javadoc and Source Code

These paths are used to show you class and parameter documentation. The docpath can take directories and URLs that are base paths to the Javadoc. Here is an example that includes JDK docs from a website, but websites are not recommended because of delays. **lisa.java.docPath=LISA_HOME\examples\javadoc,[http://java.sun.com/j2se/1.3/docs/api/**

lisa.java.docPath=LISA_HOME\examples\javadoc

lisa.java.sourcePath=LISA_HOME\examples\src

This sourcepath can take directories as base paths and JAR or zip files of source.

lisa.axis.compiler.version=1.4

This is lisa.axis.compiler.version 1.4.

System Properties

- **file.encoding**
  Encoding for files that DevTest reads or writes.
  Default: UTF-8

- **lisa.supported.html.request.encodings**
  To include the encodings to support in the HTTP/HTML Request step, change the comma-separated list. The underlying JVM must also support all encodings in this list. If a webpage uses an encoding that is not supported in the list, then the encoding is replaced with the DevTest default encoding (file.encoding key in **lisa.properties**). Also, if an encoding is not selected when creating a HTTP/HTML Request step, then the DevTest default encoding is assumed.
  Default: ISO-8859-1, UTF-8, Shift_JIS, EUC-JP

- **lisa.supported.jres**
  Default: 1.7

- **org.jfree.report.LogLevel**
  Default: Error

- **javax.xml.parsers.DocumentBuilderFactory**
  Default: org.apache.xerces.jaxp.DocumentBuilderFactoryImpl

- **javax.xml.parsers.SAXParserFactory**
  Default: org.apache.xerces.jaxp.SAXParserFactoryImpl

- **javax.xml.transform.TransformerFactory**
  Default: org.apache.xalan.processor.TransformerFactoryImpl

- **beanshell.script.default.imports**
  Defines imports that should always be available in a BeanShell Java script**.**
  Format: Enter class and package names, delimited with commas.
  Packages must end with ".*" to be recognized as packages.
  For example, in the following string, two classes and two packages would be imported:

  ```
  beanshell.script.default.imports=com.ca.sv.devtest.util.GenerateString,org.
  slf4j.*,com.itko.lisa.core.helpers.MobileHelper,com.itko.util.*
  ```

Server Properties

- **lisa.net.bindToAddress**
  The IP address that we listen on. By default, we listen on all our IP addresses. You can restrict this value to a specific IP address. To prevent other computers connecting but allow local connections, set the value to *127.0.0.1* or *localhost*.

OS X Properties

- **apple.awt.brushMetalLook**
  Default: false

- **apple.awt.brushMetalRounded**
  Default: false

- **apple.laf.useScreenMenuBar**
  Default: true

- **apple.awt.showGrowBox**
  Default: true

- **com.apple.mrj.application.growbox.intrudes**
  Default: true

- **com.apple.macos.smallTabs**
  Default: true

- **com.apple.mrj.application.apple.menu.about.name**
  Default: LISA

- **com.apple.mrj.application.live-resize**
  Default: true

Update Notifications

lisa.update.every=1

Controls how often DevTest checks to see whether a newer version is available to download. To disable checking, set the value to blank. The other valid values are:

- 0 (check at every startup)
- 1 (check once a day)
- 2 (check every two days), and so on.

lisa.update.URL=http://www.itko.com/download/ga/

Basic Defaults

lisa.registryName=registry

Default name of the registry to attach to, and the default name of the registry when you start it without a name.

lisa.coordName=coordinator

The coordinator server default name when started without an explicit name AND when the Test Runner needs one and you do not specify a coordinator server on the command line.

lisa.simulatorName=simulator

The default name of a simulator daemon if you do not provide one on the command line.

lisa.vseName=VSE

The virtual environment server default name when started without an explicit name.

lisa.defaultRegistry.pulseInterval=30

Status log interval for the registry. The default value is 30 seconds.

lisa.coordinator.pulseInterval=30

The status log interval for the coordinator. The default value is 30 seconds.

lisa.simulator.pulseInterval=30

The status log interval for the simulator. The default value is 30 seconds.

lisa.vse.pulseInterval=30

The status log interval for VSE. The default value is 30 seconds.

lisa.server.projectmap.refresh.pulseInterval=600

The time interval after which the DevTest servers (coordinator and simulator) refresh the map of project names to file paths.

lisa.defaultRegistryConnectionTimeoutSeconds=90

Timeout value in seconds that coordinators and simulators use when connecting to a DevTest registry. A value of 0 indicates an infinite timeout; that is, we wait forever trying to connect.

lisa.regex.helper.tutorial.url=http://download.oracle.com/javase/tutorial/essential/regex/

For the Regex helper window, the URL to show for the regular expression tutorial.

lisa.hooks=com.itko.lisa.files.SampleHook

To register hooks with DevTest; these values are comma-separated.

lisa.project.ignore=CVS|SCCS|RCS|rcs|\.DS_Store|\.svn|vssver\.scc|vssver2\.scc|\.sbas|\.IJI\..*|.*\.pyc|.*\.pyo|\.git|.*\.hprof|_svn|\.hg|.*\.lib|.*~|__pycache__|\.bundle|.*\.rbc

This property is a Java regular expression that lets you hide the types of files that you do not want to see in the Project tree. By default, various well-known control files for third-party products are hidden. To show files that are hidden by default, modify the regular expression.

HTTP Header Keys Properties

The default values for header keys in the HTTP support are in the following list. To get the benefit of the change, remove or change them for all tests. To change them for a test, or even the execution of one HTTP transaction, use TestNode-specific header directives.

- **ice.browser.http.agent**
  Values: compatible, MSIE 6.0, Windows NT 5.0
  Default: Mozilla/4.0

- **lisa.http.header.0.key**
  Default: Pragma

- **lisa.http.header.0.value**
  Default: no-cache

- **lisa.http.header.1.key**
  Default: Cache-Control

- **lisa.http.header.1.value**
  Default: no-cache

- **lisa.http.header.0.key**
  Default: Accept

- **lisa.http.header.0.value**
  Default: image/gif, image/x-xbitmap, image/jpeg

- **lisa.http.header.1.key**
  Default: Accept-Language

- **lisa.http.header.1.value**
  Default: en

- **lisa.http.header.2.key**
  Default: Accept-Charset

- **lisa.http.header.2.value**
  Default: iso-8859-1,*,utf-8

- **lisa.http.header.3.key**
  Default: User-Agent

- **lisa.http.header.3.value**
  Default: Mozilla/4.0, MSIE 6.0; Windows NT 5.0

HTTP Field Editor Properties

The following list shows defaults for fields that appear in the HTTP Field editor. Do not include "Authentication" because the editor adds it automatically.

- **lisa.gui.http.fieldNames**
  Values: Accept,Accept-Language,User-Agent,Connection

- **lisa.webrecorder.textMIMEs**
  Values: html,text,magnus-internal,application/pdf

- **lisa.webrecorder.notTextMIMEs**
  Values: css,script

- **lisa.webrecorder.alwaysIgnore**
  Values: .gif,.jpg,.jpeg,.css,.js,.ico

- **lisa.web.ntlm**
  Enables NTLM authentication in the Test Runner.
  Default: true

Test Case Execution Parameters

lisa.hotDeploy=C\Projects\Lisa\custom_classes

This setting tells the ClassLoader built into DevTest where to look for custom classes. The default is $LISA_HOME\hotDeploy.

lisa.overloadThreshold=1000

The TestNode attempts to determine if the simulator is thrashing by checking the actual amount of time slept in think time as opposed to the amount of think time it was supposed to take. This setting is the amount of additional "slip" in think time that is acceptable before sending a warning TestEvent that the simulator is overloaded. The default of 1000 means that if the simulator sleeps 1 second more than it was supposed to, (presumably because the computer is CPU starved), then raise the TestEvent.

lisa.webservices.encode.empty.xmlns=true

Some web service server stacks require empty xmlns strings in the SOAP request (for example, jbossWs). Others, such as Amazon, do not work with empty xmlns strings. Change this property to suit the stack you are calling.

lisa.webservices.encode.version=1.1

Set the encoding version to force for client stub generation. The default is 1.1.

lisa.tm.sys.min.millis=0

lisa.tm.sys.max.millis=0

The default think time for new system steps, in milliseconds. The system steps include subprocesses, continue, continue (quiet), fail and end.

lisa.numFilters.warning=100

lisa.numAsserts.warning=100

Sometimes, filters and assertions are dynamically added to test steps. In a load test, there could be many thousands of asserts/filters. The preceding two numbers are the threshold before the log generates a WARN level message.

lisa.exception.on.num.exceeded=true

If the threshold is exceeded, should we raise a TestDefException (kills the test)?

lisa.urltrans.encode.queryparams=false

Setting this property to false ensures that the query parameters in an HTTP/HTML URL are not decoded/encoded when executing the step.

lisa.generic.url.decoder=true

Set this property to true when opening a test case in DevTest 7.1.1 or later that has an HTTP /HTML Request step that was created in 7.0.0 or earlier.

TestEvent Handling Customizations

lisa.perfmon.snmp.port=1161

The StatKeeper can load a Perfmon integration class that wraps or implements a platform-specific monitor, like Windows Perfmon, JMX, SNMP, and others. DevTest comes with a Perfmon DLL class and an SNMP class to support producing our stats output to either the Windows Performance monitor OR (not both) as an SNMP agent. For more information, see the documentation on SNMP. The usual port for SNMP is 161, but you must be root to use 161.

lisa.perfmon.class=com.itko.lisa.stats.snmp.SnmpPerfmon

When we do have something that can pump native OS data into a performance monitor, implement a class that can push data into that tool and can put that class name here.

lisa.perfmon.dll=/c:/Projects/Lisa/PerfmonJNI/LISAPerfmonJNI/Debug/LISAPerfmonJNI.dll

Windows Perfmon integration to the StatKeeper has a "DLL" setting for the Windows (native) implementation. Put the full path/file here. You only need this if you are using PerfmonStatKeeperWindows.

Simulators use a separate thread and queue to send TestEvents to the coordinator to cut down on the chattiness of RMI. These are the thresholds that cause the daemon thread to push events. We take the minimum of the size or the max-wait (if either we take too long or have too many, we post them).

lisa.eventPoolPoll=250

How often we check the queue size or we see if we have overrun our max wait (in milliseconds).

lisa.eventPoolSize=64

The maximum size that we let it get before sending.

lisa.eventPoolMaxWait=1000

How long we are willing to go with an event being unforwarded.

Test Manager/Editor Properties

- **gui.show.memory.status**
  Default: false

- **lisa.screencap.delay.seconds**
  Default: 6

- **lisa.screencap.dir**
  Default: LISA_HOME\screens

- **lisa.screen.cap.prefix**
  Default: lisa-screencap-

- **lisa.earsubdir.endingnamepart**
  Default: -contents

- **lisa.model.editor.inspector.scale**
  This property sets the "scale" (primarily, font size) for things in the model and step inspectors of the main model editor. 1.0 is 12 points, so determine a value for this property by dividing the size that you want in points by 12. For example, 11 points is 11/12 = 0.92, 10 points is 10 /12 = 0.83 (the default), 14 points is 14/12 = 1.17.
  Default: 0.83

- **lisa.stats.decimalFormat**
  Some built-in metrics (steps for each second) use a Java DecimalFormat to display floating-point values. To not use the decimal point, make this value "######" or select from a range of displays; see http://download.oracle.com/javase/6/docs/api/java/text/DecimalFormat.html.
  Default: ###,###.#

- **lisa.editor.custJavaNodeEditor.classes**
  This is the comma-separated list of all custom Java test nodes to include in your test case.

- **lisa.editor.combined.report.type**

- **lisa.gui.log4jfmt**
  This property drives the formatting of messages to the **System Messages** window.
  Default: %-5p - %m%n

- **lisa.editor.http.recorderPort**
  The HTTP recorder binds to this port.
  Default: 8010

- **lisa.editor.proxy.webProxySupport**
  Default: on

J2EE Server Parameters

lisa.prefill.jndiNames=	JBOSS=org.jnp.interfaces.NamingContextFactory Weblogic=weblogic.jndi.WLInitialContextFactory Websphere=com.ibm.websphere.naming. WsnInitialContextFactory Borland Enterprise Server=com.inprise.j2ee.jndi.CtxFactory iPlanet/Sun AS=com.sun.jndi.cosnaming.CNCtxFactory
lisa.prefill.jndiUrlPrefix=	JBOSS=jnp:// Weblogic=t3:// Websphere=iiop:// Borland Enterprise Server=iiop:// iPlanet/Sun AS=iiop://
lisa.prefill.jndiDefPort=	JBOSS=1099 Weblogic=7001 Websphere=2809 Borland Enterprise Server=1099 iPlanet/Sun AS=1099
lisa.prefill.jndiNeedsClass=	JBOSS=false Weblogic=false Websphere=true Borland Enterprise Server=true iPlanet/Sun AS=true
lisa.prefill.jndiFactories=	org.jnp.interfaces.NamingContextFactory weblogic.jndi.WLInitialContextFactory com.ibm.websphere.naming.WsnInitialContextFactory com.webmethods.jms.naming.WmJmsNamingCtxFactory com.tibco.tibjms.naming.TibjmsInitialContextFactory com.inprise.j2ee.jndi.CtxFactory com.sun.jndi.cosnaming.CNCtxFactory fiorano.jms.runtime.naming.FioranoInitialContextFactory
lisa.prefill.jndiServerURLs=	jnp://SERVER:1099&t3://SERVER:7001 iiop://SERVER:PORT tibjmsnaming://SERVER:7222&iiop://SERVER:PORT iiop://localhost:9010&wmjmsnaming://Broker #1@SERVER:PORT /JmsAdminTest
lisa.editor.URLTransEditor. protos=	http,https
lisa.editor.URLTransEditor. hosts=	
lisa.editor.URLTransEditor. ports=	80,443
lisa.editor.URLTransEditor. files=	
lisa.prefill.jdbc.names=	Oracle SQL Server WLS Oracle JDataStore Sybase DB2 MySQL Derby

lisa.prefill.jdbc.jdbcDrivers=	oracle.jdbc.driver.OracleDriver com.microsoft.sqlserver.jdbc.SQLServerDriver com.microsoft.jdbc.sqlserver.SQLServerDriver weblogic.jdbc.oci.Driver com.borland.datastore.jdbc.DataStoreDriver com.sybase.jdbc.SybDriver com.ibm.db2.jcc.DB2Driver org.gjt.mm.mysql.Driver org.apache.derby.jdbc.ClientDriver
lisa.prefill.jdbc. jdbcConnectionURLs=	jdbc:oracle:thin:@SERVER:1521:SIDNAME jdbc:sqlserver://SERVER:PORT;databasename=DBNAME jdbc:microsoft:sqlserver://SERVER:PORT jdbc:weblogic:oracle:TNSNAME jdbc:borland:dslocal:DBNAME jdbc:sybase:Tds:SERVER:PORT/DBNAME jdbc:db2://SERVER:PORT/DBNAME jdbc:mysql://SERVER:PORT/DBNAME jdbc:derby://DBSERVER:DBPORT/DBNAME

Native Browser Information to Use for Internal Rendering

lisa.internal.browser.on=true

lisa.internal.browser.win=com.itko.lisa.web.jfxbrowser.JFxBrowserUtils

lisa.internal.browser.win.jxb.type=Default

lisa.internal.browser.osx=com.itko.lisa.web.jfxbrowser.JFxBrowserUtils

lisa.internal.browser.linux=com.itko.lisa.web.jfxbrowser.JFxBrowserUtils

#lisa.internal.browser.sol-sparc=null

#lisa.internal.browser.imgs=false

WR type: mozilla, safari, msie

#lisa.internal.browser=msie

lisa.internal.browser.swing.heavy=false

lisa.internal.browser.usejsinviews=yes

lisa.example.wsdls=http://localhost:8080/itko-examples/services/UserControlService?wsdl

Test Manager/Monitor Properties

- **monitor.events.maxrows**
  The maximum number of event rows that are kept in the Test Manager as a test is run. The number of objects that are kept is twice this number.
  Default: 500

- **lisa.tm.sysmess.size**
  The system messages window maximum size. The memory that is consumed is twice this value.
  Default: 10240

Built-In String-Generator Patterns

lisa.patterns.stringgenerator.types=&Phone=(DDD)DDD-DDDD, &SSN=DDD-DD-DDDD, &Date=Lll-DD-DDDD, &Zip=D*(5)

JMX Information

lisa.jmx.types	com.itko.lisa.stats.jmx.JSE5Connection com.itko.lisa.stats.jmx.TomcatConnection com.itko.lisa.stats.jmx.JBossConnection com.itko.lisa.stats.jmx.JSR160RMIConnection com.itko.lisa.stats.jmx.WeblogicConnector com.itko.lisa.stats.jmx.Weblogic9Connector com.itko.lisa.stats.jmx.WebsphereSOAPConnection com.itko.lisa.stats.jmx.ITKOAgentConnection com.itko.lisa.stats.jmx.OracleASConnector
lisa.jmx.typeprops	com.itko.lisa.stats.jmx.JSE5Connection=LISA_JMX_JSE5 com.itko.lisa.stats.jmx.TomcatConnection=LISA_JMX_TOMCAT5 com.itko.lisa.stats.jmx.JBossConnection=LISA_JMX_JBOSS3240 com.itko.lisa.stats.jmx.JSR160RMIConnection=LISA_JMX_JSR160RMI com.itko.lisa.stats.jmx.Weblogic9Connector=LISA_JMX_WLS9 com.itko.lisa.stats.jmx.WeblogicConnector=LISA_JMX_WLS6781 com.itko.lisa.stats.jmx.OracleASConnector=LISA_JMX_OC4J com.itko.lisa.stats.jmx.WebsphereSOAPConnection=LISA_JMX_WASSOAP5X com.itko.lisa.stats.jmx.ITKOAgentConnection=LISA_JMX_ITKOAGENT

You do not generally need anything for these parameters.

- **LISA_JMX_JSR160RMI**
  Default: LISA_HOME/lib/mx4j.lib

- **LISA_JMX_ITKOAGENT**
  Default: LISA_HOME/lib/mx4j.lib

- **LISA_JMX_JSE5**
  If you are running JSE 5, you do not need to change this property. We ship a jbossall-client that includes what you need, assuming the version is right.

- **LISA_JMX_JBOSS3240**
  Default: LISA_HOME/lib/mx4j.lib{{path.separator}}LISA_HOME/hotDeploy/jbossall-client.jar

- **LISA_JMX_TOMCAT5**
  This parameter is for Tomcat.
  Default: LISA_HOME/lib/mx4j.lib{{path.separator}}LISA_HOME/hotDeploy/mx4j-tools.jar

- **LISA_JMX_WLS9**
  Default: LISA_HOME/hotDeploy/weblogic.jar{{path.separator}}LISA_HOME/hotDeploy/wljmxclient.jar

- **LISA_JMX_WLS6781**
  This parameter must change to the location of your weblogic.jar. No wlclient.jar will not work.
  Default: LISA_HOME/hotDeploy/weblogic.jar

Oracle AS

- **LISA_JMX_OC4J**
  Default: LISA_HOME/lib/oc4jclient.jar{{path.separator}}LISA_HOME/lib/adminclient.jar

IBM WebSphere

- **LISA_JMX_WASSOAP5X**
  It is easier to use your IBM/WAS CLASSPATH before you start DevTest and leave this blank.
  Default: LISA_HOME/lib/mx4j.lib

- **lisa.alert.email.emailAddr**
  If you use performance monitoring alerts, this parameter is the "from" email address for those alerts.
  Format: lisa@itko.com

- **lisa.alert.email.defHosts**
  This parameter is the email server that we try to route emails with (SMTP server).
  Default: localhost

- **lisa.alert.email.smtpPort**
  This parameter specifies the port on which the SMTP server runs.

- **lisa.rundoc.builtins**
  Values:

  - **com.itko.lisa.files.1user1cycle.stg**
    Run the test case once with one simulated user

  - **com.itko.lisa.files.1user1cycle0think.stg**
    Run the test case once with one simulated user and no think time

  - **com.itko.lisa.files.1user1min.stg**
    Run the test case with one simulated user for one minute, restarting the test as needed

  - **com.itko.lisa.files.1user5min.stg**
    Stage the test for 5 minutes, restarting as needed

  - **com.itko.lisa.files.1usernonstop.stg**
    Execute this test until manually stopped

  - **com.itko.lisa.files.5user1min.stg**
    Execute this test with 5 virtual users for 1 minute

- **lisa.auditdoc.builtins**
  Default: com.itko.lisa.files.DefaultAudit.aud

Test Manager/Interactive Test Run Properties

- **lisa.tm.itr.max.delay.seconds**
  Default: 5

External Command Shells

test.cmde.win.shell=cmd /c

test.cmde.unix.shell=sh -c

test.cmde.Windows.NT.(unknown).shell=cmd /c

Testing Parameters

lisa.props.blankOnMissing=true

Property to use if you want DevTest to replace key with an empty string if key is not in state.

lisa.test.custevents=&101="Custom Event 101"; &102="Custom Event 102"

Custom events: the first allowed event number is 101, so we have registered two as examples here.

lisa.SimpleWebFilter.responseCodeRegEx=[45] d d

lisa.fsss.dateformat=MM/dd/yyyy hh:mm:ss a

Properties for Use by StdSchedulerFactory to Create a Quartz Scheduler Instance

Configure Main Scheduler Properties

- **org.quartz.jobStore.class**
  Default: org.quartz.simpl.RAMJobStore
- **org.quartz.threadPool.class**
  Default: org.quartz.simpl.SimpleThreadPool
- **org.quartz.threadPool.threadCount**
  Default: 5
- **lisa.meta-refresh.max.delay**
  Default: 5

Platform-Specific Parameters in the TM

- **lisa.tm.exec.unix**
  Default: xterm -e {0}
- **lisa.tm.exec.win**
  Default: cmd /c start {0}
- **lisa.tm.exec.osx**
  Default: open -a /Applications/Utilities/Terminal.app {0}

Properties Used by Swing Testing Support

- **lisa.swingtest.client.logging.properties.file**
  Uncomment to use a custom Log4J logging properties file in SwingTestProgramStarter.
  Default: C:/Lisa/swingtestclient-logging.properties

License Settings

- **laf.request**
  Default: laf/license.do

- **laf.default.url**
  Default: https://license.itko.com

- **laf.displaysetting**
  Default: true

Reporting Properties

- **lisa.reporting.defaultPageSize**
  Values: A4 | letter

Reporting JPA Properties

- **rpt.eclipselink.ddl-generation**
  Default: create-tables

- **rpt.eclipselink.ddl-generation.output-mode**
  Default: database

- **rpt.eclipselink.validateschema**
  Default: false

- **perfmgr.rvwiz.whatrpt.autoExpire**
  Default: true

- **perfmgr.rvwiz.whatrpt.expireTimer**
  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).
  Default: 30d

- **rpt.hibernate.validateschema**
  Validate the report database schema. By default, only the registry validates the schema.
  Default: false

- **lisa.0.registry.local.autoshutdown**
  Default: true

- **lisa.8.registry.local.autoshutdown**
  Default: true

- **lisa.10.registry.local.autoshutdown**
  The default behavior is to not automatically shut down the local registry if it was started automatically. The exceptions are DevTest Workstation, VSE, and VSE Workstation.
  Default: true

- **lisa.4.registry.local.autoshutdown**
  Default: false

- **lisa.5.registry.local.autoshutdown**
  JUnit and TestRunner should never auto shut down the registry.
  Default: false

Property Used for the Eclipse Connector

- **lisa.eclipse.connector.port**
  Default: 8546

VSE Properties

- **lisa.magic.string.min.length**
  Defines the minimum length of the argument value in a VSE transaction request that is required to consider that argument for constructing a magic string.
  Default: 3

- **lisa.magic.string.word.boundary.type**
  Defines how searches in VSE for magic string content relate to word boundaries.
  Values:

  - **none:** The word boundaries do not matter
  - **start:** Magic string candidates must start on a word boundary
  - **end:** Magic string candidates must end on a word boundary
  - **both:** Magic string candidates must be found as whole words; that is, a word boundary on both ends

  Default: both

- **lisa.magic.string.exclusion**
  Specifies whether to exclude specific strings from eligibility for magic stringing.
  Values: Yes, YES, yes, No, NO, no, true, True, TRUE, false, False, FALSE, __NULL
  Default: Yes, YES, yes, No, NO, no, true, True, TRUE, false, False, FALSE, __NULL

- **lisa.magic.string.xml.tags**
  Specifies whether to change text in XML tags to magic strings.
  Default: false

- **lisa.vse.server.dir.full.service.name=false**
  Specifies whether multiple VSE instances run on different computers from the same installation directory. If they do, uncomment this property and set it to true.
  Default: false

- **lisa.vse.response.xml.prettyprint**
  Specifies whether to format the XML in the service image if VSE records an XML response. The default, false, indicates DevTest does not prettyprint, format, or add line breaks to XML. If the XML arrives as one long string, DevTest displays it that way.
  Default: false

- **lisa.vse.remember.execution.mode**
  Specifies whether VSE remembers the execution modes you set with the VSE Dashboard for virtual services. Comment out this property if you do not want VSE to remember the execution modes across shutdowns.
  Default: true

- **lisa.vse.match.event.buffer.size**
  Sets the number of events for each VS model to define how many matching related events VSE buffers. The property sets the number of events for each VS model. For example, if two

VS models are deployed with the default event buffer size of 100, then 200 events are buffered. These events are the source for the **Match** tab of the VS model inspection page in the VSE dashboard.
Default: 100

- **lisa.vse.request.event.set.buffer.size**
  Defines the number of inbound VSE requests for which a full set of DevTest events is buffered. The property sets the number of events for each VS model. For example, if two VS models are deployed with the default request buffer size (5), ten sets of events (grouped by inbound request) are buffered. These sets of events are the source for the **Events** tab of the VS model inspection page in the VSE dashboard.
  For performance reasons, VSE holds 50 events for processing. This limit can result in the removal of events from the processing queue and the absence of in the Inspection View. To prevent the truncation of events, you can increase the size of the event processing queue. However, this action can increase memory usage and can reduce performance.
  Default: 5

- **lisa.vse.si.text.editor.order**
  Defines the order in which registered VSE text response editors are queried when using autodetect. Only enough of the "right end" of the class name to make it unique is required.
  Values: XMLTextEditor, JSONTextEditor
  Default: The default text editor

- **lisa.tm.def.min.millis**
  Defines (in milliseconds) the default minimum think time for new "regular" steps.
  Default: 500

- **lisa.tm.def.max.millis**
  Defines (in milliseconds) the default maximum think time for new "regular" steps.
  Default: 1000

- **lisa.vse.rest.max.optionalqueryparams**
  Specifies the maximum number of optional query parameters to use per method in a WADL, Swagger, or RAML file. If there are more than *lisa.vse.rest.max.optionalqueryparams* optional query parameters specified in one of these files, only the first *lisa.vse.rest.max. optionalqueryparams* are used to create transactions.
  Default: 5

Date-Checker Properties

The VSE date utilities use the following properties to determine which date patterns are considered valid for date sensitivity conversions. Each of the following entries represents a regular expression that VSE considers as a part of a date.

- **lisa.vse.datechecker.dayregex**
  ((\[12\]\\d)\|(3\[01\])\|(0?\[1-9\]))

- **lisa.vse.datechecker.monthnumberregex**
  ((1\[012\])\|(0 \\d)\|0\[1-9\]\|\[1-9\])

- **lisa.vse.datechecker.monthalpharegex**
  (\\bJAN\\b\|\\bFEB\\b\|\\bMAR\\b\|\\bAPR\\b\|\\bMAY\\b\|\\bJUN\\b\|\\bJUL\\b\|\\bAU
  \\b)

- **lisa.vse.datechecker.yearlongregex**
  \\d\\d\\d\\d

- **lisa.vse.datechecker.yearshortregex**
  \\d\\d

- **lisa.vse.datechecker.timeregex**
  (\\s?((\[012\]? \\d)\|(2\[0123\]))\:((\[012345\] \\d)\|(60))\:((\[012345\] \\d)\|(60)))

- **lisa.vse.datechecker.time.hhmmssregex**
  ((\[012\]? \\d)\|(2\[0123\]))\:((\[012345\] \\d)\|(60))\:((\[012345\] \\ d)\|(60))

- **lisa.vse.datechecker.time.millisregex**
  ((\[012\]?\\d)\|(2\[0123\]))\:((\[012345\]\\d)\|(60))\:((\[012345\]\\d)\|(60))\\.((\\d \\d \\d) \|0)

- **lisa.vse.datechecker.time.millis.zoneregex**
  ((\[012\]?\\d)\|(2\[0123\]))\:((\[012345\] \\d)\|(60))\:((\[012345\] \\ d)\|(60)) \\.((\\d \\d \\d)\|0) \\s(\[A-Za-z\]\[A-Za-z\]\[A-Za-z\])

- **lisa.vse.datechecker.wstimestampregex**
  \\d\\d\\d\\d-((1\[012\])\|(0\\d)\|0\[1-9\]\|\[1-9\])-((\[12\]-\\-d)\|(3\[01\])\|(0?\[1-9\]))T((\[012\]?-\\-d)\|(2\[0123\]))\:((\[012345\]-\\-d)\|(60))\:((\[012345\]-\\-d)\|(60))-\\-.-\\-d-\\-d-\\-d\[-+\]\\d\\d\\d \\d

- **lisa.vse.datechecker.ddmmmyyyyregex**
  ((\[12\]\\d)\|(3\[01\])\|(0?\[1-9\]))
  (JAN\|FEB\|MAR\|APR\|MAY\|JUN\|JUL\|AUG\|SEP\|OCT\|NOV\|DEC)\\d \\d \\d \\d

- **lisa.vse.datechecker.mmmddyyyyregex**
  (JAN\|FEB\|MAR\|APR\|MAY\|JUN\|JUL\|AUG\|SEP\|OCT\|NOV\|DEC)((\[12\]\\d)\|(3\[01\])\|(0?\[1-9\]))\\d \\d \\d \\d

- **lisa.vse.datechecker.yyyyddmmmregex**
  \\d\\d \\d \\d((\[12\] \\d)\|(3\[01\])\|(0?\[1-9\]))
  (JAN\|FEB\|MAR\|APR\|MAY\|JUN\|JUL\|AUG\|SEP\|OCT\|NOV\|DEC)

- **lisa.vse.datechecker.yyyymmmddregex**
  \\d\\d\\d\\d(JAN\|FEB\|MAR\|APR\|MAY\|JUN\|JUL\|AUG\|SEP\|OCT\|NOV\|DEC)((\[12\] \\d)\|(3\[01\])\|(0?\[1-9\]))

- **lisa.vse.datechecker.ddmmmregex**
  ((\[12\]\\d)\|(3\[01\])\|(0?\[1-9\]))
  (JAN\|FEB\|MAR\|APR\|MAY\|JUN\|JUL\|AUG\|SEP\|OCT\|NOV\|DEC)

- **lisa.vse.datechecker.mmmddregex**
  (JAN\|FEB\|MAR\|APR\|MAY\|JUN\|JUL\|AUG\|SEP\|OCT\|NOV\|DEC)((\[12\]\\d)\|(3\[01])|(0?\[1-9\]))

- **lisa.vse.datechecker.ddmmmyyregex**
  ((\[12\]\\d)\|(3\[01\])\|(0?\[1-9\]))
  (JAN\|FEB\|MAR\|APR\|MAY\|JUN\|JUL\|AUG\|SEP\|OCT\|NOV\|DEC)\\d\\d

In VSE, each of the following entries represents a valid pattern for part of date:

lisa.vse.datechecker.dayformat

Format: dd

- **lisa.vse.datechecker.monthnumberformat**
  Format: MM

- **lisa.vse.datechecker.monthalphaformat**
  Format: MMM

- **lisa.vse.datechecker.yearlongformat**
  Format: yyyy

- **lisa.vse.datechecker.yearshortformat**
  Format: yy

- **lisa.vse.datechecker.timeformat**
  Format: HH:mm:ss

- **lisa.vse.datechecker.time.hhmmssformat**
  Format: HH:mm:ss

- **lisa.vse.datechecker.time.millisformat**
  Format: HH:mm:ss.SSS

- **lisa.vse.datechecker.time.millis.zoneformat**
  Format: HH:mm:ss.SSS z

- **lisa.vse.datechecker.wstimestampformat**
  Format: yyyy-MM-dd'T'HH:mm:ss.SSSZ

The following date patterns cannot be constructed by using a combination for the previous patterns that involve at least day, month, and year:

lisa.vse.datechecker.mmmddyyyy.separatorformat

Format: MMM*dd*yyyy

- **lisa.vse.datechecker.mmddyyyy.separatorformat**
  Format: MM*dd*yyyy

- **lisa.vse.datechecker.ddmmmyyyy.separatorformat**
  Format: dd*MMM*yyyy

- **lisa.vse.datechecker.ddmmyyyy.separatorformat**
  Format: dd*MM*yyyy

- **lisa.vse.datechecker.yyyymmmdd.separatorformat**
  Format: yyyy*MMM*dd

- **lisa.vse.datechecker.yyyymmdd.separatorformat**
  Format: yyyy*MM*dd

- **lisa.vse.datechecker.ddmmmyyyyformat**
  Format: ddMMMyyyy

- **lisa.vse.datechecker.mmmddyyyyformat**
  MMMddyyyy

- **lisa.vse.datechecker.yyyyddmmmformat**
  Format: yyyyddMMM

- **lisa.vse.datechecker.yyyymmmddformat**
  Format: yyyyMMMdd

- **lisa.vse.datechecker.ddmmmyyformat**
  Format: ddMMMyy

- **lisa.vse.datechecker.ddmmmformat**
  Format: ddMMM

- **lisa.vse.datechecker.mmmddformat**
  Format: MMMdd

- **lisa.vse.datechecker.separators**
  Defines the valid separator characters to use in the date formats.
  Values: -/.
  Example: lisa.vse.datechecker.separators=/ sets '/' as a separator as in 10/15/2011.

- **lisa.vse.datechecker.top.priorityorder=lisa.vse.datechecker.wstimestampformat**
  Defines the order in which to match the date pattern. The separator characters that **lisa.vse.datechecker.separators** defines replace the asterisks.
  Example: MM*dd*yy generates four date patterns: "MM-dd-yyyy", "MM dd yyyy", "MM/dd /yyyy", "MM.dd.yyyy".

lisa.vse.datechecker.date.priorityorder

lisa.vse.datechecker.mmmddyyyy.separatorformat&\
lisa.vse.datechecker.mmddyyyy.separatorformat&\
lisa.vse.datechecker.ddmmmyyyy.separatorformat&\
lisa.vse.datechecker.ddmmyyyy.separatorformat&\
lisa.vse.datechecker.yyyymmmdd.separatorformat&\
lisa.vse.datechecker.yyyymmdd.separatorformat&\
lisa.vse.datechecker.mmmddyyyyformat&\
lisa.vse.datechecker.ddmmmyyyyformat&\
lisa.vse.datechecker.yyyyddmmmformat&\
lisa.vse.datechecker.yyyymmmddformat&\
lisa.vse.datechecker.ddmmmyyformat

lisa.vse.datechecker.time.priorityorder

lisa.vse.datechecker.time.millis.zoneformat&\
lisa.vse.datechecker.time.millisformat&\
lisa.vse.datechecker.time.tenthsformat&\
lisa.vse.datechecker.time.hhmmssformat

lisa.vse.datechecker.mmmddformat&\
lisa.vse.datechecker.ddmmmformat

lisa.vse.datechecker.months

Values: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC

lisa.vse.datechecker.coarse.year.regex

Format: (?ism).*((\\d[./-](19|20)[0-9]{2})|
((JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC)[. /-](19|20)[0-9]{2})|((19|20)[0-9]{2}[. /-][0-9]{2})).*

Defines a coarse level test to determine whether the payload contains anything that resembles a year. To avoid matching too much, it looks nearby for other identifying information such as month information and common date separators (for example, - or / or .).

When the year is difficult to distinguish from any other four-digit number, this pattern may not match. In that case, your magic dates are not found. Try one of the following solutions:

- Tweak this RegEx
- Try the more permissive form that follows

lisa.vse.datechecker.coarse.year.regex

Format: (?ism).*((\\d[./-]?(19|20)[0-9]{2})|((JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC)[. /-]?(19|20)[0-9]{2})|((19|20)[0-9]{2}[. /-]?[0-9]{2})).*

Defines a more permissive coarse level test than the previous test to determine whether the payload contains anything that resembles a year. If the dates are not recognized, this Regex may allow them to be found. To avoid matching too much, it looks nearby for other identifying information such as month information and common date separators (for example, - or / or .).

However, this Regex can have some performance impact. If this Regex matches too many dates, the patterns above run in situations where they are not applicable. Running large numbers of Regex patterns against large payloads can be time consuming.

VSE JMS Messaging - Custom JMS Properties to Ignore

- **vse.jms.ignore.proplist**
  Some JMS platforms include extra custom properties in the JMS messages they deliver. These properties can interfere with VSE creation. **vse.jms.ignore.proplist** contains a list of properties to ignore when recording a JMS service with the VSE messaging recorder. The format is a comma-delimited list of regular expressions. By default, it excludes standard JMS extension properties (JMSX) and JBoss custom bookkeeping properties (JMS_.).
  Default: JMSX.*, JMS_.*

VSE Recorder Conversation Batch Size

- **lisa.vse.recorder.conversation.batch.size**
  Defines how many conversations we process before committing them to the database. If you have a few large conversations, make this number small. If you have many smaller conversations, make this number large. A batch size of <= 0 is the same as a batch size of 1.
  Default: 10

VSE Service Image Database Settings

- **eclipselink.ddl-generation**
  Default: create-tables

- **eclipselink.ddl-generation.output-mode=database**
  Defines the database to use as the repository for service images. VSE uses the open source EclipseLink JPA provider. By default we use our reports database as the repository for service images.
  Default: LISA reports database

Validate the VSE Schema

- **lisa.eclipselink.query.warn.threshold**
  If EclipseLink is configured to do query timings (the default), this value defines the maximum interval (in milliseconds) allowed for a database query before the **com.itko.lisa.vse.stateful.model.SqlTimer logger** raises a WARN-level message. If you set the threshold on that logger

to DEBUG, timings return for all SELECT statements EclipseLink issues. This logging is the first step in debugging VSE performance issues. If the database is local and all the indexes are created, then you should not see anything over about 20 ms. However, a database on the network that is not under your control and possibly missing indexes could easily take 100ms or more to return a result. In that case, VSE can seem to be slow when in fact it is fast if correctly deployed. Most service images are small enough or have a working set sufficiently small enough to fit into the VSE entity cache. Therefore, the number of SELECTs that VSE issues should dwindle to zero, provided the VSE has a large enough heap. The cache maintains soft references by default.
Default: 100

VSE Delimited Content Detection Properties

- **lisa.vse.delimited.delimiter.check.maxchars**
  Defines the maximum number of characters to inspect to determine the content type.
  Default: 1000

- **lisa.vse.delimited.delimiter.check.maxpercent**
  Defines the maximum percentage of characters to inspect to determine the content type. This property is used if it is higher than **lisa.vse.delimited.delimiter.check.maxchars**.
  Default: 2

- **lisa.vse.delimited.delimiter.list**
  Defines the characters that the Delimited Content Detector looks for as delimiters.
  Default: :,\\,;,|,\u00A6,\t,\n

- **lisa.vse.delimited.delimiter.minimum.threshold**
  Defines the minimum number of delimiters that are counted for the Delimited Content Detector. The payload must have at least this many delimiters.
  Default: 3

- **lisa.vse.delimited.namevalue.separator.list**
  Defines the Name-Value separators that are used to detect the delimited name-value payload. These values must not be in the **lisa.vse.delimited.delimiter.list**.

REST Data Protocol Properties

- **lisa.protocol.rest.editor.observedtraffic.max**
  Defines the maximum number of items of matching traffic that are populated in the lower pane of the URI rules wizard. If the wizard takes a long time to populate the lower pane with matching traffic, try using a lower value in this field.
  Default: 100

- **lisa.protocol.rest.editor.unmatchedtraffic.max**
  Defines the maximum number of items of unmatched traffic that display in the unmatched traffic dialog. If the wizard takes a long time to populate this wizard, try using a lower value in this field.
  Default: 100

- **lisa.protocol.rest.idPattern**
  Defines for the REST data protocol a regular expression that detects the parts of an HTTP request that it can consider to be an identifier. The data protocol converts such identifiers to dynamic parameters. When you have dynamic parameters that you know follow a specific format, you can use this property to detect them.
  Default: [a-zA-Z]+[0-9]{5,}[a-zA-Z]*
  Example: The value **id12345** can denote a stock item and the subsequent rule contains {URLPARAM0}.

- **lisa.protocol.rest.maxChanges**
  Defines the maximum number of changes that VSE allows for a token before the variability is considered significant enough to generate a rule.
  Default: 1

- **lisa.protocol.rest.parameterBaseName**
  Defines the prefix that the REST data protocol uses for the parameters in rules.
  Default: URLPARAM

- **lisa.protocol.rest.startPosition**
  Defines the position in the URL at which the REST data protocol starts looking for variable tokens.
  Default: 3
  Example: In the URI **GET /a/b/c/d**, **GET** is at position 0, **a** is at position 1, and **b** is at position 2.

- **vse.log.trace.truncate.response.at=2048**
  Limits the size of responses that DevTest writes to the vse.log. This property is only applied when trace-level logging is enabled: specifically, when **com.itko.lisa.vse.http.Transaction** =TRACE or when **com.itko.lisa.vse.stateful.protocol.http.Coordinator**=TRACE.
  Values:

  - **0**: The entire response is logged.

  - **>0**: The logged response is truncated at the character specified.

  Default: 2048

Network Port Properties

The network port properties take the form of **lisa.net.APPID.port**.

If you must change these defaults, override them in the **site.properties** file. Clients make assumptions about what port to connect to their server based on these values.

- **lisa.net.0.port**
  DevTest Workstation
  Default: 2008

- **lisa.net.2.port**
  Coordinator
  Default: 2011

- **lisa.net.3.port**
  Registry
  Default: 2010

- **lisa.net.4.port**
  JUnit exec
  Default: 2012

- **lisa.net.5.port**
  Test Runner (command line)
  Default: 2005

- **lisa.net.6.port**
  Others
  Default: 2007

- **lisa.net.8.port**
  VSE
  Default: 2013

- **lisa.net.9.port**
  VSEManager
  Default: 2004

- **lisa.net.11.port**
  ServiceManager
  Default: 2006

- **lisa.net.1.port**
  Simulator (we start here so it is easier to have many on the same computer).
  Default: 2014

Application Insight Properties

- **lisa.pathfinder.broker.host**
  Default: 0.0.0.0

- **lisa.pathfinder.broker.port**
  Default: 2009

- **broker.webserver.port**
  Default: 1508

- **broker.webserver.host**
  Default: 0.0.0.0

- **broker.webapps.directory**
  Default: broker

The following properties are miscellaneous properties:

- **lisa.webserver.port**
  Our embedded web server.
  Default: 1505

- **devtest.port**
  Default: 1507

- **lisa.enable.workstation.pathfinder.browser.ui**
  Default: true

- **lisa.portal.enable.pathfinder.browser.ui**
  Default: true

- **lisa.webserver.host**
  The host of our embedded web server. *0.0.0.0* binds to all local addresses.
  Default: 0.0.0.0

- **lisa.webserver.acl.authmodule.baseurl**
  Default: /acl/

- **lisa.database.migration.safeguards**
  This property is only used when upgrading DevTest Solutions. When enabled, the installer confirms the required database permissions before executing the actual schema upgrade.
  Default: true

These properties set the values for all the portal app launchers from DevTest Workstation. Currently, the TR host name is used for the Host value so we get that value dynamically. The URL is launched externally in your system's browser.

- **lisa.portal.url.prefix**
  Default: http://* (http://*)

- **lisa.portal.root.base.url**
  Default: /index.html

- **lisa.portal.cvsdashboard.base.url**
  Default: /index.html?lisaPortal=cvsdashboard

- **lisa.portal.pathfinder.console.base.url**
  Default: /index.html?lisaPortal=pathfinder

- **lisa.portal.server.console.base.url**
  Default: /index.html?lisaPortal=serverconsole

- **lisa.portal.model.execution.url**
  Default: /index.html?lisaPortal=serverconsole

- **lisa.portal.reporting.context**
  Default: reporting

- **lisa.portal.reporting.console.base.url**
  Default: /index.html?lisaPortal=reporting

- **lisa.portal.defect.capture.url**
  Default: /pathfinder/lisa_pathfinder_agent/defectcapture

- **lisa.portal.save.defect.data.url**
  Default: /pathfinder/lisa_pathfinder_agent/saveDefectData

- **lisa.portal.invoke.base.url**
  Default: /lisa-invoke

- **lisa.portal.invoke.report.url**
  Default: /reports

- **lisa.portal.invoke.server.report.directory**
  Default: lisa.tmpdirlisa.portal.invoke.report.url

- **devtest.portal.base.url.path**
  Default: /devtest/#/main

- **devtest.portal.homepage.url.path**
  Default: {{devtest.portal.base.url.path}}/dashboard

- **lisa.portal.pathfinder.explorer.base.url**
  Default: {{devtest.portal.base.url.path}}/createartifacts

- **lisa.portal.pathfinder.management.base.url**
  Default: {{devtest.portal.base.url.path}}/pathManageAgents

- **lisa.portal.invoke.test.root**
  Default: LISA_HOME

Reporting Graph View Properties

- **rpt.lisa.graph.view.threshhold**
  Default: 100

- **rpt.lisa.graph.view.infomessage**
  Results more than maximum threshold. Modify a filter to fetch the details.

- **rpt.lisa.graph.scatter.view.threshhold**
  Default: 10000

- **rpt.lisa.graph.scatter.view.infomessage**
  Results more than maximum threshold. To fetch the details, modify a filter.

Async Reporting Support

- **lisa.reporting.useAsync**
  If you run load tests and you find that the bottleneck in the test cases is writing the events to the reporting database, consider removing everything except metrics from the report generator. If you still see the reporting engine as the bottleneck, consider enabling this property. The property uses JMS to send the reporting event and background threads in the simulators and coordinator write the events to the database asynchronously. This means that your load test finishes before all the events are written to the database, so the report does not appear for some time (how long depends on your test cases and how many events they generate). The simulator queue typically takes the longest to flush; you get a message at the INFO level in the simulator log showing the percentage complete.
  This feature is considered as "advanced usage" for now and is disabled by default. Feedback at http://ca.com/support is welcome and encouraged.
  Default: false

- **lisa.reporting.step.max.propsused.buffersize**
  Default: 100

- **lisa.reporting.step.max.propsset.buffersize**
  These properties control the collection of statistics for a test step during the execution of a test case. The values specify the maximum number of occurrences that are recorded for the properties that are used and properties that are set. You should not need to change the default values.
  Default: 100

- **lisa.threadDump.generate**
  Default: true

- **lisa.threadDump.interval**
  Default: 30

- **lisa.threadDump.loggerName**
  Enable periodic thread dumps. It is a good idea to leave this property enabled. It efficiently checks to see if the threadDumpLogger is at INFO or below and not do anything if it is set to

WARN or higher. Properties are only read once at startup; the l**ogging.properties** file is checked every 10 seconds for changes. All that you must do is leave this property alone and set the log level of the threadDumpLogger to INFO and wait for at most 30 seconds to get periodic thread dumps of a running DevTest Server. These logs are invaluable when debugging performance issues. See the comments in **logging.properties** for more information.
Default: threadDumpLogger

These properties are used to control the metrics collection in VSE servers.

- **lisa.vse.metrics.collect**
  Main property to turn on the overall metrics collection (true) or off (false).
  Default: true

- **lisa.vse.metrics.txn.counts.level**
  This property controls the level at which transaction counts are recorded. When the named transaction counts are summed, you get transactions for any time period.
  Values: none (or false), service, operation, arguments

  - **Service:** The name of the service names transaction counts

  - **Operation:** The service name plus the request operation names transaction counts

  - **Arguments:** The service name plus the request operation names transaction counts plus a list of request arguments

  Default: service

- **lisa.vse.metrics.sample.interval**
  This property controls how often transaction rate and response times are sampled.
  Default: 5m

- **lisa.vse.metrics.delete.cycle**
  Default: 1h

- **lisa.vse.metrics.delete.age**
  These properties control how often old metric data is scanned for and deleted, and what is considered old.
  Default: 30d

- **lisadb.internal.enabled**
  Indicates whether to start the internal Derby database instance in the registry.
  Default: true

- **lisadb.internal.host**
  The network interface that the internal Derby database uses. The default value 0.0.0.0 indicates that all interfaces are used.
  Default: 0.0.0.0

- **lisadb.internal.port**
  The port number that the internal Derby database listens on.
  Default: 1528

- **lisa.acl.audit.logs.delete.frequency**
  Default: 1d

- **lisa.acl.audit.logs.delete.age**
  These properties control how often old ACL audit log data is scanned for and deleted, and what is considered old.
  Default: 30d

Database Properties

The following components interact with a database: reporting, Agent broker, VSE, and ACL. Typically you would point them all to the same connection pool, but you can define a separate pool for each or mix and match. To define a new pool, set up properties like **lisa.db.pool.myPool.url**. The underlying pool implementation is the open source c3p0 pool, and the various properties are passed down. For available settings, see http://www.mchange.com/projects/c3p0/index.html#configuration_properties.

- **lisadb.reporting.poolName**
  Default: common

- **lisadb.acl.poolName**
  Default: common

- **lisadb.broker.poolName**
  Default: common

- **lisadb.pool.common.driverClass**
  Default: org.apache.derby.jdbc.ClientDriver

- **lisadb.pool.common.url**
  Default: jdbc:derby://localhost:1528/database/lisa.db;create=true

- **lisadb.pool.common.user**
  Default: rpt

- **lisadb.pool.common.password_enc**
  Set the password by removing the trailing _enc from the property name and adding *=MyPlaintextPassword*. The password is automatically encoded at startup.
  Default: 76f271db3661fd50082e68d4b953fbee

The following pool properties keep the number of connections to a minimum when DevTest is idle.

- **lisadb.pool.common.minPoolSize**
  Default: 0

- **lisadb.pool.common.minPoolSize**
  Default: 0

- **lisadb.pool.common.maxPoolSize**
  Default: 10

- **lisadb.pool.common.acquireIncrement**
  Default: 1

- **lisadb.pool.common.maxIdleTime**
  Default: 45

- **lisadb.pool.common.idleConnectionTestPeriod**
  Default: 5

Other common database settings: copy and paste the relevant user, password, and pool size parameters from the common template.

- **lisadb.pool.POOLNAME.driverClass**
  Default: oracle.jdbc.OracleDriver

- **lisadb.pool.POOLNAME.url**
  Default: jdbc:oracle:thin:@HOST:1521:SID

- **lisadb.pool.POOLNAME.driverClass**
  Default: com.ibm.db2.jcc.DB2Driver

- **lisadb.pool.POOLNAME.url**
  Default: jdbc:db2://HOST:50000/DBNAME

- **lisadb.pool.POOLNAME.driverClass**
  Default: com.microsoft.sqlserver.jdbc.SQLServerDriver

- **lisadb.pool.POOLNAME.url**
  Default: jdbc:sqlserver://durry;databaseName=LISA

- **lisadb.pool.POOLNAME.driverClass**
  Default: com.mysql.jdbc.Driver

- **lisadb.pool.POOLNAME.url**
  Default: jdbc:mysql://HOST:3306/DBNAME

- **lisa.jdbc.asset.pool.size**
  When defining a JDBC Connection asset, this property configures the connection pool size.
  Default: 5

- **lisa.jdbc.pool.minPoolSize**
  The minimum number of connections a pool maintains at any given time.**Default**: 0

- **lisa.jdbc.pool.initialPoolSize**
  The number of connections a pool tries to acquire upon startup. This value should be between **minPoolSize** and **maxPoolSize.**
  Default: 0

- **lisa.jdbc.pool.maxPoolSize**
  The maximum number of connections a pool maintains at any given time.
  Default: 10

- **lisa.jdbc.pool.acquireIncrement**
  Determines how many connections at a time c3p0 will tries acquire when the pool is exhausted.
  Default: 1

- **lisa.jdbc.pool.maxIdleTime**
  The number of seconds a connection can remain pooled but unused before being discarded. Zero means idle connections never expire.
  Default: 45

Mainframe Properties

- **lisa.mainframe.bridge.enabled**
  Indicates whether to enable the CICS mainframe bridge.
  Default: false

- **lisa.mainframe.bridge.mode**
  Indicates whether the mainframe bridge runs in client mode or server mode.
  Default: server

- **lisa.mainframe.bridge.port**
  When the mainframe bridge is running in server mode, this port is the well-known port that the mainframe bridge listens on.
  Default: 61617

- **lisa.mainframe.bridge.server.host**
  Default: 127.0.0.1

- **lisa.mainframe.bridge.server.port**
  When the mainframe bridge is running in client mode, these values are the IP address and the well-known port of the LPAR agent.
  Default: 3997

- **lisa.mainframe.bridge.connid**
  Two-character unique ID for each client.
  Default: AA

WebSphere MQ Properties

lisa.mq.ccsid.default=819

You can use this property to override the default Coded Character Set Identifier (CCSID) for all IBM WebSphere MQ messages sent from DevTest. This value can also be overridden for each case in the Publisher Info's Message Properties of the IBM WebSphere MQ step. For more information, see *Using CA Application Test.* For the complete list of CCSIDs, see http://www-01.ibm.com/software/globalization/ccsid/ccsid_registered.html. The default value for United States locales is 819 (ASCII).

Localization Options

lisa.locale.languages=en

You can use this property to indicate the languages to support in the UI. The valid values are **en** and **ja**, indicating English or Japanese.

If you specify both **en** and **ja** here, you can switch the UI between English and Japanese by using the **System**, **Language** option off the **Main** menu.

- **lisa.supported.html.request.encodings=**
  Valid values are ISO-8859-1,UTF-8,Shift_JIS,EUC-JP,Windows-31J.

Selenium Integration Properties

- **selenium.enable.waitfor=true**
  Specifies whether to add an implicit **waitForElementPresent** step for each step in the test case that must locate a web element.

- **Values:**
  - **True:** Adds the implicit step that executes before the real step to ensure that the element exists and that the page has loaded.
  - **False:** Does not add the implicit step. If you set this property to false, scenarios can occur where the script runs successfully in Selenium Builder but fails in DevTest because of delays in page loading. If you have already defined **waitForElementPresent** steps in Selenium Builder, you can set this property to false.

- **selenium.browser.type**
  Defines the type of browser for running a Selenium Integration test case. Use this property in a project configuration file.
  Values:
  - Chrome
  - IE
  - Firefox

  Note: If this property is not specified, the test runs in Firefox by default.

- **selenium.ie.driver.path**
  Defines the full path to the Selenium driver on a local computer that is used to run a Selenium Integration test in a Microsoft Internet Explorer browser. Use this property in a project configuration file.
  - **Example:**

    ```
    C:\lisa-se\IEDriverServer.exe
    ```

- **selenium.chrome.driver.path**
  Defines the full path to the Selenium driver on a local computer that is used to run a Selenium Integration test in a Chrome browser. Use this property in a project configuration file.
  - **Example:**

    ```
    C:\lisa-se\ChromeDriverServer.exe
    ```

- **selenium.remote.url**
  Defines the URL of a remote Selenium Server hub to run Selenium Integration test cases on a remote browser. Use this property in a project configuration file.
  - **Example:**

    ```
    http://your_remote_hostname:4444/wd/hub
    ```

Note: You can define **selenium.chrome.driver.path** and **selenium.ie.driver.path** in the same project configuration file. A configuration file that contains **selenium.chrome.driver.path** or **selenium.ie.driver.path** cannot also contain **selenium.remote.url**.

- **selenium.WebDriver.DesiredCapabilities.filePath**
  Specifies the location of the parameter file that is used to set advanced options in the Selenium web driver.

VSEasy Properties

The port properties control the dynamic assignment of ports when using auto configuration with the HTTP protocol in VSEasy.

If the minimum port is less than 1024, the value is set to 1024.

If the maximum port is greater than 65535, the value is set to 65535.

If the maximum port is less than or equal to the minimum port, both values are reverted to the defaults.

- **lisa.vseasy.http.min.dynamic.port**
  Default: 8000
  This property is applied at record time only. It does not apply when deploying a virtual service.

- **lisa.vseasy.http.max.dynamic.port**
  Default: 65535
  This property is applied at record time only. It does not apply when deploying a virtual service.

- **lisa.vseasy.default.group.tag**
  Specifies the name of the default virtual service group.
  Default: VSEasy

Solr Properties

The Application Insight feature uses Apache Solr for its search functionality.

- **lisa.pathfinder.solr.syncUpHour**
  Specifies when Apache Solr synchronizes with the database. The value represents the hour. For example, the default value indicates 2 o'clock in the morning.
  Default: 2

- **lisa.pathfinder.solr.batchSize**
  Specifies how many rows to retrieve from the database at a time. We recommended that you leave the value blank, as JDBC uses its default value. For most JDBC drivers, the default value is 500.

Application Insight Stateful Baseline Parameterization

The following properties control the use of parameterization in stateful baselines:

- **lisa.pathfinder.stateful.baseline.parameter.string.min.length**
  Defines the minimum length of a value in a Application Insight transaction request/response that is required to consider that key/value for generating a parameter or filter.
  Default: 3

- **lisa.pathfinder.stateful.baseline.parameter.string.exclusion**
  Allows you to exclude certain strings from being eligible for generating a parameter or filter.
  Default: Yes, YES, yes, No, NO, no, true, True, TRUE, false, False, FALSE, __NULL

Application Insight Shelve Frames

- **lisa.pfinvoke3.qshelveframes.trans.max**
  Specifies the maximum number of transactions to process when the user tries to shelve frames by category. If you set the value to -1, all of the transactions are processed.
  Default: 500

CA Agile Central Integration Properties

- **rally.api.key**
  Specifies the CA Agile Central integration API key.

- **rally.api.url=https:/rally1.rallydev.com/slm/webservice/v2.0/**
  The URL to the CA Agile Central API for the instance of CA Agile Central used. Change only if you are not using the cloud-based service.

SAP Properties

- **lisa.vse.rfc.server.state.timeout.ms**
  Property for RFC protocol
  Example: 30000 ms
  Default: 30 Seconds

- **lisa.vse.idoc.server.state.timeout.ms**
  Property for IDoc protocol
  Example: 30000 ms
  Default: 30 Seconds

Community Edition Properties

- **l2w.update.URL**
  This property is not used by DevTest Solutions. It is used by CA Service Virtualization Community Edition to check for available updates to that product.
  Default: http://cainc.to/

Identity and Access Manager Properties

- **devtest.iam.host**
  Address for the IAM host.
  Default: localhost

- **devtest.iam.port**
  Address for the IAM port
  Default: 51111

- **devtest.iam.autoAddUsers**
  By default, if a user does not exist in the DevTest database, it will be added without any roles. Only applicable to DevTest registries.
  Default: true

- **devtest.iam.authenticateOnly**
  The default value will use IAM for authentication and authorization. To use only authentication with IAM, make the value "true." Only applicable to DevTest registries.
  Default: false

VS Recorder

- **lisa.transactions.isStateless**=true
  Specifies if "Treat all transactions as Stateless" is enabled.
  Default: true

Custom Property Files

You can use two other property files for your custom properties:

- local.properties
- site.properties

Site properties can be stored at the test server, which automatically pushes the **site.properties** file to all workstations that connect to it.

Properties files are evaluated in the following precedence: Command line and vmoptions files always take precedence over properties files. Then, **local.properties** take precedence over **site.properties**, which takes precedence over **lisa.properties**.

Note: When DevTest Workstation is started, you are prompted to connect to a registry. After the registry is connected, a site.properties is sent to the workstation. If you switch the registry and change from Registry1 to Registry2, the **site.properties** from Registry2 is sent to the workstation.

To use a custom property file:

1. Go to the LISA_HOME directory.
2. Copy the **_site.properties** or **_local.properties** file and paste it in the same directory.
3. Change the name of the file that you pasted to **site.properties** or **local.properties** (without the leading underscore).

More Information:

- Local Properties File
- Site Properties File
- logging.properties
- Enterprise Dashboard Properties

Local Properties File

The properties in the **local.properties** file take precedence over the **lisa.properties** and **site.properties** files.

- Autoconnection Properties
- License Properties
- VSE Properties
- Enterprise Dashboard Properties
- Interconnectivity Properties
- License Properties if Using an HTTP Proxy Server
- SDK Properties
- SSL Properties
- HTTP Authorization Properties
- Kerberos Authentication Properties
- HTTP Proxy Server Properties
- XML Serialization Settings
- Workstation Management Properties
- IP Spoofing
- Quick Start Recent Items Properties
- lisa-invoke Properties
- Server Host Name Properties
- DevTest to DevTest Communication Encryption Properties
- IBM WebSphere MQ Properties
- JMS Properties
- Miscellaneous Properties
- User Session Lifetime Properties

Autoconnection Properties

- **lisaAutoConnect**
  To load site-wide properties automatically from a DevTest registry, uncomment this property and make it the right URL to your DevTest registry. The **hostname/lisa.TestRegistry** property usually works. Your properties in this file override any properties that are defined at the site level.
  To load site-wide properties from a DevTest registry automatically, uncomment this property and make it the right URL to your DevTest registry. The **hostname/lisa.TestRegistry usually** works. Your properties in this file override any properties that are defined at the site level for non-GUI components.
  DevTest Workstation does not use this property. You can select the registry that you want to connect to with the **Select Registry** dialog or by the **Change Registry** button in the workstation. If you disable the "prompt on startup" check box in the **Select Registry** dialog, then DevTest Workstation connects to the last registry that it was using.
  Format: tcp://somehost/Registry

License Properties

- **laf.server.url**
  Format: https://license.itko.com

- **laf.domain**
  Format: iTKO/LISA/YOURCO

- **laf.username**
  Format: YOURUSERNAME

- **laf.password**
  Format: YOURPASSWORD

VSE Properties

- **lisa.vse.deploy.dir**
  Lets you override the directory where run-time files are managed. Set this property to an absolute directory path. If you do not start the path with a drive specification, it becomes relative to LISA_HOME. If you are specifying a multilevel directory, specify in this format: C:\\Temp\\myVSE.

  > **Note:** In properties files, you MUST double backslashes for them to be recognized.

- **lisa.vse.si.editor.prefer.xml**
  The valid values are the list of VSE SI text editor class names that are found in **typemap.properties** assigned to the **vseTextBodyEditors** name. If you write a custom text editor and make it available through standard SDK methods, that class name can also be a value for this property.

The following properties let VSE use old-style socket I/O rather than new I/O ("NIO"). Be aware of the following restrictions when using old style socket support:

- The ability of VSE to handle being a proxy in front of SSL automatically cannot be supported. The solution is to create a VSM that is in Live System mode and points to the actual virtual service, with SSL turned on.

- The ability of VSE to handle plain text traffic, even when the listen step has been configured to use SSL, cannot be supported. The solution is to provide two virtual services, one with SSL configured and one without SSL configured.

- Using old style socket support does not scale as efficiently as the default socket support VSE uses.

- **lisa.vse.tcp.uses.nio**
  Setting this property to false causes both plain and SSL sockets to use old-style I/O.
  Default: true

- **lisa.vse.plain.tcp.uses.nio**
  This property controls only plain sockets.
  Default: sa.vse.tcp.uses.nio

- **lisa.vse.ssl.tcp.uses.nio**
  This property controls only SSL sockets.
  Default: isa.vse.tcp.uses.nio

- **lisa.vse.execution.mode**

  - EFFICIENT uses the most efficient path through a VS model.

- TRACK records VSE activity at the VS level.
- LIVE routes requests received by a VS model to a live system (somewhat like a pass through mode).
- VALIDATION uses both VSE and the live system to determine a response. Both are recorded as tracking information and feed the model healing process. The live response becomes the response of the virtual service.
- DYNAMIC invokes a script or subprocess for every request that must resolve to one of the other four modes. This approach is useful for tracking requests that only the virtual service model sees.
- LEARNING automatically "heals" or corrects the virtual service to have the new or updated response from the live system.
- STAND_IN first routes a request to the virtual service (the same as Most Efficient mode). However, if the virtual service does not have a response, the request is then automatically routed to the live system.
- FAILOVER first routes a request to the live system (the same as Live System mode). However, if the live system does not have a response, the request is then automatically routed to the virtual service.

Default: EFFICIENT

- **lisa.vse.body.cache.weight**
  Defines the size in bytes of cache values that are allowed to persist in specific caches.
  Default: 1024 * 1024 * 2 (2 MB)

- **lisa.vse.metric.collect**
  When set to false, DevTest stops the collection of periodic samples that calculate Response Time, Forced Delay, and Transactions per Second. The remainder of the metrics data that is tracked in the VSE_METRICS_TXN_COUNTS table is still captured to produce the charts for the following items:

  - Server Charts
  - Total Lifetime Transactions
  - Daily Transaction Counts
  - Server Availability
  - Service Charts
  - Total Transactions per Day.

  When set to false, these items do not show any data:

  - Service Charts
  - Transaction Throughput
  - Transaction Hits and Misses
  - Response Time

- Forced Delay
- Transactions per second.

Default: true

- **lisa.vse.max.hard.errors**
  Lets you configure the number of errors that cause a VSE service to stop.
  To designate the number of errors that cause the service to stop, enter a valid number, 0 or greater. A value of 0 means that no errors are allowed; the service shuts down after the first error.
  To designate an unlimited number of errors, enter any negative number.
  If you enter an invalid value or no value, the default number of 3 errors is used.
  Default: 3

  > **Note:** Depending on the type of error and model, the final error count can be greater than the value set in **lisa.vse.max.hard.errors**. Shutting down the virtual service can take some time. While the shutdown is in process, some types of errors can continue to increase.

- **lisa.tcp.tcprecorder.close.immediately**
  Specifies if sockets are closed immediately at the end (or termination) of a TCP recording. Both local (listen port) and remote sockets are affected.
  If a socket read or write is in progress (if there is a long delay by server or client) and recording is terminated, then this property specifies whether to close or to let the communication complete.
  For long running conversations (for example, large file downloads) this property specifies whether to terminate the conversation or let it run until the end.
  Values: blank, true, and false. If the value is blank, the value defaults to true.
  Default: true

- **lisa.vse.body.cache.size**
  Specifies the amount of memory that is set aside to cache the uncompressed body for VSE request and response objects. Units are in MB.
  Default: 10

- **lisa.vse.body.cache.timeout**
  Specifies the length of time cached uncompressed bodies of VSE requests and responses remain in the cache. Units are in seconds.
  Default: 60

- **lisa.vse.argument.match.allow.whitespace**
  This property tells VSE to allow leading or trailing whitespace, or both, on incoming request arguments when matching to a service image. The default value is *false*, which causes VSE to strip off leading and trailing whitespace on incoming request arguments before matching the request to a service image.
  Default: false

These properties define the IMS Connect message parameters for the protocol to use for recording and playback purposes.

- **lisa.vse.protocol.ims.encoding**
  Default: EBCDIC

- **lisa.vse.protocol.ims.header.length**
  Default: 80

- **lisa.vse.session.timeout.ms**
  If an existing session cannot be found, 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 2 minutes after it has last been "seen" by VSE. You can change this behavior by setting **lisa.vse.session.timeout.ms**. 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.
  Default: 120000

- **lisa.vse.tcp.oldio.read.timeout**
  Controls the period that we wait for a TCP read to be satisfied before timing out and running retry logic. Its default setting of 500 milliseconds is sufficient for "normal" or well-tuned networks where network latency is not an issue. For networks experiencing delays longer than 1/2 a second, you can set this property to allow the code to wait for a longer time before retrying. The retry logic should be sufficient for communications on the TCP socket to be maintained at the loss of some throughput (a lower number of transactions per second). NIO must be turned off for the **lisa.vse.tcp.oldio.read.timeout** property to work. Set **lisa.vse.ssl.tcp.uses.nio**=false.
  Default: 500

- **lisa.vse.conversational.back.navigation.allowed**
  VSE supports unrecorded back navigation during playback in a conversational transaction. Set this property to *true* to enable back navigation for all services. The conversational session then tracks received requests and allows any next request to be one of the previous ones.
  Default: false

- **lisa.vse.tracking.delete.data.age**
  Defines how long to keep session tracking information, in hours.
  Default: 8

- **lisa.vse.protocol.ims.response.includes.llll**
  Indicates the existence of the 4-byte IMS Connect LLLL length field in the request payload. A value of true includes the LLLL field. A value of false excludes it.
  Default: false

- **lisa.vse.ims.connect.llzz.request**
  Indicates the existence of the 4-byte IMS Connect LLZZ length field in the request payload. A value of **true** includes the LLZZ field. A value of **false** excludes it.
  Default: false

- **lisa.vse.ims.connect.llzz.response**
  Instructs the protocol to generate the 4-byte IMS Connect LLZZ length field in the response messages. A value of **true** creates the LLZZ field. A value of **false** does not create it.
  Default: true

- **lisa.vse.http.server.lookup.client.names**
  Controls whether we do reverse DNS lookup on our clients.
  Values: true, false
  Default: false

- **lisa.vse.performance.enabled**
  Specifies whether to enable CA Service Virtualization performance mode.
  Values: true, false
  Default: false

- **lisa.vse.http.response.allowNonStandardResponseCode**
  Allows for non-standard response codes when using the HTTP virtual service.
  Values: true, false
  Default: false

- **lisa.baseline.cache.timeout.seconds**

  After baselines are created, they are maintained in cache for a short period to allow time for our web-based applications to retrieve them. The length of time they remain in cache is managed with this property,

  Default: 60 seconds

- **lisa.baseline.cache.maximum.entries**

  Determines which baselines to evict when the cache size becomes larger than desired. This property does not ensure a maximum size, it only provides a suggested size limit to the cache manager. When this size limit is reached, the cache manager considers the least-recently accessed baseline for eviction.

  Default: 100

- **lisa.baseline.conversationThread.maximumTransactions**

  Baselines of virtual services support the concept of "conversation threads." Every stateful conversation in a service image starts with a single transaction that defines a "start node". This start node has branches to other transactions, those transactions lead to others, and so on, yielding a hierarchy of transactions.

  Each path from a start node to a leaf node constitutes a "conversation thread". Each conversation thread yields one baseline test case. Usually, these conversation threads are not long and contain only a handful of transactions. Inevitably, pathological cases arise, and these pathological cases have the potential of introducing runtime errors such as Out Of Memory exceptions and Stack Overflow exceptions. To mitigate these problems, this property can be set to limit the number of transactions that are allowed in a single conversation thread.

  With the default, the first 100 transactions in a conversation thread are included in the baseline test, and the remaining transactions along that thread are ignored.
  Default: 100

- **com.itko.lisa.vse.sio.loggerName**
  Enables NIO session logic to log higher levels of state information during playback. This logging is at the TRACE log level instead of the INFO level of the current logging. In order to turn on NIO traces, make this property value *NIOSession.* To see the higher-level traces, add **log4j.logger=NIOSession** to the **logging.properties** file.

- **lisa.vse.close_socket_after_response.enabled**
  TCP-specific property to control whether or not the **lisa.vse.close_socket_after_response** metadata property is set when recording responses or handling live responses. The **lisa.vse.close_socket_after_response** metadata property instructs the VSE whether or not to close the socket after sending back a particular response during playback.

When recording live TCP traffic, we set the **lisa.vse.close_socket_after_response** metadata property on a response if (and only if) it the system we are recording traffic from closed its socket after sending it. It also applies to the TCP Live Invocation Step, where we will the **lisa.vse.close_socket_after_response** metadata property on a response if (and only if) it the live system closed its socket after sending it. If the **lisa.vse.close_socket_after_response.enabled** property is set to **false**, then we will never explicitly add the **lisa.vse.close_socket_after_response** metadata property to a response, even if the live system closed its socket.
Default: true

Enterprise Dashboard Properties

The Enterprise Dashboard database is used for the Solutions Usage Audit Report data, registry status and component information, historical event logs, and historical component metrics.

The Enterprise Dashboard database is not able to reside in the same database schema as a registry database. We recommend that the database have at least 50 GB of storage.

Derby is not supported in a distributed environment.

A custom Enterprise Dashboard database must be configured in the **dradis.properties** file.

- **lisa.activation.retry.count**
  Defines the number of times (count) that DevTest retries the initial communication with Enterprise Dashboard.
  Default: 3

The default connection is to the internal Derby database. For information about connecting to external databases, see External Database Configuration.

Interconnectivity Properties

- **lisa.default.keystore**
  Default: {{LISA_HOME}}webreckeys.ks

- **lisa.default.keystore.pass**
  Default: passphrase

DevTest to DevTest communication encryption. Typically the network traffic is not encrypted. To use encryption, we support SSL out of the box. Instead of naming your server endpoints with "tcp," name them with "ssl" - for example: ssl://hostname:2010/Registry. You do not have to set any of the following properties; simply naming the endpoints (and the server name) with SSL is enough. For example, you can start a new simulator:

```
Simulator -name ssl://thishost:2014/Simulator -labName ssl://regHost:2010/Registry
```

We provide a default internal self-signed certificate (in **LISA_HOME\webreckeys.ks**). For a stronger certificate, specify the **lisa.net.keyStore** property and the plaintext password in **lisa.net.keyStore.password**.

The next time DevTest starts, an encrypted string (**lisa.net.keyStore.password_enc**) replaces the plaintext password.

- **lisa.net.keyStore**
  Default: {{LISA_HOME}}lisa.ks

- **lisa.net.keyStore.password**

This is where we keep our identification certificate. If we are a server installation, this is the certificate that clients must add to their list of trusted servers. If we are a client installation and we are doing mutual (client) authentication, this is the certificate that must be added to the truststore on the server side. If we are a client installation that is not using mutual authentication, then you do not need to specify this.

- **lisa.net.trustStore**
  Default: {{LISA_HOME}}lisa.ts

- **lisa.net.trustStore.password_enc**
  Default: 079f6a3d304a978146e547802ed3f3a4

This is where we keep trusted certificates. If we are primarily a client installation, this holds a list of trusted servers. If we are a server installation and we are doing mutual (client) authentication, this is where we put trusted client certificates.

- **lisa.net.clientAuth**
  Indicates whether to use mutual authentication.
  Default: false

- **lisa.net.default.protocol**
  Values: SSL or TCP
  Default: SSL
  Default: ssl

License Properties if Using an HTTP Proxy Server

- **laf.usehttpproxy.server**
  Default: true

- **laf.httpproxy.server**
  Defines the proxy server, in the format *my_proxyserver.com*.

- **laf.httpproxy.port**
  If your proxy server requires credentials, leave blank to use the native NTLM authentication.
  Default: 3128

- **laf.httpproxy.domain**
  Defines the proxy domain, if needed for NTLM.

- **laf.httpproxy.username**
  Optional

- **laf.httpproxy.password**
  Optional

- **laf.email.alert**
  If there is a license error, send an email. Refer to the following example. Replace *mailhost* with the hostname of your mail host. Replace *recipient@mycompany.com* with the target email address. Replace *from@mycompany.com* with the email address from which the email is sent.
  Example: *mailhost,recipient@mycompany.com,from@mycompany.com*

SDK Properties

The DevTest SDK examples require:

- **asserts**
  Format: com.mycompany.lisa.AssertFileStartsWith

- **com.mycompany.lisa.AssertFileStartsWith**
  Format: com.itko.lisa.editor.DefaultAssertController,com.itko.lisa.editor.DefaultAssertEditor

- **filters**
  Format: com.mycompany.lisa.FilterFileFirstLine

- **com.mycompany.lisa.FilterFileFirstLine**
  Format: com.itko.lisa.editor.FilterController,com.itko.lisa.editor.DefaultFilterEditor

- **nodes**
  Format: com.mycompany.lisa.node.FTPTestNode

- **com.mycompany.lisa.node.FTPTestNode**
  Format: com.mycompany.lisa.node.FTPTestNodeController,com.mycompany.lisa.node.FTPTestNodeEditor

SSL Properties

Change the default behavior for validating the SSL certificates.

- **ssl.checkexpiry**
  A value of "true" says to validate the validity dates for the certificate.
  Default: false

- **ssl.checkcrl**
  A value of "true" says to validate the cert revocation list that is specified in the certificate.
  Default: false

Enable a client cert and password for SSL (used by HTTP step; also used by Web Service step if not overridden).

- **ssl.client.cert.path**
  A full path to the keystore.

- **ssl.client.cert.pass**
  The password for the keystore. This password is automatically encrypted when DevTest runs.

- **ssl.client.key.pass**
  An optional password for the key entry if using a JKS keystore and key has a different password from keystore. This password is automatically encrypted when DevTest runs.

Override the client certificate and password for SSL (ssl.client.cert.path and ssl.client.cert.pass) for Web Service step.

- **ws.ssl.client.cert.path**
  A full path to the keystore.

- **ws.ssl.client.cert.pass**
  The password for the keystore. This password is automatically encrypted when DevTest runs.

- **ws.ssl.client.key.pass**
  An optional password for the key entry if using a JKS keystore and key has a different password from keystore. This password is automatically encrypted when DevTest runs.

Use a custom keystore when acting as the SSL server during VSE recording and playback (Listen Step).

- **ssl.server.cert.path**
  A full path to the SSL server keystore file.

- **ssl.server.cert.pass**
  The password for the SSL server keystore. When DevTest runs, this password is automatically encrypted as a new property named **ssl.server.cert.pass.encrypted.**

- **lisa.ssl.reuse.session**

  If you are reusing the same socket (for example, within the same test it reuses sockets) DevTest creates a new session.
  Default: false

HTTP Authorization Properties

These credentials are automatically encrypted when DevTest runs. To reset the values, use the unencrypted property names. To use the native NTLM authorization (Windows only), leave these settings commented out.

- **lisa.http.domain**
  The domain name; use this option for NTLM.

- **lisa.http.user**
  Username

- **lisa.http.pass**
  Password

- **lisa.http.preemptiveAuthenticationType**
  Preemptively send the authorization information rather than waiting for a challenge.
  Values: basic, ntlm, negotiate
  Default: ntlm

- **lisa.http.forceNTLMv1**
  NTLMv2 is used by default but by setting this property to *true* it can be forced to use NTLMv1 when not using native integrated Windows authentication.
  Default: true

- **lisa.vse.http.live.invocation.request.authorization.header**
  Used to replace or add an HTTP Authorization header to outgoing requests to the live service under the HTTP Live Invocation step.
  Format: an Authorization header - <type> <credentials> For example: Basic YWxhZGRpbjpvcGVuc2VzYW1l or Negotiate TlRMTVNTUAABAAAAB4IIogAAAAAAAAAAAAAAAAAAAAAFAs4OAAAADw==Typically this is set dynamically in a Script step or a filter on a Java Execution step, but can also be set as a configuration parameter or as part of local.properties file.

Kerberos Authentication Properties

- **lisa.java.security.auth.login.config**
  The location of the login configuration file.

- **lisa.java.security.krb5.conf**
  The location of the Kerberos configuration file that would be used to override any preset locations.

- **lisa.http.kerberos.principal**
  The name of the principal to be used for the login when using DevTest support for principal and password authentication. This parameter is encrypted when DevTest Workstation is started.

- **lisa.http.kerberos.pass**
  The password to be used for the login when using DevTest support for principal and password authentication. This parameter is encrypted when DevTest Workstation is started.

HTTP Proxy Server Properties

- **lisa.http.webProxy.host**
  The machine name or IP address.

- **lisa.http.webProxy.nonProxyHosts**
  The machine name or IP address to exclude from proxy authentication. To enter multiple values, delimit values with pipes (|). Use * as a wildcard.
  Default: 127.0.0.1

- **lisa.http.webProxy.port**

- **lisa.http.webProxy.ssl.host**
  The machine name or IP address.

- **lisa.http.webProxy.ssl.nonProxyHosts**
  The machine name or IP address to exclude from SSL proxy authentication. To enter multiple values, delimit values with pipes (|). Use * as a wildcard.
  Default: 127.0.0.1

- **lisa.http.webProxy.ssl.port**
  Leave blank to use integrated NTLM authentication.

- **lisa.http.webProxy.host.domain**
  Used for NTLM authentication.

- **lisa.http.webProxy.host.account**

- **lisa.http.webProxy.host.credential**

- **lisa.http.webProxy.nonProxyHosts.excludeSimple**
  Exclude simple host names from proxy use.
  Default: true

- **lisa.http.webProxy.preemptiveAuthenticationType**
  Preemptively send authorization information rather than waiting for a challenge.
  Values: basic, ntlm
  Default: ntlm

- **lisa.http.timeout.connection**
  HTTP timeout (in milliseconds) - To extend the timeout to wait indefinitely, set the values to zero.
  Default: 15000

- **lisa.http.timeout.socket**
  HTTP timeout (in milliseconds). To extend the timeout to wait indefinitely, set the value to zero.
  Default: 180000

XML Serialization Settings

- **lisa.toxml.serialize.mode**=NOREFERENCES

NOREFERENCES means a simple tree is rendered. Circular references are not allowed. XPATHREFERENCES means the XPATH notation is used for references. Circular references can be used. If there is a circular reference, we fall back to XPATHREFERENCES. However, any serialization before version 3.6 that is reread fails and could require you to set a default of XPATHREFERENCES.

Workstation Management Properties

- **lisa.ui.admin.tr.control**=no

- **lisa.ws.jms.SoapAction.quoted**=false

SOAP over JMS with TIBCO BusinessWorks/Active Matrix: BW/AM requires that the SOAPAction header is quoted.

When DevTest serializes a date, time, or dateTime, it uses the following formats by default. To change the default formats/time zone, use these properties. You can also dynamically set them during the test case. You can set the format to one that does not comply with the XML schema specification. However, doing so is not recommended (except for a negative test case).

- **lisa.ws.ser.dateFormat**=yyyy-MM-dd

- **lisa.ws.ser.timeFormat**=HH:mm:ss.SSS'Z'

- **lisa.ws.ser.timeFormat.timeZone**=GMT

- **lisa.ws.ser.dateTimeFormat**=yyyy-MM-dd'T'HH:mm:ss.SSS'Z'

- **lisa.ws.ser.dateTimeFormat.timeZone**=GMT

- **ws.raw.format**=true

To format the SOAP response for RAW Soap Steps, set this property to **true. T**his formatting can also be done dynamically at run time.

- **lisa.ws.endpoint.fullautoprop**=false

The entire WS Endpoint URL is automatically converted into a single DevTest property. To convert the URL to use the WSSERVER and the WSPORT properties, set this property to **false**.

- **stats.unix.xml.folder**={ {LISA_HOME } }/umetrics

IP Spoofing

- **lisa.ipspoofing.interfaces**=2-34-56-78-90-AB, eth0, Realtek PCIe GBE Family Controller

Interfaces: a comma-separated list of interfaces that is used for the IP spoofing. These interfaces can be named using the MAC address (JDK 1.6+), interface name, or interface display name.

- **lisa.ui.useNativeFileDialog**=true

Force the use of a native file dialog.

Quick Start Recent Items Properties

- **lisa.prefill.recent**
  Specify the maximum number of recent items to show in the Prefill combo boxes (no less than 10).
  Default: 10

- **lisa.quickstart.recent**
  Specify the maximum number of recent items to show in the **Open Recent Quick Start** tab (no less than 5).
  Default: 5

lisa-invoke Properties

- **lisa.portal.invoke.base.url**
  Default: /lisa-invoke

- **lisa.portal.invoke.report.url**
  Default: /reports

- **lisa.portal.invoke.server.report.directory**
  Default: lisa.tmpdirlisa.portal.invoke.report.url

- **lisa.portal.invoke.test.root**
  Default: d:/lisatests/

Server Host Name Properties

- **lisa.net.externalIP**
  This value must be the external IP address or DNS name for the registry to be accessible remotely. This parameter is referenced in the relevant connection properties. If the value is left at "localhost," it is automatically overridden in the external IP address of the registry server when sent to connecting applications.
  Default: localhost

DevTest to DevTest Communication Encryption Properties

Typically the network traffic is not encrypted. SSL encryption is supported out of the box. Instead of naming your server endpoints with "tcp," name them with "ssl": for example, **ssl://hostname:2010/Registry**. You do not have to set any of the following properties: naming the endpoints (and the server name) with ssl is enough. For example, to start a new simulator: **Simulator -name ssl://thishost:2014/Simulator -labName ssl://regHost:2010/Registry**.

- **LISA_HOME\webreckeys.ks** is a default internal self-signed certificate. For a stronger certificate, specify the **lisa.net.keyStore** property and the plaintext password in **lisa.net.keyStore.password**. The next time DevTest starts, an encrypted string **lisa.net.keyStore.password_enc** replaces the plain text password.

- **lisa.default.keystore**={{LISA_HOME}}webreckeys.ks

- **lisa.default.keystore.pass**=passphrase

Where the identification certificate is kept. For a server installation, this value is the certificate to add to the list of trusted servers. For a client installation doing mutual (client) authentication, this value is the certificate to add to the truststore on the server side. For a client installation that is not using mutual authentication, you do not need to specify this parameter.

- **lisa.net.keyStore**={{LISA_HOME}}lisa.ks
- **lisa.net.keyStore.password**=PlainTextPasswordWilBeConvertedToEncrypted

Where trusted certificates are kept. For a primarily client installation, this field holds a list of trusted servers. For a server installation doing mutual (client) authentication, this field is where to put trusted client certificates.

- **lisa.net.trustStore**={{LISA_HOME}}lisa.ts
- **lisa.net.trustStore.password_enc**=079f6a3d304a978146e547802ed3f3a4

Whether or not to use mutual authentication.

- **lisa.net.clientAuth**=false

Defines the default protocol for ActiveMQ connections. The default is **tcp**.

- **lisa.net.default.protocol**=tcp

IBM WebSphere MQ Properties

The string-value is supplied by a data set such as the Unique Code Generator.

- **lisa.mq.correlation.id**
  A run-time value that sets the correlation ID for an outgoing MQ message.
  Default: string-value
- **lisa.mq.correlation.id.bytes**
  A run-time value that sets the correlation ID as a byte[] for nonprintable values.
  Default: byte[]
- **lisa.mq.message.id**
  A run-time value that sets the message ID for an outgoing MQ message.
  Default: string-value
- **lisa.mq.message.id.bytes**
  A run-time value that sets the message ID as a byte[] for nonprintable values.
  Default: byte[]

JMS Properties

The string-value is often supplied by a data set such as the Unique Code Generator.

- **lisa.jms.correlation.id**
  A run-time value that sets the JMSCorrelationID for an outgoing JMS message.
  Value: string-value
- **lisa.jms.ttl.milliseconds**
  A run-time value that sets the time-to-live for an outgoing JMS message.
  Value: num-value

- **jms.always.close.jndi**
  A value of true always uses the JMS step to close the JNDI context at the end of its execution.
  Values: true, false

Miscellaneous Properties

- **lisa.coord.failure.list.size**
  If too many errors are reported in a test, the coordinator uses up all its available heap space. When the heap space is used up, DevTest Server must be restarted. This situation is bad if multiple tests are running, and one test gets staged and throws nothing but errors. This property limits the size of the coordinator failure list.
  Default: 256

- **testexec.lite.longMsgLen**
  To view the full text of long responses, set this property to the length of your longest response. Run your test, capture what you need, verify that the correct response is indeed being sent back, and then comment out the added line in the local.properties file to revert to the default length.
  Default: 1024

- **java.rmi.server.hostname**
  Forces the communication through a specific NIC on the computer.
  Value: IP address of selected NIC

- **lisa.xml.xpath.computeXPath.alwaysUseLocalName**
  This property configures whether the XPath local-name() function is always used during XPath generation. The default value is false, meaning that the local-name() is only used when necessary. To generate an XPath that works regardless of the namespace of an XML node, set the value of this property to true.
  Default: false

- **lisa.commtrans.ctstats**
  To capture the information that is necessary to populate the Cumulative HTTP Traffic Summary report, enter this property into one of the custom property files.
  Default: true

- **gui.viewxml.maxResponseSize**
  The property to designate the size at which the view of an XML response in an editor or in the ITR is plain and no DOM view is provided.
  Default: 5 MB

- **rpt.cleaner.initDelayMin**
  Default: 10

- **rpt.cleaner.pulseMin**
  These properties were added to make the report cleanup thread run more often. The first is the initial delay, in minutes, before the cleaner starts. The second is the time between runs, in minutes.
  Default: 60

- **lisa.LoadHistoryWriteSupport.max.errors**
  Set to limit the number of errors that are reported to prevent the report generator from backing up. The counter for errors increases based on four events; Abend, Abort, Warning, and Error. If the number of warnings and errors or errors (if warnings are excluded) is greater than maxErrors, then DevTest attempts to stop the test through the coordinator.
  Default: 100

- **lisa.LoadHistoryWriteSupport.includeWarnings**
  You can exclude warnings from incrementing the counter by setting lisa.LoadHistoryWriteSupport.includeWarnings=false.
  Default: true

- **lisa.metric.initialization.timeout**
  Initialization of the Metric Collector occurs during the staging of the test or suite and the test or suite does not start until the collectors are initialized. The default value of 90000 means that initialization times out after 90 seconds (for each collector).
  Default: 90000

- **lisa.graphical.xml.diff.report.numberofextralines**
  When the graphical XML diff displays an error, this property controls how many lines before and after the differing lines are displayed. The default is two lines before and two lines after.
  Values: 0, 1, and 2.
  Default: 2

- **lisa.gui.dashboard.sleep.ms**
  There is a small sleep() right before the dashboard refreshes at end-of-test. The sleep() allows a timeslice for the metric collectors to wrap up their work. Adjust the timeslice with this property.
  Format: milliseconds

- **lisa.scripting.default.language**
  Designates the default scripting language to use.
  Values:

- applescript (for OS X)
  - beanshell
  - freemarker
  - groovy
  - javascript
  - velocity

  Default: beanshell

- **lisa.serverSocket.bind.backlog**
  Specifies the backlog argument while binding Server Socket. Backlog is the requested maximum number of pending connections on the server socket. Its exact semantics are implementation-specific. In particular, an implementation may impose a maximum length or may choose to ignore the parameter altogether.
  Values: numeric, greater than 0.
  Default: 100

- **qc.pulseInterval**
  Specifies the interval, in seconds, that HP ALM Quality Center tries to ping the registry,
  Default: 300

- **lisa.server.https.cipher.suites**
  Lets you restrict the enabled cipher suites for DevTest.

> **Note:** Some of the cipher suites that Java ships with enabled have known vulnerabilities, and may appear on a security scan against running virtual services that are using SSL. You can use this property to restrict DevTest to only the ciphers that you want to allow.

The rules for this property are:

- Configure one or more ciphers by setting this property to a comma-separated list of cipher names.
- If this property is not included in the properties file or if the property is included but with an empty value, then the default JVM-supported ciphers are enabled.
- If none of the cipher names that are specified in this property are supported by the JVM, then the default JVM-supported ciphers are enabled.
- If at least one cipher name specified in this property is supported by the JVM, then those ciphers alone are enabled.

- **FileNode.cache.max.size.mb**
  For the Read a File (Disk, URL or Classpath) step, specifies when the cache file should be flushed, by size.
  Default: 20 MB

- **FileNode.cache.duration.mins**
  For the Read a File (Disk, URL or Classpath) step, specifies when the cache file should be flushed, by time.
  Default: 30 minutes

- **lisa.portal.url.prefix**
  Prefix for the URL of DevTest Portal.
  Default: http://

- **devtest.hostname**
  The hostname for DevTest Portal.
  Default: the registry server

- **devtest.itr.size.pref**
  The default width of the ITR, in pixels.
  Default: 800

- **lisa.cvs.monitor.startup.delay**
  Used to delay the CVS start, which gives additional time for a coordinator to start and prevents CVS jobs failing when a coordinator is still unavailable. Specified in seconds.
  Default: 0

- **lisa.builderSession.cleanup.task.interval**
  Used to specify the intermittent time interval in which the process will run to clean up the builder sessions on the disk, in hours.
  Default: 1

- **list.builderSession.cleanup.interval**
  Used to specify the time interval after which every builder session needs to be removed or cleaned up from the file system, in hours.
  Default: 24

- **lisa.ant.junit.password**Used to specify the password to use when running JUnit tests.
- **lisa.ant.junit.user**Used to specify the userid to use when running JUnit tests.

User Session Lifetime Properties

All the user session lifetimes in different DevTest modules are configurable with properties.

The ACL session-related properties and their default values are:

- **registry.max.user.lifetime.seconds**

The **registry.max.user.lifetime.seconds** property is the main session lifetime property that determines how long a user session is valid after the last activity done by a user in a session.

Specifies how many seconds the SSO security token is kept in memory for Portal to bypass authentication for the DevTest Portal.

When you click a button to access a UI, DevTest bypasses ACL authentication by using an SSO security token. That SSO security token is stored in memory for reuse. However, if Portal is idle for 20 minutes (or the number of seconds specified for this property) that SSO security token becomes expired. It then requires reauthentication. If you start Portal, you can click these buttons and can go directly to the UI without entering a userid and password. But after Portal is idle (no activity) for 20 minutes, if you click these buttons, Portal will prompt for a userid and password to proceed so that a new security token is kept in memory again.

Default: 1200

The following modules have a user session lifetime, but they always communicate with the registry to decide if the session is still alive.

- **vse.max.user.lifetime.seconds**=1200
- **console.max.user.lifetime.seconds**=1200
- **coordinator.max.user.lifetime.seconds**=1200
- **simulator.max.user.lifetime.seconds**=30

The console refresh interval can be configured with the following property.

- **lisa.portal.server.console.polling.interval.seconds**=5

The interval at which the DevTest Console checks whether the current session is valid and is not expired.

Site Properties File

The properties in the **site.properties** file are sent from the registry to any DevTest application or service that connects to it. These properties take precedence over any properties that are defined locally in **lisa.properties**. However, these properties do not take precedence over the properties that are defined in **local.properties** or defined on the command line using -D command-line option.

- DevTest Enterprise Dashboard Properties
- Database Properties
- Oracle Properties

- MS SQL Server Properties
- DB2 Properties
- MySQL Properties
- Derby Properties
- DCM Settings

DevTest Enterprise Dashboard Properties

- **devtest.enterprisedashboard.host**=somehost

- **devtest.enterprisedashboard.port**=1506

> **Important!** The Enterprise Dashboard database is not able to reside in the same database schema as a registry database. Because the site.properties is handled solely by a registry process, a custom Enterprise Dashboard database needs to be configured in the dradis.properties file. For more details on configuration of the Enterprise Dashboard database, look at the _dradis.properties template in the home directory of your DevTest installation.

- **devtest.enterprisedashboard.https.enabled**=false
  Specifies that Enterprise Dashboard runs with schema HTTP or HTTP/S.

Database Properties

These are the default properties that are used by DevTest to connect to the registry database.

Derby is not supported in a distributed environment. You must use an enterprise database in a distributed environment.

The following components interact with a database: reporting, VSE, ACL, and the broker. By default, these components use the common connection pool. However, you can define a separate pool for each or mix and match. To define a new connection pool, add properties such as **lisadb.pool.newpool.url**. The underlying pool implementation is the open source c3p0 pool. DevTest passes the various properties down. For information about the c3p0 settings that you can select from, see http://www.mchange.com/projects/c3p0/index.html#configuration_properties.

- **lisadb.reporting.poolName**
  Default: common

- **lisadb.acl.poolName**
  Default: common

- **lisadb.broker.poolName**
  Default: common

- **lisadb.pool.common.driverClass**
  Default: org.apache.derby.jdbc.ClientDriver

- **lisadb.pool.common.url**
  Default: jdbc:derby://localhost:1528/database/lisa.db;create=true

- **lisadb.pool.common.user**
  Default: rpt

- **lisadb.pool.common.password_enc**
  Default: 76f271db3661fd50082e68d4b953fbee

The following pool properties keep the number of connections to a minimum when DevTest is idle.

- **lisadb.pool.common.minPoolSize**
  Default: 0

- **lisadb.pool.common.initialPoolSize**
  Default: 0

- **lisadb.pool.common.maxPoolSize**
  Default: 10

- **lisadb.pool.common.acquireIncrement**
  Default: 1

- **lisadb.pool.common.maxIdleTime**
  Default: 45

- **lisadb.pool.common.idleConnectionTestPeriod**
  Default: 5

Should the internal Derby database instance in the registry be started?

- **lisadb.internal.enabled**
  Controls whether the registry starts an internal Derby database.
  Default: true

For more information about configuring the registry database, see the following video.

Oracle Properties

- **lisadb.pool.common.driverClass**

Default: oracle.jdbc.driver.OracleDriver

- **lisadb.pool.common.url**

Default: jdbc:oracle:thin:@[HOST]:1521:[SID]

- **lisadb.pool.common.user**

Default: none

- **lisadb.pool.common.password**

Default: none

MS SQL Server Properties

- **lisadb.pool.common.driverClass**

Default: com.microsoft.sqlserver.jdbc.SQLServerDriver

- **lisadb.pool.common.url**

Default: jdbc:sqlserver://[SERVER]:[PORT];databaseName=[DATABASENAME]

- **lisadb.pool.common.user**

Default: none

- **lisadb.pool.common.password**

Default: none

DB2 Properties

- **lisadb.pool.common.driverClass**

Default: com.ibm.db2.jcc.DB2Driver

- **lisadb.pool.common.url**

Default: jdbc:db2://[HOSTNAME]:[PORT]/[DATABASENAME]

- **lisab.pool.common.user**

Default: none

- **lisadb.pool.common.password**

Default: none

MySQL Properties

- **lisadb.pool.common.driverClass**

Default: com.mysql.jdbc.Driver

- **lisadb.pool.common.url**

Default: jdbc:mysql://[DBHOST]:[DBPORT]/[DBNAME]

- **lisadb.pool.common.user**

Default: none

- **lisadb.pool.common.password**

Default: none

Derby Properties

- **lisadb.pool.common.driverClass**

Default: org.apache.derby.jdbc.ClientDriver

- **lisadb.pool.common.url**

Default: jdbc:derby://[SERVER]:[PORT]/[DATABASE];create=true

- **lisadb.pool.common.user**

Default: none

- **lisadb.pool.common.password**

Default: none

DCM Settings

- **lisa.dcm.labstartup.min**=6
- **lisa.dcm.lisastartup.min**=4
- **lisa.dcm.lisashutdown.min**=2
- **lisa.dcm.lab.cache.sec**=<180 default>
- **lisa.dcm.lab.factories**=com.itko.lisa.cloud.serviceMesh.ServiceMeshCloudSupport;com.itko.lisa.cloud.vCloud.vCloudDirectorCloudSupport;;
- **lisa.dcm.SERVICEMESH.baseUri**=<url>
- **lisa.dcm.SERVICEMESH.userId**=<userId>
- **lisa.dcm.SERVICEMESH.password**=<password>
- **lisa.dcm.vCLOUD.baseUri**=<https://<fqdn>/api/versions>
- **lisa.dcm.vCLOUD.userId**=<userId>
- **lisa.dcm.vCLOUD.password**=<password>
- **lisa.net.timeout.ms**=60000

logging.properties

The **logging.properties** file lets you set properties that control logging levels within DevTest and third-party software.

- **log4j.rootCategory**
  Default: INFO, A1

To provide centralized logging for cloud runs, add the **registry** appender and uncomment the registry appender properties in the logging.properties file. For example:

```
log4j.rootCategory=INFO,A1,registry
```

The following lines adjust the log levels of third-party libraries that are used by DevTest. Specifying log levels means that they do not clutter the logs with messages unrelated to DevTest.

- **log4j.logger.com.teamdev**
  Default: WARN

- **log4j.logger.EventLogger**
  Default: WARN

- **log4j.logger.org.apache**
  Default: ERROR

- **log4j.logger.com.smardec**
  Default: ERROR

- **log4j.logger.org.apache.http**
  Default: ERROR

- **log4j.logger.org.apache.http.header**
  Default: ERROR

- **log4j.logger.org.apache.http.wire**
  Default: ERROR

- **log4j.logger.com.mchange.v2**
  Default: ERROR

- **log4j.logger.org.hibernate**
  Default: WARN

- **log4j.logger.org.jfree**
  Default: ERROR

- **log4j.logger.com.jniwrapper**
  Default: ERROR

- **log4j.logger.sun.rmi**
  Default: INFO

- **log4j.logger.com.itko.util.ThreadDumper**
  Default: INFO

- **log4j.logger.profiler**
  Set this property to *INFO* if you want your profile events to be logged:
  Default: OFF

- **log4j.appender.A1**
  Default: com.itko.util.log4j.TimedRollingFileAppender

- **log4j.appender.A1.File**
  Default: ${lisa.tmpdir}/${LISA_LOG}

- **log4j.appender.A1.MaxFileSize**
  Default: 10MB

- **log4j.appender.A1.MaxBackupIndex**
  Default: 5

- **log4j.appender.A1.layout**
  Default: org.apache.log4j.EnhancedPatternLayout

- **log4j.appender.A1.layout.ConversionPattern**
  Default: %d{ISO8601}{UTC}Z (%d{HH:mm}) [%t] %-5p %-30c - %m%n

- **log4j.logger.VSE**
  Keep a separate log for VSE transaction match/no-match events, Having a separate log makes debugging much easier.
  Change INFO to WARN for production systems. The logging can slow down systems with high transaction rates. Do not simply comment out the following line. Explicitly set the log level to OFF or WARN instead of INFO.
  Default: INFO, VSEAPP

- **log4j.additivity.VSE**
  To add VSE logging to other log destinations, comment out this line.
  Default: false

- **log4j.appender.VSEAPP**
  Default: com.itko.util.log4j.TimedRollingFileAppender

- **log4j.appender.VSEAPP.File**
  Default: ${lisa.tmpdir}/vse_matches.log

- **log4j.appender.VSEAPP.MaxFileSize**
  Default: 10MB

- **log4j.appender.VSEAPP.MaxBackupIndex**
  Default: 20

- **log4j.appender.VSEAPP.layout**
  Default: org.apache.log4j.EnhancedPatternLayout

- **log4j.appender.VSEAPP.layout.ConversionPattern**
  Default: %d{ISO8601}{UTC}Z (%d{HH:mm})[%t] %-5p - %m%n
  Keep a separate log for advisory events. This logger warns of potential configuration issues such as potential memory leaks. The log is deliberately kept separate from the application log to minimize noise.

- **log4j.logger.ADVICE**
  Default: INFO, ADVICE_APP

- **log4j.additivity.ADVICE**
  Default: false

- **log4j.appender.ADVICE_APP**
  Default: org.apache.log4j.RollingFileAppender

- **log4j.appender.ADVICE_APP.File**
  Default: ${lisa.tmpdir}/advice.log

- **log4j.appender.ADVICE_APP.MaxFileSize**
  Default: 10MB

- **log4j.appender.ADVICE_APP.MaxBackupIndex**
  Default: 20

- **log4j.appender.ADVICE_APP.layout**
  Default: org.apache.log4j.EnhancedPatternLayout

- **log4j.appender.ADVICE_APP.layout.ConversionPattern**
  Default: %d{ISO8601}{UTC}Z (%d{HH:mm}) %-5p - %m%n

- **log4j.logger=NIOSession**
  Enables NIO session logic to log higher levels of state information during playback. This logging is at the TRACE log level instead of the INFO level of the current logging. In order to turn on NIO traces, make **com.itko.lisa.vse.sio.loggerName**=**NIOSession** in local.properties.

If enabled, periodic thread dumps are sent here. DevTest writes logs at the INFO level. Therefore, to get the thread dumps, change WARN in the next line to INFO, even with DevTest servers or DevTest Workstation running. This action results in a thread dump in the named file in 30 seconds. For more information, search for "threadDump" in **lisa.properties**. This action also simplifies getting thread dumps to debug performance issues. Change WARN in the next line to INFO, wait for 1 minute or 2 minutes, then revert the setting to WARN.

You can also generate a point-in-time thread dump with the LISA_HOME/bin/ServiceManager application. For example, use standard Java tools such as jstack, or issue the following command:

```
ServiceManager -threadDump tcp://hostname:2014/Simulator
```

- **log4j.logger.threadDumpLogger**
  Default: WARN, THREAD_DUMPS

- **log4j.additivity.threadDumpLogger**
  Default: false

- **log4j.appender.THREAD_DUMPS**
  Default: org.apache.log4j.RollingFileAppender

- **log4j.appender.THREAD_DUMPS.File**
  Default: ${lisa.tmpdir}/threadDumps/TD_${LISA_LOG}

- **log4j.appender.THREAD_DUMPS.MaxFileSize**
  Default: 10MB

- **log4j.appender.THREAD_DUMPS.MaxBackupIndex**
  Default: 20

- **log4j.appender.THREAD_DUMPS.layout**
  Default: org.apache.log4j.EnhancedPatternLayout

- **log4j.appender.THREAD_DUMPS.layout.ConversionPattern**
  Default: %d{ISO8601}{UTC}Z (%d{HH:mm}) [%t] %-5p - %m%n

- **log4j.appender.registry**
  Mirrors DevTest logging to the (remote) registry.
  Default: com.itko.lisa.net.LoggingToRegistryAppender

- **log4j.appender.registry.layout**
  Default: org.apache.log4j.EnhancedPatternLayout

- **log4j.appender.registry.layout.ConversionPattern**
  Default: %d{ISO8601}{UTC}Z [%t] %-5p - %m%n

The following properties display the requests and responses for HTTP-based traffic in the vse.log. This information is useful for debugging a virtual service that returns a "no match found" response. Such a result can indicate that the VS did not receive the expected request.

These log messages reveal the same requests and response that TCPMON shows. The requests and responses display in the vse.log, without having to inject TCPMON between the virtual service and its client application. These options only log HTTP requests and responses. These options can help to debug virtual services, especially when a virtual service responds with an unexpected message and you want to match a raw request with the raw response.

- **log4j.logger.com.itko.lisa.vse.http.Transaction**
  Prints the requests that travel through this class in the vse.log. These log messages begin with "Raw playback response."
  Values: TRACE, null
  DEFAULT: null

- **log4j.logger.com.itko.lisa.vse.stateful.protocol.http.Coordinator**
  Prints the requests that travel through this class in the vse.log. These log messages begin with "Raw request start."
  Values: TRACE, null
  DEFAULT: null

- **log4j.logger.com.itko.lisa.vse.stateful.protocol.http.HttpListenStep**
  Prints the requests that travel through this class in the vse.log. These log messages begin with "Raw request start."
  Values: TRACE, null
  DEFAULT: null

The following lines adjust the log levels of third-party libraries that are used by DevTest so that they do not clutter the logs with messages unrelated to DevTest. If not specified, the default value is the value specified in the **log4j.rootCategory** property.

- **log4j.logger.com.teamdev**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.EventLogger**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.org.apache**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.com.smardec**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.org.apache.http**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.org.apache.http.header**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.org.apache.http.wire**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.com.mchange.v2**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.org.hibernate**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.org.jfree**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.com.jniwrapper**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.sun.rmi**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.com.itko.util.ThreadDumper**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.org.eclipse**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.org.codehaus**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.org.grails**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.grails**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.org.springframework**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.jndi**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.org.flywaydb**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.com.jayway**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.net.sf**
  Values: INFO, WARN, ERROR, FATAL

- **log4j.logger.org.quartz**
  Values: INFO, WARN, ERROR, FATAL

Enterprise Dashboard Properties (dradis.properties)

The **dradis.properties** file contains default values for properties that are used by Enterprise Dashboard.

- **dradis.webserver.port**=1506
  Our embedded web server

- **dradis.webserver.host**=0.0.0.0
  The host of our embedded web server: '0.0.0.0' binds to all local addresses.

- **dradis.db.internal.enabled**=false
  Start the internal Derby database instance in the Enterprise Dashboard?

- **dradis.db.internal.host**=0.0.0.0
  Internal Derby DB network interface to use (0.0.0.0 = all network interfaces)

- **dradis.db.internal.port**=1530
  Internal Derby DB port number to use
- **dradis.db.internal.start.retries**=3
- **dradis.db.internal.start.delay**=5000

Alternative Database Connection String Templates

Oracle Template

- **dradis.db.driverClass**=oracle.jdbc.driver.OracleDriver
- **dradis.db.url**=jdbc:oracle:thin:@[HOST]:1521:[SID]
- **dradis.db.user**=[USER]
- **dradis.db.password**=[PASSWORD]

MS SQL Server Template

- **dradis.db.driverClass**=com.microsoft.sqlserver.jdbc.SQLServerDriver
- **dradis.db.url**=jdbc:sqlserver://[SERVER]:[PORT];databaseName=[DATABASENAME]
- **dradis.db.user**=[USER]
- **dradis.db.password**=[PASSWORD]

DB2 Template

- **dradis.db.driverClass**=com.ibm.db2.jcc.DB2Driver
- **dradis.db.url**=jdbc:db2://[HOSTNAME]:[PORT]/[DATABASENAME]
- **dradis.db.user**=[USER]
- **dradis.db.password**=[PASSWORD]

MySQL Template

- **dradis.db.driverClass**=com.mysql.jdbc.Driver
- **dradis.db.url**=jdbc:mysql://[DBHOST]:[DBPORT]/[DBNAME]
- **dradis.db.user**=[USER]
- **dradis.db.password**=[PASSWORD]

Derby Template

- **dradis.db.driverClass**=org.apache.derby.jdbc.ClientDriver
- **dradis.db.url**=jdbc:derby://[SERVER]:[PORT]/[DATABASE];create=true
- **dradis.db.user**=[USER]

- **dradis.db.password**=[PASSWORD]

Enable HTTP/S and Set up the Web Server SSL Keystore

- **dradis.webserver.https.enabled**=true
- **dradis.webserver.ssl.keystore.location**={{DRADIS_HOME}}webserver.ks
- **dradis.webserver.ssl.keystore.password**=yourpassword
- **dradis.webserver.ssl.keymanager.password**=yourpassword

For more information about configuring Enterprise Dashboard, see the following video.

Portal Properties File (phoenix.properties)

Phoenix Properties

The **phoenix.properties** file contains default values for properties that are used by Portal.

- **phoenix.transactions.isStateless**=true
  Specifies if "Treat all transactions as Stateless" is enabled.
  Default: true
- **phoenix.monitoring.autoRefresh.enabled**=false
  Specifies if auto refresh is enabled by default in VSE monitoring view.
  Default: false
- **phoenix.monitoring.autoRefresh.interval**=15
  Specifies the default value (in seconds) the refresh interval time in VSE monitoring view.
  Default: 15
- **phoenix.monitoring.pagesize**=10
  Specifies the default number of rows displayed in the VS table in VSE monitoring view.
  Values: 5, 10, 15, 20, 25.
  Default: 10

Registry Configuration

The registry properties allow for configuration of the endpoint with the remote registry. Portal connects to this registry and the location is defined using the following properties:

- **registry.host**=localhost
- **registry.port**=2009
- **registry.portal.port**=1505
- **registry.https.enabled**=true

Test Step Descriptions

A *test step* is an element in the test case workflow that represents a single test action to be performed.

This section describes the following test steps:

- Standard Options in Test Steps
- Web-Web Services Steps
- Java-J2EE Steps
- Other Transaction Steps
- Utilities Steps
- External-Subprocess Steps
- JMS Messaging Steps
- Apache Kafka Steps
- BEA Steps
- Sun JCAPS Steps
- Oracle Steps
- TIBCO Steps
- Sonic Steps
- webMethods Steps
- IBM Steps
- RabbitMQ Steps
- SAP Steps
- Selenium Integration Steps
- Virtual Service Environment Steps
- CAI Steps
- Custom Extension Steps

Standard Options in Test Steps

The following options let you control the flow of a test case:

Abort the Test

Follow these steps:

1. Right-click the test step after which you want to abort the test case.
2. Select **For Next Step, Abort the Test**.
   The Abort the Test step quits the test case and marks the step as having aborted.

End the Test

Follow these steps:

1. Right-click the test step after which you want to end the test case.
2. Select **For Next Step, End the Test**.
   The step completes the test and marks the test as having ended successfully.

Fail the Test

Follow these steps:

1. Right-click the test step after which you want to fail the test case.

2. Select **For Next Step, Fail the Test**.
   The Fail the Test step fails the test case and marks the test as having failed.

Web-Web Services Steps

The following steps are available in the **Web/Web Services** category:

- HTTP-HTML Request Step
- REST Step
- Web Service Execution (XML) Step
- WSDL Validation
- Base64 Encoder
- Multipart MIME Step
- SAML Assertion Query

HTTP-HTML Request Step

The HTTP/HTML Request step is used while testing a traditional web application to send and receive HTTP(S) requests. Requests can include GET parameters and POST parameters and optionally, embedded images as a response. You can also record the HTTP steps using the Website Proxy Recorder.

The HTTP/HTML Request step has a default name using this convention: *HTTP(s) ("GET" or "POST") (leaf of URL).* An example is **HTTP GET rejectCard.jsp**. You can change step names at any time.

You can manually execute the HTTP/HTML step at design time. When you select **Actions**, **Replay through here**, DevTest saves the step responses so the step editors can display the response values.

When you add this step to a test case, the step editor includes the following tabs:

- URL Transaction Info Tab
- HTTP Headers Tab
- Response Tab
- SSL Tab

URL Transaction Info Tab

Specify the information that is used to construct the URL.

You can set up the URL transaction information with either of the following options:

- Specify URL in parts
- Use property

Specify URL in Parts

Select this option to specify the URL in its essential pieces.

- **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 property SERVER or enter hostname or IP address of your application server. The host name 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) If necessary, use the PORT property or the port on the web server that is used to access the web server. For example, the port that is 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**.

- **User**
  Enter if a user ID is required for the application server.

- **Password**
  Enter if a password is required for the application server.

- **Encoding**

  The list of available values is controlled by the **lisa.supported.html.request.encodings** property. You can change the comma-separated list to include the encodings to support. The underlying JVM must also support all encodings in this list. If a web page uses an encoding that is not supported in the list, the drop-down entry is blank. If you save in that situation, DevTest replaces the encoding with the DevTest default (file.encoding key in lisa.properties). Also, if you do not select an encoding when creating a HTTP/HTML Request step, the DevTest default encoding is assumed.

- **URL Parameters**
  GET (or URL) Request Parameters: These request parameters are passed as part of the URL. They are exposed to the user in the address bar of the web browser.

- **POST Parameters**
  POST Request Parameters. These request parameters are passed as part of the body of the page request. 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 as they are sent. The MIME type is **application/x-www-form-urlencoded**.

- **All Known State**
  All known properties, such as test case properties, data sets, and filters, are listed.

- **Download files referenced (img, link, script, embed)**
  If this check box is selected, the step downloads web page images into the test environment. If you do not select this check box, no images are downloaded.

The following table describes other functions that are part of the toolbar that is available on the **URL Parameters** and **POST Parameters** sections.

Field	Icon	Description

Add	Add icon	Add a parameter
Up	Up icon	Move a parameter up in the list of parameters
Down	Down icon	Move a parameter down in the list of parameters
Delete	Delete icon	Delete a parameter
Find:		Find text
Auto Generate a Filter from the referring Step to make this parameter dynamic.	Auto Generate a Filter icon	Create a new filter to auto-populate this property at run time. For more information on filters, see the Filters section.
Apply selected All Known State property to current parameter	Apply selected All Known State property filter	Apply state to the parameter. For more information on applying state, see the All Known State section that follows.
Auto Apply all All Known State properties to all properties by patterns	Auto Apply all Known State properties icon	Apply all state to all properties possible by patterns. For more information on applying state, see the All Known State section.

All Known State

All known properties, such as test case properties, data sets, and filters, are displayed in the **All Known State** panel.

All Known State panel

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 previous example:

1. Select the **u_login** key in the **URL Parameters** pane.
2. Select the **LISA_USER** key in the **All Known State** panel.
3. Click **Apply selected All Known State property to current parameter** .
   A warning message opens asking you to confirm the impending change.
4. Click **OK**.
   The new property is displayed in the **URL Parameters** pane.
   If all the names of the URL Parameter keys equal the names of the **All Known State** keys, you can click **Apply to All** to assign all the properties to the associated parameters quickly.

Use Property

If this option is selected, you can specify the following parameters:

- **Property Key**
  Specify a property that contains the connection information.
- **Download images referenced (img, link, script, embed)**
  If this check box is selected, the step downloads web page images into the test environment. If you do not select this check box, no images are downloaded.

HTTP Headers Tab

On the **HTTP Headers** tab, create any custom HTTP headers.

The **Custom HTTP Headers (Current Only)** section is for headers that are only sent to the server for this request.

The **Custom HTTP Headers (Persist)** section 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 target values.

Response Tab

On the **Response** tab, view the HTTP response that the server returns when this test was recorded. You can view:

- The source of the response
- The DOM tree of the response

SSL Tab

The SSL tab lets you enter information for either single or multiple SSL certificates.

Using a Single SSL Certificate

Enter the following information:

- **SSL Keystore File**
  The name of the keystore file where the client identity certificate is stored. The file can be in JKS or PKCS format.

- **SSL Keystore password**
  Password for the keystore file.

- **SSL Key alias**
  The keystore attribute that defines the alias that is used to store and retrieve the private key for the server.

- **SSL Key password**
  A password for the key entry if using a JKS keystore, and key has a different password from keystore.

Using Multiple SSL Certificates

To use multiple certificates on one step, store your SSL information in a data set. The following example shows how you can use a data sheet to store information about multiple certificates.

In this example, the demo server is running in the background and it has been configured to require client-side authentication. The demo server accepts five certificates. The keystore files for those certificates are in the project structure in a **Data** folder with the name **keystore**. These keystore files are pkcs12 keystores.

Using Multiple SSL Certificates window

Next, an HTTPS Request step is added and https is specified as the protocol and the port and path are entered.

HTTPS Request step is added and https is specified as the protocol and the port and path are entered

To ensure this process works with all the certificates and only run the test once, a data set is created for this test step. Use the Create your own Data Sheet data set to create the data set. The data set has the name **Certificate Information** and it is configured to run through the data set rows once, and then exit. Each certificate has five certificates and four parameters. When all the information is complete, use the **Create Data Sheet Skeleton** button to open the editor.

Create Data Sheet editor

For keystoreFile, we use the short file name of the keystore so that relative paths can be used later. To encrypt the password columns, right-click the column label and select **Encrypt**.

Create Data Sheet editor certificate information

Return to the HTTP step and select the **SSL** tab. Use property substitution for the variables. For the password fields, enter **{{keystorePassword}}** and **{{keyPassword}}**.

HTTP/HTML Request step SSL information dialog

The test is staged with **Stage a Quick Test**, and you can see that all of the tests completed successfully. All five certificates were tested against the demo server and client authentication was successful for all of them.

Test staged with Quick Stage Test

REST Step

The REST step is used when testing REST applications to send and receive HTTP(S) requests, including GET and POST parameters.

The following methods are supported:

- GET
- POST
- DELETE
- PUT
- HEAD
- PATCH
- OPTIONS

To view parameters, click **Test State** on the right vertical bar. The **Test State** area slides open. You can dock, pin, and hide the list.

To encode the parameters that are read from the **project.config** file, select the **Encode Properties Values in URL** check box.

To view the HTTP response, click **Response**.

If the response is a JSON string, the string is formatted automatically. To view the string in its original format, right-click and select **Remove whitespace**. To view the formatted string again, reopen the **Response** panel.

The bottom of the **Response** panel has a selection of filters and an assertion that you can add to the response.

When a service responds with a response code of 302 (redirect), the REST step automatically continues with the redirect URL from the response. To force the REST step to not redirect, add the property **lisa.http.followRedirects** with the value "false" to the project properties.

The demo server contains an example of invoking a JSON service to retrieve information from the LISA Bank user database. The URL is **http://localhost:8080/rest-example/**.

The **rest-example** test case in the examples project demonstrates how to use this step.

JSON Content

If you select the JSON type in the **Content** tab, the following tabs are displayed:

- **Visual JSON**
  Shows the content in a tree structure. Columns are provided for editing the names, types, and values.

- **Raw JSON**
  Shows the content as text. To apply formatting, right-click and select **Format**. To go to a specific line, right-click and select **Navigate**, **Line**.

Changes that you make in one tab are reflected in the other tab.

The following graphic shows the **Visual JSON** tab.

Screen capture of Visual JSON tab.

Web Service Execution (XML) Step

The Web Service Execution (XML) step lets you execute an operation on a SOAP-based Web Service using an HTTP POST or JMS message.

Access to a WSDL is not required; 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. This step lets you manipulate the raw SOAP message (XML) directly. This feature provides flexibility and power, but does expose you to the details of how web services work.

In general, the top portion of the editor is dedicated to how and where to send the SOAP message. The bottom portion is dedicated to the contents of the message.

The Web Service Execution (XML) step has a default name using this convention: *Web Service webServiceOperation name*.

If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

After the test step opens, it has two tabs, with each tab having multiple sub tabs.

The **PRO** icon switches between basic and advanced options. Some tabs and options are only available when **PRO** is selected.

- Connection Tab
- Basic Configuration

Connection Tab

The **Connection** tab has fields for the connection. The tab has sub tabs on the top bar and bottom bar.

- The top bar - for viewing the **Visual XML**, **Raw XML**, **Headers**, **Attachments**.
- The bottom bar - for **Request** and **Response**.
  - **Basic Configuration**
  - **Design Time Execution**

Basic Configuration

Connection

- **WSDL URL**
  The **WSDL URL** is an optional but recommended field (denoted by its slightly gray color). The **WSDL URL** must be a URL (either file:/, http:/, or https:/). From the **More Options** menu you can:
  - Browse the file system for a local WSDL or WSDL Bundle file.
  - Search a UDDI Registry (which populates the advanced UDDI access point lookup).
  - To migrate from the legacy WS step, select WSDL from hotDeploy.
  - Create and use a WSDL bundle. You can also create a WSDL Bundle from the **Actions** menu, but from the Options menu, DevTest automatically populates the WSDL URL with the resulting WSDL bundle file URL. Or if you already have a WSDL URL populated, it prepopulates the WSDL URL in the WSDL bundle dialog.

  When you enter a WSDL URL that is not already a WSDL bundle, DevTest creates a WSDL bundle and stores it locally in the Data/wsdls directory for the project. This action caches the WSDL locally for quicker access. DevTest parses the WSDL and uses its schema is used to build sample SOAP messages. The Visual XML editor also uses the WSDL to help you manually edit the SOAP message. DevTest first tries to 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. You can manually drop a WSDL bundle into the Data/wsdls any time. When the step tries to process the "live" WSDL URL, it uses the cached bundle instead.

- **Service, Port, Operation**
  If the **WSDL URL** is populated, the WSDL is processed and the **Service**, **Port**, and **Operation** selections are populated. These optional but recommended fields can be used to build a sample SOAP request message. Selecting a port also updates the **Endpoint URL** to match the definition in the WSDL. Changing the **WSDL URL** causes these items to be refreshed. If the **Endpoint URL** and **SOAP Message** are unchanged, they update also to correspond to the new WSDL, service, port, and operation that are selected.
  If the endpoint was changed and no longer matches the endpoint in the WSDL, a **Warning** button appears next to the field. A tooltip on the button indicates the differences between the entered value and the WSDL definition. Clicking the button updates the field to match the WSDL definition.
  If the **SOAP Request Message** no longer matches the default, it is not 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 empty SOAP request message.

  - Build full SOAP request message.

- **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.

- **Build Options**
  When building sample SOAP messages, various build options are used to determine what to do in various situations.

  - **Use String Pattern for Value:** When selected, it populates element values using DevTest string patterns as opposed to 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, only the first element in an XML schema choice is generated. To build all possible choice elements, select this option.
    Note: The SOAP request is not valid if you include multiple choice elements. However, it provides a sample for each possible choice, which makes it easier to build a message when you do not use the first choice.

  - **Maximum Elements:** Defines the maximum number of elements to include when building the sample message.

- **Maximum Type:** Defines the maximum number of complex schema types to include when building the sample message.

- **Insert Comments:** By default, comments that are related to the schema are generated. For example, when an element is optional, alternative choices, nillable elements, are generated. You do not see these comments in the Visual XML Editor, but they are visible in the Raw Editor.

Web Service Execution Tab

The Web Service Execution (XML) step provides a graphical editor and a text-based editor. You can also insert headers and edit attachment data.

- Visual XML Tab
- Raw XML Tab
- Headers Tab
- Attachments Tab

Visual XML Tab

The Visual XML Editor (or VXE) is a graphical editor for any XML document. A SOAP message is an XML document that conforms to the SOAP specification (SOAP schema). You can use the editor to build and edit the SOAP message.

Visual XML Editor

The table shows the XML document, including the SOAP Envelope and Body elements.

- **Type**
  Shows the type for each element.

- **Occurs**
  Indicates how many elements are expected. The first number indicates the minimum number of times the element can occur. Zero means that it is optional and can be removed. The second number indicates the maximum number of times the element can occur. Infinity means that there can be an unlimited number of elements of that name.

- **Nil**
  Lets you set the element value as **nil** or **un-nil**. Selecting the check box removes any element children or values but leaves all attributes alone. Clearing the check box populates all expected children and attributes as defined in the WSDL schema.

- **Nillable**
  Indicates whether the element can be nil. A red icon indicates that the element is non-nillable, but is set to **nil**. A green icon indicates that the element is non-nillable and is not set to **nil**.

- **Value**
  You can type the element values directly into the **Value** column. If the element type is a known type, specialized edit buttons appear.

To select the columns to display or hide, right-click the column heads.

When you right-click the editor, a context-sensitive menu provides options for manipulating the document. This menu contains the following options:

- **Add Schema Attribute/Element**
  Lets you select from a list of valid attributes or elements. If a schema is present and an element or attribute is selected in the editor, child elements and attributes can be selected to be added. If more than 20 schema objects are available, a dialog can be used to select the schema object. This dialog contains a search text field, which makes it easier to find a specific schema object when there are dozens of them.

- **Add Element**
  Add an element to the document.

- **Add Attribute**
  Add an attribute to the document.

- **Add Text**

- **Remove Element/Attribute**
  Remove the selected element or attribute.

- **Move Up/Down**
  Move the selected node up or down.

- **Convert to Attachment**
  A shortcut method for creating a standard referenced attachment. For more information, see Attachments Tab.

- **Convert to XOP Attachment**
  A shortcut method for creating a standard referenced XOP Include attachment. For more information, see Attachments Tab.

- **Create XML Data Set**
  A shortcut method for generating an XML Data Set. A typical use case is 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 an XML Data Set automatically populates the first record with the selected XML element tree. The new data set property is set as the value in the editor.

- **Hide Text Nodes**
  By default, DevTest hides text nodes that are typically redundant (such as white space). However, it is occasionally useful to view them, including when the XML element is a mixed type element that supports intermixed elements and text.

- **Hide Namespace Nodes**
  By default, DevTest hides namespace declarations and namespace prefix declarations. You can show them to confirm a prefix value or to change prefixes or namespace scoping.

- **Validate**
  Validates the XML and displays the results in the **Validation Results** pane at the bottom of the window.

To switch between displaying and hiding the results of the XML validation, click **Validation Results** .

You can also use keyboard shortcuts for some tasks:

- To remove the selected element or attribute on Windows, press Ctrl+Backspace.

- To move the selected node up on Windows, press Ctrl+up arrow.

- To move the selected node down on Windows, press Ctrl+down arrow.

Right-clicking an attribute displays a menu with some of the functionality of the general right-click menu.

Editing the Type Field

The **Type** column shows the XML schema type (local name) and has a tool tip to show the fully qualified name (qName) with namespace. This column can be edited for derived XSD types. Only base types with derived types can be edited.

When the column is available for editing, a list of available derived types and the base types are presented in a combo box. You can select a type, and the associated element is now associated with the selected type.

Changing the type removes all child elements and attributes from an element and sets the nil attribute on the element to nil=true.

Raw XML Tab

The Raw XML Editor is a text-based editor that is XML aware and lets you manually edit the raw XML SOAP message. Any changes that are made are seen when you switch back to the Visual XML Editor (and conversely).

Raw XML Editor

The Raw XML Editor has a right-click menu for formatting the raw XML.

> **Note**: The **Remove whitespace** option trims whitespace and all ASCII control characters from the beginning and end of each line in the document.

If you edit the document to make it invalid XML, the Visual XML Editor could show an error message.

Fix your changes in the Raw XML Editor, and the Visual XML Editor begins working again.

Headers Tab

The **Headers** tab lets you insert headers that are transmitted with the SOAP message (for example, HTTP Headers or JMS properties).

Headers Tab

To add a header row and select a header from the drop-down list, click the plus sign.

Accept

The **Accept request-header** field can be used to specify specific media types that are acceptable for the response. Use accept headers to indicate that the request is specifically limited to a small set of types. For example, in the case of a request for an in-line image.

This field contains a semicolon-separated list of representation schemes that are accepted in the response to this request.

```
Accept          = "Accept" ":"
                  #( media-range [ accept-params ] )
```

Accept - Language

The **Accept - Language** header field is similar to **Accept**, but lists the language values that are preferable in the response. A response in an unspecified language is not illegal.

```
Accept-Language = "Accept-Language" ":"
 1#( language-range [ ";" "q" "=" qvalue ] )
   language-range  = ( ( 1*8ALPHA *( "-" 1*8ALPHA ) ) | "*" )
```

User - Agent

The **User-Agent** request-header field contains information about the user agent originating the request.

The headers tab is for statistics, tracing protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid specific user agent limitations. User agents should include this field with requests.

By convention, the product tokens are listed in order of their significance for identifying the application.

```
User-Agent     = "User-Agent" ":" 1*( product | comment
```

The **Connection** general-header field lets the sender specify options that are appropriate for the specific connection. Proxies and *must not* communicate the field over further connections.

```
Connection = "Connection" ":" 1#(connection-token)
connection-token  = token
```

Authorization

To authenticate itself with a server, a user agent (usually, but not necessarily, after receiving a 401 response) includes an **Authorization** request-header field with the request. The **Authorization** field value consists of credentials containing the authentication information of the user agent for the realm of the resource being requested.

```
Authorization  = "Authorization" ":" credential
```

DevTest does not support using properties in the **Headers** tab. The UI takes a user name and password during design and calculates the value of the authorization header. To use properties, set the **lisa.http.user** and **lisa.http.pass** properties either in local.properties or in config files and DevTest uses those properties during run time.

Attachments Tab

The **Attachments** tab lets you edit attachment data.

Attachments Editor

Referenced Attachments

Referenced attachments are referenced in the SOAP message. To use referenced attachments, in the VXE, right-click the element that you want to be the referenced attachment and select the appropriate action:

- **Convert to Attachment** (for MIME and DIME)
- **Convert to XOP Attachment** (for MTOM/XOP Include style)

This action automatically creates the necessary elements and attributes and configures the content ID that is used to match up the element with the attachment. The action then completes the following actions:

- Switches to the **Attachments** tab
- Prepopulates a new attachment with the content ID
- Selects a default content type and attachment type
- Populates the value with any existing element data from the VXE

Unreferenced Attachments

If you plan to use unreferenced (anonymous) attachments, use the **Attachments** tab to add an attachment manually.

Use the **Add**, **Up**, **Down**, and **Delete** icons to add, remove, or rearrange the attachments in the table.

- **MIME DIME XOP MTOM**
  This field controls how the attachments are sent, using MIME, DIME, XOP, or MTOM standards. XOP sends different content headers that are based on the SOAP version.
  When **MTOM** is selected, any base64binary schema types are automatically optimized using the XOP standard. Adding attachments manually is not necessary. If an element is already configured as an attachment, it is left alone. Any extra attachments added manually are also sent.
  If you select **Force** (even if the document contains no base64binary elements) the document is formatted and sent as an attachment (Microsoft MTOM method).
  The limitation of automatically optimizing elements is that the VXE must understand the element as a base64binary schema type (or extension/restriction thereof). If you use a data set or property, the expanded property or first data set entry must contain all possible elements to optimize. If the VXE does not display the element that must be optimized, the element is not optimized.

- **cid**
  The Content ID that can be used in an **href** attribute in the SOAP message to link the element to the attachment data.

- **content type**
  The mime encoding type that is used to assist the server in how to process the attachment data.

- **type value**
  The DevTest attachment type determines how to edit and interpret the value data. Each type has its own editor.

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 sends the attachment as text; otherwise it sends 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 content is loaded and sent as the attachment data.

Design Time Execution

After you configure the connection information and build the SOAP Request Message, you can run the Web Service Execution (XML) step at design time to test it.

Execute the Web Service operation by clicking **Execute** in the upper right corner. After it has executed, the **Request** and **Response** tabs are populated and it will switch to the **Response** tab automatically.

- Request Tab
- Response Tab

Request Tab

The **Request** tab shows the resulting request data that was sent after any post processing (for example, substituting DevTest properties). If the message contained any attachment, it does not show the raw MIME or DIME encoded message, but rather the processed message and attachments. To see the raw message, use a tool like TCPMon.

Web Service Execution (XML) step Request tab

Header Tab

The **Header** tab shows the Transport Headers that were sent for the request.

Web Service Execution (XML) step Request tab Header tab

XML Tab

The **XML** tab shows the raw SOAP message that was sent after any advanced processing.

Web Service Execution (XML) step Request tab XML tab

DOM Tree Tab

The **DOM Tree** tab shows a DOM tree for the SOAP message.

Web Service Execution (XML) step Request tab DOM Tree tab

Attachments Tab

The **Attachments** tab shows any attachments that were sent.

Response Tab

The **Response** tab shows the resulting response data that was received. If the message contained any attachment, it does not show the raw MIME or DIME encoded message, but rather the processed message and attachments. To see the raw message, use a tool like TCPMon. If any advanced post-processing options are set (see the following window), the SOAP response message is shown post-processed.

Web Service Execution (XML) step Response tab

Header Tab

The **Header** tab shows the Transport Headers that were received from the response.

Web Service Execution (XML) step Response tab Header tab

XML Tab

The **XML** tab shows the raw SOAP message that was received after any advanced processing.

Web Service Execution (XML) step Response tab XML tab

DOM Tree Tab

The **DOM Tree** tab shows a DOM tree for the SOAP message. From this tab, you can quickly add filters and assertions on the resulting SOAP message.

Web Service Execution (XML) step Response tab DOM Tree tab

Attachments Tab

The **Attachments** tab shows any attachments that were received. You can access the received attachments using automatically generated DevTest properties (for use in later steps, filters, or assertions). For each attachment, the following properties are set.

- **lisa.<step name>.rsp.attachment.<cid>**
  Attachment value, byte, or String

- **lisa.<step name>.rsp.attachment.contenttype.<cid>**
  Content type (mime type, for example, text/plain)

- **lisa.<step name>.rsp.attachment.<index>**
  Attachment value, byte, or String

- **lisa.<step name>.rsp.attachment.contenttype.<index>**
  Content type (mime type, for example, text/plain)

- **lisa.<step name>.rsp.attachment.contentid.<index>**
  Content Id (cid)

The parameter **step name** is the DevTest step name that is being executed.

The parameter **cid** is the Content ID that is usually referenced in the SOAP message.

The parameter **index** starts at 0 and is increased for each attachment in the response attachments list.

Advanced Settings

The Web Service Execution (XML) step includes advanced settings.

To open the **Advanced settings** tabs, click **PRO** . Five new tabs open in the top of the panel, and two new tabs open at the bottom level.

Web Service Execution (XML) step Advanced Settings tabs

- Transport Tab
- SSL Tab

- UDDI Tab
- WS-I Tab
- Advanced Tab
- Request Editor Tabs

Transport Tab

Web Service Execution (XML) step Transport tab

- **HTTP Version**
  This parameter controls which HTTP protocol is used when sending the operation request. The default is 1.1.

- **SOAP Version**
  The SOAP version is auto-populated based on the WSDL definition. **SOAP Version** controls the generation of a number of transport headers (for example, SOAPAction and contentType).

- **Call Timeout (ms)**
  This parameter defines how to wait while trying to execute the operation. After the timeout is hit, an exception will be thrown and the On Error handling will occur.

SSL Tab

Web Service Execution (XML) step SSL tab

- **SSL Keystore File**
  The name of the keystore file where the client identity certificate is stored. The file can be in JKS or PKCS format.

- **SSL Keystore password**
  Password for the keystore file.

- **SSL Key alias**
  The keystore attribute that defines the alias that is used to store and retrieve the private key for the server.

- **SSL Key password**
  An optional password for the key entry if using a JKS keystore, and key has a different password from keystore.

For an example of using multiple SSL certificates, see HTTP_HTML Request Step.

Specify global certificates properties for SSL in your **local.properties** file.

For global certificates (web server and web service steps):

- **ssl.client.cert.path**
  A full path to the keystore.

- **ssl.client.cert.pass**
  Password for the keystore (this password is automatically encrypted when DevTest 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 is automatically encrypted using AES (Advanced Encryption Standard) when DevTest runs.

Note: This option is not available for the WS Test step. To set this option, use 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 is automatically encrypted when DevTest 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 is automatically encrypted using AES (Advanced Encryption Standard) when DevTest runs.

The SSL configuration in this step is only used for execution. If the WSDL access requires SSL authentication, set the **ssl.client.*** properties in **local.properties**. The **ws.ssl.client.*** properties, as well as the SSL tab in the UI, assigns the SSL configuration to be used only during the execution of the WS step. The SSL configuration for WSDL access may not necessarily be the same as the SSL configuration for the service.

Note: This option is not available for the WS Test step. To set this option, use the **local.properties** file.

> **Note:** If you have duplicate values in **local.properties** and in the general tab, the values in the general tab are used.

UDDI Tab

Web Service Execution (XML) step UDDI tab

- **Perform Access Point (Web Service URL) Lookup**
  Select the **Inquiry URL** and **Binding Template**. To perform the lookup, use the **Search UDDI Server Find** button to navigate to the correct binding.

> **Note:** When creating the step, if you used the UDDI Search function when specifying the web service WSDL URL, these values are automatically populated. If the **Inquiry URL** is specified but the **Binding Template** is not, you could have performed a Model search. To locate the **Binding Template**, perform a search at a higher level of the hierarchy. Then drill down to the TModel through a specific Binding Template.

WS-I Tab

Web Service Execution (XML) step WS-I tab

- **WS-I Basic Profile 1.1**
  You can select four different validation levels in the pull-down menu.
  - **Display All Assertions**
  - **Display All But Info Assertions**
  - **Display Only Failed Assertions**
  - **Display Only Not Passed Assertions**

- **Validate**
  Select to validate the WSDL, the SOAP message, or both.

- **On Failure Go To**
  Select the step to redirect to on error.

> **Note:** Validation failures are common, but usually do not affect the outcome of the test. A good practice is to set the next step to continue so that you can complete the test.

Advanced Tab

Web Service Execution (XML) step Advanced tab

- **SOAP Action**
  This field is auto-populated based on the WSDL operation definition. The field is used as the value of the SOAPAction transport header for SOAP 1.1 request message. Change this field manually only in rare cases.

- **Style**
  This field is auto-populated based on the WSDL operation definition. The field is used to determine how a sample SOAP message is generated. Change this field manually only in rare cases.

- **Use**
  This field is auto-populated based on the WSDL operation definition. The field 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. Change these fields manually only in rare cases.

- **SOAP Fault is Error**
  If a SOAP fault is returned, perform On Error handling.

- **Do Not Send Request**
  When selected, the step execution performs all of the normal SOAP message processing, but it does not send the generated SOAP message. Instead it sets the response to be the request message that would have been sent.
  This method is recommended to use transports other than HTTP/S to send SOAP requests. For example, when using a JMS client, you can configure the WS step to only create the request and not send it. Then you can append a Generic JMS step to actually send the request and receive the response.

- **Maintain Session**
  Select to maintain cookies across invocations.

- **Clear Session**
  Select to clear cookies across invocations. Clearing the session cookies acts like a new session was created. Old session cookies are not used and any new cookies in the response are not set on the session. However, because the session is not cleared, future steps can still use it.

Request Editor Tabs

Addressing Tab

You can send a WS-Addressing header with your request. The WSDL does not specify if WS-Addressing information is required so you must configure it.

Web Service Execution (XML) step Addressing tab

Click the **Addressing** tab, and then specify:

- **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 (for example, .NET) still use the older Draft specifications. Determine which your web service platform is using.

DevTest populates as many values as possible. Then you can select to use the default value or override it. You can select not to send some of the default elements by clearing the **Default** check box for that element.

To assure that the web service can understand the WSAddressing header, select the **Must Understand** check box.

Security Tab

Web Service Execution (XML) step Security tab

Click the **Security** tab and click **Send**.

- **Must Understand**
  Select to ensure that the server processes the WS-Security header.

- **Actor/Role**
  Enter the name if needed: most web services do not use multiple Actors/Roles.

Click **Add** and select the security action type to add. You are presented with the configuration panel for that security action type.

Adding the security verification to the Response is similar:

Click the **Security** tab. Click **Add** , and select **Receive**.

- Enter the **Actor/Role** name (if needed)

- Click **Add** and select the security action type. The configuration panel for that security action type opens.

> **Note:** You can add as many security types as are necessary to execute your web service.

When you use a keystore in a security action type configuration, you can verify in the keystore settings that you are using the correct format, password, alias, and alias password. Clicking the **Verify** button on the editors for **Signature**, **Encryption/Decryption**, and **SAML Assertion Token** invokes the Keystore Verifier, which produces a verification report.

If you do not know the expected alias name for a WS-Security setting, use the **Keystore Verifier** to list all of the aliases in the keystore. Leave the **Keystore Alias** and **Alias Password** boxes empty and click **Verify**. The **Keystore Verifier** is described at the end of this section on WS-Security.

> **Note:** You can load and save the security configuration information from and to a .wss file, using the **Load** and **Save** icons. This capability allows for quick and easy creation of new steps connecting to the same service.

Security Example

This page describes the parameters that are necessary to run the WS-security example.

- XML Encryption-Decryption
- XML Signature Token-Signature Verification
- Timestamp-Timestamp Receipt
- Username Token-Username Token Verifier
- SAML Assertion Token-SAML Assertion Receipt

XML Encryption-Decryption

Encryption

Select the **Use Encryption** check box.

- **Keystore File**

  The location of the keystore file.

- **Keystore Password**

  Enter the password for the keystore.

- **Keystore Alias**
  Enter an alias for a public key.

- **Alias Password**

  Leave empty or make the same as Keystore Password for PKCS #12 files.

- **Key ID Type**
  Select the appropriate key ID type from the pull-down menu.

- **Algorithm**
  Select Triple DES, AES 128, AED 192, or AES 256.

- **Transport**
  Select PKCS#1: RSA Encryption Standard v1.5 or Optimal Asymmetric Encryption Padding with RSA Encryption.

The default behavior is to encrypt only the SOAP Body contents.

- **Encrypt Only Parts**
  To specify different parts to encrypt, click **Select** to identify the parts to be encrypted.

- **Type**
  Values:

  - **Element:** Encrypt the element and the content.

  - **Content:** Encrypt only the content.

- **Namespace URL**

  Enter the value for the element.

- **Element**

  Enter the name of the element.

Click **Add** to repeat the process.

To include the Body element, manually add it. To include the Binary Security Token as a part, use the Element name **Token**.

Decryption

- **Keystore File**

  The location of the keystore file.

- **Keystore Password**

  Enter the password for the keystore.

- **Keystore Alias**
  Enter an alias for a public key.

- **Alias Password**

  Leave empty or make the same as Keystore Password for PKCS #12 files.

XML Signature Token-Signature Verification

Signature Token

Select the **Add Signature** check box.

- **Keystore File**

  The location of the keystore file.

- **Keystore Password**

  Enter the password for the keystore.

- **Keystore Alias**
  Enter an alias for a private key.

- **Alias Password**

  Leave empty or make the 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.

- **Digest Algorithm**
  Values: SHA-1, SHA-256, SHA-384, SHA-512, RIPEMD-160, or MD5 (not recommended).

The default behavior is to sign only the SOAP Body contents.

- **Sign Only Parts**
  To specify different parts to sign, click **Select** to identify the parts to be signed.

- **Type**
  Values:
  - **Element:** Encrypt the element and the content.
  - **Content:** Encrypt only the content.
- **Namespace URL**

  Enter the value for the element.
- **Element**

  Enter the name of the element.

Click **Add** to repeat this process.

To include the Body element, add it manually. To include the Binary Security Token as a part, use the Element name **Token**.

Signature Verification

The parameters that are required to configure the signature verification are a subset of the parameters that are required for signing.

- **Keystore File**

  The location of the keystore file.
- **Keystore Password**

  Enter the password for the keystore.
- **Keystore Alias**
  Enter an alias for a public key.
- **Alias Password**

  Leave empty or make the same as Keystore Password for PKCS #12 files.

Timestamp-Timestamp Receipt

Timestamp

Select the **Add Timestamp** check box.

- **Time-To-Live (sec)**
  Enter the lifetime of the message in seconds. Enter 0 to exclude an Expires element.

> **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 milliseconds. For these web services, clear the **Use Millisecond Precision in Timestamp** check box.

Timestamp Receipt

The parameters that are required for Timestamp Receipt are a super set of those parameters that are required for Timestamp. The additional parameter is:

- **Don't allow expired**
  Select this parameter if you do not want to allow expired timestamps.

Username Token-Username Token Verifier

Username Token

Select the **Add Username Token** check box.

- **User Name**
  Enter the user name.

- **Password**
  Enter the appropriate password.

- **Password Type**
  Select the password type from the drop-down list (Text, Digest, None). None is typically used with the Add Signature option.

- **Add Nonce**
  Specifies whether a nonce is required to protect against replay attacks.

- **Add Created**
  Select if a timestamp is required.

- **Use Millisecond Precision in Timestamp**
  To use millisecond precision, select the check box. Some Web services, particularly .NET 1.x/2.0 with WSE 2.0, are not compliant with standard timestamp formatting and do not allow the use of milliseconds.

- **Add Signature**
  Select to add a signature that is built using a combination of the username and password as the key.

- **Sign Only Parts**
  To specify different parts to sign, click **Select** to identify the parts to be signed.

- **Type**
  Values:

  - **Element:** Encrypt the element and the content.

  - **Content:** Encrypt only the content.

- **Namespace URL**

  Enter the value for the element.

- **Element**

  Enter the name of the element.

Click **Add** to repeat for as many elements as you want.

To include the Body element, manually add it. To include the Binary Security Token as a part, use the Element name **Token**.

UserName Token Verifier

Select the **Verify Username Token** check box.

- **User Name**
  Enter the user name.

- **Password**
  Enter the appropriate password.

- **Use Millisecond Precision in Timestamp**
  To use millisecond precision, select the check box. Some Web services, particularly .NET 1.x/2.0 with WSE 2.0, are not compliant with standard timestamp formatting and do not allow the use of milliseconds.

- **Verify Signature**
  Select the check box if signature verification is required.

SAML Assertion Token-SAML Assertion Receipt

SAML Assertion Token

Select the **Add SAML Token** check box.

Perform one of the following options:

- Select the **From Step Results** check 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)

- Select the **From Property** check box and enter the property that contains the XML SAML Assertion.

Click **Verify** to have DevTest parse the SAML Assertion XML and build the SAML Assertion object as it would when sending the SOAP request. Verifying is useful to confirm that the SAML Assertion that could have been created manually is a valid SAML Assertion. Verifying also attempts to verify any signatures that are associated with the assertion. However, it is likely that DevTest cannot verify the assertion without configuring a public certificate with which to verify.

Select the **Signed Sender Vouches** check box if the sender must sign the assertion (the sender instead of the bearer/creator of theassertion vouches for its authenticity). When selected, the following information is required:

- **Keystore File**

  The location of the keystore file.

- **Keystore Password**

  Enter the password for the keystore.

- **Keystore Alias**
  Enter an alias for a private key.

- **Alias Password**

  Leave empty or make the 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.

- **Digest Algorithm**
  Values: SHA-1, SHA-256, SHA-384, SHA-512, RIPEMD-160, or MD5 (not recommended).

The default behavior is to sign only the SOAP Body contents.

- **Sign Only Parts**
  To specify different parts to sign, click **Select** to identify the parts to be signed.

- **Type**
  Values:

  - **Element:** Encrypt the element and the content.

  - **Content:** Encrypt only the content.

- **Namespace URL**

  Enter the value for the element.

- **Element**

  Enter the name of the element.

To repeat this process, click **Add**. To include the Body element, add it manually.

To include the Binary Security Token as a part, use the Element name **Token**.

- **SAML Assertion Receipt**
  To scan the response for a SAML Assertion Receipt header in the response, select the **Process SAML Assertion** check box. If you select this option and there is no SAML Assertion Receipt header, an exception occurs.

- **Signature Confirmation**
  To scan the response for a signature confirmation header in the response, select the **Signature confirmation** check box. If you select this option and there is no confirmation header, an exception occurs.

- **Using the Keystore Verifier**
  You can verify your keystore settings to ensure that you are using the correct format, password, alias, and alias password. To produce verification reports, click **Verify** on the editors for SSL, Signature, Encryption/Decryption, and SAML settings.
  SSL verification validates the Keystore password only. The SSL verification also confirms at least one of the keys in the keystore can be loaded using the keystore password.
  WS-Security verification validates the Keystore password, the alias, and the alias password. A correct validation is indicated with a green entry. Any validation errors that are found are 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 an incorrect choice of algorithm. These issues must be validated independently.

- **Alias Search**
  If you do not know the expected alias name for a WS-Security setting, use the keystore verifier. The keystore verifier lists all of the aliases in the keystore. Leave the **Keystore Alias** and **Alias Password** boxes empty and click **Verify**.
  Aliases have a blue background.
  Verification fails because the **Keystore Alias** and **Alias Password** boxes were left blank.

- **WS-I Report**
  When you click **Execute** on the **Object Editor** window, the validation runs. A report is generated and saved (in the reports directory in the DevTest install directory). You can view the report by pressing the **WS-I Report** tab at the bottom of the window.

> **Note:** The format of this report is standard, and the Web-Services-Interoperability Organization (WS-I) dictates the format.

WSDL Validation

The WSDL Validation step lets you load a WSDL and add one or more assertions to validate the WSDL. This step is different in that it lets you load the static WSDL file and perform assertions on it. In most steps, the assertions are made on the response.

The following list describes the most useful assertions:

- **XML Diff Assertion**
  Checks that the WSDL has not 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 Assertion Descriptions.

Prerequisites: Familiarity with the three assertions that were named previously.

Parameter Requirements: Location of the WSDL to validate.

To configure the WSDL Validation step, enter a WSDL in the **WSDL URL** field and click **Load**.

The WSDL appears in the editor. You can view it in XML or DOM View.

You are now ready to add the assertions.

Base64 Encoder

The Base64 Encoder step is used to encode a file using the Base-64 encoding algorithm. The result can be stored into a 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 property. To encode a file, click **Load**. The Base64 encoded text that is displayed in the editor is read-only.

Complete the following fields:

- **File**
  Enter the full path and path name, or browse to the file to be encoded.
- **Property Key [opt]**
  The name of the property in which to store the encoded file.
- **Load**
  Click to load and test the encoding of the file. Optionally, store it in the specified property.
- **If environment error**
  Action to take if an environment error occurs.

After the file is encoded, you can add filters and assertions. The valid options are:

- Random Selection Filter
- Parse Value Filter
- XML Xpath Filter
- Create HTML Table ResultSet Filter
- Make Assert on Selection

When you have added filters and assertions, click **Load** to load the file or **Save** to save the editor contents in a new file.

Multipart MIME Step

The Multipart MIME (Multipurpose Internet Mail Extensions) step lets DevTest load data from a file, encode it, and store it in a property for use as a post parameter on an HTTP request. The encoded document is stored in the property that has been defined previously in an HTTP/HTML Request step.

When a multipart MIME form submit request is recorded, the contents of the file that was uploaded are recorded. Subsequent playback results 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 exist, and it must be before the Multipart MIME step.

Complete the following fields:

- **Step**
  Select the name of the HTTP/HTML Request step, or select from the pull-down menu, the step that receives the property containing the encoded document.

- **Parameter**
  Select the name of the property listed in the step named in the **Step** field from the pull-down menu.

- **File**
  Enter the pathname or browse to the document to be encoded.

- **MIME Type**
  Enter the MIME type that the server expects.

Click **Load** to encode the file.

Multipart MIME step encoded

To save the contents of the editor to a file, click Save .

In the example in the previous graphic, the Account Activity step has a post-parameter user ID that contains the encoded version of the file **DataSet1.lds**.

SAML Assertion Query

The SAML Assertion Query step lets you obtain a SAML assertion from an identity provider for 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 that you must perform. Obtain this information from either the developer of the system that uses SAML Assertions as the form of identity security or from the identity provider administrator.

Parameter Requirements: At a minimum, know:

- The URL to the SAML Query interface (endpoint) for the identify provider.

- The Subject information (who/what you want to obtain a SAML Assertion for).

- The type of query you want to perform.
- Some extra information depending on the type of query.

SAML Assertion Query step

The SAML Assertion Query Editor has four tabs. The **Editor** tab lets you configure the query information. After you configure the query, you can test the query using the **Test** button in the **Query** section of the editor. After you test 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. For example, if the query returned multiple assertions, you can see the assertions in the **Raw Query Result** tab. You can view the step response in the **Last Response** tab. The **Last Response** tab shows, for example, what is used in the Web Service WS-Security token.

The **Advanced** button lets you enter more information.

SAML Assertion Query step Advanced Settings

Select the **Add Signature** check box.

- **Keystore File**
  The location of the keystore file.
- **Keystore Passwor**d
  Enter the password for the keystore.
- **Keystore Alias**
  Enter an alias for a private key.
- **Alias Password**
  Leave empty or make the same as Keystore Password for PKCS #12 files.
- **Signature Algorithm**
  Select DSA with SHA-1.
- **Digest Algorithm**
  Values: SHA-1, SHA-256, SHA-384, SHA-512, RIPEMD-160, or MD5 (not recommended).

- Connection
- Subject
- Response (deprecated)
- Query

Connection

This information describes where the SAML Query API server is and how to connect to it.

- **Endpoint**
  The URL to the SAML Query API of the identity provider.

- **SSL Keystore**
  To use client-side identification certificates to connect to the endpoint, select the keystore file using the **Select** button. Or, select a previously entered 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 to use to query the identity provider.

Subject

This information describes the recipient of a SAML assertion. The subject could be a user, user group, or other entity for which you want to provide an assertion about the current authorization /privileges.

- **Name**
  The name of the entity (for example, username).

- **Name Qualifier**
  A group or categorization that is used to qualify the **Name** (for example, domain).

- **Format**
  This field describes what format the name is being sent (for example, Full Name as opposed to Username).

- **Confirmation Methods**
  Select which confirmation method types you want to include in the query. The query only returns assertions that contain at least one of the specified types. If you leave all types cleared, the query returns any assertion regardless of the confirmation method.

Response (deprecated)

This information describes which assertion statements you want to be returned as part of the SAML Assertion. **Response** is deprecated as of SAML 1.1.

- **Local Part**
  The element name (for example, AuthenticationStatement, AuthorizationDecisionStatement, and AttributeStatement).

- **Namespace**
  The element namespace (for example, urn:oasis:names:tc:SAML:1.0:assertion).

Click **Add** to add more XML elements to the set to be returned.

Click Delete to remove any elements you have already added.

Query

A description of which type of query to perform. The query types are:

- Attribute

- Authorization

- Authorization Decision

- **Attribute**
  An attribute query responds with a set of attribute statements. For example, it could tell you which groups a subject is a member of.

  - **Resource**
    To limit your query to a specific resource (for example, a specific web service, domain, file) specify the resource name.

  - **Attribute Designators**
    A name and namespace (like XML elements) identifies each attribute. You can filter the set of attribute statements that are returned by specifying each attribute type to be returned.
    Example:

    ```
    Name = urn:mace:dir:attribute-def:
    eduPersonScopedAffiliation, Namespace = urn:mace:shibboleth:1.0:
    attributeNamespace:uri)
    ```

- **Authorization**
  Used to request authentication statements that are related to a specific Subject SAML Assertions.

  - **Authorization Method**
    This parameter limits your query to returning Authorization Statements that are for a specific authorization method.
    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 is available from the pull-down list.

- **Authorization Decision**
  Used to request SAML Assertions for specific actions that a subject wants to perform, given the evidence.

  - **Resource**
    To limit your query to a specific resource (for example, a specific web service, domain, or file), specify the resource name.

  - **Actions**
    Specify at least one action for which 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 Assertions to include with the Authorization Decision Query as advice to the Identity Provider. Specify the property that holds the SAML Assertion XML. To use the response from a previous step (for example, another SAML Assertion Query or Parse Text step) use **lisa.<stepname>.rsp**.

  - **Evidence (Reference IDs)**
    Optionally specify assertion reference IDs.

Java-J2EE Steps

The following steps are available in the **Java/J2EE** category:

- Dynamic Java Execution
- RMI Server Execution
- Enterprise JavaBean Execution

Dynamic Java Execution

The Dynamic Java Execution step lets you instantiate and manipulate a Java object. All Java classes on the DevTest classpath are available, including the classes in the JRE classpath. Any user classes can be placed on the classpath by copying them into the hotDeploy directory. The class under test is loaded into the Complex Object Editor where it can be manipulated without having to write any Java code.

This example uses a Java date instance of class **java.util.Date**.

1. Enter the following parameters in the Dynamic Java Execution editor:

   - **Use JVM**
     Select **Local**.

     > **Note:** You can use In-Container Testing (ICT) to execute a Java object remotely by clicking the **Remote** option button. However, this mode requires some extra setup before it can be used. For more information, see the Using the SDK.

   - **Local JVM Settings**
     Select one of the following options:

     - **Make New Object of Class:** Select this option button and enter, select, or browse to the Java class to instantiate. This value must be the fully qualified class name including the package of the Java class; for example, **com.example.MyClass**.

     - **Load from Property:** Click the option 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.

   > **Note:** If you require that the Java object be loaded by its own classloader, add the **Class Loader Sandbox Companion**.

2. Click **Construct/Load Object**.
   The **Complex Object Constructor** window opens and lists the available constructors for your object.

3. Select a constructor and click **Next**.

4. Enter any input parameters that the constructor needs.

Dynamic Java Execution step Complex Object Constructor

5. In this example, enter a string representation of a day (5/10/2011). You can enter a value, a property, or null. DevTest constructs the object and loads it into the Complex Object Editor.

Dynamic Java Execution step Complex Object Editor

6. You can now manipulate the object, and execute methods, using the Complex Object Editor.
   For more information about how to use the Complex Object Editor, see Complex Object Editor (COE).

7. You can use either of the following methods to add filters and assertions:

   - Use the inline filter/assertion form (part of the Complex Object Editor)

   - Manually, by selecting filter under your test step in the test case tree

   For example, the following graphic is a window before we execute the before method on the Date class.

COE before we execute the before method on the Date class

In the **Status/Result** section, you can add an inline filter in the **Save Results in Property** text box. You can also add an inline assertion in the **Comparison on Result Like** text box. The Dynamic Java Execution step has a default name using this convention: Dynamic Java Execution. You can change step names at any time.

RMI Server Execution

The RMI Server Execution step lets you acquire a reference to a remote Java object through Remote Method Invocation (RMI) and call the Java object.

Prerequisites: Knowledge of the Complex Object Editor is assumed. You must also copy the interface and stub classes for the remote object into the hot deploy directory. These actions are required to contact and interact with the remote object. Get these classes from the remote object developer.

Parameter Requirement: You must know how to connect to the RMI Server (usually a host name and port) and you must know the RMI name of the object you want to invoke.

The RMI Server step editor lets you enter the following parameters:

- **RMI Name**
  Complete one of the following actions:
  - Enter or select the full RMI name of the object (as shown previously)
  - Enter the RMI Server name and click **Apply** to open a list of available objects.

DevTest constructs the object and loads it into the Complex Object Editor.

COE with getName method selected

You can now manipulate the object, and execute methods, using the Complex Object Editor.

In the previous graphic, the getName method was selected. If you double-click it, it gets added in the Object Call Tree. You can then click **Execute** in the dialog, to execute it with any required method arguments and information about how to process the result.

RMI Server Execution step before method is executed

The previous graphic shows null as the return value because the method is not yet executed. After you click **Execute**, it gets executed and the correct return type is shown. The following graphic shows how the **Object Call Tree** tracks the results of execution of several methods.

RMI Server Execution step after method is executed

You can also use either of the following methods to add filters and assertions:

- Use the inline filter/assertion form
- 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** text box. You can see an inline assertion in the **Comparison on Result Like** text box.

> **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 IP address.

The RMI Server Execution step has a default name using this convention: *RMI Server Execution - operation name*.

If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

Enterprise JavaBean Execution

The Enterprise JavaBean Execution step lets you acquire a reference to and make calls on an Enterprise JavaBean (EJB) running in a J2EE application server.

Testing an EJB is similar to testing a Java object. DevTest dynamically connects to the EJB using the Home EJB interface, and then from it creates an instance of an EJB object. This process is a little different for EJBs, as they do not require a home interface. The EJB under test is loaded into the Complex Object Editor, where it can be manipulated without having to write any Java code.

The EJB step has a default name using this convention: *EJB javaMethod dynamic java execution*. Before the step is saved, the default step name is *EJB*. If another step also uses the default step name, a number is appended to it. You can change step names at any time.

Prerequisites:

- Knowledge of the Complex Object Editor is assumed.
- Copy the application JAR files for the EJB under test and the client EJB JAR file to the **LISA_HOME\hotDeploy** directory. The **hotDeploy** directory contains the **jbossall-client.jar** file for the JBoss application server and the examples JAR file so you can run the EJB examples immediately. For more information about JBoss requirements, see Generate an EJB Baseline.

Parameter Requirements:

- Server connection information (JNDI connection) and user ID and password (if necessary).
- The global JNDI lookup name of your EJB home interface.

The EJB deployer should provide this information.

- Example
- Connecting to WebSphere with DevTest Solutions using SIBC

Example

This example is based on the demo server, which uses JBoss 4.2.3. To use your local demo server, enter **localhost** as the host name.

1. Select the application server from the list and enter the following parameters:
   - **Host Name or IP Address**
     Enter the hostname or IP address of your application server.
   - **Port Number**
     Enter the port number.
   - **User**
     Enter if a user ID is required for the application server.
   - **Password**
     Enter if a password is required for the application server.
2. If your application server is not on the list, you select the **Other / You Specify** option and enter the following parameters:

- **JNDI Factory**
  Enter or select the fully qualified JNDI factory class name for your application server.

- **JNDI 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.

3. Click **Next**.
   The **New EJB Setup** window lists the JNDI names that are registered with the application server.

Enterprise JavaBean Execution step New EJB Setup window

4. Select the name of the appropriate EJB home interface.
   In this example, the JNDI name is **com.itko.examples.ejb.UserControlBean**. The EJB 3 specification enables stateful and stateless beans to bind to the JNDI tree directly and not require a home interface. If so, the bean can be selected directly and DevTest does not need to create an instance.

5. Click **Next**.
   The object is constructed and loaded into the Complex Object Editor.

Enterprise JavaBean Execution step in COE

In the **Execution Info** area, the current EJB information is displayed. If you plan to reuse this EJB, you can keep the references to the EJB object and the EJB Home by selecting the **Keep EJB Home Reference** and **Keep EJB Object Reference** check boxes. If the bean is an EJB 3 bean without a home interface, the **Keep EJB Home Reference** check box is disabled.

6. You can now manipulate the object, and execute methods, using the Complex Object Editor.
   The usage is the same as in the RMI Server Execution step.

Connecting to WebSphere with DevTest Solutions using SIBC

IBM has an EJB and JMS client that you can download and use with the Sun JVM.

Follow these steps:

1. Download this file; then run their installer.
2. Issue the following command:

   ```
   java -jar sibc_install-o0902.06.jar jms_jndi_sun <output_directory>
   ```

3. Get the following files from your <output_directory>:
   - **lib\sibc.jms.jar**
   - **lib\sibc.jndi.jar**

- **lib\sibc.orb.jar**

4. To reference the previous three JAR files, Create a **LISA_PRE_CLASSPATH** environment variable.
   For example: LISA_PRE_CLASSPATH=C:\sibc.jms.jar;C:\sibc.jndi.jar;C:\sibc.orb.jar;

5. Edit **local.properties** and add the following line:

   ```
   com.ibm.CORBA.ORBInit=com.ibm.ws.sib.client.ORB
   ```

6. On the JMS step, use the following settings:

   - **JNDI factory class:** com.ibm.websphere.naming.WsnInitialContextFactory
   - **JNDI URL:** iiop://SERVER:PORT

Other Transaction Steps

The following steps are available in the **Other** category:

- SQL Database Execution (JDBC)
- SQL Database Execution (JDBC with Asset)
- CORBA Execution

SQL Database Execution (JDBC)

The SQL Database Execution step lets you connect to a database using JDBC (Java Database Connectivity) and make SQL queries on the database.

Full SQL syntax is supported, but your SQL is not validated. The SQL is passed through to the database where it is validated. If you get an SQL error, it is captured in the response. You can assert on the error. Verify that the SQL is valid for the database manager you are using.

Prerequisites: The JDBC driver appropriate for your database must be on the DevTest classpath. You can place the driver JAR file in the hot deploy directory. The DevTest classpath includes the Derby client driver, so you do not need to re-add it.

Parameter Requirements: Have the name of the JDBC driver class, the JDBC URL for your database, and a user ID and password for the database. You also must know the schemas for the tables in the database to construct your SQL queries.

SQL Database Execution (JDBC) Step

1. Enter the following parameters in the SQL Database step editor:
   Connection Info:

   - **JDBC Driver**
     Enter or select the full package name of the appropriate driver class. Standard driver classes are available in the drop-down list. You can also use the **Browse** button to browse the DevTest class path for the driver class.

   - **Connect String**
     The connect string is the standard JDBC URL for your database. Enter or select the URL. The JDBC URL templates for common database managers are available in the drop-down list.

   - **Max Rows to Fetch**
     Enter the maximum number of rows you want returned in the result set. This field is required. Enter -1 for unlimited rows.

   Execution Info:

- **User ID**
  Enter a user ID (if the database requires it).

- **Password**
  Enter a password (if the database requires it).

- **Keep Connection Open**
  If this option is selected, the database connection that is opened the first time that the step executes is cached. That database connection is then closed when garbage collection happens for the step. If **Keep Connection Open** is not selected, the connection is closed each time that the step executes.

- **Use Connection Pool**
  By default, if you enable connection pooling on the SQL execution step, we pool the connection that was created for 45 seconds after the test has released the connection before closing the connection to the database. If you need more than one connection during that 45-second window, then you can use the connection pool (at its absolute minimum). If you are running a load test that uses this step, you should turn on connection pooling. If you are running a test that is in a tight loop that is using this step in the loop, then enable the connection pool. If you are running a series of functional tests, and during the run of those tests, a couple of calls are made out to the database in a 10-minute window, then do not turn on connection pooling.

  in general, you should always use connection pooling unless you want to have a single connection per step execution specifically for testing the ability of the server to accept large number of connections. When using connection pooling for load tests (multi-VUs), you may need to configure the **lisa.jdbc.pool.maxPoolSize** property not to run out of connections (starvation),

- **Returns Result Set**
  Select this check box if your query results in a Result Set being returned; that is, a SELECT type query. Leave cleared for an UPDATE, INSERT, or DELETE. If this check box is set incorrectly, your query causes an error.

- **If SQL error**
  Select the step to redirect to if an error occurs.

To communicate with the local demo server, use:

- **JDBC Driver**
  org.apache.derby.jdbc.ClientDriver

- **Connect String**
  jdbc:derby://localhost:1527/reports/lisa-reports.db

- **User ID**
  sa

- **Password**
  sa

2. After you have entered the database connection information, including the user ID and password (if necessary), click **Test Connection** to test your connection.
   If the information is correct, a success message in a window appears. Otherwise an error message appears.

3. You are now ready to enter your SQL statement in the lower window.
   Properties can be used in your SQL. DevTest substitutes the parameter before passing the SQL string to the database.
   The JDBC step supports stored procedure calls. Basic data types (strings, numbers, dates, Boolean) are supported as arguments that a stored procedure uses as input and returns.
   Click **Add** to add a parameter. You cannot edit the numbers in the **Parameter** column. As you add, delete, and move rows, the numbers in the **Parameter** column are automatically renumbered.
   The JDBC step can also use JDBC prepared statements. You can use question marks in a SQL statement and can add named {{properties}}. This ability allows you to avoid being concerned about the type of the argument or escaping single quotation marks in parameter values. A statement "insert into MYTABLE(COL1,COL2) values (?, ?)" with a reference to {{col1}} and {{col2}} is easier to understand. The type and escape characters are automatically converted.
   To include a null in your statement, use this syntax: **{{<<NULL>>}}**.

4. After you have created the SQL query, click **Test/Execute SQL** to execute the query.
   A message indicates the result status.

5. Click **OK**.
   Your results display in the **Result Set** tab.

6. You are now ready to create filters and assertions on the result set.
   The icons on the bottom of the **Result Set** tab provide 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 the cell value in the search field is found, the value in the **Value** field in that row is set as the value of the property that is entered.

   - **Parse Result for Value filter**
     The value in the selected cell is set as the value of the property that is entered.

   - **Result Set Contents Assertion**
     The value in the chosen field (column) is compared to the regular expression that is entered.

   For more information about these and other filters and assertions appropriate for result sets, see Filter Descriptions and Assertion Descriptions.

The SQL Database Execution step has a default name using this convention: *JDBC SQLfunction tablename*. The supported functions in step name defaults are *select*, *insert*, *delete*, *update*, and *perform*. You can change step names at any time.

SQL Database Execution (JDBC with Asset)

The SQL Database Execution step lets you connect to a database using JDBC (Java Database Connectivity) and make SQL queries on the database. Use this step when you have a JDBC connection asset defined.

> **Note:** If you have a legacy SQL Database Execution (JDBC) step, you can export the connection information from the existing step instead of creating it manually. For more information see Create Assets from Test Steps.

Full SQL syntax is supported, but your SQL is not validated. The SQL is passed through to the database where it is validated. If you get an SQL error, it is captured in the response. You can assert on the error. Verify that the SQL is valid for the database manager you are using.

SQL Execution step (JDBC with Connection Asset)

1. Enter the following parameters in the SQL Database step editor:
   Connection Info:

   - **Connection Asset**
     Select a connection asset that contains the connection parameters for the JDBC connection.

   - **Max Rows to Fetch**
     Enter the maximum number of rows you want returned in the result set. This field is required. Enter -1 for unlimited rows.

   Execution Info:

   - **Returns Result Set**
     Select this check box if your query results in a Result Set being returned; that is, a SELECT type query. Leave cleared for an UPDATE, INSERT, or DELETE. If this check box is set incorrectly, your query causes an error.

   - **If SQL error**
     Select the step to redirect to if an error occurs.

2. After you have entered the database connection information, including the user ID and password (if necessary), click **Test Connection** to test your connection.
   If the information is correct, a success message in a window appears. Otherwise an error message appears.

3. You are now ready to enter your SQL statement in the lower window.
   Properties can be used in your SQL. DevTest substitutes the parameter before passing the SQL string to the database.
   The JDBC step supports stored procedure calls. Basic data types (strings, numbers, dates, Boolean) are supported as arguments that a stored procedure uses as input and returns.
   Click **Add** to add a parameter. You cannot edit the numbers in the **Parameter** column. As you add, delete, and move rows, the numbers in the **Parameter** column are automatically renumbered.
   The JDBC step can also use JDBC prepared statements. You can use question marks in a SQL statement and can add named {{properties}}. This ability allows you to avoid being concerned about the type of the argument or escaping single quotation marks in parameter values. A statement "insert into MYTABLE(COL1,COL2) values (?, ?)" with a reference to {{col1}} and {{col2}} is easier to understand. The type and escape characters are automatically converted.
   To include a null in your statement, use this syntax: **{{<<NULL>>}}**.

4. After you have created the SQL query, click **Test/Execute SQL** to execute the query.
   A message indicates the result status.

5. Click **OK**.
   Your results display in the **Result Set** tab.

6. You are now ready to create filters and assertions on the result set.
   The three icons on the bottom of the **Result Set** tab provide 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 the cell value in the search field is found, the value in the **Value** field in that row is set as the value of the property that is entered.

   - **Parse Result for Value filter**
     The value in the selected cell is set as the value of the property that is entered.

   - **Result Set Contents Assertion**
     The value in the selected field (column) is compared to the regular expression that is entered.

   For more information about these and other filters and assertions appropriate for result sets, see Filter Descriptions and Assertion Descriptions.

The SQL Database Execution step has a default name using this convention: *JDBC with Connection Asset SQLfunction tablename*. The supported functions in step name defaults are *select*, *insert*, *delete*, *update*, and *perform*. You can change step names at any time.

CORBA Execution

The CORBA Execution step is used to make CORBA calls using the Java RMI-IIOP library. You are required to provide appropriate skeleton classes.

Follow these steps:

1. Copy the **corbaserver.jar** file to the DevTest lib directory before starting execution. This JAR is available in the CORBA server lib directory.

2. To open the step editor, select the CORBA step.

3. Complete the fields.

   - **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.

     > **Note:** Copying and pasting from nameserver printed output to the Object IOR field can introduce spaces. The string must not contain spaces.

   - **Class Name**
     IOR references this object class.

   You can also use the IOR construction dialog. When open, it parses any IOR string that is entered and fills in the individual parts. Ignore the strange looking key. IORs store the key in a byte format so not every byte can be displayed properly. To use the parsed version of a raw IOR, do not edit the field.

4. Click **Construct/Load Object**.
   The dynamic object editor shows the call sheet for the object.

5. Select the method to call and execute it.

The default CORBA Execution step uses the following convention: *CORBA classname*. You can change step names at any time.

Utilities Steps

The following steps are available in the **Utilities** category:

- Save Property as Last Response
- Output Log Message
- Write to Delimited File
- Read Properties from a 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
- Send Email

Save Property as Last Response

The Save Property as Last Response step lets you save the value of a property as the last response.

Enter the property name of an existing property or select it from the pull-down menu. The value of the property is loaded as the last response (and the step response). The value 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 the current step.

Each step in a test case has a response that is associated with it. When that step runs, the response is automatically saved in two properties: **LASTRESPONSE** and **lisa.thisStepName.rsp**. Use this step so you can have filters and assertions apply to a property value instead of the real response of the step.

Output Log Message

The Output Log Message step lets you write a text message to the log. This step is useful for tracking and logging test cases as they execute. Your log message is usually a combination of text and properties.

Enter your log message in the editor. This log message appears when this step executes in the Interactive Test Run (ITR), and the message is logged during a test run.

Write to Delimited File

The Write to Delimited File step lets you save the current value of some properties in a CSV file. This step is commonly used when properties are shared among multiple test cases, or for debugging purposes when a test has failed. The properties can be existing properties or new properties. You can save existing properties under a different property name.

Complete the following fields:

- **File Name**
  Enter the path name of the file, or use the **Browse** button to browse to the file.

- **Encoding**
  Accept the default encoding of UTF-8 or select an alternate encoding from the drop-down list. If you select **Platform Default**, the JVM chooses an encoding, which can change across machines and even across JVMs on the same machine.
  We recommend that you select a specific character encoding, such as **UTF-8**.

- **Include BOM**
  If the encoding you select includes a Byte Order Mark, you can select this check box to include that BOM at the beginning of the file.

- **Delimiter**
  Enter the delimiter character to use for your file.

- **Line end**
  Select from the drop-down list the correct line end character for your file.

- **Properties to write**
  Click **Add** to add a row. Then enter the properties (Header) and corresponding values (Value) to save. Your list of properties can contain existing or new properties. When specifying existing properties (Header), you can override their current value by specifying a new value, or you can use their existing value.

In the previous example:

- The first two properties are being stored without change

- The third property was stored under a new name (Header), Date
- The fourth property is using Reg Expression as its value

The properties are stored in a CSV file where the first line in the file is the set of property names being saved. The second line contains the values corresponding to each of the properties.

Note: This step is used to pass data between test cases. For example, assume Test1 adds customers to the bank and then returns a list of new account numbers. Those accounts could be written out to a file. A second test could get the accounts and could make deposits. The second test would use a data set to read in the file that was created in the first test.
Be aware of the following warning when writing to a delimited file. The Data directory can be used as the location of the CSV file only if both tests support rerun. The first test must be run again to create the CSV in the **lads** folder so the second test can run. The **lads** directory stores files temporarily while a test case or suite is running.
To let the second test run without the first, you must put the data set in a common location outside the project or MAR.

The Write to Delimited File step writes out one (and only one) data row for each execution. The only exception is on the first execution when (presumably) the target file does not exist. If the file does not exist, if indicated, a byte order mark for the selected encoding is written, followed by a row containing the keys.

A delimited file is an SV (comma-separated value) file. CSV files use double-quote marks to delimit field values that have spaces, so a value like **Santa Clau**s gets saved as “Santa Claus” to preserve the space. If a field value contains double-quotes, then the double-quotes get doubled up so the parser can tell the difference. For example, if the CSV field is “Santa Claus” the returned value is **Santa Claus**. If the CSV field is “”Santa Claus””, the returned value is **“Santa Claus” –** the double-quote marks are *part* of the value, not only a delimiter around the value.

In JSON, fields have quotations, such as **{ “firstName” : “Santa” }** . As such, the quotation marks are part of the value (from a CSV file’s point-of-view), so these get saved as “”firstName””,””Santa””.

The Save to Delimited File step has no idea the content is JSON; it just sees quotes and presumes the quotes are part of the field’s value and wraps the quotes in quotes accordingly.

One idea to get around this is to Base64-encode the JSON and save the Base64 value to a delimited file. When reading the file back, after reading a record, Base64-decode the value to get back the original JSON. To read the JSON back as a data set, that is one approach. To save the JSON to a file, save it to a text file (not delimited file).

The default name for the Write to Delimited File step is *Write Properties to File <file>*.

If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

Read Properties from a File

The Read Properties from a File step is used to read the properties from an external file.

You can read the properties in two ways:

- In name/value pairs
- In XML tags

Complete the following fields:

- **File Name**
  Enter the path name of the file, or use the **Browse** button to browse to the file.
- **File Encoding**
  Accept the default encoding of UTF-8 or select an alternate encoding from the drop-down list. You can also select **Auto-detect** and click **Detect** to have DevTest select an encoding type for you. If you select **Platform Default**, the JVM chooses an encoding, which can change across machines and even across JVMs on the same machine.
  We recommend that you select a specific character encoding, such as **UTF-8**.
- **Type of File**
  Select the type of a file; either **Name-Value-Pairs** or **XML Tags**.

Notepad file as input to Read Properties from File step

The previous graphic shows the Name/Value pair type of properties file, where *name* is a property and *Mary* is a value of the name property.

XML file as input to Read Properties from File step

The previous graphic shows the XML Tags type of properties file, where *fname* is a property and *Sam* is a value of the fname property.

The Read Properties from a File step has a default name using this convention: *Properties Reader from <filename>* where *filename* is the leaf name of the entered or selected file.

If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

Do-Nothing Step

The Do-Nothing step does not take any parameters, nor does it perform any actions. However, the step 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 compare numbers or dates, or some numerical comparison test.

Parse Text as Response

The Parse Text as Response step lets you enter textual content from a file that can be saved as the last response. The content can be stored in a property. You can type or paste the text into the editor, or you can load it from a file.

Complete the following fields:

- **Property Key**
  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 **Test** to show the resulting text.

Audit Step

The Audit step lets you apply an audit document to a current test step, remote test, or virtual service.

This step allows a model to be verified in terms of the events it produces during execution.

To open its editor, click the step.

- **Mode**
  This step has two modes:

  - **Start Monitoring**

  - **Apply Audit Document**.

Start Monitoring Mode

If you select the **Start Monitoring** Mode:

- **Audit Document**
  Enter or browse for the audit document.

- **Target**
  Select the target for the audit document:

  - **This Model:** Applies to the current model.

  - **Test Run:** Applies to the test run.
  - **Virtual Model:** Applies to the virtual model.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

Apply Audit Document Mode

If you select the **Apply Audit Document** mode:

- **If Audit Fails**
  Select the step to run if the audit fails.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

Base64 Encoder Step

The Base64 Encoder step is used to encode a file using Base 64 encoding algorithm.

The result can be stored in a property file to be used anywhere in the test run.

To open its editor, click the step.

Complete the following fields:

- **File**
  Enter the name of file to be encoded or browse to the target location.

- **Property Key**
  The name of the property to store the encoded file. This field is optional.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

Click **Load** to load the file in the editor.

Here you can apply filters or assertions, or both, if necessary, from the **Command** menu at the bottom.

Checksum Step

The Checksum step calculates the checksum of a file and can save that value in a property.

Complete the following fields:

- **File**
  Enter the pathname or browse to the file on which you want the checksum to be calculated.

- **Property key**
  Enter the name of the property that stores the checksum value.

Click **Load**.

The checksum is displayed as the response, and if a property name was entered the value is in that property.

The checksum value is now available for you to filter and add assertions.

Convert XML to Element Object

The Convert XML to Element Object step converts raw XML into an object of one of the following types:

- Message Element Array

- Message Element

- DOM Element

This step is useful when you have a web service API that takes any type using strict processing. This type of WSDL element requires 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 property. That property becomes an input parameter for this step.

Prerequisites: The XML must be already stored in a property.

Complete the following fields:

- **Load XML from Property**
  Enter the property that contains the XML. This property can be a user-defined property or a built-in DevTest property.

- **Treat as just Text**
  Select this check box if you must use plain text as your input instead of XML. This selection results in a message element that contains plain text.

Select the type of object you want from the available types by clicking the respective option button.

Click **Test** to do the conversion.

If you select the **Treat as Just Text** check box, you see the results as seen in the following graphic. The **Create Message Element**, **Create DOM Element**, and **Deserialize a WS Object of Type** options are dimmed.

Convert XML to Element Object step

Use the response from this step when this object is required as a parameter in another step. To save the response in a filter, use the Save Step Response as a Property filter. Or, you can refer to it as **lisa.<stepname>.rsp**.

Compare Strings for Response Lookup

The Compare Strings for Response Lookup step is used to inspect an incoming request to a virtual service and determine the appropriate response. You can match the incoming requests using partial text match, regular expression, and others.

This step is automatically filled out and added to a virtual service when using the Virtual Web Service HTTP Recorder.

To open a step editor, click the step.

Complete the following fields:

- **Text to match**
  Enter the text against which criteria is matched. This value is typically a property reference, such as LASTRESPONSE.

- **Range to match**
  Enter the **Start** and **End** of the range.

- **If no match found**
  Select the step to be executed, if no match is found.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **Store responses in a compressed form in the test case file**
  Selected by default. This option compresses the responses in the test case file.

- **Case Response Entries**
  In this table you can add, move, and delete entries by clicking **Add**, **Move**, and **Delete**. The columns in the table are as follows:

  - **Enabled**
    Selected by default when you add an entry. Clear 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 from 1000 through 10000 milliseconds. (The syntax is the same format as Think Time specifications.)

  - **Criteria**
    This area provides the response of this step if the entry matches the **Text to match** field. To edit the criteria, in the **Criteria Column** area select the appropriate row, and enter a different setting in the criteria list. Click **Enter** to update it in the subsequent row.

  - **Compare Type**
    Select an option for the Compare type from the list:

    - Find in string (default)

    - Regular expression

    - Starts with

    - Ends with

    - Exactly equals

  - **Response**
    Allows for the update of the step response for an entry.

- **Criteria**
  Updates the criteria string for an entry.

- **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 enter a different setting in the **Response** list. To get it updated in the previous row, click **Enter**.

Compare Strings for Next Step Lookup

The Compare Strings for Next Step Lookup step is used to inspect an incoming request and determine the appropriate next step. You can match incoming requests using partial text match, regular expression, and others. Each matching criterion specifies the name of the step to which to transfer if the match succeeds.

This step is automatically filled out and matches to a Virtual Service when using the JDBC Database Traffic Recorder.

To open a step editor, click the step.

Enter the following parameters:

- **Text to match**
  Enter the text against which criteria is matched. This value is typically a property reference, such as LASTRESPONSE.

- **Range to match**
  Enter the **Start** and **End** of the range.

- **If no match found**
  Select the step to be executed, if no match is found.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **Next Step Entries**
  Click **Add** to add an entry. Use **Move** and **Delete** to move or delete an entry. Enter text in the **Find** field to find an entry.
  The columns in the **Next Step Entry** are:

  - **Enabled**
    Selected by default when you add an entry. Clear 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 from 1000 through 10000 milliseconds. The syntax is the same format as Think Time specifications.

  - **Criteria**
    This area defines the criteria to compare to the **Text to match** field.

- **Compare Type**
  Select from five options:
  - Find in string (default)
  - Regular expression
  - Starts with
  - Ends with
  - Exactly equals

- **Next Step**
  The step name.

- **Criteria**
  Updates the criteria string for an entry.

Send Email

The Send Email step lets you send an email notification from a test case.

Complete the following fields:

- **Connection**
  Specifies the asset that contains the connection parameters to the SMTP mail server. Select a connection from the drop-down list. To add an asset, click **Add New Asset**.

- **From**
  Identifies the email sender. Enter an email address or a property.

- **To**
  Identifies the email recipient. To send to multiple recipients, use a comma between email addresses. You can also use a property or list of properties.
  Example:

  ```
  {{Ops_email}},{{QA_email}},InfoTechnology@company.com
  ```

- **Cc**
  Identifies the secondary email recipient. To send to multiple recipients, use a comma between email addresses. You can also use a property or list of properties.

- **Bcc**
  Identifies the tertiary email recipient. To send to multiple recipients, use a comma between email addresses. The tertiary recipients are not identified to other recipients. You can also use a property or list of properties.

- **Subject**
  Specifies the subject of the email. You can use a property or multiple properties in the **Subject** field.

- **Message**
  Specifies the body of the e-mail with HTML support. If you use a buffered image as a property in the email body, the image is embedded in the message. You can use a property or multiple properties in the **Message** field. For the Selenium Integration step, you can use the **selenium.last.screenshot** property to embed a screenshot from the Selenium Integration step in the email message.

- **Attachments**
  Lists the attachments to the email. Click **Add** , then **Browse** to locate the path and file name of the file to be attached to the email.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test

To test the email connection, click **Send Test Email**.

The Send Email step has a default name using this convention: *Send Email Step*.
If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

External-Subprocess Steps

The following steps are available in the **External/Subprocess** category:

- Execute External Command
- File System Snapshot
- Execute Subprocess
- Execute JUnit Test Case-Suite
- Read a File (Disk URL or Classpath)
- External - FTP Step

Execute External Command

The Execute External Command step lets you run an external program (such as an operating system script, an operating system command, or an executable) and capture its contents for filtering or making assertions.

The syntax of the external program depends on your operating system.

Enter the following parameters:

- **Execute from directory**
  The directory that is considered current when the external command is run. DevTest creates the directory (subject to file system permissions) if it does not exist on the system that is running the test. If the directory does not exist and cannot be created, the step fails.

- **Time Out (Seconds)**
  How long to wait before transferring to the step defined by **On Time Out Execute**.

- **If timeout**
  The step to run if the external command does not complete execution before the timeout value.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **Output Encoding**
  Accept the default encoding of UTF-8 or select an alternate encoding from the drop-down list. You can also select **Auto-detect** to have an encoding type be selected for you. If you select **Platform Default**, the JVM chooses an encoding, which can change across machines and even across JVMs on the same machine.
  We recommend that you select a specific character encoding, such as **UTF-8**.

- **Allow Properties**
  This check box determines whether properties are allowed for the next four parameters. This option changes the look of the command editor interface.
  With the **Allow Properties** check box cleared, five check boxes are available. Here the only choice is to select the parameter or not.

- **Wait for Completion**
  If you select this check box, the step waits until the execution finishes to filter or assert on the results. If this check box is cleared, filters and assertions execute; however execution does not wait for the result of the command being run.

- **Kill at Test End**
  If the **Wait for Completion** check box is cleared, select this check box to kill the process after the test case finishes. This parameter lets a process run while a test case runs, then shuts down the process. A property contains the process ID of the started command.

- **Spawn Process**
  Creates a process in the operating system in which to run the command. This parameter is helpful in the following cases:

  - You want a long-running background process

  - You must ensure that a new set of environment variables is set and your environment contains nothing that DevTest set

- **Exec Shell**
  Lets you run the contents of the command line in a system shell. This option is required if you must use shell process features 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 such as dir or ls. This check box must be selected for a Windows operating system.

- **Append to Environment**
  When the step defines environment variables, DevTest appends them to the existing environment, instead of creating an environment that defines only these variables.

With the **Allow properties** check box selected, there are pull-down menus that have the same functionality as shown previously, but each parameter can now be a property.

Execute External Command step

- **Command Line**
  The external command is typically only one command that is written as a shell script or batch file. You can run multiple commands when the **Exec Shell** option is also selected. The command string must be valid for the operating system that you are running.

- **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 you define an environment variable, the new variable sets are used instead of the environment variables that were used to start DevTest.

- **Exit Codes**
  Lets you base the outcome of the test on the exit code of the completed process. Enter a comma-delimited string of exit codes with a corresponding step to run when the process exits with this code.

To test your command, click **Execute**.

The content is now available for you to filter and add assertions.

The Execute External Command step has a default name using this convention: *Command commandfirstword*.

If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

File System Snapshot

The File System Snapshot step lets you 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.

To open a step editor, click the step.

Enter the following parameters:

- **Execute from directory**
  Enter the path name or browse to the file or directory.

- **Recurse Subdirectories**
  Select if you want the complete directory tree including sub directories.

- **Include File Size**
  Select if you want the file sizes to be listed.

- **Include Date/Time**
  Select if you want the last modified date to be listed.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

Click **Execute Now** to run and start scanning the file system.

The content is now available for you to filter and add assertions.

Execute Subprocess

The Execute Subprocess step lets you call a subprocess test case as a single step.

This step is used to call a subprocess and receive the outputs. This step is commonly used when a specific function is performed in numerous test cases. For example, if a specific validation always works the same way, you can create a validation subprocess and can add it to different test cases.

For more information, see Building Subprocesses.

Complete the following fields:

- **Subprocess**
  Select the subprocess from the pull-down menu.

- **Fully Expand Props**
  When the parameters have nested properties, fully expand the properties before sending to the subprocess.

- **Send HTTP Cookies**
  Select to forward cookies to the subprocess.

- **Get HTTP Cookies**
  Get the HTTP cookies from the subprocess.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

The documentation that was entered into the **Documentation** area of the subprocess is displayed.

The **Parameters to Subprocess** panel contains a list of the parameters that the subprocess requires. These keys and values must be present in your current test case. Edit the **Value** column as necessary to supply the correct values.

The **Result Properties** panel lists all the properties that are produced in the subprocess. Select the properties to be returned from the subprocess. These properties are used in your test case. You are not limited to a single return value.

When this step runs, it appears to run as a single step. The Interactive Test Run (ITR) utility shows the events that fired while the step ran. The short name of the events combines the name of the current step and the name of the subprocess step where the event was fired..

If an assertion is triggered in the subprocess, then the appropriate event appears in the **Test Events** tab of the Interactive Test Run (ITR) utility.

The default Execute Subprocess step uses the following convention: *Subprocess subprocessname*. You can change step names at any time.

Execute JUnit Test Case-Suite

The Execute JUnit Test Case/Suite step lets you run a JUnit test case or a JUnit test suite in a DevTest step. If the JUnit test passes, then so does the test step. After a failure, you can redirect to another test step.

Prerequisite: Your JUnit test must be on the classpath. Drop it into the hot deploy directory.

Complete the following fields:

- **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 technique also confirms that your class is on your classpath.

- **If environment error**
  Select the step to redirect to if the JUnit test fails.

Click **Load** to load the class files. The class tree is displayed in the left panel.

Click **Execute** to run the JUnit test. The standard JUnit results are displayed in the right panel.

The JUnit Test Case/Suite step has a default name using this convention: *Junit Test Case Suite testclassname*.

If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

Read a File (Disk URL or Classpath)

The Read a File (Disk, URL, or Classpath) step reads a file from your file system, a URL, or the classpath.

Files are used as a source of data for testing. This step can be paired with the Load a set of File Names data set to provide source data for testing.

You can read a text file or a binary file. The contents of the file can optionally be stored in a property.

Complete the following fields:

- **File**
  Enter the path name, a URL, or a classpath, or browse to the file using the **Browse** button.

- **File Encoding**
  Accept the default encoding of UTF-8 or select an alternate encoding from the drop-down list. You can also select **Auto-detect** and click **Detect** to have DevTest select an encoding type for you. If you select **Platform Default**, the JVM chooses an encoding, which can change across machines and even across JVMs on the same machine.
  We recommend that you select a specific character encoding, such as **UTF-8**.

- **Property Key**
  Enter the name of the property in which to store the file contents (optional).

- **Load as Byte []**
  To load contents as a byte array, select this check box. This capability is useful when loading a file to be used as a binaryData type in a web service execution parameter.

- **Cache file in memory**
  To avoid memory leaks, adjust the size and time duration to specify when the cache is flushed. The maximum file size has a default of 20 MB, but you can adjust the size with the **FileNode.cache.max.size.mb** property.
  If a cached entry is inactive for 30 minutes, that entry is removed from cache. The default is 30 minutes but you can adjust the **FileNode.cache.duration.mins** to change the time.

- **If environment error**
  Select the step to redirect to if the Read a File test fails.

- **Display as characters**
  The contents are displayed as hexadecimal encoded bytes unless you select this check box. This check box is only visible if the **Load as Byte []** box is selected.

Click **Load** to load and display the file. The content is now ready for you to filter and add assertions.

> **Note:** If a binary file is loaded but you do not select to load as byte, DevTest converts the data to characters (many of which are unreadable) and the step response is a string.

The Read a File step has a default name using this convention: *Read file [set the File Name variable]*.

If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

External - FTP Step

The FTP step lets you send or receive a file using the FTP protocol. After entering the FTP information, user name, and user password, you can either upload a file or download a file.

You can drag the FTP step editor window to the left to display the **Execute Now** button.

Complete the following fields:

- **Host**
  Enter the host name of the FTP server (without the protocol).

- **Port**
  Enter the port for FTP server access. A port is optional; the default port is 21.

- **User**
  Enter the user ID for FTP server access.

- **Password**
  Enter the password for FTP server access.

- **Direction**
  Indicate if the data flow is an upload or a download.

- **Mode**
  Specify passive or active FTP, or enter a property that indicates passive or active.

- **Transfer Type**
  Select the file transfer type; Binary or ASCII, or enter a property that indicates binary or ASCII.

- **Host Path**
  Enter the path to the source file (either on the FTP server or local computer).

- **Local Path**
  Enter the path for the destination file (either on the FTP server or local computer).

  > **Note: Host Path** is the path of the file on the remote computer. **Local Path** is always the path to the file on the local computer that is running DevTest. When you upload a file, you are uploading the file from the **Local Path** to the **Host Path**. When you download a file, you are downloading the file from the **Host Path** to the **Local Path**.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

Click **Execute Now** to initiate the send/receive action.

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 exists, it is overwritten without a warning message.

The FTP step has a default name using this convention: *FTP action (put or get) hostname*; for example, **FTP get ftp.download.com**. You can change step names at any time.

JMS Messaging Steps

The following steps are available in the **JMS Messaging** category:

- JMS Messaging (JNDI) (Deprecated)
- JMS Messaging - Message Consumer (Deprecated)
- JMS Send Receive Step
- Tutorial - Send and Receive a JMS Message

JMS Messaging (JNDI) (Deprecated)

Note: This step has been deprecated. Refer to the following for creating new tests and virtual services:

- JMS Client Assets
- JMS Send Receive Step

The JMS Messaging (JNDI) step lets you send messages to, and receive messages from, topics and queues. You can also receive, change, and forward an existing message. The list of possible queues and topics can be browsed using JNDI. Provide client libraries where DevTest can read them.

All the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended) are supported.

The JMS Messaging (JNDI) step is configured using a single editor regardless of the messaging requirements. The input options vary on the messaging requirements. The editor only allows valid configurations, so when you enable some features, others could become inactive.

The JMS Messaging (JNDI) step has a default name using this convention: *JMS publish-queuename publish*. If there is not a publish queue name, the default step name is *JMS subscribe-queuename subscribe*.

If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Parameter Requirements: This step requires the connection parameters and the queue or topic names that are used in the application under test. Other parameters could be required, depending on your environment. Get these parameters from the application developers. In most cases, you can browse for server resources to get some of these required parameters.

The **jms.tst** test case in the examples project shows the step that this section describes.

The **jms.tst** test case uses a publish/subscribe JMS step to send a message and listen on a temporary queue. A Message Driven Bean (MDB) on the server handles the message and drops the message onto the temporary queue. The message type is text. The message is an XML payload that is created by inserting properties dynamically into the XML elements. The properties

are read from the **order_data** data set. After the response message is received, the XML from the JMS message is put into a property. The next step does an assertion validating the order ID. After this check asserts true, the existing message object is changed, and the message is sent to another JMS destination.

The **jms.tst** test case shows how to listen for and intercept messages as they move through a multipoint messaging service backbone. You can run this test case against the demo server on your computer. The application backend is available there.

The editor for the JMS Messaging (JNDI) step contains the following tabs:

- The **Base** tab is where you define the connection and messaging parameters.
- The **Selector Query** tab lets you specify a selector query to be run when listening for a message on a queue.
- The **Send Message Data** tab is where you create the message content.
- The **Response Message** tab is where the response messages are posted.

- Base Tab
- Selector Query Tab
- Send Message Data Tab
- Response Message Tab

Base Tab

The **Base** tab is where you define the connection and messaging parameters.

The following graphic shows the **Base** tab. The tab is divided into the following sections:

- **Server Connection Info**
- **Subscriber Info**
- **ReplyTo Info**
- **Publisher Info**
- **Error Handling and Test**

JMS Messaging (JNDI) step Base Tab

To enable and disable the **Subscriber Info**, **Publisher Info**, and **ReplyTo** Info, use the enable check box in the top left corner of each section. This option allows you to configure the step to be a publish step, a subscribe step, or both. You can also select to include a JMS reply to component in the step.

When you finish configuring the test step, click **Test** in the **Error Handling and Test** section to test the configuration settings.

Server Connection Info

Enter the JNDI information in the **Server Connection Info** section of the **Base** tab.

To simplify changing the application under test, parameterize these values with properties that are in your configuration. The previous graphic shows an example of this approach.

The following parameters are available to you for the system under test:

- **JNDI Factory Class**
  The fully qualified class name of the context factory for the JNDI provider.

- **JNDI Server URL**
  The URL for connecting to the JNDI server. The format of the URL depends on the specific JNDI provider being used.

- **JMS Connection Factory**

  Use **Search** to browse available resources on the server. Select or enter a connection factory to use for this step execution according to the JMS specification.

The pull-down menus contain common examples or templates for these values.

The user and password could be optional.

- **User**
  The user name for connecting to the JNDI provider and getting a handle to the connection factory.

- **Password**
  The password for connecting to the JNDI provider and getting a handle to the connection factory.

- **Share Sessions and Share Publishers**

  To share JMS sessions and publishers throughout the test case, use these check boxes. This approach can lower overhead, but does not always provide a realistic simulation because typically JMS clients want to release resources. If you select the **Share Publishers** check box, the **Share Sessions** check box is also selected. You cannot share publishers without sharing sessions. For more detailed information about these parameters, see the Deliberate Delays in VSE knowledge base article.

- **Stop All**

  Lets you stop any listeners at design time now. Some listeners can get orphaned, but still consume messages. When they do, it is difficult to create test cases.

- **Advanced**
  Displays a panel where you can add custom properties that are sent with the connection information and you can configure the second-level authentication.

> **Note:** The **Server Connection Info** user and password fields are for connecting to the JNDI provider and getting a handle to the connection factory. The user and password fields in the **Second Level Authentication** tab are for getting a handle to the actual JMS connection.

Publisher Info

To set up the ability to send messages, select the **enable** check box.

To execute a commit when the message is sent, select the **use transaction** check box.

Enter the following parameters:

- **Name**

  The name of the topic or queue. Use **Search** to browse the JNDI server for the topic or queue name.

- **Type**

Select whether you are using a topic or queue. To see what messages are waiting to be consumed from a queue (only), use **Browse** to the right of this field.

- **Message**
  Select the type of message you are sending. The supported types are Empty, Text, Object, Bytes, Message, and Mapped (Extended).

- **Advanced**
  Displays a panel where you can edit the message headers and can add message properties.

Subscriber Info

To set up the ability to receive messages, select the **enable** check box.

Enter the following parameters:

- **Name**

  The name of the topic or queue. Use **Search** to browse the JNDI server for the topic or queue name.

- **Type**
  Select whether you are using a topic or queue, and whether to listen in synchronous or asynchronous mode. For asynchronous mode, you also must have an entry in the **Async Key** field. To see what messages are waiting to be consumed from a queue, click **Browse** to the right of this field.

- **Timeout (secs)**
  The period to wait before there is an interrupt waiting for a message. Use 0 for no timeout.

- **Async Key**
  The value that is necessary for identifying asynchronous messages. This field is necessary only in asynchronous mode. The field is 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 lets you receive all of your messages from a topic even if you log out and then you log in again.

- **Session Mode**
  The options available are:

  - **Auto Acknowledge:** The JMS client libraries acknowledge the JMS Messages as soon as they are received.

  - **Client Acknowledge:** The JMS client must explicitly acknowledge the JMS Messages.

  - **Use Transaction:** The JMS Session operates under a transaction. The acknowledge mode is ignored.

  - **Auto (Duplicates Okay):** The JMS client library automatically acknowledges at unknown intervals. As a result, duplicate messages could be received if the automatic acknowledgment does not arrive before the JMS Provider retries the delivery.

Auto Acknowledge and **Client Acknowledge** and **Auto (Duplicates Only)** have no practical differences. With **Client Acknowledge**, each received message is acknowledged immediately upon receipt. The only difference is that the acknowledge call is made explicitly instead of letting the JMS client library do it. With **Auto (Duplicates Only)**, the behavior is the same as **Auto Acknowledge** except under high loads.

The **Use Transaction** option is not strictly an acknowledgment mode setting. The option is included in the list for two reasons:

- If the JMS Session is operating under a transaction, then the acknowledgment modes are ignored. Messages are acknowledged by committing the session transaction.
- The **Use Transaction** mode is still a way of controlling guaranteed delivery of messages from JMS. If a message is received and the session transaction is not committed, the message is resent, as if it was not acknowledged.
  - **Use temporary queue/topic**
    If you want the JMS provider to set up a temporary queue/topic on your behalf, select this check box. When a temporary queue/topic is used, DevTest automatically sets the J **MS ReplyTo** parameter of the message you send to the temporary queue/topic. The temporary queue/topic feature must always be used with a publisher so that a reply can be sent. If you use a temporary queue/topic, the **ReplyTo** section is disabled.
  - **Make payload last response**
    To make the payload response the last response, select this check box.

ReplyTo Info

To set up a destination queue/topic, select the **enable** check box.

If your application requires a destination, it is set up in this section.

Enter the following parameters:

- **Name**

  The name of the topic or queue. Use **Search** to browse the JNDI server for the topic or queue name.

- **Type**

  Select whether you are using a topic or queue. To see what messages are waiting to be consumed from a queue (only), use **Browse** to the right of this field.

Error Handling and Test

If an error occurs, the **Error Handling and Test** section lets you redirect to a step.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

Click **Test** to test your step configuration settings.

Selector Query Tab

You can enter a JMS selector query in the **Selector Query** tab. The syntax closely follows SQL. The query is a subset of SQL92.

A JMS selector query can be specified when listening for a message on a queue that is a response to a published message.

The following graphic shows a query looking for a **JMSCorrelationID** that matches the **lisa.jms.correlation.id** property as sent with the original message.

JMS Messaging (JNDI) step Selector Query Tab

Send Message Data Tab

If your step is configured to publish, you compose the message in the **Send Message Data** tab.

You can type the text, or you can click **Read Message from File** to read from a file. You can also store text in a property, in which case you would place the property in the editor, for example, **{{property_name}}**.

The following graphic shows an XML fragment with properties. Using the properties allows the message to be created dynamically during the test run.

JMS Messaging (JNDI) step Send Message Data Tab

Response Message Tab

If your step is configured to subscribe, the response appears in the Response Message tab after you click **Test** in the Base tab.

The tab shows the Complex Object Editor for the returned object. The returned object varies with the type of application server. You have access to all the JMS parameters returned in addition to the message itself. The object is loaded into the Complex Object Editor, where it can be manipulated like any other Java object.

The following graphic shows a text response from a JBoss object.

JMS Messaging (JNDI) step Response Message Tab

JMS Messaging - Message Consumer (Deprecated)

Note: This step has been deprecated. Refer to the following for creating new tests and virtual services:

- JMS Client Assets
- JMS Send Receive Step

The Message Consumer step lets you consume asynchronous messages in a test case. This step can attach to a known queue or topic and can get messages posted for this subscriber. You identify yourself with a unique key. You must have already subscribed to the queue or topic and the messages were pushed to that destination.

Prerequisite: You must have the demo server running before you execute the example test case.

Parameter Requirements: Knowledge of the queue or topic being used in the application under test.

The Message Consumer step has a default name using this convention: *Listen on subscribe-queuename*. Before the queuename is entered, the default step name is Async JMS.

If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

Review the **async-consumer-jms.tst** test case in the examples project to see what is being described in this section. The following graphic shows the **async-consumer-jms.tst** test case.

async-consumer-jms.tst test case

The **create-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). The number of messages to be published is controlled using a data set (counterA). The message consumer step has an Async queue (EXAMPLE-ASYNC-WRAPPER) using which message subscribed by the create-consumer step is consumed. The number of messages to be consumed is controlled using the data set (DataSetB).

Note: The **Async Key** specified in create-consumer and consumer steps must match.

The following graphic shows an example of the subscriber section of a step.

JMS Messaging - Message Consumer step Subscriber Info

To set up and enable the ability to listen to (subscribe to) messages, select the **enable** check box.

Notice that an asynchronous topic is specified in the **Type** field, and an **Async Key** parameter is defined. This key is necessary as an input in the current step.

The following graphic shows an example of the publisher section of a step.

JMS Messaging - Message Consumer step Publisher Info

To set up the ability to send (publish) messages, select the **enable** check box. To execute a commit when the message is sent, select the **use transaction** check box.

Enter the following parameters:

- **Name**

  The name of the topic or queue. Use **Search** to browse the JNDI server for the topic or queue name.

- **Type**

  Select whether you are using a topic or queue. To see what messages are waiting to be consumed from a queue (only), use **Browse** to the right of this field.

- **Message**
  Select the type of message you are sending. The supported types are Empty, Text, Object, Bytes, Message, and Mapped (Extended).

- **Advanced**
  Displays a panel where you can edit the message headers and can add message properties.

JMS Messaging - Message Consumer step

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 interval, in seconds, to wait for the next message.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **Wrapper Status**
  The wrapper status contains two output status values:

  - **Current Wrapper Depth:** Number of messages that are left to be read in the current wrapper.

  - **Total Wrappers:** Number of wrappers (destinations).

- **Make payload last response**

  - To make the payload as the last response in the step, select the option.

If some messages are waiting, they can be read by clicking **Next Message**. The Complex Object Editor displays the message.

You are now ready to manipulate this object.

A wrapper is a FIFO list for holding responses from asynchronous topics and queues. A wrapper provides a place for the application to put responses for consumption later. Messages wait in this list for subsequent processing (in this Message Consumer step).

JMS Send Receive Step

The **JMS Send Receive** step lets you connect to any JMS-compatible message server and do the following actions:

- Send a request message

- Receive a response message

The step editor has basic and advanced parameters. To display the advanced parameters, click **PRO** at the top of the editor.

The following graphic shows the step editor. The advanced parameters are not shown.

Screen capture of send receive step.

Each parameter has a tooltip that describes the purpose of the parameter.

Some parameters include the value **Automatic** as an option. This value indicates that the actual setting is taken from another parameter. If you click the drop-down arrow and you place the mouse pointer over the value **Automatic**, a tooltip displays the name of the other parameter.

In the following graphic, the tooltip indicates that the value of the **Destination** parameter is automatically populated from the **Send Destination** parameter.

Screen capture of Automatic tooltip.

Some parameters let you change the editor so that you can enter a property as the value.

Some parameters that provide a discrete set of values let you change the editor so that you can enter the value directly. For example, the **JMS Delivery Mode** parameter has the following values: **Persistent** and **Non-persistent**. In the JMS API, these values map to the numbers 2 and 1, respectively. To enter the value directly, switch to the direct editor. The direct editor also lets you specify a value that is not in the official enumeration of values.

- Send and Receive
- JMS Message
- Test the JMS Send Receive Step
- Monitor and Close Cached Asset Instances

Send and Receive

The **JMS Send Receive** step has separate areas for configuring the JMS send operation and the JMS receive operation.

Specify two destinations in the step editor: one for the send operation, and one for the receive operation. The destinations can be the same or different. The lists are populated with the JMS destination assets from the active configuration.

To send a message without receiving a response, disable the JMS receive operation. To wait for a message without having to send one first, disable the JMS send operation.

If you disable both operations, the step does nothing.

By default, the receive operation uses a synchronous consumer. The advanced parameters include a check box for specifying an asynchronous consumer.

To make sure that the response goes to the correct client, enable the correlation scheme.

Each asset has a runtime scope. The **JMS Send Receive** step lets you specify a minimum runtime scope for the send and receive operations. The scope parameters are advanced parameters.

JMS Message

The **JMS Send Receive** step lets you configure the message that is sent or received.

The following message types are supported:

- Text
- Bytes
- Stream
- Map
- Empty

You configure the payload of the message in the **Content** area.

The available editors in the **Content** area depend on the message type.

Text messages have the following editors:

- Plain text
- JSON
- EDI
- XML

Bytes messages have the following editors:

- Binary
- Graphic image

Stream messages have an editor that lets you add objects.

Map messages have an editor that lets you add key/value pairs.

Empty messages do not have an editor.

If you change the message type, the step checks whether the existing payload can be converted. If the conversion is not possible, the step discards the existing payload.

In JMS, a message includes a set of headers and an optional set of custom properties. You can configure these headers and properties in the step editor.

Test the JMS Send Receive Step

You can verify the functionality of the **JMS Send Receive** step in the editor.

The execution log provides a high-level view of the activity that happens behind the scenes. For example, the following lines in the execution log show the creation of various JMS client assets:

```
Creating JMS Connection
Starting JMS Connection
Creating JMS Session
Performing JNDI lookup with name: queue/B
Creating JMS Consumer on Queue B
Performing JNDI lookup with name: queue/A
Creating JMS Producer
```

Follow these steps:

1. Click the green **Execute** button in the step editor.
2. To cancel the step while it is executing, click **Cancel**.
   The **Response** tab shows the response that was received when the operation finishes.
3. To view the step activity, click **Execution Log**.
4. To monitor the asset instances, click **Runtime Monitor**.
5. To view the request that was sent, click the **Request** tab.

Monitor and Close Cached Asset Instances

When you test the **JMS Send Receive** step, a runtime monitor in the **Response** tab lets you monitor asset instances. You can also manually close the assets.

The following graphic shows an example of the runtime monitor.

Screen capture of runtime monitor.

By default, the runtime monitor automatically removes assets that are closed. In the following procedure, you disable this behavior.

Follow these steps:

1. In the **Response** tab, click **Runtime Monitor**.
2. Clear the check box that appears inside the **Clear** button.
3. Execute the step again.
   The runtime monitor displays the assets that were created during the execution of the step.
4. View the following information:
   - **Name**: The name of the asset.
   - **Type**: The type of asset.
   - **Scope**: The name of the test step, test case instance, test case, or DevTest component that corresponds to the runtime scope of the asset. The value has a tooltip that shows the scope.
   - **Status**: The color green indicates that the asset is active. The color yellow indicates that the asset is idle. The color gray indicates that the asset is closed.
5. To display more information about an asset, click the status icon.
6. To close an asset immediately, click the status icon and select **Close** or **Force Close**.
7. To close a set of assets, click **Force Close**.

Tutorial - Send and Receive a JMS Message

This tutorial provides an introduction to using the **JMS Send Receive** test step.

Prerequisites

- The registry is running.
- The demo server is running.

Step 1 - Create JMS and JNDI Assets

In this procedure, you create JMS and JNDI assets from test steps in an example test case.

Follow these steps:

1. Go to DevTest Workstation and open the examples project.
2. In the **Tests** folder, open the **jms.tst** test case.
3. Select the first **JMS Messaging (JNDI)** step. This step has the name **jms-1**.
4. Click **Generate assets from the selected steps** in the model toolbar.

5. Select the second **JMS Messaging (JNDI)** step. This step has the name **send-msg-post-update**.

6. Click **Generate assets from the selected steps** in the model toolbar.

7. In the **Configs** folder, open the project configuration.

8. Confirm that the JMS and JNDI assets were generated. In the next procedure, you select two of the JMS destination assets.

Step 2 - Configure the JMS Send Receive Step

In this procedure, you perform the following actions:

- Add the **JMS Send Receive** step to a new test case.
- Configure the step to send a JMS text message to a queue and receive the response from a different queue.

The following graphic shows a portion of the step editor. The fields that you change are highlighted.

Screenshot of step editor with fields highlighted.

The **Send Destination** field specifies the queue to which the message is sent.

The **Receive Destination** field specifies the queue from which the response is received.

The **Content** area specifies the text of the message.

Follow these steps:

1. Create a test case in the examples project.

2. Add a **JMS Send Receive** step.
   The step editor appears.

3. In the **Send Destination** list, select **queue/A**.

4. In the **Receive Destination** list, select **queue/B**.

5. In the **Content** area, type a sentence.

6. Save the test case.

Step 3 - Examine JMS and JNDI Assets

In this procedure, you examine one of the JMS destination assets that the **JMS Send Receive** step uses. You also examine the JNDI context asset that the JMS destination asset uses.

Follow these steps:

1. Click the **Edit Selected Asset** icon that appears to the right of the **Send Destination** list. The editor for the JMS destination asset appears.
2. Review the parameters, but do not change them. You can place the mouse pointer over the parameter names to display tooltips.
3. Click the **Edit Selected Asset** icon that appears to the right of the **JNDI Context** list. The editor for the JNDI context asset appears.
4. Review the parameters, but do not change them.
5. Click **Cancel** to return to the editor for the JMS destination asset.
6. Click **Cancel** to return to the step editor.

Step 4 - Test the JMS Send Receive Step

In this procedure, you verify that the **JMS Send Receive** step is configured correctly.

Follow these steps:

1. Click the green **Execute** button in the upper right portion of the step editor.
2. Confirm that the **Request** tab and the **Response** tab contain the same sentence.
3. Close the step editor.

Apache Kafka Steps

The following step is available in the **Apache Kafka** category:

- Kafka Send Receive Step

Kafka Send Receive Step

This step lets you send and receive messages over Apache Kafka.

The following graphic shows the step editor. The advanced parameters are not shown.

Screen capture of the Kafka Send Receive Step

Each parameter has a tooltip that describes the purpose of the parameter.

Some parameters include the value **Automatic** as an option. This value indicates that the actual setting is taken from another parameter. If you click the drop-down arrow and you place the mouse pointer over the value **Automatic**, a tooltip displays the name of the other parameter.

For more conceptual information about Apache Kafka and the corresponding assets, see Apache Kafka Assets.

Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

- Send and Receive
- Message Contents
- Correlation Schemes
- Test the Kafka Send Receive Step
- Advanced Use Cases
- Monitor and Close Cached Asset Instances

Send and Receive

The **Apache Kafka Send Receive** step has separate areas for configuring the send operation and the receive operation.

Specify two topics in the step editor: one for the send operation, and one for the receive operation. The lists are populated with the Kafka Topic assets from the active configuration.

Message Contents

The **Kafka Send Receive** step lets you specify the message content. The following types are supported:

- XML document
- DevTest property reference
- Text
- Graphic image
- Binary
- JSON document
- EDI document

Correlation Schemes

The following correlation scheme is available:

- Payload Scheme

Apache Kafka does not support filtering messages that a consumer receives through correlation IDs and message properties. Kafka only supports payload-based correlation where you receive every message on a topic and then do your own filtering and routing. The following payload scheme options are available:

- **Regex and XPath**
  Lets you specify a correlation ID that is embedded in the message value.

- **Header**
  The Header payload scheme lets you specify a correlation ID that is stored in custom header inside the message. The Request Header parameter specifies which header to use in request messages. The Response Header parameter specifies which header to use in response messages. If the header is the same for both the request and the response, you can leave the Response Header blank.

  > **Note:** If you specified the Request Header property in the Headers field under the Send operation, this correlation will use that header. If you entered more than one header with the same header key, the first one in the list is used.

- **Key**
  The Key payload scheme uses message keys as correlation IDs. This scheme has no parameters.

Test the Kafka Send Receive Step

You can verify the functionality of the **Kafka Send Receive** step in the editor.

The execution log provides a high-level view of the activity that happens behind the scenes.

Follow these steps:

1. Click the green **Execute** button in the step editor.

2. To cancel the step while it is executing, click **Cancel**.
   The **Response** tab shows the response that was received when the operation finishes.

3. To view the step activity, click **Execution Log**.

4. To monitor the asset instances, click **Runtime Monitor**.

5. To view the request that was sent, click the **Request** tab.

Advanced Use Cases

For more information about configuring the Kafka Send Receive Step for advanced use cases, see Kafka Advanced Use Cases.

Monitor and Close Cached Asset Instances

When you test the **Kafka Send Receive** step, a runtime monitor in the **Response** tab lets you monitor asset instances. You can also manually close the assets.

By default, the runtime monitor automatically removes assets that are closed. In the following procedure, you disable this behavior.

Follow these steps:

1. In the **Response** tab, click **Runtime Monitor**.

2. Clear the check box that appears inside the **Clear** button.

3. Execute the step again.
   The runtime monitor displays the assets that were created during the execution of the step.

4. View the following information:

   - **Name**: The name of the asset.

   - **Type**: The type of asset.

   - **Scope**: The name of the test step, test case instance, test case, or DevTest component that corresponds to the runtime scope of the asset. The value has a tooltip that shows the scope.

   - **Status**: The color green indicates that the asset is active. The color yellow indicates that the asset is idle. The color gray indicates that the asset is closed.

5. To display more information about an asset, click the status icon.

6. To close an asset immediately, click the status icon and select **Close** or **Force Close**.

7. To close a set of assets, click **Force Close**.

Kafka Advanced Use Cases

This section describes the following advanced use cases for Apache Kafka:

- Group ID and Queue-like Behavior
- Transaction ID
- Serializers
- Custom Headers
- Consuming on Multiple Topics

Group ID and Queue-like Behavior

Apache Kafka provides a way to configure multiple consumers on the same topic so that a message that is sent to that topic is routed to a single consumer, rather than going to all consumers. This allows you to treat a Kafka topic more like a queue than a topic. The typical use case is a scenario where any single message should only be processed once, but you still want to use multiple consumers that are load balanced together.

This behavior is accomplished through the Group ID. To enable this behavior, simply ensure that the load balanced consumers are all using the same Group ID.

Group ID is available as a parameter on the Receive operation in the Kafka Send Receive Step.

Screen capture of the Group ID field

Group ID is also available as a parameter in the Consumer Config section of the Kafka Consumer asset and in the Default Consumer Config section of the Kafka Connection asset.

> **Note:** The Group ID parameter on the Receive operation takes precedence over all other defined Group ID parameters. The Group ID parameter on the Consumer Asset takes precedence over the Connection Asset.

Transaction ID

If you are using transaction methods in Apache Kafka, you can specify the specific transaction methods to call in the **Transactional ID** field in the Send operation in the Kafka Send Receive Step
.

Screen capture of the Transaction ID field

Transactional ID is also available as a parameter in the Producer Config section of the Kafka Producer asset and in the Default Producer Config section of the Kafka Connection asset.

> **Note:** The Transactional ID parameter on the Send operation takes precedence over all other defined Transactional ID parameters. The Transactional ID parameter on the Producer Asset takes precedence over the Connection Asset.

Serializers

By default in DevTest, Kafka message bodies are in text. However, the character set that is used to encode the text is configurable. Kafka handles this encoding through serializers for producers and deserializers for consumers. To configure these in DevTest, you have to create your own producer and/or consumer.

Producer

By default, the producer is automatically configured by the Send operation in the Kafka Send Receive Step. To override the automatic configuration, click **Add** and select **Kafka Producer** .

Screen capture of the Producer field

You can configure the serializers that are used for the message key and value in the Kafka Producer asset.

Screen capture of the Kafka Producer asset

Note: You do not have to define the Topic field in the Kafka Producer asset. This value is overwritten by the topic that is defined in the Send operation in the Kafka Send Receive Step.

Consumer

By default, the consumer is automatically configured by the Receive operation in the Kafka Send Receive Step. To override the automatic configuration, click **Add** and select **Kafka Consumer**.

Screen capture of the Consumer field

You can configure the serializers that are used for the message key and value in the Kafka Consumer asset.

Screen capture of the Kafka Consumer asset

Note: You do not have to define the Topic field in the Kafka Consumer asset. This value is overwritten by the topic that is defined in the Receive operation in the Kafka Send Receive Step.

Configuring the Send Receive Step for Built-in Serializers

If you are using the built-in serializers, there is no special configuration required for the Key and Value fields in the Kafka Send Receive Step. The Send operation converts the string values of these fields to the value required for the serializer. The only exception is the **Binary** option, but you can use the binary editor under both the Key and Value editors for entering bytes directly.

Custom Serializers

Kafka includes built-in serializers and deserializers for String and for most primitives. The **Custom** option lets you define any custom serializers or deserializers that are required for your own implementations. Click the **Custom** option to enter the serializer or deserializer class name.

Screen capture of the Custom option for serializers

Screen capture of the Custom Serializer

If additional configuration properties are required, you can enter them in a separate window under the Configuration parameter.

Note: You must enter the serializer classes under the Kafka Producer asset and the deserializer classes under the Kafka Consumer asset.

Configuring the Send Receive Step for Custom Serializers

Typically, a custom serializer requires a custom data object that is used by that serializer. In this scenario, you must use a script or custom step that creates the data object and saves it to a testExec property. You can then configure the Kafka Send Receive step by using the **DevTest Property Reference** option under the Key/Value field.

Screen capture of the DevTest Property Reference option

Enter the name of the property.

Screen capture of the Property Name

Custom Headers

Header properties in Kafka are user-defined. You can edit the message headers in the Kafka Send Receive Step.

There are a number of possible header value types available, including String, most primitives, and binary.

> **Note:** Kafka allows multiple headers with the same key.

Header Serializers

At the API level, all Kafka header values are byte arrays. There is no built-in serialization /deserialization for header values in Kafka.

In the Kafka Producer and Kafka Consumer, assets you can define this in the Header Serialization and Header Deserialization sections.

Screen capture of the header serializer configuration

Each header serializer/deserializer is associated with a particular key. When a header with that same key is encountered while constructing a Kafka message for that producer, or parsing a Kafka message from that consumer, that serializer/deserializer is used.

Default Behavior without Header Serializers

If no header serializers are defined in the Kafka Producer asset, DevTest assigns a serializer, based on the type of the header value. For example, for a header with an Integer value, DevTest uses Kafka's Integer serializer to encode the header value.

If no header deserializers are defined in the Kafka Consumer asset, DevTest attempt to convert the Kafka header value into valid text. If that fails, it is read as a byte array.

Consuming on Multiple Topics

With other message protocols, DevTest supports listening on multiple destinations simultaneously by opening multiple consumers. This approach works with Kafka too, and might still be required if different topics have different deserializer requirements. However, if you want to consume from multiple topics, and you can use the same consumer/connection settings for each topic, you can use a single consumer for all of them.

To use a single consumer, you must leave the Topic field on the Receive operation in the Kafka Send Receive Step blank. Instead, create a custom Consumer asset.

Screen capture for consuming multiple topics

List

To configure a Kafka Consumer asset to listen on multiple topics, click **Add** to add topics to the list.

Screen capture of a topic list

Pattern

You can also listen to multiple topics by passing a regular expression for matching on topic names. To use this approach, leave the topic list blank and fill in the **Pattern** field. This field follows the Java Regular Expressions format.

Screen capture of a topic regex pattern

Note: You cannot include a list of topics and a pattern regex to match additional topics. You must either use a topic list or a pattern regex.

BEA Steps

The following steps are available in the **BEA** category:

- Message Consumer
- Read a File
- Web Service Execution (XML)
- FTP Step
- WebLogic JMS (JNDI) (Deprecated)

Message Consumer

For detailed information about this step, see JMS Messaging - Message Consumer.

Read a File

For detailed information about this step, see Read a File (Disk, URL or Class Path).

Web Service Execution (XML)

For detailed information about this step, see Web Service Execution (XML) Step.

FTP Step

For detailed information about this step, see External - FTP Step.

WebLogic JMS (JNDI) (Deprecated)

- Base Tab
- Selector Query Tab
- Send Message Data Tab
- Response Message Tab

Note: This step has been deprecated. Refer to the following for creating new tests and virtual services:

- JMS Client Assets
- JMS Send Receive Step

The WebLogic JMS (JNDI) step lets you send messages to, and receive messages from, topics and queues. You can also receive, change, and forward an existing message.

WebLogic JMS (JNDI) supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

The WebLogic JMS (JNDI) step is configured using a single editor, regardless of the messaging requirements. The input options vary on the messaging requirements. The editor only allows valid configurations. When you enable some features, others can become inactive.**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Parameter Requirements: This step requires the connection parameters and the subject names that are used in the application under test. The following sections describe the necessary parameters. There can be other parameters that are required, depending on your environment. Get these parameters from the developers of the application.

The editor for the WebLogic JMS (JNDI) step contains the following tabs:

The **Base** tab is where you define the connection and messaging parameters.

The **Selector Query** tab lets you specify a selector query to be run when listening for a message on a queue.

The **Send Message Data** tab is where you create the message content.

The **Response Message** tab is where the response messages are posted.

Base Tab

The **Base** tab is divided into the following sections:

- **Server Connection Info**
- **Subscriber Info**

- **Publisher Info**
- **ReplyTo Info**
- **Error Handling and Test**

To enable and disable the **Subscriber Info**, **Publisher Info**, and **ReplyTo Info** sections, use the **enable** check box in each section. Using these check boxes, you can configure the step to be a publish step, a subscribe step, or both. You can also include a JMS reply to component in the step.

When you finish configuring the test step, click **Test** in the **Error Handling and Test** section to test the configuration settings.

Server Connection Info

Enter the JNDI information.

These values are parameterized with properties from your configuration. These propertiessimplify changing the application under test. By default, the WLS_SERVER property in the **JNDI Server URL** is used. This property must be added to your configuration if you plan to use it.

The five parameters are available to you for the system under test. The pull-down menus contain common examples or templates for these values.

- **JNDI Factory Class**
  The fully qualified class name of the context factory for the JNDI provider.
- **JNDI Server URL**
  The URL for connecting to the JNDI server. The format of the URL depends on the specific JNDI provider being used.
- **JMS Connection Factory**

  Use **Search** to browse available resources on the server. Select or enter a connection factory to use for this step execution according to the JMS specification.
- **User**
  The user name for connecting to the JNDI provider and getting a handle to the connection factory.
- **Password**
  The password for connecting to the JNDI provider and getting a handle to the connection factory.
- **Share Sessions and Share Publishers**

  To share JMS sessions and publishers throughout the test case, use these check boxes. This approach can lower overhead, but does not always provide a realistic simulation because typically JMS clients want to release resources. If you select the **Share Publishers** check box, the **Share Sessions** check box is also selected. You cannot share publishers without sharing sessions. For more detailed information about these parameters, see the Deliberate Delays in VSE knowledge base article.
- **Stop All**
  Lets you stop any listeners at design time now. Some listeners can get orphaned, but still consume messages. When they do, it is difficult to create test cases.

- **Advanced**
  Displays a panel where you can add custom properties that are sent with the connection information and you can configure the second-level authentication.

Publisher Info

To set up the ability to send (publish) messages, select the **enable** check box. To execute a commit when the message is sent, select the **use transaction** check box.

Enter the following parameters:

- **Name**

  The name of the topic or queue. Use **Search** to browse the JNDI server for the topic or queue name.

- **Type**
  Select whether you are using a topic or queue. To see what messages are waiting to be consumed from a queue (only), use **Browse** to the right of this field.

- **Message**
  Select the type of message you are sending. The supported types are Empty, Text, Object, Bytes, Message, and Mapped (Extended).

- **Advanced**
  Displays a panel where you can edit the message headers and can add message properties.

Subscriber Info

Select the **enable** check box to set up to enable the ability to receive (subscribe to) messages.

Enter the following parameters:

- **Name**

  The name of the topic or queue. Use **Search** to browse the JNDI server for the topic or queue name.

- **Type**
  Select whether you are using a topic or queue, and whether to listen in synchronous or asynchronous mode. For the asynchronous mode, you must also have an entry in the **Async Key** field. Use **Browse** to the right of this field to see what messages are waiting to be consumed from a queue (only).

- **Timeout (secs)**
  The period to wait before the application interrupts waiting for a message (this field can be left blank for no timeout).

- **Async Key**
  The value that identifies asynchronous messages. This value is only necessary in asynchronous mode. This value is 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 lets you receive all of your messages from a topic, even if you log out and then you log back in.

- **use transaction**
  To execute a Commit when a message is received, select the **use transaction** check box.

- **use temporary queue/topic**
  To have the JMS.provider set up a temporary queue/topic on your behalf, select the **use temporary queue/topic** check box. When a temporary queue/topic is used, the JMS **ReplyTo** parameter of the message you send to the temporary queue/topic is automatically set. The temporary queue/topic feature must always be used with a publisher so that a reply can be sent. If you use a temporary queue/topic, the **ReplyTo** section is disabled.

- **make payload last response**
  To make the payload the response of this step, select the **make payload last response** check box.

ReplyTo Info

To set up a destination queue or topic, select the **enable** check box.

If your application requires a destination, it is set up in this section.

Enter the following parameters:

- **Name**

  The name of the topic or queue. Use **Search** to browse the JNDI server for the topic or queue name.

- **Type**
  Select whether you are using a topic or queue. To see what messages are waiting to be consumed from a queue (only), use **Browse** to the right of this field.

Error Handling and Test

If an exception occurs, the **Error Handling and Test** section lets you redirect to a step.

- **If Environment Error**
  Specifies the action to take or the step to go to if the test fails because of an environment error.
  Default: Abort the test.

Click **Test** to test your step configuration settings.

Selector Query Tab

You can enter a JMS selector query in this editor. The syntax closely follows SQL and is a subset of SQL92. A JMS selector query can be specified when listening for a message on a queue that is a response to a published message. The previous graphic shows a specific query for a **JMSCorrelationID** that matches one set in a property as sent with the original message.

A built-in mechanism allows a test creator to set the **JMSCorrelationID** for a message before sending it. Before the message is sent, you can set the correlation ID by setting the lisa.jms.correlation.id property.

A non-zero value is detected, and the message **JMSCorrelationID** property is set before the message is sent.

Send Message Data Tab

If your step is configured to publish, this tab lets you compose your message. The **Send Message Data** tab view in the following example shows a text message.

This example shows an XML fragment with properties being used. You can type the text, click **Read Message from File** to read from a file, or you can store the fragment in a property. If you store the text in a property, simply place the property in the editor: for example, **LISA_PROP**.

Notice that 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 is shown here. For more information, see JMS Messaging (JNDI) (Deprecated).

Sun JCAPS Steps

The following steps are available in the **Sun JCAPS** category:

- JCAPS Messaging (Native) (Deprecated)
- JCAPS Messaging (JNDI) (Deprecated)

More Information:

- Web Service Execution (XML) Step
- SQL Database Execution (JDBC)
- Read a File (Disk URL or Classpath)
- External - FTP Step

- Message Consumer

JCAPS Messaging (Native) (Deprecated)

Note: This step has been deprecated. Refer to the following for creating new tests and virtual services:

- JMS Client Assets
- JMS Send Receive Step

The JCAPS Messaging (Native) step lets you send and receive messages from topics and queues. You can also receive, change, 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 editor regardless of the messaging requirements. The input options vary on the messaging requirements. The editor only allows valid configurations, so when you enable some features others can become inactive.

The default JCAPS Messaging (Native) step has a name uses the following convention: *JCAPS queuename publish*. If there is not a publish queue name, the default step name is *JCAPS queuename subscribe*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Parameter Requirements: This step requires the connection parameters and the subject names that are used in the application under test. The following sections describe the parameters that you require. There can be other required parameters, depending on your environment. Get these parameters from the developers of the application.

The messaging step editor for JCAPS Messaging (Native) is used to configure this step.

JCAPS Messaging (Native) step

The editor for the JCAPS Messaging (Native) step contains the following tabs.

- The **Base** tab is where you define the connection and messaging parameters.
- The **Selector Query** tab lets you specify a selector query to be run when listening for a message on a queue.
- The **Send Message Data** tab is where you create the message content.
- The Response Message tab is where the response messages are posted.

Base Tab

The **Base** tab is divided into the following sections:

- **Server Connection Info**
- **Subscriber Info**
- **ReplyTo Info**
- **Publisher Info**
- **Error Handling and Test**

To enable and disable the **Subscriber Info**, **Publisher Info**, and **ReplyTo Info** sections, use the **enable** check box in the top left corner of each section. Using these check boxes, you can configure the step to be a publish step, a subscribe step, or both. You can also select to include a JMS reply to component in the step.

When you finish configuring the test step, click **Test** in the **Error Handling and Test** section to test the configuration settings.

Server Connection Info

The **Server Connection Info** section displays two parameters 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 panel where you can add custom properties that are sent with the connection information.

All other tabs are defined in detail in JMS Messaging (JNDI) (Deprecated).

JCAPS Messaging (JNDI) (Deprecated)

Note: This step has been deprecated. Refer to the following for creating new tests and virtual services:

- JMS Client Assets

- JMS Send Receive Step

The JCAPS Messaging (JNDI) step lets you send and receive messages from topics and queues. You can also receive, change, 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 DevTest editor regardless of the messaging requirements. The input options vary on the messaging requirements. The editor only allows valid configurations, so when you enable some features others can become inactive. **Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Parameter Requirements: This step requires the connection parameters and the subject names that are used in the application under test. Other parameters may be required, depending on your environment. Get these parameters from the developers of the application.

Detailed information about parameters and fields for JCAPS Messaging can be found in JMS Messaging (JNDI) (Deprecated).

Default Step Names: The default JCAPS Messaging (JNDI) step name uses the following convention: *JCAPS queuename publish*. If there is not a publish queue name, the default step name is *JCAPS queuename subscribe*.

If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

Oracle Steps

The following steps are available in the **Oracle** category:

- Oracle OC4J (JNDI) (Deprecated)
- Oracle AQ (JMS) (Deprecated)

More Information:

- Web Service Execution (XML) Step
- SQL Database Execution (JDBC)
- Read a File (Disk URL or Classpath)
- External - FTP Step
- Message Consumer

Oracle OC4J (JNDI) (Deprecated)

Note: This step has been deprecated. Refer to the following for creating new tests and virtual services:

- JMS Client Assets
- JMS Send Receive Step

The Oracle OC4J (JNDI) step lets you 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 editor regardless of the messaging requirements. The input options on the messaging requirements vary. The editor only allows valid configurations, so when you enable some features others could become inactive.
Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Parameter Requirements: You need the connection parameters and the subject names that are used in the application under test. There may be other parameters that are required, depending on your environment. Get these parameters from the developers of the application.

DevTest, 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.

Detailed information about the parameters and fields for this step can be found in JMS Messaging (JNDI) (Deprecated).

Oracle AQ (JMS) (Deprecated)

Note: This step has been deprecated. Refer to the following for creating new tests and virtual services:

- JMS Client Assets
- JMS Send Receive Step

The Oracle AQ step is added to a test case to allow sending messages, receiving messages, receiving messages asynchronously, or some combination of the three.

Oracle AQ is a messaging provider, like IBM WebSphere MQ, webMethods Broker, TIBCO EMS, and others. Oracle AQ fits into DevTest like any of those other messaging providers.

The Oracle AQ step has the standard configuration sections for messaging steps.

- A **Connection** section for configuring the connection information.
- An optional **Subscriber** section for configuring the location from which the step receives its message and the type of message it receives.
- An optional **Publisher** section for configuring the location to which the step sends its message and the type of message it sends. The contents of the message to send are configured in a separate tab.
- The step sends one message, receives one message, or both. The step can run multiple times in a loop in the same test case to send or receive multiple messages.
- When the step is used as an asynchronous subscriber, then it is only run once in a test case. An extra Consumer step is run to consume each message received from the provider.

Oracle Advanced Queuing (AQ) is a messaging provider that is built into the Oracle database itself. AQ is used as the default JMS provider for many Oracle products, such as Oracle Enterprise Service Bus.

Screenshot of Oracle AQ (JMS) step

Using the JMS library works like any other normal JMS provider, with a few notable differences:

- The JMS connection is not made through JNDI. The connection is made using a JDBC connection and involves entering a JDBC URL, driver class name, username, and password.
- Queues and topics are tied to schemas in the database. To send to a queue or receive from a queue, give both the queue name and queue schema.
- Each queue or topic is restricted to a specific type of JMS Message. For example, if a queue typically transports JMS Text Messages, then you cannot use that same queue to transport JMS Object Messages, or JMS Byte Messages.
- Setting up JMS queues and topics in the Oracle database involves running stored procedures.

Four tabs are available at the bottom of the editor.

- The **Base** tab is where you define your connection and messaging parameters.
- The **Selector Query** tab lets you specify a selector query to be run when listening for a message on a queue.
- The **Send Message Data** tab is where you create your message content.
- The **Response Message** tab is where your response messages are posted.

Base Information Tab

The **Base** tab view is shown in the previous example contains five 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. You can use the **enable** check box in the top left corner of each section to enable or disable **Subscriber Info**, **Publisher Info**, and **ReplyTo Info**. Using these check boxes, you can configure the step to publish a step, subscribe a step, or both. You can also select to include a replyto component in the step.

When you have configured your test step, click **Test** in the **Error Handling and Test** section to test your configuration settings.

Server Connection Info

Here you enter the JDBC-related information.

Parameterize these values with properties that are in your configuration, making it easy to change the application under test.

DevTest, by default, uses the **oracle.jdbc.driver.OracleDriver** in the JDBC Driver location.

The following parameters are available to you for the system under test. The pull-down menus contain common examples or templates for these values.

- **JDBC URL**
  This field is prepopulated with default values.
- **JDBC Server**
  This field is prepopulated with default values.
- **User**
  Enter the user name.
- **Password**
  Enter the password.
- **Share Sessions and Share Publishers**
  To share JMS sessions and publishers throughout the test case, use these check boxes. This approach can lower overhead, but does not always provide a realistic simulation because typically JMS clients want to release resources. If you select the **Share Publishers** check box, the **Share Sessions** check box is also selected. You cannot share publishers without sharing sessions. For more detailed information about these parameters, see the Deliberate Delays in VSE knowledge base article.
- **Stop All**
  Lets you stop any listeners at design time now. Some listeners can get orphaned, but still consume messages. When they do, it is difficult to create test cases.

Publisher Info

To set up the ability to send (publish) messages, select the **enable** check box.

To execute a commit when the message is sent, select the **use transaction** check box.

Enter the following parameters:

- **Schema**
  Enter the name of the schema to use.

- **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. The supported types are Empty, Text, Object, Bytes, Message, and Mapped (Extended).

- **Advanced**
  Displays a panel where you can edit the message headers and can add message properties.

Subscriber Info

To set up to enable the ability to receive (subscribe to) messages, select the **enable** check box.

Enter the following parameters:

- **Schema**
  Enter the name of the schema to use.

- **Name**
  Enter the name of the topic or queue to use.

- **Type**
  Select whether you are using a topic or queue, and whether to listen in synchronous or asynchronous mode. For asynchronous mode, you also must have an entry in the Async key field. To see what messages are waiting to be consumed from a queue (only), click **Browse**, to the right of this field.

- **Timeout (secs)**
  Indicates the number of seconds before DevTest interrupts waiting for a message. For no timeout, leave this field blank.

- **Async Key**
  Enter the value that is necessary to identify asynchronous messages. This value is only required in asynchronous mode. This value is 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 lets you receive all of your messages from a topic even if you log out, and then you log in again.

- **Session Mode**
  Select the appropriate mode from the available options by clicking the drop-down list. Options are: Auto Acknowledge, Client Acknowledge, Use Transaction, Auto (Duplicates Okay).

ReplyTo Info

If your application requires a destination, it is set up in this section.

To set up a destination queue/topic, select the **enable** check box.

Enter the following parameters:

- **Schema**
  Enter the name of the schema to use.
- **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

If an error occurs, the **Error Handling and Test** section lets you redirect to a step.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

Click **Test** to test your step configuration settings.

TIBCO Steps

The following steps are available in the **TIBCO** category:

- TIBCO Rendezvous Messaging
- TIBCO EMS Messaging (Deprecated)
- TIBCO Direct JMS (Deprecated)

More Information:

- Web Service Execution (XML) Step
- SQL Database Execution (JDBC)
- Read a File (Disk URL or Classpath)
- External - FTP Step
- Message Consumer

TIBCO Rendezvous Messaging

The TIBCO Rendezvous Messaging step lets you send and receive messages from Rendezvous "Subjects" using Native Rendezvous protocol. You can also receive, change, and forward an existing message.

The TIBCO Rendezvous Messaging step is configured using a single editor regardless of the messaging requirements. The input options vary on the messaging requirements. The editor only allows valid configurations, so when you enable some features others could become inactive.

The default TIBCO Rendezvous Messaging step uses the following convention: *RV queuename publish*. If there is not a publish queue name, the default step name is *RV queuename subscribe*. If another step also uses the default step name, a number is appended to the step name. You can change step names at any time.**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Parameter Requirements: This step requires the connection parameters and the subject names that are used in the application under test. The following sections describe the parameters that you require. There could be other required parameters, depending on your environment. Get these parameters from the developers of the application.

TIBCO Rendezvous Messaging step

The editor for the TIBCO Rendezvous Messaging step contains the following tabs:

- The **Base** tab is where you define the connection and messaging parameters.
- The **Send Message Data** tab is where you create the message content.
- The **Response Message** tab is where the response messages are posted.

Base Tab

The **Base** tab is divided into the following sections:

- **Server Connection Info**
- **Subscriber Info**
- **Certified Transport Info**
- **Publisher Info**
- **ReplyTo Info**
- **Error Handling and Test**

To enable and disable the Subscriber Info, Publisher Info, and ReplyTo Info sections, use the **enable** check box in the top left corner of each section. To configure the step to be a publish step, a subscribe step, or both, use these check boxes. You can also select to include a JMS reply to component in the step.

When you finish configuring the test step, click **Test** in the **Error Handling and Test** section to test the configuration settings.

Server Connection Info

Enter the connection information specific to Rendezvous information in the **Server Connection Info** area.

The following parameters are available for the system under test:

- **Service, Network, and Daemon**
  These parameters enable connection to the RV network that you want to communicate on.
- **Client Mode**
  Select either the Rendezvous Native client or Java Client mode. Usually you use the more versatile client mode.

To simplify changing the system under test, parameterize these values with properties from your configuration.

Publisher Info

To set up the ability to send messages, select the **enable** check box.

Complete the following fields:

- **Subject**
  The name of the subject to use. You can define your own subjects. A valid subject name is: **queue.sample**. An invalid subject name is: **queue…..My_Samples** (null element) or **.My. Queue..** (three null elements).
- **Message**
  Select the type of message you are sending. The supported types are Empty, Text, Object, Bytes, Message, and Mapped (Extended).

- **Send Field**
  RV messages are actually maps of fields and values. This field is used 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 value is overridden with Mapped (Extended) type messages as they let you do multiple fields and values in a single message.

- **Enable Inbox Type**
  To enable the **Inbox Timeout** and **Enable sendReply** fields, select this check box.

- **Enable sendReply**
  To specify an **Inbox Timeout** or to enable sendReply functionality for the publisher, select the **Enable Inbox Type**.

Certified Transport Info

To provide transport information, select the **enable** check box.

- **Sender Name**
  The name that is the correspondent name of the CM transport.

- **Advisory Subject**
  Rendezvous software constructs the subject names of system advisory messages using this template: **_RV.class.SYSTEM.name**. Rendezvous certified message delivery software constructs the subject names of advisory messages using these templates: **_RV.class.RVCM.category.condition.subject** and **_RV.class.RVCM.category.condition.name**. Distributed queue software constructs the subject names of advisory messages using this template: **_RV.class.RVCM.category.role.condition.name**. Rendezvous fault tolerance software constructs the subject names of advisory messages using this template: **_RV.class.RVFT.name.group**.

- **Time Limit**
  The time limit in which a message exists.

Subscriber Info

Selecting the **enable** box turns on the subscriber function and lets you set up the ability to receive messages.

Complete the following fields:

- **Subject**
  The name of the subject to use. You can define your own subjects.

- **Timeout (secs)**
  Indicates the number of seconds before DevTest interrupts waiting for a message. For no timeout, leave this field blank.

- **Async Key**
  Enter the value that is necessary to identify asynchronous messages. This value is only required in asynchronous mode. This value is used in a subsequent Message Consumer step to retrieve asynchronous messages.

ReplyTo Info

To set up a destination subject, select the **enable** check box.

If your application requires a destination, it is set up in this section.

Enter the following parameter:

- **Subject**
  The name of the subject to use.

Error Handling and Test

If an error occurs, the Error Handling and Test section lets you redirect to a step.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

Click **Test** to test your step configuration settings.

Send Message Data

If your step is configured to publish, compose the message on this tab. The **Send Message Data** tab view in the following example shows a text message.

TIBCO Rendezvous Messaging step Send Message Data tab

This example shows an XML fragment with properties being used. You can type the text, or you can click **Read Message From File** to read from a file. The text can also be stored in a property, in which case you would place the property in the editor: for example, LISA_PROP.

Notice that 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, the response is shown. For more information, see JMS Messaging (JNDI) (Deprecated).

TIBCO EMS Messaging (Deprecated)

Note: This step has been deprecated. Refer to the following for creating new tests and virtual services:

- JMS Client Assets
- JMS Send Receive Step

The TIBCO EMS Messaging step lets you send and receive messages from topics and queues. You can also receive, change, and forward an existing message.

All the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended) are supported.

The default TIBCO EMS Messaging step names uses the following convention: *EMS queuename publish*. If there is not a publish queue name, the default step name is *EMS queuename subscribe*.

If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Parameter Requirements: This step requires the connection parameters and the subject names that are used in the application under test. The default is the TIBCO_SERVER property in the JNDI Server URL. This property must be added to your configuration if you plan to use it. Your environment may require other parameters. Get these from the application developers.

For more detailed information about parameters and fields, see JMS Messaging (JNDI) (Deprecated).

TIBCO Direct JMS (Deprecated)

Note: This step has been deprecated. Refer to the following for creating new tests and virtual services:

- JMS Client Assets
- JMS Send Receive Step

The TIBCO Direct JMS step lets you send messages and receive messages from topics and queues without using the JNDI libraries. You can also receive, change, and forward an existing message.

All the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended) are supported.

The TIBCO Direct JMS step is configured using a single editor regardless of the messaging requirements. Input options vary on the messaging requirements. The editor only allows valid configurations. Enabling some features can make other unavailable.

The default TIBCO Direct JMS step name uses the following convention: *EMS queuename publish*. If there is not a publish queue name, the default step name is *EMS queuename subscribe*. If another step also uses the default step name, DevTest appends a number to the step name. You can change step names at any time.**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Parameter Requirements: This step requires the connection parameters and the subject names that are used in the application under test. DevTest, 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. Your environment may require other parameters. Get these from the application developers.

For more detailed information about parameters and fields, see JMS Messaging (JNDI) (Deprecated).

Sonic Steps

The following steps are available in the **Sonic** category:

- SonicMQ Messaging (Native) (Deprecated)
- SonicMQ Messaging (JNDI) (Deprecated)

More Information:

- Web Service Execution (XML) Step
- SQL Database Execution (JDBC)
- Read a File (Disk URL or Classpath)
- External - FTP Step
- Message Consumer

SonicMQ Messaging (Native) (Deprecated)

Note: This step has been deprecated. Refer to the following for creating new tests and virtual services:

- JMS Client Assets
- JMS Send Receive Step

The SonicMQ Messaging (Native) step lets you send and receive messages from topics and queues using native Sonic protocol. You can also receive, change, and forward an existing message.

SonicMQ Messaging (Native) supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

The SonicMQ Messaging (Native) step is configured using a single editor regardless of the messaging requirements. The input options vary on the messaging requirements. The editor only allows valid configurations, so when you enable some features others could become inactive.
Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Parameter Requirements: This step requires the connection parameters and the subject names that are used in the application under test.

The following parameters are required for the system under test.

- Broker Host
- Broker Port
- User
- Password

There could be other parameters that are required, depending on your environment. Get these parameters from the developers of the application.

Default Step Names: The default SonicMQ Messaging (Native) step name uses the following convention: *Sonic queuename publish*. If there is not a publish queue name, the default step name is *Sonic queuename subscribe*.

If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

For more detailed information about parameters and fields, see JMS Messaging (JNDI) (Deprecated).

SonicMQ Messaging (JNDI) (Deprecated)

Note: This step has been deprecated. Refer to the following for creating new tests and virtual services:

- JMS Client Assets
- JMS Send Receive Step

The SonicMQ Messaging (JNDI) step lets you send and receive messages from topics and queues. You can also receive, modify, and forward an existing message.

SonicMQ Messaging (JNDI) supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

The SonicMQ Messaging (JNDI) step is configured using a single DevTest editor regardless of the messaging requirements. The input options vary on the messaging requirements. The editor only allows valid configurations, so when you enable some features others could become inactive.
Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Parameter Requirements: You need the connection parameters and the subject names that are used in the application under test. DevTest, 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. There could be other parameters that are required, depending on your environment. Get these parameters from the developers of the application.

Default Step Names: The SonicMQ Messaging (JNDI) step has a default name using the following convention: *Sonic queuename publish*. If there is not a publish queue name, the default step name is *Sonic queuename subscribe*.

If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

For more detailed information about parameters and fields, see JMS Messaging (JNDI) (Deprecated).

webMethods Steps

The following steps are available in the **webMethods** category:

- webMethods Broker
- webMethods Integration Server Services

webMethods Broker

The webMethods Broker step lets you send and receive messages from the Broker. You can also receive, change, and forward existing Broker Events/ Messages.

webMethods Broker supports Mapped (Extended) messages that create Broker Events.

The webMethods Broker step is configured using a single editor, regardless of the messaging requirements. The input options vary on the messaging requirements. The step editor only allows valid configurations, so when you enable some features others can become inactive.
Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Parameter Requirements: This step requires the connection parameters and the subject names that are used in the application under test. The following sections describe the parameters that you require. Other parameters could be required, depending on your environment. Get these parameters from the developers of the application.

webMethods Broker step

The messaging step editor for webMethods Broker includes three tabs at the bottom of the page:

- The **Base** tab is where you define your connection and messaging parameters.
- The **Send Message Data** tab is where you create your message content.
- The **Response Message** tab is where your response messages are posted.

Base Tab

The **Base** tab view that the previous graphic shows is divided into the following 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. To enable or disable the **Subscriber Info**, **Publisher Info**, and **ReplyTo Info** sections, use the **enable** check box in the top left corner of each section. Using these check boxes, you can configure the step to be a publish step, a subscribe step, or both. You can also include a **replyto** component in the step.

When you have configured your test step, click **Test** in the **Error Handling and Test** section to test your configuration settings.

Server Connection Info

In the **Server Connection Info** section, enter the connection information specific to webMethods Broker.

The four parameters must be available to you for the system under test.

- **Broker Host**
- **Host Port**
- **Broker Name**
- **Client ID**
- **Client Group**
  This value is the Client group that is able to see the Broker destinations to use.
- **App Name**
  Specify the application using the Broker here. This parameter is optional and mostly used in server logs for debugging. The default is "DevTest". Using this parameter is a good practice, but if you must use something else for application logic you can.

To simplify changing the system under test, parameterize these values with properties from your configuration.

Publisher Info

To set up the ability to send (publish) messages, select the **enable** check box.

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**
  The selected docType is inspected and the message with the required fields is preloaded. This check box lets you change fields and decide whether to re-add any missing fields. DevTest does not write over existing fields with the same name. This property is only a design time effect and does nothing at test run time.
- **Deliver Enabled**
  Select to enable the **Deliver Client ID** field.

- **Deliver Client ID**
  The broker Client Identification for the connection. If the value is null, the broker generates an identifier automatically. An error can be returned if the value is already in use by another connection.

- **Envelope Tag**
  This parameter lets you set the env.tag property on a broker event message.

Subscriber Info

To set up to enable the ability to receive (subscribe to) messages, select the **enable** check box.

Enter the following parameters:

- **docType**
  Enter the name of the docType to use.

- **Timeout (secs)**
  Indicates the number of seconds before DevTest interrupts waiting for a message. For no timeout, leave this field blank.

- **Async Key**
  Enter the value that is necessary to identify asynchronous messages. This value is only required in asynchronous mode. This value is used in a subsequent Message Consumer step to retrieve asynchronous messages.

- **Auto convert to**
  To return a string representation of the payload, enter a string that calls the toString() function on the payload object; xml returns the payload in XML format.

ReplyTo Info

To set up a destination queue/topic, select the **enable** check box.

If your application requires a destination, it is set up in this section.

Enter the following parameters:

- **Name**

  The name of the topic or queue. Use **Search** to browse the JNDI server for the topic or queue name.

Error Handling and Test

If an error occurs, the **Error Handling and Test** option lets you redirect to a step.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

Click **Test** to test your step configuration settings.

Send Message Data Tab

This tab is where you compose your message, if your step is configured to publish.

Response Message Tab

If your step is configured to subscribe, your response is shown here. For more information, see JMS Messaging (JNDI) (Deprecated).

Default Step Names

The default webMethods Broker step name uses the following convention: *webM queuename publish*. If there is not a publish queue name, the default step name is *webM queuename subscribe*.

If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

webMethods Integration Server Services

The webMethods Integration Server Services step lets you execute Integration Server services through the native Java APIs. This is done using IData objects so it works with services not exposed through HTTP transports.**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Parameter Requirements: This step requires the connection parameters and the subject names that are used in the application under test. The following sections describe the parameters that you need. Other parameters could be required, depending on your environment. Get these parameters from the application developers.

Base Tab Server Connection Info

Enter the following parameters:

- **Host**
  The host name.
- **User**
  The userid.
- **Use SSL**
  To record without using SSL, leave the **Use SSL** check box cleared. To record using SSL, select the **Use SSL** check box.
- **Password**
  The password.
- **Package**
  The package in which the service is located.
- **Service**
  The name of the actual service that you want to call.
- **Input Type**
  Select the input type from **Property**, **IData Object**, or **Force IData Pre-fill**.

- **Output Type**
  Select the output type from **XML** or **IData Object**.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

Click **Execute** to connect. You see an object response. To pull the payload or other properties from the response, which is itself an IData object, export it to a Java Execution step. To complete this task, create a Java Step in DevTest and load from a property, specifying the step name pattern for a last response. Use the lisa.<stepName>.rsp property.

webMethods Integration Server step

Pipeline Input Tab

webMethods Integration Server step Pipeline Input tab

Pipeline Output Tab

webMethods Integration Server step Pipeline Output tab

Default Step Names

The default webMethods Integration Server Services step name uses the following convention: *IntegrationServerInvoker ServiceName@HostName*.

If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

IBM Steps

The following steps are available in the **IBM** category:

- IBM WebSphere MQ Step (Deprecated)
- IBM MQ Native Send Receive Step

More Information:

- Message Consumer

IBM WebSphere MQ Step (Deprecated)

Note: This step has been deprecated. Refer to the following for creating new tests and virtual services:

- Native Mode, Bindings Mode:
  - IBM MQ Native Send Receive Step
- JMS Mode:
  - JMS Client Assets
  - JMS Send Receive Step

The IBM WebSphere MQ step lets you send messages to, and receive messages from, topics and queues. You can also receive, modify, and forward an existing message.

All the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended) are supported.

The IBM WebSphere MQ step is configured using a single editor regardless of the messaging requirements. The input options vary on the messaging requirements. The editor only allows valid configurations, so when you enable some features, others could become inactive.

The default IBM WebSphere MQ step name uses the following convention: *MQ queuename publish*. If there is not a publish queue name, the default step name is *MQ queuename subscribe*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Parameter Requirements: You must have the connection parameters for your system under test. The following sections describe the required parameters.

The editor for the IBM WebSphere MQ step contains the following tabs:

- The **Base** tab is where you define the connection and messaging parameters.
- The **Selector Query** tab lets you specify a selector query to be run when listening for a message on a queue.
- The **Send Message Data** tab is where you create the message content.

- The **Response Message** tab is where the response messages are posted.

> **Note:** This page describes the **Base** tab. For information about the other tabs, see JMS Messaging (JNDI) (Deprecated).

IBM WebSphere MQ Base Tab

The following graphic shows the **Base** tab. The tab is divided into the following sections:

- **Server Connection Info**
- **Subscriber Info**
- **ReplyTo Info**
- **Publisher Info**
- **Error Handling and Test**

IBM Websphere MQ Server Connection Info

To enable and disable the **Subscriber Info**, **Publisher Info**, and **ReplyTo** Info sections, use the **enable** check box in the top left corner of each section. Using these check boxes, you can configure the step to be a publish step, a subscribe step, or both. You can also include a reply to component in the step.

When you finish configuring the test step, click **Test** in the **Error Handling and Test** section to test the configuration settings.

Server Connection Info

To connect to WebSphere MQ, enter the following information:

- **Host Name**
  The name of the host.
- **TCP/IP Port**
  The port for a client connection.
- **Channel**
  A connection property that is used for routing and management in the message bus.
- **Queue Manager**
  A connection property that is used for routing and management.
- **CCID**
  Optional for connections and only applies if character transformation must occur between the client (DevTest) and server.
- **User Name**
  The login user name, if applicable.
- **Password**
  The login password, if applicable.
- **Client Mode**
  Lets you select how you want to interact with the WebSphere MQ server.
  - **JMS:** A pure Java implementation that is based on the JMS specification. We recommend that you use the JMS Transport Protocol instead of MQ for this implementation.
  - **Native Client:** A pure Java implementation using IBM-specific APIs.
  - **Bindings:** Requires access to the native libraries from a WebSphere MQ client installation. Verify that these libraries are accessible by the DevTest application run time. In most cases, having these libraries available in the PATH environment works.
- **Share Sessions**
  Select to specify sharing everything in MQ Native Mode, including the connection.

Publisher Info

To set up the ability to send (publish) messages, select the **enable** check box.

To execute a commit when the message is sent, select the **use transaction** check box.

Enter the following parameters:

- **Name**

  The name of the topic or queue. Use **Search** to browse the JNDI server for the topic or queue name.

- **Type**

  Select whether you are using a topic or queue. To see what messages are waiting to be consumed from a queue (only), use **Browse** to the right of this field.

- **Message**
  Select the type of message you are sending. The supported types are Empty, Text, Object, Bytes, Message, and Mapped (Extended).

- **Alt QManager**
  The queue manager that hosts the publish queue, if it is different from the queue manager to which you are connecting.

- **Message Properties**
  Lets you set publish properties on the message. The list of properties changes depending on the client mode. For more information, see the WebSphere MQ documentation and the *JMS and MQ Message Properties* knowledge base article. If the client mode is JMS, then you can add custom message properties.

Subscriber Info

To set up to enable the ability to receive (subscribe to) messages, select the **enable** check box.

Enter the following parameters:

- **Name**

  The name of the topic or queue. Use **Search** to browse the JNDI server for the topic or queue name.

- **Type**
  Select whether you are using a topic or queue, and whether to listen in synchronous or asynchronous mode. For asynchronous mode, you also must have an entry in the Async key field. To see what messages are waiting to be consumed from a queue (only), click **Browse**, to the right of this field.

- **Timeout (secs)**
  Indicates the number of seconds before DevTest interrupts waiting for a message. For no timeout, leave this field blank.

- **Queue Model**
  MQ requires this value to create temporary destinations. The queue model is configured on the MQ server, and 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 that is necessary to identify asynchronous messages. This value is only required in asynchronous mode. This value is 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 lets you receive all of your messages from a topic even if you log out, and then you log in again.

- **Session Mode**
  Select the appropriate mode from the available options by clicking the drop-down list. Options are: Auto Acknowledge, Client Acknowledge, Use Transaction, Auto (Duplicates Okay).

  - **Auto Acknowledge:** The session automatically acknowledges the receipt of a message by a client.

  - **Client Acknowledge:** This option instructs the client to acknowledge messages programmatically.

  - **Use Transaction:** To execute a commit when a message is received.

  - **Auto (Duplicates Okay):** This option instructs the session to acknowledge the delivery of messages.

- **Use temporary queue/topic**
  If you want JMS.provider to set up a temporary queue/topic on your behalf, select the use temporary queue/topic check box. When a temporary queue/topic is used, the JMS ReplyTo parameter of the message you send to the temporary queue/topic is automatically set. The temporary queue/topic feature must always be used with a publisher so that a reply can be sent. If you use a temporary queue/topic, the ReplyTo section is disabled.

- **Make payload last response**
  To make the payload as the last response, select this option.

- **Use correlation ID for subscribe**
  Enables filtering of incoming messages using the values in **Subscribe Properties**.

- **Subscribe Properties**
  Lets you set subscribe properties on the message. For more information, see the WebSphere MQ documentation and the JMS and MQ Message Properties knowledge base article.

ReplyTo Info

To set up a destination queue/topic, select the **enable** check box.

If your application requires a destination, it is set up in this section.

Enter the following parameters:

- **Name**

  The name of the topic or queue. Use **Search** to browse the JNDI server for the topic or queue name.

- **Type**

  Select whether you are using a topic or queue. To see what messages are waiting to be consumed from a queue (only), use **Browse** to the right of this field.

- **Queue Manager**
  Allows the ReplyTo to be on a different Queue Manager than the Publisher (in this step).

Error Handling and Test

If an error occurs, the **Error Handling and Test** section lets you redirect to a step.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

Click **Test** to test your step configuration settings.

IBM MQ Native Send Receive Step

The **IBM MQ Native Send Receive** step lets you connect to a WebSphere MQ queue manager and do the following actions:

- Send a request message
- Receive a response message

> **Note:** This step is for WebSphere MQ in native mode. If you are using WebSphere MQ in JMS mode, we recommend that you use the **JMS Send Receive** step and the **IBM MQ JMS** assets.

This feature is supported on WebSphere MQ 5.2, 6.0, 7.1, 8.0, and 9.0.

Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

The step editor has basic and advanced parameters. To display the advanced parameters, click **PRO** at the top of the editor.

The following graphic shows the step editor. The advanced parameters are not shown.

Screen capture of send receive step.

Each parameter has a tooltip that describes the purpose of the parameter.

Some parameters include the value **Automatic** as an option. This value indicates that the actual setting is taken from another parameter. If you click the drop-down arrow and you place the mouse pointer over the value **Automatic**, a tooltip displays the name of the other parameter.

Some parameters let you change the editor so that you can enter a property as the value.

Some parameters that provide a discrete set of values let you change the editor so that you can enter the value directly. For example, the **Get Options** parameter corresponds to an integer value in the WebSphere MQ API. Instead of opening the **Get Options** dialog, you can switch to the direct editor and enter the value. The direct editor also lets you specify a value that is not in the official enumeration of values.

- IBM MQ Native Put and Get
- IBM MQ Native RFH2 Header
- Test the IBM MQ Native Send Receive Step
- Monitor and Close Cached Asset Instances

IBM MQ Native Put and Get

The **IBM MQ Native Send Receive** step has separate areas for configuring the put operation and the get operation.

Specify two queues in the step editor: one for the put operation, and one for the get operation. The lists are populated with the IBM MQ Native queue assets from the active configuration.

To send a message without receiving a response, disable the get operation. To wait for a message without having to send one first, disable the put operation.

If you disable both operations, the step does nothing.

To make sure that the response goes to the correct client, enable the correlation scheme.

Each asset has a runtime scope. The **IBM MQ Native Send Receive** step lets you specify a minimum runtime scope for the put and get operations. The scope parameters are advanced parameters.

You can configure the properties of the WebSphere MQ message descriptor (MQMD). For example, you can configure the following properties:

- Message Type
- Expiry
- Format
- Priority

When the advanced parameters are displayed, click **Open Editor** to the right of the **Header Properties** label and click the plus sign to select the properties.

IBM MQ Native RFH2 Header

The **Contents** area of the step editor lets you configure the message payload. You can include an RFH2 header by setting the payload type to **IBM RFH2 Header**.

To edit the properties in the fixed portion of the RFH2 header, click **Open Editor** to the right of the **RFH Header Properties** label and click the plus sign to select the properties.

To add a folder to the RFH2 header, perform the following steps:

1. Display the advanced parameters.
2. Click the plus sign in the **RFH2 Folders** area.
3. Select the folder type.

You can now click **Open Editor** to the right of the folder name to configure the properties in the folder.

Test the IBM MQ Native Send Receive Step

You can verify the functionality of the **IBM MQ Native Send Receive** step in the editor.

The execution log provides a high-level view of the activity that happens behind the scenes.

Follow these steps:

1. Click the green **Execute** button in the step editor.
2. To cancel the step while it is executing, click **Cancel**.
   The **Response** tab shows the response that was received when the operation finishes.
3. To view the step activity, click **Execution Log**.
4. To monitor the asset instances, click **Runtime Monitor**.

5. To view the request that was sent, click the **Request** tab.

Monitor and Close Cached Asset Instances

When you test the **IBM MQ Native Send Receive** step, a runtime monitor in the **Response** tab lets you monitor asset instances. You can also manually close the assets.

The following graphic shows an example of the runtime monitor.

Screen capture of runtime monitor.

By default, the runtime monitor automatically removes assets that are closed. In the following procedure, you disable this behavior.

Follow these steps:

1. In the **Response** tab, click **Runtime Monitor**.
2. Clear the check box that appears inside the **Clear** button.
3. Execute the step again.
   The runtime monitor displays the assets that were created during the execution of the step.
4. View the following information:
   - **Name**: The name of the asset.
   - **Type**: The type of asset.
   - **Scope**: The name of the test step, test case instance, test case, or DevTest component that corresponds to the runtime scope of the asset. The value has a tooltip that shows the scope.
   - **Status**: The color green indicates that the asset is active. The color yellow indicates that the asset is idle. The color gray indicates that the asset is closed.
5. To display more information about an asset, click the status icon.
6. To close an asset immediately, click the status icon and select **Close** or **Force Close**.
7. To close a set of assets, click **Force Close**.

RabbitMQ Steps

The following step is available in the **RabbitMQ** category:

- RabbitMQ Send Receive Step

RabbitMQ Send Receive Step

This step lets you send and receive messages over RabbitMQ.

The following graphic shows the step editor. The advanced parameters are not shown.

Screen capture of RabbitMQ Send Receive step.

Each parameter has a tooltip that describes the purpose of the parameter.

Some parameters include the value **Automatic** as an option. This value indicates that the actual setting is taken from another parameter. If you click the drop-down arrow and you place the mouse pointer over the value **Automatic**, a tooltip displays the name of the other parameter.

For more conceptual information about RabbitMQ and the corresponding assets, see RabbitMQ Assets.

Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

- Send and Receive
- Message Properties
- Message Contents
- Correlation Schemes
- Test the RabbitMQ Send Receive Step
- Monitor and Close Cached Asset Instances

Send and Receive

The **RabbitMQ Send Receive** step has separate areas for configuring the send operation and the receive operation.

Specify two queues in the step editor: one for the send operation, and one for the receive operation. The lists are populated with the RabbitMQ queue and temp queue assets from the active configuration.

Each asset has a runtime scope. The **RabbitMQ Send Receive** step lets you specify a minimum runtime scope for the send and receive operations. The scope parameters are advanced parameters.

Message Properties

RabbitMQ has messages, which consist of header data and a payload.

Like JMS, you can add custom properties to the header data.

Like WebSphere MQ, the payload is text or binary. Internally, the payload is stored as binary.

Like JMS and WebSphere MQ, header fields are available for Message ID, Correlation ID, and ReplyTo.

Unlike JMS and WebSphere MQ, RabbitMQ does not specify anything about the Message ID, Correlation ID, and ReplyTo fields. These fields are provided as a convenience. RabbitMQ itself does not care how, or if, the fields are used.

With Message ID, this means that a message is not assigned a Message ID automatically. The application must fill in this field before sending the message, if it wants to. The application also determines the uniqueness of the Message ID.

With ReplyTo, it is again up to the application to determine how this this field is populated on the client side and interpreted on the service side. The field might contain the name of a response queue, or a routing key, or something completely different.

Message Contents

The **RabbitMQ Send Receive** step lets you specify the message content. The following types are supported:

- XML document
- DevTest property reference
- Text
- Graphic image
- Binary
- IBM RFH2 header
- IBM MQMD header
- JSON document
- EDI document

Correlation Schemes

The following correlation schemes are available:

- Default ReplyTo
- RabbitMQ Direct
- RabbitMQ Topic
- RabbitMQ Headers
- RabbitMQ Payload

Default ReplyTo

The simplest correlation scheme is not technically called that. It is the default behavior of the **RabbitMQ Send Receive** step.

The ReplyTo field of the request message is filled in with the name of the response queue, whether it is a temp queue or not, and it is assumed that the service will use the default exchange to send a response directly back to that queue. This corresponds to the typical ReplyTo scheme in other messaging providers.

RabbitMQ Direct and RabbitMQ Topic

In this correlation scheme, the request message's ReplyTo field contains the exact routing key that was used to bind the response queue to some direct or topic exchange. It is assumed that the service will copy the routing key used for its response from the ReplyTo field of the request. It is also assumed that the service knows which direct or topic exchange to use when sending the response.

For the purposes of RPC, we assume that direct and topic exchanges work basically the same. That is, the routing key is matched exactly between the response queue's binding and the response message.

If no explicit exchange is provided on the response queue, it will use **amq.direct** or **amq.topic** by default. It also can generate a routing key, and will set up the message and queue binding parameters automatically. This overrides the default ReplyTo behavior because it uses the same ReplyTo field on the request message.

Direct and Topic are two separate correlation schemes in the **RabbitMQ Send Receive** step, but that is just to avoid confusion. The schemes are almost identical, with the exception of the default exchange name.

RabbitMQ Headers

In this correlation scheme, the request message contains one property used for correlation. It is assumed that the service will copy the property value verbatim to the response message. It is also assumed that the service knows which headers exchange to use when sending the response.

If no explicit exchange is provided on the response queue, it will use **amq.headers** by default. It also can generate a value, and will set up the message and queue binding parameters automatically. This overrides the default ReplyTo behavior because headers exchanges do not use routing keys.

You only need to provide the name of the property, and the name of the property on the response message if different from the request.

RabbitMQ Payload

Like JMS and MQ, we have payload-based correlation with RabbitMQ. This scheme works exactly like it does with the other providers.

Test the RabbitMQ Send Receive Step

You can verify the functionality of the **RabbitMQ Send Receive** step in the editor.

The execution log provides a high-level view of the activity that happens behind the scenes.

Follow these steps:

1. Click the green **Execute** button in the step editor.
2. To cancel the step while it is executing, click **Cancel**.
   The **Response** tab shows the response that was received when the operation finishes.
3. To view the step activity, click **Execution Log**.
4. To monitor the asset instances, click **Runtime Monitor**.
5. To view the request that was sent, click the **Request** tab.

Monitor and Close Cached Asset Instances

When you test the **RabbitMQ Send Receive** step, a runtime monitor in the **Response** tab lets you monitor asset instances. You can also manually close the assets.

By default, the runtime monitor automatically removes assets that are closed. In the following procedure, you disable this behavior.

Follow these steps:

1. In the **Response** tab, click **Runtime Monitor**.
2. Clear the check box that appears inside the **Clear** button.
3. Execute the step again.
   The runtime monitor displays the assets that were created during the execution of the step.
4. View the following information:
   - **Name**: The name of the asset.
   - **Type**: The type of asset.
   - **Scope**: The name of the test step, test case instance, test case, or DevTest component that corresponds to the runtime scope of the asset. The value has a tooltip that shows the scope.
   - **Status**: The color green indicates that the asset is active. The color yellow indicates that the asset is idle. The color gray indicates that the asset is closed.
5. To display more information about an asset, click the status icon.

6. To close an asset immediately, click the status icon and select **Close** or **Force Close**.

7. To close a set of assets, click **Force Close**.

SAP Steps

The following steps are available in the **SAP** category:

- SAP RFC Execution
- SAP IDoc Sender
- SAP IDoc Status Retriever

SAP RFC Execution

The SAP RFC Execution step lets you connect to an SAP system to execute an RFC (Remote Function Call).**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

To create an SAP RFC Execution step:

1. Select a **Destination** asset that includes the connection information for either an application server or a message server.

2. If you are using a message server:
   In Windows, add the following line to the end of the C:\Windows\System32\drivers\etc\services file: *sapmsCR2 3600/tcp.*
   In Linux, add the following line to the end of /etc/services: *sapmsCR2 3600/tcp*.

SAP RFC Execution step connection screen

3. Enter the following parameters in the SAP RFC Execution step editor. You can use properties for RFC input parameters.

   - **RFC name filter**
     To filter the RFC function names, enter a filter value, which can include a wildcard character of *. When you click the **RFC name** field drop-down arrow, DevTest requests the SAP system to retrieve the RFC names that the filter qualified, then populates the field.
     After you select or enter an RFC function name, DevTest sends a request to the SAP system to retrieve the RFC function input parameters. Both **Key** and **Description** of each parameter are returned from the SAP system. Although the description is available for most parameters, not all of them have it. You can enter the parameter values before executing the RFC function.

- **RFC name**
- **Import Parameters**
  - **Name**
    The name of a parameter
  - **Description**
    The description of a parameter
  - **Value**
    The value of a parameter
    Each table cell in the value column is a drop-down list of the available properties.
- **Table Parameters**
  - **Name**
    The name of a parameter
  - **Description**
    The description of a parameter
  - **Value**
    The value of a parameter
    Each table cell in the value column is a drop-down with the available properties to select from.
- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

4. Click **Test** to populate the **Response** tab, which contains subtabs for the output for XML and DOM Tree.

SAP IDoc Sender

The SAP IDoc Sender step lets you connect to an SAP system to send SAP IDocs.**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

To create an SAP IDoc Sender step:

1. Select a **Destination** asset that includes the connection information for either an application server or a message server.
2. If you are using a message server:
   In Windows, add the following line to the end of the C:\Windows\System32\drivers\etc\services file: *sapmsCR2 3600/tcp.*
   In Linux, add the following line to the end of /etc/services: *sapmsCR2 3600/tcp*.
3. Enter the following parameter in the SAP IDoc Sender step editor.
   - **If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

Default: Abort the test.

4. To test the connection to the SAP server, click **Test Connection**.

5. To display the file locator, click **Read IDoc From File**. Navigate to the location of your IDoc and select it.
   SAP IDocs are supported in XML and raw text formats.

SAP IDoc Status Retriever

The SAP IDoc Status Retriever step lets you connect to an SAP system and poll SAP IDocs status periodically until the IDocs completes or a specified interval ends.

Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

To create an SAP IDoc Status Retriever step:

1. Select a **Destination** asset that includes the connection information for either an application server or a message server.

2. If you are using a message server:
   In Windows, add the following line to the end of the C:\Windows\System32\drivers\etc\services file: *sapmsCR2 3600/tcp.*
   In Linux, add the following line to the end of /etc/services: *sapmsCR2 3600/tcp*.

SAP IDoc Status Retriever step base tab

3. Enter the following parameters in the SAP IDoc Status Retriever step editor. You can use properties for SAP input parameters.

   - **Timeout (secs)**
     The duration of polling operation for SAP IDoc status

   - **Polling Interval (secs)**
     The frequency of polling operation for SAP IDoc status
     For example, with the default values, the SAP IDoc Status Retriever step polls the IDoc status every 30 seconds for a total duration of 10 minutes. If the IDoc is completed any time in the duration, the SAP IDoc Status Retriever step stops immediately, without waiting for the entire 10 minutes.

- **SAP Transaction ID**
  An alphanumeric transaction ID or a DevTest property. If an SAP IDoc Sender step is used before this IDoc Status Retriever step, use the default property {{lisa.SAP_TRANSACTION_ID}}.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **If timeout**
  Select the step to be executed or action to take if there is a timeout.

4. To validate the connection to the SAP server, click **Test Connection**.

Selenium Integration Steps

The Selenium steps let you import test scripts for web-based user interfaces from Selenium Builder into CA Application Test. You can use the exported JSON test script to build test steps for testing a user interface.

To integrate with Selenium, complete the following tasks:

- Create and Export a Selenium Builder Recording
- Import a Selenium Builder Recording

Note: Browser support for recording Selenium Integration tests is limited to Mozilla Firefox. After you import these steps to CA Application Test, you can run the test cases on Google Chrome, Mozilla Firefox, Apple Safari, or Internet Explorer. For information about the versions of these browsers, see System Requirements.

You can also export a test case with Selenium steps to a JSON script. For more information, see:

- Export a Test Case with Selenium Test Steps to a JSON Script

More Information:

- Run a Selenium Integration Test Case

Create and Export a Selenium Builder Recording

The first step in creating a Selenium test step in CA Application Test is to create a Selenium Builder test script. Selenium Builder is a Firefox add-on that lets you record actions in a web-based user interface and create Selenium tests.

The following video provides an introduction to integrating CA Application Test with Selenium Builder.

If Selenium Builder is not installed, download and install it from the following URL: http://seleniumbuilder.github.io/se-builder/.

> **Note:** We recommend that you turn off the automatic upgrade option:
>
> 1. Open the main Firefox menu and select **Add-ons**.
> The Add-ons Manager opens.
> 2. Click **Extensions** and double-click **Selenium Builder**.
> 3. Select the Off option for **Automatic Updates**.

For detailed information about creating Selenium Builder tests, see the Selenium Builder documentation at the following URL: http://seleniumbuilder.github.io/se-builder/.

Follow these steps:

1. Open the main Firefox menu and select **Developer**, **Launch Selenium Builder**.
 The **Selenium Builder** dialog opens.
2. Enter the URL of the web application that you want to test in the **Start recording at** field.
3. Click **Record**.
4. Perform the actions that you want to test.
5. Return to the Selenium Builder dialog and click **Stop recording**.
6. Click **File**, **Export**.
 The **Choose export format** menu opens.
7. Click **Save as JSON**.
8. Browse to the directory where you want to save the JSON test script.
9. Enter a name for the JSON test script and click **Save**.
10. Close the **Selenium Builder** dialog.

Import a Selenium Builder Recording

Once you have created and exported a Selenium Builder recording, you can import the JSON test script into DevTest Workstation. The result is a test case that contains one or more Selenium steps.

You cannot edit the JSON-related information after importing the JSON script to the Selenium script step.

Follow these steps:

1. Open an existing test case or create a new test case.

2. Perform one of the following actions:

   - Click **Selenium** .

   - Click **Add Step** , **Selenium**, **Selenium Import/Export**.

   The **Selenium Integration** dialog opens.

3. Click **Browse** and select the JSON test script.
   The tree structure is populated.

4. Select one of the following options:

   - **Selenium Step**
     Creates a separate test step for each action in the script. This option does not support Selenium Builder suites.

   - **Selenium Script**
     Creates one test step that includes all of the actions in the script. This option also supports Selenium Builder suites.

5. Click **Build**.
   One or more steps are added to the test case.
   The **Selenium Integration Import** dialog provides information about the import, such as the number of steps that were created. The dialog also notifies you of any warnings or errors that occurred.

6. Click **Close**.

7. Double-click each step to view the details in the right panel.

8. Complete the following fields for each step:

   - **Alert Behavior (Optional)**
     The Selenium step can provide "inline" alert handling. The fields in the **Alert Behavior** area work in parallel with the value of the **unexpectedAlertBehaviour** parameter, which is defined in the file that is named by the **selenium.WebDriver. DesiredCapabilities.filePath** property.
     These fields let you specify how the test step should react to a modal alert dialog displayed in the web application.

   - **Alert Action (Optional)**
     Defines the action to take in response to a modal alert dialog.
     Values: Accept, Dismiss, Answer, Ignore

   - **Input Text (Optional)**
     Defines the text to input when the **Alert Action** field is Answer. The step inputs the text that you put into this text box, then clicks **OK**.

   - **On Step Fail**
     Defines the action to take if a specific step in the test case fails. Select the step to execute (Go to:) or the action to take if the step fails. For more information, see Configure Next Step and Generate Warnings and Errors.

> **Note:** The JSON script supports variable substitution by property for any value that you input. You can also use properties with encrypted values, such as **{{password_enc}}**, to avoid exposing sensitive data.

The following Selenium Builder steps are mapped to your test case as described:

- **Store**
  The name/value pair becomes a standard property in your test case. The name is prefixed with **selenium** to distinguish it from other properties. For example, the following JSON definition becomes a new property with the name **selenium.window_title**. The value is populated after the step runs.

  ```
  {
    "type": "storeTitle",
    "variable": "window_title"
  },
  ```

- **Verify**
  The verify step in Selenium Builder is used to validate user interface elements. If the validation fails, the state of the step is set to ERROR, but the test case execution flow continues to the next step. An associated DevTest error event (in red) is created for the failure.

- **Assertion**
  The assertion step in Selenium Builder validates user interface elements. If the validation fails, the state of the step is set to FAIL, and the test case execution flow stops. An associated DevTest error event (in red) is created for the failure.

- **saveScreenshot**
  If a saveScreenshot step in your script does not include a full path for saving the screen shot, DevTest tries to create the file under a $LISA_HOME\tmp\selenium directory. You can also use variables for the supplied file name. For example:
  c:\testcase1\snapshot1-{{LISA_TEST_RUN_ID}}.png
  or
  c:\{{testCase}}\{{LISA_TEST_RUN_ID}}\snapshot1.png
  If any part of the parent directory for the target file does not exist, that portion of the directory is automatically created. If the target file already exists, it is deleted before writing new data.

Install the CA Application Test for Selenium Builder 1.0 Plugin

The following actions in DevTest Workstation require the CA Application Test for Selenium Builder 1.0 plugin:

- Edit a Selenium test case in the test step editor
- Export a Selenium test case

Prerequisites

Prerequisites for using the CA Application Test for Selenium Builder 1.0 plugin include:

- Firefox version 43 or higher

- If you have a version of Selenium Builder installed prior to 3.0.5, you must uninstall it.
  Follow these steps:
  1. From the Tools menu in Firefox, click Add-ons.
  2. Select Selenium Builder, then click Remove.
  3. Restart Firefox.

- Install the Selenium Builder add-on for Firefox (in the Firefox browser). Make sure that you run the add-on at least once before installing the CA Application Test for Selenium Builder 1.0 Plugin. From the Firefox menu, select **Developer, Launch Selenium Builder**.

Installation

Follow these steps:

1. Go to the **LISA_HOME\addons\sebuilder-plugin** directory.

2. Copy the entire **lisa** directory, and its contents, from **LISA_HOME\addons\sebuilder-plugin**.

3. Go to your [Firefox profile]\SeBuilder3\plugins directory. To find your Firefox profile:

   a. For Firefox 3.6 and later, from Firefox, select Help, Troubleshooting information.

   b. Under Application Basics:

      - On Windows and Linux, depending on the Firefox version, click Show Folder (Windows), Open Directory (Linux), or Open Containing Folder.

      - On OS X, click Show in Finder.

> **Note:** The Firefox menu bar contains the File, Edit, View, History, Bookmarks, Tools, and Help menu items. On Windows, the menu bar may be hidden. To show a hidden menu bar temporarily, click the Alt key.

4. Paste the **lisa** directory to the [Firefox profile]\SeBuilder3\plugins directory.

5. Verify that the plugin was successfully installed. From the Firefox menu, select **Developer, Launch Selenium Builder** to view the Plugins directory.

Screen capture of Plugins directory.

Edit Selenium Test Steps Within CA Application Test

Once you install the CA Application Test for Selenium Builder 1.0 plugin, extra functionality is available for editing a Selenium recording. DevTest Workstation lets you edit the Selenium test once screenshots and screen JSON data are captured. Without this plugin, editing is best done via Selenium Builder (via exporting and re-importing the Selenium script).

Follow these steps:

1. Make sure that the CA Application Test for Selenium Builder 1.0 plugin is installed.

2. Once a test is imported and displays in the Test Editor, hover over a test step to see the screen shot for the step.

3. Click the test step, then click the **Selenium Step** tab in the right panel.
   The details for the step open, including the JSON script (for reference), its action (or gesture), and image. More fields display depending on the action type.
   Actions include:

   Get

   Loads the webpage into the browser.

   GoBack

   Simulates the user clicking the Back button on their browser.

   ClickElement

   Clicks in the middle of the given element.

   DoubleClickElement

   Performs a double-click at the middle of the given element.

   SendKeys

Lets you send keystrokes to the active window just as if you had manually typed them on the keyboard.

SetValue

Ensures focus on an input field and sets its value.

ElementPresent

Verifies that the specified element is somewhere on the page.

ElementValue

Verifies the value of the element, such as text in an input field, or true/false for a check box.

Switch to Frame

Retargets script execution to the specified iFrame.
Switch to Window
Retargets script execution to the specified browser window

4. Some actions enable editing for XPath expressions. Click the **Locator** link to open the XPath editor.

   The left side of the dialog displays a tree that shows the UI hierarchy in the current test step. The middle tree displays the attributes of the currently selected element or, for a multiple selection, attributes that are common to all selected elements. These attributes are informational only. The right side displays a screenshot of the current step.

   a. Click an element in the left-side tree or right-side image.

   b. Edit the XPath expression in the **Expression** field for the selected element.

      The XPath expression is validated while typing. If the XPath expression is empty or invalid, an error message displays.

   c. Click **Apply** if you edited the XPath expression.

      The tree and screenshot are updated with the new expression.

5. To add a new action, right-click a screen element and select an option from the menu. You can drag-and-drop the actions to change the order.

As you edit the test step, the JSON object script expands to accommodate your updates.

Export a Test Case with Selenium Test Steps to a JSON Script

You can export a test case with Selenium steps to a JSON script that can be stored on the file system.

Before you export a Selenium test case, you must install the CA Application Test for Selenium Builder 1.0 plugin in the Firefox Selenium Builder plugins directory.

After you export a test case, you can use Selenium Builder to rerecord web applications. You can then reimport the newly generated JSON from Selenium Builder to DevTest Workstation and merge the changes into an existing test case.

Load exported JSON scripts into Selenium Builder using the **Open a script exported by CA Application Test** menu item. This menu item is available after you install the DevTest plugin.

After you load the JSON script into Selenium Builder, you can make updates using regular Selenium Builder functionalities, such as:

- Edit step
- Add step
- Delete step
- Change the execution order of steps

Follow these steps:

1. Make sure the CA Application Test for Selenium Builder 1.0 plugin is installed.
2. Open an existing test case.
3. Perform one of the following actions:
   - Click **Selenium** .
   - Click **Add Step** , **Selenium**, **Selenium Import/Export**.

   The **Selenium Integration** dialog opens.
4. Click the **Export** tab.
5. Click **Browse** next to the **Output JSON Script** field and browse to the location of the JSON test script that you want to export, or enter a path and file name.
6. Click **Save**.

Virtual Service Environment Steps

The CA Service Virtualization test steps are described in Editing a VSM.

CAI Steps

The following steps are available in the **CAI** category:

- Execute Transaction Frame
- Execute webMethods Transaction Frame

Execute Transaction Frame

The Execute Transaction Frame step lets you run a transaction frame on a DevTest Java Agent.

This step is useful when you want to verify the functionality in a system under test, but you do not have a public access point. In addition, this step can be automatically included in baselines.

This step can be used on any type of transaction frame that the agent can capture.

> **Note:** If you are using webMethods, use the Execute webMethods Transaction Frame step instead.

If you manually add the step to a test case, do either of the following:

- Import a transaction
- Specify the property in which to find the frame

If you do both, DevTest examines the property first and uses the imported transaction only if the property does not refer to a frame, either as an actual frame or as an XML representation of it. The frame can be imported from an XML file or from a zip file. You can obtain frames by exporting from DevTest Portal.

The following graphic shows the initial view of the step. To specify the frame, click **Import**. To have the import process create an assertion on the response for the imported frame and add it to the step automatically, select the **Assert** check box before you click **Import**.

Execute Transaction Frame step

After a successful import, three tabs are added:

- **State**
- **Request**
- **Response**

You can now configure and invoke the frame.

The following graphic shows the step, after a frame was imported.

Execute Transaction Frame step after a transaction has been imported

The upper portion contains the following items:

- **Executing agent**
  The agent that runs the frame. The color of the agent in the drop-down list indicates whether it is active. Execution from this editor works only if the agent is active. To refresh the contents, click **Refresh** to the right of the drop-down list. A property can also be specified.

- **Frame property**
  The property from which the frame to execute is obtained when the step is run. This field is automatically filled in for consolidated baseline tests that DevTest Portal creates.

- **Class name**
  The name of the class of the method.

- **Method name**
  The name of the method that the agent intercepted.

The **State** tab contains metadata that the underlying protocol requires. For example, the state for an EJB frame includes the JNDI lookup information. You can change the state before execution.

The **Request** tab contains the input to the method for the frame. You can change the request before execution.

In the **State** and **Request** tabs, the heading bar indicates the type of payload that is being shown. You can change the payload type by clicking the icon in the right portion of the heading bar and selecting the appropriate type.

The **Response** tab compares the expected response from invoking the frame with the actual response. The format is the same as the Graphical XML Side-by-Side Comparison assertion.

The lower portion contains the following items:

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **Execute**
  Invoke the frame on the currently selected agent. You can use this button to test the step "in place" in the editor.

- **Import**
  Import a frame.

- **Assert**
  Add an assertion to the step that verifies the response. The assertion is when you import a frame.

- **Clear**
  Remove the imported frame.

Execute webMethods Transaction Frame

The Execute webMethods Transaction Frame step is based on the Execute Transaction Frame step. The upper portion contains the following additional items:

- **Integration Server Port**
  The port number of webMethods Integration Server.

- **Use SSL**
  Indicates whether to use SSL to connect to webMethods Integration Server.

Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements.

Custom Extension Steps

The following steps are available in the **Custom Extensions** category:

- Custom Test Step Execution
- Java Script Step (deprecated)
- Execute Script (JSR-223) Step
- Opaque Data Listener Step

Custom Test Step Execution

The Custom Test Step Execution step runs a test step that is written by your team using the Software Developer Kit (SDK). This step is documented in Using the SDK.

Java Script Step (deprecated)

> **Note:** The Java Script step has been deprecated. Use the Execute Script (JSR-223) step instead.

The Java Script step gives 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 properties in the test case, including built-in objects.

Prerequisites: Some knowledge of BeanShell. For more information about BeanShell, see http://www.beanshell.org/.

The step includes a script editor. Double-clicking an item in the **Available Objects** list pastes that variable name into the editor.

The last value that is exposed in the script is saved as the response of this step.

The following graphic shows the script editor. The script contains the following statements:

- A new Date object is created, initialized to the current date and time.
- This object is stored in a new property, **newProp**, using one of the DevTest exposed objects, testExec. For more information, see Using the SDK.
- The **toString()** value of the Date object is set as the response of the step.

The **Test** button lets you test the script. You see the result from executing the script, or an error message describing the error that occurred.

DevTest property name syntax is flexible and can include spaces. The property names that are not valid Java identifiers are converted for use in this step. An underscore (_) replaces any invalid characters.

If you use properties **{{exampleprop}}** in a script, DevTest substitutes the properties for the actual property values at run time before it runs the script.

Properties with "." in their names are imported into the script environment with "." replacing "_". So **{{foo.bar}}** in a script is the same as **foo_bar**.

You can produce a log event inside a script step or assertion. A **testExec** object is useful. To produce a log event, code the following instead of using the log4j logger. The **testExec.log()** object causes an actual event to be raised and you can see it in the ITR.

```
testExec.log("Got here");
```

A test case run has only one scripting instance. If there are multiple Java scripting steps, whether in the same test case or in subprocesses, DevTest uses the same instance to run the entire test case.

By default, variables are global to the instance. This behavior extends to subprocesses.

If you want the scope of a variable to be local, place the code inside an opening curly brace and a closing curly brace. For example:

```
{
     String var= "local";
     return var;
}
```

Parameter names for subprocesses are treated as global variables.

Execute Script (JSR-223) Step

The Execute Script (JSR-223) step lets you write and run a script to perform some function or procedure. You can select from:

- Applescript (for OS X)
- Beanshell
- Freemarker
- Groovy
- JavaScript
- Velocity

To use additional scripting languages, see Enabling Additional Scripting Languages.

You have access to all the properties in the test case, including built-in objects.

Complete the following fields:

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

  Default: Abort the test.

- **Script Language**
  Designates the scripting language to use.
  Values:

  - Applescript (for OS X)
  - Beanshell
  - Freemarker
  - Groovy
  - JavaScript
  - Velocity

Default: Beanshell

- **Copy properties into scope**
  Allows you to specify which properties to download for use in the step.
  Values:
  - Test state and system properties: all properties for the test case and system
  - Test state properties: properties that provide information about the test case
  - TestExec and logger only: only the TestExec and logger properties

  Default: Test state properties

Click **Test** to open a window with the result of the script execution or a description of the errors that occurred.

Anywhere you use {{expressions}}, you can specify a scripting language by using this syntax:

```
{{=%language%}}
```

Examples are in the **scripting** test case in the Examples project.

Set the default scripting language by using the lisa.scripting.default.language property.

> **Note:** Instances of TestExec object are not allowed to be added to the state map (using TestExec.setStateValue method in the script).

Opaque Data Listener Step

Use the Opaque Data Listener step to simulate TCP/IP connections to a server application. This step is an extension of the Virtual TCP/IP Listener step. The step listens for incoming TCP/IP traffic and converts it to a request format that the ODP algorithm understands to determine the response at playback time.

Complete the following fields:

- **Listen port**
  Enter the port on which DevTest listens for the Opaque Data traffic.
- **Bind address**
  Enter the local IP address on which connections can come in. With no bind address specified, the listen step accepts connections on the specified port regardless of the NIC (or the IP address) on which it comes in.
- **Use SSL to client**
  Specifies whether DevTestuses a custom keystore when acting as an SSL server during recording and playback.
  - If you select **Use SSL to client,** an HTTPS connection is used between the client and virtual service. You can also specify a custom server-side keystore file and password that is associated with it.

  - If you do not select **Use SSL to client**, an HTTP connection is used between the client and virtual service.

- **SSL keystore file**

- Specifies the name of the server-side keystore file. DevTestuses this keystore when handling HTTPS connections from the client.

- By default, the **ssl.server.cert.path** property is used for the server-side keystore file. This property can also be selected from the **Defaults** section of the drop-down list.

- **Keystore password**

- Specifies the password that is associated with the specified server-side keystore file.

- You can specify the password directly using the Password Editor option, or you can use the DevTestProperty Reference Editor option to specify a property expression that will be evaluated to provide the password.

- By default, the property expression **{{ssl.server.cert.pass}}** is used for the server-side keystore password. This property expression can be selected from the **Defaults** section of the drop-down list.

- **Treat request as text**
  Specifies whether the request is treated as text. For more information, see Virtualize TCP.

- **Request Encoding**
  Lists the available request encodings on the machine where DevTest Workstation is running. The default is UTF-8.

- **Request Delimiter**
  Select the request delimiter that you want to use. Some delimiters require you to enter additional information. For example, if you select **Regular expression matches the record**, you must also specify the regular expression to match.

- **Format step response as XML**

  The VSE framework expects Response steps to accept one of the following:

  - A response object

  - A list of response objects

  - An XML document that represents either

  > If this check box is cleared, the step produces a list of response objects. The step produces the list, even if it contains only one response.

  Default: The step response is formatted as XML.

- **If Environment Error**

  Specifies the action to take or the step to go to if the test fails because of an environment error.

Default: Abort the test.

Connect

This section provides quick links to the DevTest Solutions Community, Support pages by product, CA Education offerings, and other resources.

User Community

Support

- CA Application Test
- CA Service Virtualization
- Application Insight

Webcasts, Success Stories, White Papers, Demos, and More

- CA Application Test
- CA Service Virtualization

DevTest Solutions Education and Training

Just one hour of training saves five hours of lost productivity. Trained users make 35% fewer support calls. A well-trained team also delivers more value from their technology investments.

Ensure that your team can leverage your DevTest investment to meet your business needs today and into the future through CA Education training offerings and CA Learning Subscriptions.

- Enable Your Teams with CA Education
- Learning Paths
- CA Learning Subscription
- Free Training

Enable Your Teams with CA Education

Your CA software investment helps you break through and produce measurable outcomes. If you are looking to do any of the following, CA Education can help.

- Lower the risk of product failure

- Improve user adoption and productivity
- Reduce reliance on external support
- Accelerate the ROI of your CA investment

Learning Paths

The DevTest Solutions Learning Paths contain recommended courses by role, for both functional and technical users. Courses are updated based on impact from release and demand, which is why you might see courses from previous releases included in the current learning path. You can click a course from the learning path to get started with your learning experience.

Courses are offered in a variety of self-paced delivery options, as well as traditional instructor and virtually led classes. Visit the DevTest Solutions Learning Path for a complete list of offerings.

CA Learning Subscription

CA Learning Subscriptions provide access to an extensive library of videos, e-learning courses, dynamic labs, and more. You can learn what you need to know, when and where you need it. Content is added and updated regularly, so you always have access to the latest courses and training materials.

Subscriptions are available in two editions:

- **Get Started**
  Unlimited access to rich, engaging content based on real-world use cases.
  - Videos
  - e-Learning Courses

  Cost: $700/year per user

- **Learn More**
  All the features of Get Started plus dynamic labs for hands-on experience and exclusive access to expert-led sessions.
  - Videos
  - e-Learning Courses
  - Dynamic Labs
  - Expert Sessions

  Cost:$2,200/year per user

Free Training

Learn about free training, as well as how to personalize your learning experience. See CA Education and Training.