

DevTest Solutions - 8.4

DevTest Solutions - Home

Date: 29-Jun-2016



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 © 2016 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Table of Contents

Release Information	38
8.4 Enhancements	38
8.4 CA Application Test Enhancements	39
8.4 CA Service Virtualization Enhancements	40
8.4 CA Continuous Application Insight Enhancements	41
8.4 Mobile Testing Enhancements	42
8.4 General Enhancements	43
8.4 Known Issues	43
End of Life Features	44
End of Life Features as of DevTest Solutions 8.2	44
End of Life Features as of DevTest Solutions 8.0.2	44
End of Life Features as of DevTest Solutions 8.0	44
End of Life Features as of LISA 7.5	45
End of Life Features as of LISA 7.1	45
End of Life Features as of LISA 6.0.9	45
Preview Features	46
8.4 Preview Features	47
DevTest Portal Functionality	48
CA Application Test	48
CA Service Virtualization	48
 Installing	 50
Preinstallation	50
System Requirements	50
Operating System Requirements	50
Supplying Your Own JVM	52
DevTest Server Requirements	55
DevTest Workstation for CA Application Test Requirements	56
CA Service Virtualization Requirements	56
CA Continuous Application Insight Requirements	57
Communication Requirements	57
Database Requirements	57
Supported Browsers	60
CAI Support Matrix	60
CAI Versions	61

CAI Agents	62
Java SE and EE JVMs	62
Operating Systems	62
Application Servers	63
Visible Transaction Protocols	64
Virtualization Features	65
Baseline Features	67
Planning Your DevTest System	68
DevTest Components	69
About DevTest Server Components	71
DevTest Process Relationships	72
DevTest Solutions Architecture	73
CA Application Test Architecture	73
CA Service Virtualization Architecture	75
CAI Architecture	75
Enterprise Dashboard Architecture	76
DevTest Server Components	77
Data Flow in DevTest Server for CA Application Test and CA Service Virtualization	78
Download DevTest Solutions Installers	80
DevTest Installation Overview	81
Installation Options	82
Options of Components to Install	82
Typical Configurations	82
Options for Installing Required Processes (Windows)	83
How to Install and Set up DevTest Solutions	84
How License Activation Works	85
Installing and Configuring DevTest Server	86
Install DevTest Server on Windows	86
Install DevTest Server on UNIX	90
Install DevTest Server on a Mac	92
Activate the Registries	95
Verify Registry Activation	96
Post-Installation	96
Configure Existing Registries	96
Using an HTTP/S Proxy Server - DevTest Server	99
Running Components on Different Systems	101
Calculate Simulator Instances	101
Load and Performance Server Sizing	102
Using DevTest Workstation with Your Java Environment	102
Change the Default Project Home	103
Project Directory Structure	104
Uninstall DevTest Server	104

Installing DevTest Workstation	105
Install DevTest Workstation on Windows	105
Install DevTest Workstation on UNIX	107
Install DevTest Workstation on a Mac	109
Using an HTTP/S Proxy Server - DevTest Workstation	111
Environment Settings	112
Installing the Demo Server	113
Verifying the DevTest Solution Installation	114
Start DevTest and Log In to UIs	114
Start the DevTest Processes or Services	115
Start Order Sequence	115
Ways You Can Start DevTest Server	116
Log in as the Standard Super User	118
Create a User with the Super User Role	119
Access the DevTest User Interfaces	120
Demo Server	120
Enterprise Dashboard	120
Portal	120
Workstation	121
DevTest Console	121
Installing Integration Tools	121
Install Performance Monitor (Perfmon)	121
Install and Configure SNMP	122
SNMP support on UNIX	123
SNMP support on Windows	123
Run TCPMon	124
Using TCPMon as an Explicit Intermediary	125
Using TCPMon as a Request Sender	125
Install the HP ALM - Quality Center Plug-in	125
Using the HP ALM - Quality Center Client to Run Test Cases	127
Install IBM Rational Quality Manager	127
Implementation	128
Install the Adapter UI	128
Run Command Line Adapter	129
General Usage Workflow	129
Configure Integration with CA APM	130
Instrumenting DevTest Processes	130
Introscope Agent Files	132
Metrics Reported by Tracers	132
Deriving DevTest Metrics	134
Tracer Configuration	134
Install Introscope Agent	136

Configuring Tracer Logging	136
Typical Metrics Reported	137
Displaying Metrics	139
Reporting	146
Set Up the SAP System Landscape Directory	146
Manually Register DevTest with SLD	147
Import the DevTest SLD XML File	147
Setting Up the Mobile Testing Environment	148
Supported Operating Systems for Mobile Testing	148
Supported Mobile Operating Systems	148
Hardware Requirements for Mobile Testing	149
Preinstallation Steps for Mobile Testing	149
Download the Latest Version of Android Studio	149
Define ANDROID_HOME	150
Configure the Android SDK Manager	150
Create an Android Virtual Device	151
Install Xcode Version 6.2 (Mac Only)	152
Prepare IPA Signing (Mac Only)	152
Enable UI Automation (Mac Only)	154
Using Genymotion	154
Managing Genymotion Devices	155
Define VBOXMANAGE_CMD	156
Installing DevTest Solutions with a Silent Install	157
Sample response file	158
Docker Containers	158
Docker Prerequisites	158
Build Baseline Docker Images	159
Docker Images	159
Example	160
Migrating DevTest Solutions	160
Key Points to Understand for DevTest 8.X	160
Supported Upgrade Paths	162
Migration System Requirements	162
Upgrading to DevTest Solutions 8.X	162
The Upgrade Process	162
Clearing Browser Cache between Upgrades	164
Licensing	164
User Preferences	164
Configuration	164
Reporting Upgrades	166
Database	166
CA Continuous Application Insight Upgrades	167

Continuous Validation Service	169
Enterprise Dashboard	169
Upgrading through Major Versions	170
Upgrading DevTest Workstation	170
Project Panel Changes	170
JDBC Virtualization Approach	171
New and Deprecated JMS Components	171
New and Deprecated WebSphere MQ Components	172
Test Invocation	173
The Negative Testing Companion	173
Model Archive (MAR) Architecture and Staging Tests	173
Web Services Testing	174
Web 2.0 Testing	174
Metric Name Changes in LISA 5	175
Test Timeouts	175
Subprocess Migration	175
Upgrading VSE	175
Upgrading Models	175
Upgrading Service Images	175
New Service Image Editor	176
Known Migration Issues	178
8.0-Specific Upgrade Information	178
Pre-Release 6 to 8.0 Upgrade	178
6.x or Later to 8.0 Upgrade	178
Migrating the SDK	180
Upgrading SDK Components	180
SDK Documentation	180
Java Version Notes	180
Assertion Updates	181
4.0 to 4.5	182
4.5 to 4.6	182
4.6 to 5.0	182
5.0 to 6.0	182
6.0 to 6.1	182
6.1 to 7.0	182
7.0 to 7.1	182
7.1 to 7.5	182
7.5 to 8.X	183
Companion Updates	183
4.0 to 4.5	183
4.5 to 4.6	183
4.6 to 5.0	183

5.0 to 6.0	183
6.0 to 6.1	184
6.1 to 7.0	184
7.0 to 7.1	184
7.1 to 7.5	184
7.5 to 8.X	184
Filter Updates	184
4.0 to 4.5	184
4.5 to 4.6	184
4.6 to 5.0	184
5.0 to 6.0	185
6.0 to 6.1	185
6.1 to 7.0	185
7.0 to 7.1	185
7.1 to 7.5	185
7.5 to 8.X	185
Step Updates	186
4.0 to 4.5	186
4.5 to 4.6	186
4.6 to 5.0	186
5.0 to 6.0	186
6.0 to 6.1	186
6.1 to 7.0	186
7.0 to 7.1	186
7.1 to 7.5	186
7.5 to 8.X	187
Data Protocol Updates	187
5.0 to 5.0.25	187
5.0 to 6.0	187
6.0 to 6.1	187
6.1 to 7.0	187
7.0 to 7.1	187
7.1 to 7.5	188
7.5 to 8.X	188
Transport Protocol Updates	188
5.0 to 6.0	188
6.0 to 6.1	188
6.1 to 7.0	188
7.0 to 7.1	188
7.1 to 7.5	189
7.5 to 8.X	189
Body Carrier Updates	189
6.0.7	189

6.0 to 6.1	189
6.1 to 7.0	189
7.0 to 7.1	190
7.1 to 7.5	190
7.5 to 8.X	190
Request Updates	190
5.0 to 6.0	190
6.0 to 6.1	190
6.1 to 7.0	190
7.0 to 7.1	190
7.1 to 7.5	190
7.5 to 8.X	191
Response Updates	191
5.0 to 6.0	191
6.0 to 6.1	191
6.1 to 7.0	191
7.0 to 7.1	191
7.1 to 7.5	191
7.5 to 8.X	191
Transient Response Updates	192
5.0 to 6.0	192
6.0 to 6.1	192
6.1 to 7.0	192
7.0 to 7.1	192
7.1 to 7.5	192
7.5 to 8.X	192
TCP Delimiter Updates	192
6.0 to 6.1	193
6.1 to 7.0	193
7.0 to 7.1	193
7.1 to 7.5	193
7.5 to 8.X	193
API Notes	193
5.0 to 6.0	193
6.1 to 7.0	193
7.0 to 7.1	194
7.1 to 7.5	194
7.5.1	194
7.5.1 to 8.X	194
Getting Started	195

CA Application Test	195
CA Service Virtualization	195
CA Continuous Application Insight	195
DevTest Portal	196
Open the DevTest Portal	196
Navigating the DevTest Portal	197
Navigation Menu	197
Home Page	199
All Resources Window	200
Manage Projects Window	200
Customize the Home Page Dashboard	202
Update Preferences	203
Enable and Disable Preview Features	204
CA Application Test Tutorials	204
Tutorial 1 - Projects, Test Cases, and Properties	205
Step 1 - Start DevTest Workstation	206
Step 2 - Create a Project	206
Step 3 - Create a Test Case	206
Step 4 - Add a Property to the Project Configuration	207
Step 5 - Add a Test Step	207
Step 6 - Add a Log Message	208
Step 7 - Add a Second Log Message	208
Step 8 - Run the My Output Log Message Step	209
Step 9 - Observe Property Values	209
Step 10 - Run the Output Log Message Step	209
Review	210
Tutorial 2 - Data Sets	210
Step 1 - Create a Data Set	211
Step 2 - Create a Test Case	211
Step 3 - Add a Property to the Project Configuration	212
Step 4 - Add a Test Step for Output Log Message	212
Step 5 - Create Another Output Log Message Step	213
Step 6 - Execute the Test	213
Step 7 - Add the Data Set	214
Step 8 - Change the Data Set Behavior	214
Tutorial 2 - Review	215
Tutorial 3 - Filters and Assertions	215
Step 1 - Create a Test Case from an Existing Test Case	216
Step 2 - Change Action of Test Step	216
Step 3 - Add an Assertion	217
Step 4 - Test the Assertion	219
Step 5 - Add a Filter	221

Step 6 - Test the Filter	221
Tutorial 3 - Review	223
Tutorial 4 - Manipulate Java Objects (POJOs)	223
Step 1 - Create a Test Case	224
Step 2 - Create a Dynamic Java Execution Test Step	224
Step 3 - Make a Call on the Java Object	226
Step 4 - Add an Inline Filter	230
Step 5 - Verify the Property Created	231
Tutorial 4 - Review	232
Tutorial 5 - Run a Demo Server Web Application	232
Step 1 - Start the Web Application	233
Step 2 - Log in to the Web Application	234
Step 3 - Create an Account	235
Step 4 - Close an Account	237
Step 5 - Log Out from the Web Application	238
Tutorial 5 - Review	238
Tutorial 6 - Test a Website	238
Tutorial 6 - Part A - Record the Test Case	239
Tutorial 6 - Part B - Run the Test Case	247
Tutorial 6 - Part C - Modify Request Test Steps	248
Tutorial 7 - Test an Enterprise JavaBean (EJB)	253
Step 1 - Create a Test Case	254
Step 2 - Create a Configuration	254
Step 3 - Add an EJB Test Step	255
Step 4 - Connect to the Server	255
Step 5 - Locate the EJB Interface	256
Step 6 - Configure the EJB	257
Step 7 - Add an Assertion	259
Step 8 - Verify the Method Execution	261
Step 9 - Add Another EJB Test Step	262
Tutorial 7 - Review	264
Tutorial 8 - Test a Web Service	264
Step 1 - Create a Test Case	264
Step 2 - Add a Web Service Execution (XML) Test Step	265
Step 3 - Create a Web Service Client	266
Step 4 - Execute the Web Service Request	268
Step 5 - View the Request and Response	268
Tutorial 8 - Review	270
Tutorial 9 - Examine and Test a Database	270
Step 1 - Create a Test Case	271
Step 2 - Add Database Properties to the Configuration	272
Step 3 - Add a SQL Database Execution (JDBC) Test Step	272
Step 4 - Connect to the Database	273

Step 5 - Execute a SQL Query	274
Step 6 - Add an Assertion	275
Step 7 - Run the Test Case	277
Step 8 - Change the Assertion	278
Step 9 - Add a Filter	279
Step 10 - Test the Filter and Assertion	280
Tutorial 9 - Review	281
Tutorial 10 - Stage a Quick Test	282
Step 1 - Open the Test Case	282
Step 2 - Review the Test Case	282
Step 2 - Part A - Run a Quick Test	283
Step 2 - Part B - View Generated Reports	286
Tutorial 10 - Review	287
Mobile Testing Tutorials	287
Tutorial 1 - Record an iOS Test Case	287
Step 1 - Start DevTest Workstation	288
Step 2 - Create a Project	288
Step 3 - Create a Config File for an iOS Simulator	289
Step 4 - Create an iOS Simulator Asset	289
Step 5 - Record a Test Case	290
Step 6 - Run the Test Case in the ITR	291
Tutorial 2 - Record an Android Test Case	292
Step 1 - Start DevTest Workstation	292
Step 2 - Create a Project	293
Step 3 - Create a Config File for an Android Simulator	293
Step 4 - Create an Android Emulator Asset	293
Step 5 - Record a Test Case	294
Step 6 - Run the Test Case in the ITR	295
CA Service Virtualization Tutorial	296
Prerequisites	296
Process for Creating and Testing a VSI	297
Step 1 - Start DevTest Workstation	297
Step 2 - Start VSE	298
Step 3 - Start Demo Server	298
Step 4 - Run a Test Case	298
Step 5 - Create a Config File	299
Step 6 - Activate Config File	300
Step 7 - Configure the VSE Recorder	300
Step 8 - Record the Test Case	302
Step 9 - Deploy the VSM	304
Step 10 - Test Against the VSM	305
Review the Tutorial	306

CA Continuous Application Insight Tutorial	306
Step 1 - Increase the Default Capture Levels	306
Step 2 - Generate Transactions from the Demo Application	307
Step 3 - View the Transactions in the Analyze Transactions Window	308
Glossary	309
assertion	309
asset	309
audit document	310
companion	310
configuration	310
Continuous Service Validation (CSV) Dashboard	310
conversation tree	310
coordinator	310
data protocol	311
data set	311
de-identify	311
event	311
filter	311
group	311
Interactive Test Run (ITR)	312
lab	312
magic date	312
magic string	312
match tolerance	312
metrics	312
Model Archive (MAR)	313
Model Archive (MAR) Info	313
navigation tolerance	313
network graph	313
node	313
path	313
path graph	313
project	314
property	314
quick test	314
registry	314
service image (SI)	314
simulator	314
staging document	314
stitching	315
subprocess	315
test case	315

test step	315
test suite	315
think time	315
transaction frame	315
virtual service model (VSM)	316
Virtual Service Environment (VSE)	316

Using 317

Using CA Application Test	318
The Registry	319
Start the Registry	320
Create a Named Registry	320
Change the Registry	320
Coordinator Server	321
Create a Coordinator Server	321
Monitor Coordinator Servers	322
Simulator Server	322
Create a Simulator Server	323
Monitor Simulator Servers	323
Command-Line Utilities	324
Using the DevTest Portal with CA Application Test	324
Opening the DevTest Portal	325
Enabling and Disabling Tool Tips	325
Create an API Test	326
Monitor Tests	330
Monitor with CVS (Continuous Validation Service)	337
Manage Test Artifacts	346
Using the Workstation and Console with CA Application Test	348
DevTest Workstation	348
DevTest Console	349
Open DevTest Workstation	349
Open the DevTest Console	369
Building Test Cases	370
Building Documents	494
Working with Model Archives (MARs)	524
Running Test Cases and Suites	533
Cloud DevTest Labs	585
Continuous Validation Service (CVS)	598
Reports	598
Recorders and Test Generators	632
Mobile Testing	639

Advanced Features	657
Using CA Service Virtualization	661
Introduction to Virtualization	661
Types of Service Virtualization	662
Service Virtualization Overview	662
High-level Virtualization Steps	662
Virtualization of Messaging Systems	664
Installation and Configuration	666
How to Install CA Service Virtualization	666
How to Configure CA Service Virtualization	667
Install the Database Simulator	668
DDLs for Major Databases	670
Understanding CA Service Virtualization	677
How to Work with CA Service Virtualization	677
CA Service Virtualization Components	677
Virtual Service Models	678
Service Images	679
How Virtualization Works	680
Magic Strings and Dates	682
Understanding VSE Transactions	687
Match Tolerance	690
Track Transactions	692
Using the DevTest Portal with CA Service Virtualization	693
Opening the DevTest Portal	693
Enabling and Disabling Tool Tips	693
Virtualize Websites	694
Virtualize JDBC Traffic	703
Virtualize Using Request-Response Pairs	712
Editing Virtual Services	713
Deploy a Virtual Service	735
REST Data Protocol Handler	737
Running Live Requests in DevTest Portal	741
Session Viewing and Model Healing in DevTest Portal	749
Create Copybook Bundles	750
Using the Workstation and Console with CA Service Virtualization	758
Creating Service Images	759
Editing Service Images	906
Editing a VSM	934
De-identifying Data	969
Virtualizing a Service	970
VSE Manager - Manage and Deploy Virtual Services	995
Install VSE Manager	995

Configure VSE Manager	996
Import a LISA Project	997
Use the VSE Manager View	997
Properties View Integration	998
VSE Commands	999
VSE Manager Command - Manage Virtual Service Environments	999
ServiceManager Command - Manage Services	1001
ServiceImageManager Command - Manage Service Images	1001
VirtualServiceEnvironment Commands	1005
Java Agent VSE Properties	1006
Using CA Continuous Application Insight	1007
Getting Started with CA Continuous Application Insight	1008
CA Continuous Application Insight Prerequisites	1009
CA Continuous Application Insight Supported Platforms	1009
Open the DevTest Portal	1010
Business Transactions and Transaction Frames	1011
SAP NetWeaver Notes	1012
Configuring Agents	1013
Agent Status Indicator	1014
VSE Mode Indicator	1014
Filter Agents	1015
View Performance Data	1016
Configure Capture Levels	1017
View Agent and Broker Information	1019
Stop and Start Agent Transaction Capture	1020
View and Update Properties	1020
De-identifying Data in CAI	1020
Upgrade an Agent in the DevTest Portal	1021
Upgrade an Agent Manually	1022
Delete an Offline Agent	1022
Create and Manage Agent Groups	1023
Analyzing Business Transactions	1025
View Transaction Details	1027
Generate Artifacts for an Agent	1036
Export Path Diagram to PDF	1037
Merge Repeated Paths	1038
View Usage Counts for Identical and Partial Transaction Paths	1039
View Producer and Consumer Relationship	1040
Pin and Unpin Transactions	1042
Exclude Data from Agent Capture	1042
Share a Link	1044
Share Point of Interest	1044

Label the Links Between Transaction Frames	1045
Search and Filter Transactions	1045
Shelve Transactions	1054
Exporting and Importing Paths	1056
Working with Defects	1059
Using the Shelf	1061
Shelf Dialog	1062
Shelf Popup	1062
How to Work with Transactions in the Shelf	1063
Creating and Managing Tickets	1066
Disabling the Tickets Feature	1067
Create Tickets in an Application	1067
Email Settings for New Tickets	1069
Send Tickets to HP ALM - Quality Center	1069
Managing Tickets	1070
Configure Browsers to Support Screenshots	1076
Creating Virtual Services	1077
Consolidation of Transactions When Creating Virtual Services	1077
Create Virtual Services from EJB Transactions	1079
Create Virtual Services from Java Transactions	1081
Create Virtual Services from JCA Transactions	1083
Create Virtual Services from JDBC Transactions	1085
Create Virtual Services from JMS Transactions	1086
Create Virtual Services from REST Transactions	1088
Create Virtual Services from SAP ERPConnect Transactions	1089
Create Virtual Services from SAP IDoc Transactions	1092
Create Virtual Services from SAP JCo Transactions	1095
Create Virtual Services from TIBCO ActiveMatrix BusinessWorks Transactions	1099
Create Virtual Services from Web HTTP Transactions	1102
Create Virtual Services from Web Service Transactions	1103
Create Virtual Services from webMethods Transactions	1105
Create Virtual Services from WebSphere MQ Transactions	1107
VRS Files in CAI	1109
Creating Baselines	1110
Baseline Types	1111
EJB Baselines	1113
JMS Baselines	1118
REST Baselines	1121
TIBCO BusinessWorks Baselines	1126
TIBCO Enterprise Message Service Baselines	1127
Web HTTP Baselines	1128
Web Service Baselines	1130
webMethods Baselines	1135

WebSphere MQ Baselines	1137
Generic Baselines	1142
Database Validation	1144
Test Templates	1147
Run a Baseline	1149
Consolidated Baseline Tab	1150
Baseline Results Panel	1151
Dynamic Responses and Transaction Frame Steps	1153
Creating Data Sets	1153
Creating Request and Response Pairs	1154
Documenting Transactions for Testing	1156
How to Document a Transaction for a Manual Test	1157
Troubleshooting for Documenting Transactions	1165
Native MQ CAI	1165
WebSphere MQ User Privileges for Native MQ CAI	1167
Create the ITKO_PATHFINDER Queue	1167
Install the Native MQ CAI API Exit	1168
Configure the Native MQ CAI API Exit	1169
Configure the Agent Properties for Native MQ CAI	1170
Generate Transactions from the ITKO_PATHFINDER Queue	1172
Native MQ CAI Troubleshooting	1173
CAI Command-Line Tool	1174
Search Criteria	1175
Querying and Display	1176
Export and Import	1177
Baselines	1177
Virtualization Artifacts	1179
Agent Condition	1179
Capture Levels	1180
Miscellaneous	1180
Example: Display Root Transaction Frames	1181
Example: Display Transaction Frame Hierarchy	1181
Example: Generate a Baseline Suite	1181
Example: Generate a Stateful Baseline	1181
Example: Generate a Baseline with a Template	1181
Example: Generate Virtualization Artifacts	1182
Example: Check the Agent Condition	1182
Example: Configure the Capture Level	1182
Example: Search Transaction Frame	1182
Example: Generate Large Data Set File	1182
VS Traffic to CA Application Insight Companion	1182
CA Continuous Application Insight Agent Light	1183
Agent Light Architecture	1184

Agent Light Prerequisites	1185
Install the Agent Light	1185
REST API	1185
Command-Line	1186
Agent Light for webMethods HTTP	1187
Agent Light for JMS	1189
Agent Light for WebSphere MQ	1193
CAI Extension for Google Chrome	1196
Installing and Configuring the CAI Extension	1196
Using the CAI Extension	1199
Using CA Application Trace Kit	1201
Start the Application Trace Kit	1201
Live Paths	1202
Saved Paths	1203
Configure the Amount of Data Captured	1204
Configure Agent Properties	1205
View and Configure Logging	1205
View and Manage Classes	1205
Manage Files	1206
Execute Remote Commands from the Terminal	1207
CA Application Trace Kit Database Tools	1207
Explore the DevTest Database	1207
Explore Databases Connected to Agents	1208
Creating Java Agent Extensions	1209
Create, Build, and Upload the Extension	1209
Using Extensions to Reduce Noise	1209
Live Instances and OQL Queries	1210
Live Instances	1210
Query the Heap with OQL	1211
Using the SDK	1212
Examples and API Documentation	1212
The Integration API	1212
Integration API Concepts	1212
Integration Flow	1213
The Integration Process	1213
Integrating Components	1214
Integrate Server-Side Components	1214
Collect Transaction Information	1216
Integrators	1217
Handle Integrated Output	1218
Testing Integrated Components	1219
Integration Filters	1219

Integration Assertions	1220
Extending DevTest Solutions	1222
Extension Concepts	1223
Extending Test Steps	1228
Custom Java Test Steps	1229
Native Test Steps	1233
Extending Assertions	1235
Create an Assertion	1235
Deploy an Assertion	1237
Define and Test an Assertion	1237
Extending Filters	1238
Create a Filter	1238
Deploy a Filter	1240
Define and Test a Filter	1240
Custom Reports	1241
Create a Report Generator	1241
Deploy a Report Generator	1241
Use a Report Generator	1242
Custom Report Metrics	1242
Create a Report Metric	1242
Deploy a Report Metric	1243
Custom Companions	1243
Create a Companion	1244
Deploy a Companion	1245
Using Hooks	1245
Create a Hook	1246
Deploy a Hook	1246
Custom Data Sets	1246
Data Set Characteristics	1247
Create a New Data Set	1247
Deploy a New Data Set	1248
Java .NET Bridge	1249
com.itko.lisa.jdbridge.JDInvoker	1249
com.itko.lisa.jdbridge.JDProxy	1249
com.itko.lisa.jdbridge.JDProxyEventListener	1250

Agents 1253

DevTest Java Agent	1253
Java Agent Architecture	1254
Java Agent Components	1254
Java Agent Data Flow	1255

Java Agent Database Schema	1257
Java Agent Data Categories	1257
Java Agent Installation	1258
Install the Pure Java Agent	1259
Install the Native Agent	1259
Options for Agent Parameters String	1260
Java Agent Install Assistant	1261
Platform-Specific Agent Installation	1265
Wily Introscope Agent	1271
Start the Broker	1272
rules.xml File	1272
Managing Agents in Groups	1274
Protocol Weight Configuration	1275
Signature Specification	1276
Instrumentation Rules for VSE	1277
Configure a Database Sink	1279
Category Settings	1279
Add a Class for Tracking	1281
JNDI Priority	1281
Adding a Method to Intercept	1281
Excluding from Interception and Virtualization	1282
Excluding Methods, Classes, and Packages	1282
Excluding URLs	1283
Java Agent Auto-Response Generation	1284
Example Auto-Response Generation for EJB Calls	1285
Java Agent REST APIs	1285
REST API Categories	1285
View the REST APIs	1286
Example: Retrieve Performance Data	1287
Developing Against the Java Agent	1287
Agent General APIs	1288
Agent Discovery APIs	1289
Agent Transaction APIs	1293
Agent VSE APIs	1296
Agent API Examples	1299
Java Agent Extensions	1300
Extending the Agent	1300
Adding Tags to Transaction Frames	1303
Extending the Broker	1304
Extending Java VSE	1305
Load Balancers and Native Web Servers	1306
Java Agent Security	1307
Extensions	1307

Remote Code Invocation	1307
Configure the Java Agent to Use SSL	1308
Java Agent Log Files	1309
Java Agent Troubleshooting	1310
Error at startup: Error occurred during initialization of VM. Could not find agent library in absolute path...	1311
The agent exits immediately (or shortly after starting the process).	1311
The agent exits with the message "GetEnv on jvmdi returned -3 (JNI_EVERSION)".	1311
The agent exits with the message "UTF ERROR" ["../../src/solaris/instrument /EncodingSupport_md.c":66]: ..."	1311
Agent hangs or throws numerous exceptions at startup (LinkageErrors, CircularityErrors, and so on).	1312
Agent throws java.lang.VerifyErrors or hot swapping has no effect.	1312
Agent starts but the consoles or broker cannot see the agent.	1312
Agent causes some operations to time out.	1312
java.lang.NoClassDefFoundError on com.itko.lisa/remote/transactions/TransactionDispatcher. class	1313
Security-related exceptions are thrown when the agent is enabled.	1313
Abnormal resource consumption (CPU, memory, file handles, ...).	1313
I can see the agent started, but CAI data is missing or incomplete.	1314
I turned on Java VSE recording or playback, but VSE does not receive any requests from the agent.	1314
Other issues with Java VSE.	1314
Case functionality is not working.	1315
DevTest is configured to use an IBM DB2 database. In the DevTest Portal, I selected a JDBC node in a path graph. The Statements and Connection Url columns are blank.	1315
Received the following exception: com.itko.lisa.remote.transactions. TransactionSizeExceededException: could not XStream ... Serialization for this frame will be disabled.	1315
The VSE log includes the following error: com.itko.lisa.vse.stateful.protocol.java.listen. DefaultJavaPlaybackCallback - Timed out waiting on response.	1315
Mainframe Bridge	1316
Mainframe Bridge Rules	1316
Mainframe Bridge Properties	1317
DevTest CICS Agent	1318
How to Install the CICS Agent	1319
CICS Agent Storage Requirements	1323
CICS Exit Considerations	1324
Coexistence with Other Exits	1324
DevTest CICS Agent Exit Behavior Summary	1324
CICS Agent User Exits	1325
ITKOUEX1 - DevTest CICS Agent Initialization Exit	1325

ITKOUEX2 - DevTest CICS Agent Termination Exit	1326
ITKOUEX3 - LISA CICS COMMAREA Consolidation Exit	1327
CICS Agent Operation	1329
Starting the DevTest CICS Agent and Exits	1329
Stopping the DevTest CICS Agent and Exits	1329
Impact on Virtualized CICS LINK Commands	1330
Impact on Virtualized CICS DTP/MRO Commands	1330
Monitor Transaction (TKOM)	1331
CICS Agent Messages	1332
DevTest CICS Agent Message Descriptions	1334
DevTest LPAR Agent	1355
How to Install the LPAR Agent	1355
LPAR Agent Storage Requirements	1355
LPAR Agent Installation	1356
LPAR Agent Operation	1359
Starting the DevTest LPAR Agent	1359
Stopping the DevTest LPAR Agent	1360
Monitoring and managing the LPAR Agent	1360
LPAR Agent Messages	1361

Administering	1362
General Administration	1362
Default Port Numbers	1362
DevTest Server Default Port Numbers	1363
DevTest Workstation Default Port Numbers	1364
Demo Server Default Port Numbers	1364
Change Port Numbers for the DevTest Portal and DevTest Console	1365
Shared Installation Type	1365
Directory Structure	1366
DevTest Workstation Directories	1367
DevTest Server Directories	1368
Running Server Components as Services	1370
Running DevTest Solutions with Ant and JUnit	1371
Run DevTest Tests as JUnit Tests	1372
DevTest Solutions Ant Tasks	1372
Ant and JUnit Usage Examples	1375
JUnit Usage Examples	1375
Integration with CruiseControl	1377
More Example Build Files	1377
Memory Settings	1378
Change Memory Allocation	1378

Third-Party File Requirements	1380
JCAPS File Requirements	1380
JMS Messaging File Requirements	1380
Oracle OC4J File Requirements	1380
SAP File Requirements	1381
SAP IDoc Virtualization Requirements	1381
SAP RFC Virtualization Requirements	1382
SonicMQ File Requirements	1382
TIBCO File Requirements	1382
TIBCO Rendezvous Messaging	1382
TIBCO EMS Messaging or TIBCO Direct JMS	1383
TIBCO Hawk Metrics	1383
WebLogic File Requirements	1383
webMethods File Requirements	1383
WebSphere MQ File Requirements	1384
Enabling Additional Scripting Languages	1385
Database Administration	1385
Internal Database Configuration	1385
External Registry Database Configuration	1386
Configure DevTest to Use DB2	1387
Configure DevTest to Use MySQL	1388
Configure DevTest to Use Oracle	1390
Configure DevTest to Use SQL Server	1391
External Enterprise Dashboard Database Configuration	1393
Database Maintenance	1394
Automatic Reporting Maintenance	1394
Automatic Deletion of Audit Log Entries	1395
Automatic Deletion of Transactions	1395
Automatic Deletion of Tickets	1396
License Administration	1396
Licensing overview	1396
Honor-Based Licensing	1397
How Licensing, ACLs, and Audit Reports Work Together	1399
Usage-Based License Agreement	1399
License Activation	1399
ACL Activation	1400
Honor-based Compliance	1400
Continuous Collection of Usage Data by User Type	1400
Usage Audit Report	1401
DevTest Solutions Usage Audit Report	1401
Overview	1401
User Type tabs (Admin User, PF Power User, SV Power User, Test User, Runtime User) ..	1402

Component By User	1402
Count Calculation Example	1403
User Types	1404
Example of How the User Type is Determined for Auditing Purposes	1405
User Type Inheritance Chart	1407
Security	1408
Using SSL to Secure Communication	1409
SSL Certificates	1410
Create Your Own Self-Signed Certificate	1410
Use SSL with Multiple Certificates	1412
Mutual (Two-Way) Authentication	1413
Using HTTPS Communication with the DevTest Console	1413
Generate a New Key Pair and Certificate	1413
Copy the New Keystore to LISA_HOME	1415
Update Webserver Properties	1415
Using Kerberos Authentication	1416
DevTest support for principal + password authentication	1416
Sample krb5.conf file	1417
Sample krb5.conf file with Active Directory as KDC	1417
Sample login.config file	1417
Access Control (ACL)	1417
Planning Deployment of ACLs	1417
ACL Adminstration	1418
Using the Lightweight Directory Access Protocol (LDAP) with DevTest Solutions	1418
Authentication	1419
Authorization	1419
User Sessions	1419
ACL and Backward Compatibility for CLIs and APIs that Ran Without Credentials	1419
ACL Database	1420
ACL and Command Line Tools or APIs	1420
Permission Types	1420
Standard User Types and Standard Roles	1421
Standard Permissions	1422
Standard Users	1430
View User Information from DevTest Workstation	1433
Manage Users and Roles	1434
Configure Authentication Providers for ACL	1439
Authorize Users Authenticated by LDAP	1443
ACL Configuration Scenarios	1444
Resource Groups	1445
Logging	1448
Main Log Files	1448

Demo Server Log Files	1449
Logging Properties File	1449
Status Messages for Server Components	1450
Automatic Thread Dumps	1451
Test Step Logger	1451
Monitoring	1452
Use the ServiceManager	1452
Service Manager Options	1453
Service Manager Examples	1454
Use the Server Console	1455
Open the Server Console	1455
View the Component Health Summary	1455
View the Component Performance Detail	1456
Create Heap Dumps and Thread Dumps	1457
Force Garbage Collection	1458
Use the Registry Monitor	1458
Registry Monitor - Test Tab	1459
Registry Monitor - Simulators Tab	1459
Registry Monitor - Coordinator Servers Tab	1460
Registry Monitor - Virtual Environments Tab	1460
Use the Enterprise Dashboard	1460
Open the Enterprise Dashboard	1460
Enterprise Dashboard Main Window	1461
Enterprise Dashboard Registry Details Window	1462
Reactivate a Registry or Enterprise Dashboard	1464
Maintain Registries	1464
Export Dashboard Data	1466
Purge Dashboard Data	1468
Reference	1470
Assertion Descriptions	1470
HTTP Assertions	1470
Highlight HTML Content for Comparison	1471
Check HTML for Properties in Page	1473
Ensure HTTP Header Contains Expression	1475
Check HTTP Response Code	1475
Simple Web Assertion	1476
Check Links on Web Responses	1476
Database Assertions	1477
Ensure Result Set Size	1477
Ensure Result Set Contains Expression	1478

XML Assertions	1478
Highlight Text for Comparison	1479
Ensure Result Contains String	1481
Ensure Step Response Time	1481
Graphical XML Side-by-Side Comparison	1482
XML XPath Assertion	1486
Ensure XML Validation	1487
JSON Assertions	1489
Ensure Result Equals	1489
Ensure Result Contains	1491
Ensure JSON Schema	1491
Virtual Service Environment Assertion	1493
Assert on Execution Mode	1493
Other Assertions	1494
Highlight Text Content for Comparison	1494
Ensure Non-Empty Result Assertion	1497
Ensure Result Contains String Assertion	1498
Ensure Result Contains Expression Assertion	1498
Ensure Property Matches Expression Assertion	1498
Ensure Step Response Time Assertion	1499
Scripted Assertion	1499
Ensure Properties Are Equal Assertion	1501
Assert on Invocation Exception Assertion	1501
File Watcher Assertion	1502
Check Content of Collection Object Assertion	1502
WS-I Basic Profile 1.1 Assertion	1503
Messaging VSE Workflow Assertion	1505
Validate SWIFT Message Assertion	1505
Mobile Testing Assertions	1507
Mobile-Specific Assertions	1507
Mobile Target Predicate	1510
Asset Descriptions	1511
Email Connection Asset	1511
IBM MQ Native Assets	1513
JDBC Connection Assets	1514
JMS Client Assets	1516
Connection Factories	1517
Connections	1518
Sessions	1518
Producers	1518
Consumers	1518
Destinations	1519

JNDI Initial Context Asset	1519
Mobile Assets	1519
RabbitMQ Assets	1522
Connections	1522
Queues	1523
Exchanges	1524
Temp Queues	1525
SAP JCo Destination Assets	1526
SSL Assets	1528
Basic Scenarios	1529
Advanced Scenarios	1531
SSL Context Parameters	1533
Companion Descriptions	1533
Web Browser Simulation Companion	1534
Browser Bandwidth Simulation Companion	1535
HTTP Connection Pool Companion	1535
Configure DevTest to Use a Web Proxy Companion	1536
Set Up a Synchronization Point Companion	1537
Set Up an Aggregate Step Companion	1537
Execute Event-driven Subprocess Companion	1538
Observed System VSE Companion	1538
Configuration Information	1539
Data Set Source Example	1540
Observed System Data Provider (Wily) Source	1545
Run Time Priorities	1547
VSE Think Scale Companion	1547
Batch Response Think Time Companion	1549
Recurring Period Think Time Companion	1550
Create a Sandbox Class Loader for Each Test Companion	1552
Set Final Step to Execute Companion	1552
Negative Testing Companion	1553
Fail Test Case Companion	1553
XML Diff Ignored Nodes Companion	1553
JSON Ignored Nodes Companion	1553
Correlation Schemes	1555
Correlation ID	1556
Message ID to Correlation ID	1557
Payload	1557
ReplyTo and Temporary Response Queues	1558
Data Set Descriptions	1558
Read Rows from a Delimited Data File Data Set	1558
Create Your Own Data Sheet Data Set	1560

Create Your Own Set of Large Data Data Set	1562
Read Rows from a JDBC Table Data Set	1564
Create a Numeric Counting Data Set	1565
Read Rows from Excel File Data Set	1566
Read DTOs from Excel File Data Set	1567
Building the Excel spreadsheet	1569
Unique Code Generator Data Set	1573
Random Code Generator Data Set	1573
Message-Correlation ID Generator Data Set	1574
Load a Set of File Names Data Set	1575
XML Data Set	1576
Basic Settings Tab	1578
Advanced Settings Tab	1578
Record Editing Panel	1579
Visual XML Tab	1580
Raw XML Tab	1581
Event Descriptions	1581
Filter Descriptions	1587
Utility Filters	1588
Create Property Based on Surrounding Values	1588
Store Step Response	1590
Override Last Response Property	1591
Save Property Value to File	1591
Parse Property Value As Argument String	1592
Save Property from One Key to Another	1592
Time Stamp Filter	1593
Database Filters	1593
Extract Value from JDBC Result Set	1594
Size of JDBC Result Set	1596
Set Size of a Result Set to a Property	1596
Get Value For Another Value in a ResultSet Row	1597
Messaging/ESB Filters	1599
Extract Payload and Properties from Messages	1599
Convert an MQ Message to a VSE Request	1600
Convert a JMS Message to a VSE Request	1600
HTTP/HTML Filters	1601
Create Resultset from HTML Table Rows	1601
Parse Web Page for Properties	1603
Parse HTML/XML Result for the Value of a Specific Tag or Attribute	1606
Parse HTML Result for Specific Value and Parse It	1608
Parse HTML Result for the Child Text of a Tag	1609
Parse HTML Result for HTTP Header Value	1609

Parse HTML Result for the Value of an Attribute	1610
Parse HTML Result for DevTest Tags	1611
Choose Random HTML Attribute	1611
Parse HTML into List of Attributes	1612
Parse HTTP Header Cookies	1613
Dynamic Form Filter	1613
Parse HTML Result by Searching Tag Attribute Values	1614
Inject HTTP Header	1615
XML Filters	1616
Parse Text from XML	1616
Read Attribute from XML Tag	1618
Parse XML Result for DevTest Tags	1620
Choose Random XML Attribute	1620
XML XPath Filter	1620
JSON Filters	1621
JSON Path Filter	1621
Java Filters	1622
Java Override Last Response Property Filter	1623
Java Store Step Response Filter	1623
Java Save Property Value to File Filter	1624
VSE Filters	1624
Data Protocol Filter	1624
CAI Filters	1625
Integration for CA Continuous Application Insight	1625
Integration for webMethods Integration Server	1625
Copybook Filters	1626
Copybook Filter	1626
Metrics Descriptions	1627
DevTest Whole Test Metrics	1627
DevTest Test Event Metrics	1628
SNMP Metrics	1629
JMX Metrics	1630
Enable JMX Metrics for Tomcat	1633
TIBCO Hawk Metrics	1633
Windows Perfmon Metrics	1634
UNIX Metrics Via SSH	1636
Property Descriptions	1638
DevTest Property File (lisa.properties)	1638
Comma-Separated List of Paths for Javadoc and Source Code	1639
System Properties	1639
Server Properties	1640
OS X Properties	1640

Update Notifications	1641
Basic Defaults	1641
HTTP Header Keys Properties	1642
HTTP Field Editor Properties	1643
Test Case Execution Parameters	1643
TestEvent Handling Customizations	1644
Test Manager/Editor Properties	1645
J2EE Server Parameters	1647
Native Browser Information to Use for Internal Rendering	1648
Test Manager/Monitor Properties	1649
Built-In String-Generator Patterns	1649
JMX Information	1649
Test Manager/ITR Properties	1651
External Command Shells	1651
Testing Parameters	1651
Properties for Use by StdSchedulerFactory to Create a Quartz Scheduler Instance	1651
VSE Properties	1653
Network Port Properties	1661
CA Continuous Application Insight Properties	1662
Database Properties	1666
Mainframe Properties	1667
Selenium Integration Properties	1668
VSEEasy Properties	1670
Solr Properties	1670
CAI Stateful Baseline Parameterization	1670
Custom Property Files	1671
Local Properties File	1671
Site Properties File	1690
logging.properties	1694
REST Invoke API	1698
Explore the REST Invoke API from the Web Interface	1699
Change the Execution Mode Using the REST Invoke API	1700
Test Step Descriptions	1701
Test Step Information	1701
Abort the Test	1701
End the Test	1701
Fail the Test	1702
Web-Web Services Steps	1702
HTTP-HTML Request Step	1702
REST Step	1710
Web Service Execution (XML) Step	1711
WSDL Validation	1748

Web-Raw SOAP Request	1749
Base64 Encoder	1750
Multipart MIME Step	1750
SAML Assertion Query	1752
Java-J2EE Steps	1756
Dynamic Java Execution	1757
RMI Server Execution	1760
Enterprise JavaBean Execution	1764
Other Transaction Steps	1768
SQL Database Execution (JDBC)	1768
SQL Database Execution (JDBC with Asset)	1771
CORBA Execution	1773
Utilities Steps	1774
Save Property as Last Response	1775
Output Log Message	1775
Write to Delimited File	1775
Read Properties from a File	1776
Do-Nothing Step	1777
Parse Text as Response	1778
Audit Step	1778
Base64 Encoder Step	1779
Checksum Step	1779
Convert XML to Element Object	1780
Compare Strings for Response Lookup	1781
Compare Strings for Next Step Lookup	1783
Send Email	1784
External-Subprocess Steps	1785
Execute External Command	1786
File System Snapshot	1788
Execute Subprocess	1788
JUnit Test Case-Suite	1789
Read a File (Disk URL or Classpath)	1790
External - FTP Step	1791
JMS Messaging Steps	1792
JMS Messaging (JNDI)	1792
JMS Messaging - Message Consumer	1800
JMS Send Receive Step	1803
Tutorial - Send and Receive a JMS Message	1807
BEA Steps	1810
WebLogic JMS (JNDI)	1810
Message Consumer	1815
Read a File	1815
Web Service Execution (XML)	1816

Raw SOAP Request	1816
FTP Step	1816
Sun JCAPS Steps	1816
JCAPS Messaging (Native)	1816
JCAPS Messaging (JNDI)	1818
Oracle Steps	1819
Oracle OC4J (JNDI)	1819
Oracle AQ Steps	1820
TIBCO Steps	1829
TIBCO Rendezvous Messaging	1829
TIBCO EMS Messaging	1834
TIBCO Direct JMS	1835
Sonic Steps	1835
SonicMQ Messaging (Native)	1836
SonicMQ Messaging (JNDI)	1836
webMethods Steps	1837
webMethods Broker	1837
webMethods Integration Server Services	1841
IBM Steps	1846
IBM WebSphere MQ Step	1846
IBM MQ Native Send Receive Step	1851
SAP Steps	1855
SAP RFC Execution	1855
SAP IDoc Sender	1857
SAP IDoc Status Retriever	1858
Selenium Integration Steps	1860
Create and Export a Selenium Builder Recording	1861
Import a Selenium Builder JSON into CA Application Test	1862
Export a Test Case with Selenium Test Steps to a JSON Script	1864
Virtual Service Environment Steps	1865
CAI Steps	1865
Execute Transaction Frame	1866
Mobile Steps	1868
Modify Mobile Test Steps	1868
Custom Extension Steps	1872
Custom Test Step Execution	1873
Java Script Step (deprecated)	1873
Execute Script (JSR-223) Step	1874
RabbitMQ Steps	1875
RabbitMQ Send Receive Step	1875

Connect	1879
User Community	1879
Support	1879
Webcasts, Success Stories, White Papers, Demos, and More	1879
CA Education	1879
CA Education DevTest Solutions Resources:	1879
Extensions	1880
OData Dynamic Virtual Services (DVS)	1880
OData Dynamic Virtual Service Features	1881
Method Support	1882
Resource Paths Supported	1885
Special Resource Paths	1887
System Query Options	1890
Filter Operators	1895
Built-in Filter Functions	1896
Date and Time Functions	1897
Mathematical Functions	1898
Request and Response Headers	1899
Extensions Outside of OData Standard	1900
.....	1903
DVS Limitations	1903
Building Dynamic Virtual Service	1904
Prerequisites	1904
Java SE Development Kit (JDK)	1905
Apache Ant	1905
Install Required Ant Extensions	1906
Install Apache Maven	1907
Build the DVS Components	1907
Next Steps	1908
Installing Dynamic Virtual Service	1908
Prerequisites for Installing the DVS Eclipse Plug-in	1909
Installation	1909
Configuring the Eclipse Plug-in	1910
Installing the DVS Servlet	1910
Installing the DVS Servlet	1911
Configuring the DVS Servlet	1911
.....	1912
Verifying DVS Servlet Operation	1912
Installing the OData Dynamic Virtualization Services Extension	1912

Installing the Example Bookshop DevTest Project	1913
Next Steps	1913
Designing Your Virtual Service	1913
Defining a VS in AEDM Format	1914
Defining a VS in RAML Format	1919
Sample Data	1921
Deploying Your Virtual Service	1922
Prerequisites for Deploying When Using the DVS Assistant (Eclipse Plug-in) and DevTest Solutions	1922
Creating the Virtual Service MAR File Through Eclipse	1922
Deploying the MAR File	1923
Prerequisites for Deploying When Using the DVS Servlet	1924
Creating and Deploying	1924
Working with the MAR as a DevTestProject	1924
Removing a Virtual Service	1925
Next Steps	1925
Managing Your Virtual Service Data	1926
Runtime Management of Virtualization Data	1926
Web Client Requests	1926
Unsupported Requests	1928
Backing up Data	1929
 Third Party Acknowledgments	1930

DevTest Solutions - Home

Release Information

Contents

- [8.4 Enhancements \(see page 38\)](#)
 - [8.4 CA Application Test Enhancements \(see page 39\)](#)
 - [8.4 CA Service Virtualization Enhancements \(see page 40\)](#)
 - [8.4 CA Continuous Application Insight Enhancements \(see page 41\)](#)
 - [8.4 Mobile Testing Enhancements \(see page 42\)](#)
 - [8.4 General Enhancements \(see page 43\)](#)
- [8.4 Known Issues \(see page 43\)](#)
- [End of Life Features \(see page 44\)](#)
 - [End of Life Features as of DevTest Solutions 8.2 \(see page 44\)](#)
 - [End of Life Features as of DevTest Solutions 8.0.2 \(see page 44\)](#)
 - [End of Life Features as of DevTest Solutions 8.0 \(see page 44\)](#)
 - [End of Life Features as of LISA 7.5 \(see page 45\)](#)
 - [End of Life Features as of LISA 7.1 \(see page 45\)](#)
 - [End of Life Features as of LISA 6.0.9 \(see page 45\)](#)

Release Date: 2015 September

8.4 Enhancements

8.4 CA Application Test Enhancements

Case	Title	Description	Updates to User Documentation
22567 3	Enhanced ACL support	Added support for configuration of multiple authentication providers, along with assigning default roles to groups of LDAP users.	Updated the following pages: Configure Authentication Providers for ACL (see page 1439)
22957 8, 00133 183	CVS monitors reappearin g	When monitored file are deleted from the cvsMonitors directory, the UI reflects that change.	Authorize Users Authenticated by LDAP (see page 1443) Added the following page: ACL Configuration Scenarios (see page 1444)
23241 7	JSON string formatter	The REST step now does automatic formatting of JSON responses.	Updated the following page: REST Step (see page 1710)
23267 5, 00176 368	Oracle 12c connection string	Updated the format of the Oracle connection string to reflect changes in format.	Updated the following pages: External Enterprise Dashboard Database Configuration (see page 1393) Configure DevTest to Use Oracle (see page 1390)

8.4 CA Service Virtualization Enhancements

Case Title	Description	Updates to User Documentation
2178 Search 74 enhance ments	Added options and filters to the search feature in the virtual service editor. The search feature and the feature that lets you find the transactions that match a request are now located in the same area of the editor. To switch between these features, you select an option from a drop-down list.	Updated the following pages: Search for Text in a Virtual Service (see page 719) Find the Transactions that Match a Request (see page 730)
2193 JSON 01, DPH 0008 issue 1260	Fixed an issue with adding extra underscores in JSON data that was causing VSIs recorded earlier than DevTest version 7.5.1 to fail on versions 7.5.1 and later.	
2265 HTTP/S 00, response 0011 step 0605	Made performance improvements during playback, especially with large request/response payloads.	
2306 Unable 81, to open 0013 VS in 3864 portal	Updated res-hub code to handle symbolic link to projects 81, to open directory.	
2309 Large 22, attachme 0013 nt VSE 5486 handling	Memory usage improvements: reduced the memory requirements during playback by not setting recorded_raw_request_bytes and recorded_raw_request attributes.	
2323 Live 21, invocatio 0017 n no 1728 events	During playback when the transaction mode is set to "Live Invocation," request data was not seen in the Inspection view. Made code changes to fix that.	
2324 Observed 37, System 0015 VSE 2415 Compani on	Updated documentation to reflect the need for the metric-data-service.jar file when using the Observed System VSE companion.	Updated the following page: Observed System VSE Companion (see page 1538)

8.4 CA Continuous Application Insight Enhancements

Case	Title	Description	Updates to User Documentation
2278 99, 2296 29	Deep validation	Enhanced the database validation feature.	Updated the following page: Database Validation (see page 1144)
2326 80	Update for agent setting	Enhancement to agent configuration. Added the ability to create agent groups to configure agent settings to multiple agents simultaneously. Changes to various agent configuration fields and buttons.	Added the following page: Create and Manage Agent Groups (see page 1023)
2329 96	Change label	Changed the name of "consolidated" baselines to "data driven."	Updated various pages for: Configuring Agents (see page 1013)
2329 98	Button and label for generating test	Changed Create Baseline button to Create Functional Test.	Updated the following page: Baseline Types (see page 1111) How to Document a Transaction for a Manual Test (see page 1157)

8.4 Mobile Testing Enhancements

Ca Title se	Description	Updates to User Documentation
23 Mobile 05 properties 29	Added a new Mobile section to the Local Properties File topic and included new properties that pertain strictly to Mobile Testing.	Updated the following page: Local Properties File (see page 1671)
23 Application 05 signing 29	Added documentation on how to sign applications while creating a mobile asset.	Created the following page: Application Signing (see page 655)
23 General 33 mobile 55 assertion	Added documentation for a new mobile assertion that can be applied to all test steps, not just those created in the Mobile Test Step Editor.	Updated the following pages: Mobile Assets (see page 1519) Tutorial 1 - Record an iOS Test Case (see page 287)
22 Remote 57 recording 00	Remote recording now requires authentication for hosts.	Created the following page: Mobile Target Predicate (see page 1510) Updated the following page: Remote Recording (see page 651)

8.4 General Enhancements

Case	Title	Description	Updates to User Documentation
22880 1, 00131 003	OSS Vulnerability - Tomcat	Fixed Tomcat 7.0.50 vulnerability issue related to CVE-2014-0230.	
22880 3	OSS Vulnerability - Send	Addressed a vulnerability issue with Visionmedia send of Node.js included in appium.zip that is distributed in the DevTest installer.	
23280 3	Update Manage Resources left hand navigation and artifact viewing	Updated the DevTest Portal left navigation panel to create a new tab for each type of artifact.	Updated the following pages: DevTest Portal (see page 196) Manage Test Artifacts (see page 346) Deploy a Virtual Service (see page 735) Editing Virtual Services (see page 713) Create Copybook Bundles (see page 750)

8.4 Known Issues

Case	Issue	Comments
228802, OSS 001310 18	Vulnerability - GNU M4	There are some vulnerability issues related to the version of Appium that is included in the DevTest installer. To fix it, delete {{Lisa_Home}}/addons/Appium folder.

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 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.
- LISAINST 1.0 by including of .jar or EJB co-deployment. If a LISAINST 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.

Preview Features

This page describes the Preview features that are available in the Sandbox for this release. We are considering adding these features to the basic user interface. We are making them 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.

For more information about enabling these features, see [Enable and Disable Preview Features \(see page 204\)](#).

8.4 Preview Features

Title	Description	User Documentation
CAI Chrome Extension	Lets you analyze calls and transactions started from the browser.	CAI Extension for Google Chrome (see page 1196)
Common APIs	Lets you view producer and consumer relationships within common APIs.	View Producer and Consumer Relationship (see page 1040)
Data Driven VSI	Lets you maintain the data for a signature definition in an external data set.	Data-Driven Virtual Services (see page 732)
Replace	Lets you find and replace text in a virtual service.	Find and Replace (see page 731)
Virtual Service	Lets you view and manage services existing within the Virtual Service Environment from the DevTest Environment Portal.	View Virtual Service Environments in DevTest Portal (see page 741) Manage and Monitor Virtual Service Environments (see page 743)
		VSE Window Matching Tab in DevTest Portal (see page 746)
		VSE Window Request Events Details Tab in DevTest Portal (see page 746)
CVS	Lets you schedule tests and test suites to run regularly over an extended time period.	Monitor with CVS (Continuous Validation Service) (see page 337) CVS Dashboard Overview (see page 338) Deploy a Monitor to CVS (see page 342) Email Notification Settings (see page 343) Run a Monitor Immediately (see page 344) View Test Details (see page 344) CVS Manager (see page 344)

DevTest Portal Functionality

This page identifies the specific features and functions that are available from the DevTest Portal in Release 8.4. Features not identified in the following table are available from the DevTest Workstation and Console.

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 JDBC
- Virtualize using RR pairs
- Data protocol handlers: REST
- Create and manage copybook bundles
- Manage virtual services (Service Image Editor)
- Monitor virtual services (uses DevTest Console)



More Information:

- [DevTest Portal \(see page 196\)](#)
- [Using the DevTest Portal with CA Application Test \(see page 324\)](#)
- [Using the Workstation and Console with CA Application Test \(see page 348\)](#)

- Using the DevTest Portal with CA Service Virtualization (see page 693)
- Using the Workstation and Console with CA Service Virtualization (see page 758)
- Using CA Continuous Application Insight (see page 1007)

Installing

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

- [Preinstallation \(see page 50\)](#)
- [DevTest Installation Overview \(see page 81\)](#)
- [Installing and Configuring DevTest Server \(see page 86\)](#)
- [Installing DevTest Workstation \(see page 105\)](#)
- [Installing the Demo Server \(see page 113\)](#)
- [Verifying the DevTest Solution Installation \(see page 114\)](#)
- [Installing Integration Tools \(see page 121\)](#)
- [Setting Up the Mobile Testing Environment \(see page 148\)](#)
- [Installing DevTest Solutions with a Silent Install \(see page 157\)](#)
- [Docker Containers \(see page 158\)](#)
- [Migrating DevTest Solutions \(see page 160\)](#)
- [Migrating the SDK \(see page 180\)](#)

Preinstallation

This section describes the steps to take before you install DevTest. It includes information about system requirements, how to plan for your installation, the DevTest architecture, and how to download the product installers. It is important that you read and understand this section before you begin the installation process.

System Requirements

Contents

- [Operating System Requirements \(see page 50\)](#)
- [Supplying Your Own JVM \(see page 52\)](#)
- [DevTest Server Requirements \(see page 55\)](#)
- [DevTest Workstation for CA Application Test Requirements \(see page 56\)](#)
- [CA Service Virtualization Requirements \(see page 56\)](#)
- [CA Continuous Application Insight Requirements \(see page 57\)](#)
- [Communication Requirements \(see page 57\)](#)
- [Database Requirements \(see page 57\)](#)
- [Supported Browsers \(see page 60\)](#)

Operating System Requirements

The following operating systems are supported:

- Microsoft Windows:
 - Windows Server 2012
 - Windows Server 2012 R2
 - Windows 7
 - Windows 8 (with latest service pack and all critical updates applied)
- Linux and UNIX
 - Fedora 19
 - Red Hat Enterprise Linux 6.3
 - SUSE Linux 10 SP2, 11.x
 - Ubuntu 11.04, 12.04, 13.x
 - Oracle Solaris 10 and 11
 - IBM AIX 6.1, 7.0
- OS X 10.9 and 10.10

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

DevTest Server JVM System Requirements

We recommend a 64-bit Java 7 for DevTest Server when requiring heap sizes above 2 GB.

Java JDK

Products composing DevTest Solutions are Java applications. An Oracle JRE is included with each operating system-specific installer (1.7.0_17 JRE including a **tools.jar** from the JDK) other than the generic UNIX installer. IBM JRE Version 7, Release 1 is also supported if you [supply your own JVM \(see page 52\)](#).

The minimum supported Java version for DevTest Workstation, DevTest Server, and VSE is Java 1.7 update 6.

This requirement is a DevTest-side requirement only. DevTest running in a Java 1.7 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 (CAI)
JDK 1.5	Not supported	Not supported	Not supported	Supported
	Not supported	Not supported	Not supported	Supported

JDK**1.6**

JDK	Required for run time	Required for run time	Required for runtime	Supported
1.7				
1.8	Not supported	Not supported	Not supported	Supported (WildFly 8.2 only)



Note: DevTest does not support IBM JRE Version 8, Oracle JRE 1.8, 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 (CAI):

- CAI 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. IBM JDKs have limited support only.

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 UNIX, Linux, 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: 3.2, 4.1, 4.2, 5.1, and 5.2.

You can get the DevTest CICS 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 \(see page 102\)](#).



Note: DevTest supports IBM JRE Version 7, Release 1. DevTest does not support IBM JRE Version 8, Oracle JRE 1.8, or OpenJDK.

Installing and Configuring the Oracle JVM

Follow these steps:

1. Download the [Java SE Development Kit \(JDK\) 7](http://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase7-521261.html#jdk-7u67-oth-JPR) (<http://www.oracle.com/technetwork/java/javase/downloads/javase7-521261.html#jdk-7u67-oth-JPR>) package for your platform from the Oracle website.
2. Download the [Java Cryptography Extension \(JCE\) Unlimited Strength Jurisdiction Policy Files 7](http://www.oracle.com/technetwork/java/javase/downloads/jce-7-download-432124.html) (<http://www.oracle.com/technetwork/java/javase/downloads/jce-7-download-432124.html>) from the Oracle website.
3. Install JDK 7 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 7 in \usr\java, which creates the \usr\java\jdk1.7.0_67 directory (JDK_HOME).
4. After you copy the jdk-7u67-*platform*.tar.gz file, enter this command:

```
tar zxvf jdk-7u67-platform-x64.tar.gz
```

5. Set environment variables:

- Set the **JDK_HOME** environment variable to the directory where you installed JDK 7.
- Set the **JAVA_HOME** environment variable to point to the **JDK_HOME** directory.

For example:

```
cd \usr\java\jdk1.7.0_67
pwd
export JAVA_HOME=$PWD
export JDK_HOME=$PWD
```

6. Extract the UnlimitedJCEPolicy folder from the UnlimitedJCEPolicyJDK7.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\jdk1.7.0_67
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 \(see page 102\)](#).



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

```
C:\DevTest8\8.3.0.72\bin>EnterpriseDashboard.exe
```

```
JVMJ9VM007E Command-line option unrecognised: ${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

Follow these steps:

1. Download the JVM for your platform from the IBM website.
2. Install JDK 7 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 7 in \usr\java, which creates the \usr\java\jdk1.7.0_67 directory (**JDK_HOME**).
3. After you copy the *jdk-7u67-platform.tar.gz* file, enter this command:

```
tar zxvf jdk-7u67-platform-x64.tar.gz
```
4. 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 .
```
5. 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\jdk1.7.0_67  
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 \(see page 102\)](#).



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

```
C:\DevTest8\8.3.0.72\bin>EnterpriseDashboard.exe
```

```
JVMJ9VM007E Command-line option unrecognised: ${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, CA Service Virtualization, and CA Continuous Application Insight require the registry in DevTest Server.

The minimum requirements for DevTest Server are:

- **CPU:** 2 GHz or faster, 4 cores minimum
- **RAM:** 4 GB
- **Disk Space:** 50 GB
- **Database:** See [Database Requirements \(see page 57\)](#). The database can reside on a different system and must have at least 200 GB of storage.

The recommended requirements for DevTest Server are:

- **CPU:** 2 GHz or faster, 8 cores minimum
- **RAM:** 8 GB
- **Disk Space:** 50 GB
- **Database:** See [Database Requirements \(see page 57\)](#). The database can reside on a different system and must have at least 500 GB of storage.

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

- 250 virtual users for each simulator

- 1 processor core and 2-GB RAM for each simulator

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

For each data center:

- 1 test registry and 1 coordinator
- 1 processor core/process = 2 processor cores
- 2 GB RAM/each = 4 GB (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. CA Continuous Application Insight 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
- **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 for VSE, in addition to the RAM requirement for DevTest Workstation, DevTest Server, or CAI
- **Disk Space:** 50 GB of free space

- **Database:** See [Database Requirements \(see page 57\)](#). The database can reside on a different system and must have at least 10 GB of storage.

CA Continuous Application Insight Requirements

The minimum requirements for CA Continuous Application Insight are as follows:

- **CPU:** 2 GHz or faster, 8 cores minimum
- **RAM:** 8 GB
- **Disk Space:** 50 GB of local disk storage
- **Database:** See [Database Requirements \(see page 57\)](#). The database can reside on a different system and must have at least 500 GB of storage.

Communication Requirements

Make sure 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.



Note: To implement secure communication, see [Using SSL to Secure Communication \(see page 1409\)](#) and [Using HTTP/S Communication with the DevTest Console \(see page 1413\)](#).

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

- [DevTest Server Default Port Numbers \(see page \)](#)
- [DevTest Workstation Default Port Numbers \(see page \)](#)
- [Demo Server Default Port Numbers \(see page 1364\)](#)

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 virtual service images.
- **CAI:** 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.

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.

DevTest, when run in a distributed configuration, strongly depends upon any 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, and any restriction to the flow of this data has adverse effects.

In order 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! Use a clean database schema for any new installation. Data from the same DevTest version can be restored into the clean schema before you install. Do not use data from other 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 and 5.6** - 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. In addition, you must set the `lower_case_table_names` system variable to the value 1.
- **Oracle 11g Release 2 and 12c** - 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 needs to be installed with UTF8 encoding.
- **Microsoft SQL Server 2008 R2 and 2012**

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 for creating the reporting tables and indexes, and for creating the Enterprise Dashboard database schema. Provide this information to your database administrator.



Note: For more information about the configuration of an external database, see [Administering \(see page 1362\)](#).



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 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 obtaining the SQL statements for CAI, see [Java Agent Database Schema v8.1 \(see page 1257\)](#).

Supported Browsers

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

- Enterprise Dashboard (<http://hostname:1506/>)
- DevTest Portal (<http://hostname:1507/devtest>)
- DevTest Console (<http://hostname:1505/>)
 - Reporting Portal
 - Continuous Validation Service
 - Server Console
- Demo Server (<http://hostname:8080/lisabank/>)

The 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 36
- Mozilla Firefox 30
- Apple Safari 7.0 (Mac)
- Microsoft Internet Explorer 11

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



Note: Browser support for recording [Selenium Integration tests \(see page 1860\)](#) is limited to Mozilla Firefox. After you import these tests to DevTest, running the test cases has been verified on Google Chrome, Mozilla Firefox 24, Apple Safari 7.0 (Mac) and 5.1.7 (Windows), and Internet Explorer 10 and 11.

CAI Support Matrix

This page provides a reference to various components and features of CAI and when they are supported.

- [CAI Versions \(see page 61\)](#)
- [CAI Agents \(see page 62\)](#)
- [Java SE and EE JVMs \(see page 62\)](#)
- [Operating Systems \(see page 62\)](#)
- [Application Servers \(see page 63\)](#)
- [Visible Transaction Protocols \(see page 64\)](#)
- [Virtualization Features \(see page 65\)](#)
- [Baseline Features \(see page 67\)](#)

CAI Versions

The following table indicates when each product version was released:

Product	2012	2013	2014	2015	2016
DevTest 8.4				Yes (from September)	Yes
DevTest 8.3				Yes (from August)	Yes
DevTest 8.2				Yes (from July)	Yes
DevTest 8.1				Yes (from June)	Yes
DevTest 8.0. x				Yes (from first half)	Yes
DevTest 8.0			Yes (from December 12)	Yes	Yes
LISA 7.5.2			Yes (from June 1)	Yes	Yes (through June 1)
LISA 7.5.1			Yes (from April 15)	Yes	Yes (through April 15)
LISA 7.5		Yes (from December 12)	Yes	Yes (through December 12)	
LISA 7.1		Yes (from September 16)	Yes	Yes (through September 16)	
LISA 6.x	Yes	Yes	Yes (through October 31)		
LISA 5.x	Yes	Yes	Yes (through April 20)		

CAI Agents

The following table indicates when each agent was released:

Agent	2012	2013	2014	2015	2016
Pure Java Agent	Yes	Yes	Yes	Yes	Yes
Native Java Agent	Yes	Yes	Yes	Yes	Yes
.NET Agent (ERPConnect only)			Yes (LISA +7.5.2)	Yes	Yes

Java SE and EE JVMs

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

JVM	2012	2013	2014	2015	2016
Oracle 1.8*				Yes (DevTest +8.2)	Yes (DevTest +8.2)
Oracle 1.7		Yes		Yes	Yes
Oracle JRockit 1.6				Yes (DevTest 8.x)	Yes (DevTest 8.x)
Sun 1.6	Yes	Yes	Yes	Yes	Yes
Sun 1.5	Yes	Yes	Yes	Yes	Yes
Sun 1.4	Yes	Yes	Yes (LISA 5.x, 6.x, 7.x)	Yes (LISA 7.x)	
IBM 1.6	Yes	Yes	Yes	Yes	Yes
IBM 1.5	Yes	Yes	Yes	Yes (LISA 7.x)	Yes (LISA 7.x)

* Limited to WildFly 8.2.

Operating Systems

The following table describes the operating system support:

Operating System	2012	2013	2014	2015	2016
Windows Server 2012		Yes	Yes	Yes	Yes
Windows Server 2008	Yes	Yes	Yes (LISA 6.x, 7.x)	Yes (LISA 7.x)	Yes (LISA 7.5.x)
Windows Vista	Yes	Yes	Yes (LISA 5.x, 6.x, 7.x)	Yes (LISA 7.x)	Yes (LISA 7.5.x)
Windows 8	Yes	Yes	Yes	Yes	Yes

Operating System	2012	2013	2014	2015	2016
Windows 7	Yes	Yes	Yes	Yes	Yes
Windows XP	Yes (LISA 5.x, 6.x, 7.1)	Yes (LISA 5.x, 6.x, 7.1)	Yes (LISA 5.x, 6.x, 7.1)		
Fedora 8	Yes	Yes	Yes (LISA 5.x, 6.x, 7. x)	Yes (LISA 7. x)	Yes (LISA 7.5. x)
Fedora +9	Yes	Yes	Yes	Yes	Yes
IBM AIX 6.1, 7.0	Yes	Yes	Yes	Yes	Yes
Red Hat Enterprise Linux 6.3 - 7.0	Yes	Yes	Yes	Yes	Yes
Solaris 10 and 11	Yes	Yes	Yes	Yes	Yes
SUSE Linux Enterprise 10 SP2, 11.x	Yes	Yes	Yes	Yes	Yes
Ubuntu 8	Yes	Yes	Yes (LISA 5.x, 6.x, 7. x)	Yes (LISA 7. x)	Yes (LISA 7.5. x)
Ubuntu +9	Yes	Yes	Yes	Yes	Yes
OS X 10.8	Yes	Yes	Yes (LISA 5.x, 6.x, 7. x)	Yes (LISA 7. x)	Yes (LISA 7.5. x)
OS X 10.9 - 10.10		Yes	Yes	Yes	Yes (LISA 7.5. x)

Application Servers

The following table describes the application server support:

Application Server	2012	2013	2014	2015	2016
IBM WebSphere 7.0	Yes (LISA +7. x)	Yes (LISA +7. x)	Yes (LISA +7. x)		Yes
IBM WebSphere 8.5			Yes (DevTest 8.x)		Yes (DevTest 8.x)
JBoss AS 4.2	Yes (LISA +6.1)	Yes (LISA +6.1)	Yes (LISA +6.1)	Yes	Yes
JBoss AS 7.3			Yes (DevTest 8.x)		Yes (DevTest 8.x)
JBoss EAP 6.2			Yes (DevTest 8.x)		Yes (DevTest 8.x)
WildFly 8.2			Yes (DevTest +8.1)		Yes (DevTest +8.1)

Application Server	2012	2013	2014	2015	2016
Jetty 8				Yes (DevTest 8.x)	Yes (DevTest 8.x)
Jetty 9.1				Yes (DevTest 8.x)	Yes (DevTest 8.x)
Jetty 9.2				Yes (DevTest +8.3)	Yes (DevTest +8.3)
Oracle WebLogic 10.3	Yes (LISA +6.1)	Yes (LISA +1)	Yes (LISA +1)	Yes	Yes
Oracle WebLogic 12.1.1				Yes (DevTest 8.x)	Yes (DevTest 8.x)
TIBCO BW 5.x	Yes (LISA +6.1)	Yes (LISA +1)	Yes (LISA +1)	Yes	Yes
TIBCO EMS				Yes (DevTest 8.x)	Yes (DevTest 8.x)
webMethods Integration Server 9.6				Yes (DevTest +8.3)	Yes (DevTest +8.3)
webMethods Integration Server 9.5				Yes (LISA +7. 5.2)	Yes
webMethods Integration Server 9.0				Yes (LISA +7. 5.2)	Yes
webMethods Integration Server 8.2	Yes (LISA +0.1)	Yes (LISA +0.1)	Yes (LISA +0.1)	Yes (LISA 7.x)	Yes (LISA 7.5.2)
webMethods Integration Server 7.1	Yes (LISA +0.1)	Yes (LISA +0.1)	Yes (LISA +0.1)	Yes (LISA 7.x)	Yes (LISA 7.5.2)
SAP NetWeaver 7.3 and 7.4				Yes (LISA 7.5.2*)	Yes (LISA 7.5.2*, DevTest +8.0.1*)

* 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	2012	2013	2014	2015	2016
Exception	Yes	Yes	Yes	Yes	Yes
Logs	Yes	Yes	Yes	Yes	Yes
HTTP	Yes	Yes	Yes	Yes	Yes

Feature	2012	2013	2014	2015	2016
Web service	Yes	Yes	Yes	Yes	Yes
REST	Yes	Yes	Yes	Yes	Yes
RMI	Yes	Yes	Yes	Yes	Yes
EJB	Yes	Yes	Yes	Yes	Yes
JDBC	Yes	Yes	Yes	Yes	Yes
JCA	Yes	Yes	Yes	Yes	Yes
JMS	Yes	Yes	Yes	Yes	Yes
WebSphere MQ	Yes	Yes	Yes	Yes	Yes
webMethods	Yes	Yes	Yes	Yes	Yes
TIBCO	Yes (LISA +6.1)	Yes (LISA +6.1)	Yes (LISA +6.1)	Yes	Yes
CICS	Yes	Yes	Yes	Yes	Yes
SAP JCo		Yes (LISA +7.1)	Yes (LISA +7.1)	Yes	Yes
SAP IDoc		Yes (LISA +7.5)	Yes (LISA +7.5)	Yes (LISA +7.5)	Yes
ERPConnect 3.0			Yes (LISA +7.5.2)	Yes	Yes
Java				Yes (DevTest +8.2)	Yes (DevTest +8.2)

Virtualization Features

The following table indicates the support for various virtualization features:

Feature	2012	2013	2014	2015	2016
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)
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)
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)
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)

DevTest Solutions - 8.4

Feature	2012	2013	2014	2015	2016
IBM DB2 9.5 JDBC driver to IBM DB2 10.5 database (db2jcc.jar 3.57.82 and db2jcc4.jar)			Yes (DevTest +8.0) +8.0)	Yes (DevTest +8.0)	Yes (DevTest +8.0)
IBM DB2 9.5 JDBC driver to IBM DB2 9.8 database (db2jcc.jar 3.57.82 and db2jcc4.jar)			Yes (DevTest +8.0) +8.0)	Yes (DevTest +8.0)	Yes (DevTest +8.0)
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 (LISA +7.5.1)	Yes (LISA +7.5.1)	Yes	
Oracle 11g JDBC driver to Oracle 11g database (ojdbc7.jar, ojdbc6-11.1.0.2.jar, and ojdbc5.jar)	Yes (LISA +7.1)	Yes (LISA +7.1)	Yes (LISA +7.1)	Yes	
Oracle 11g JDBC driver to Oracle 12c database (ojdbc7.jar, ojdbc6-11.1.0.2.jar, and ojdbc5.jar)		Yes (DevTest +8.0) +8.0)	Yes (DevTest +8.0)	Yes (DevTest +8.0)	
Apache Derby 10.8.3.0 JDBC driver to Apache Derby database (derbyclient-10.10.2.0.jar)	Yes (LISA +7.1)	Yes (LISA +7.1)	Yes (LISA +7.1)	Yes	
Teradata 14.10	Yes (LISA +7.5)	Yes (LISA +7.5)	Yes (LISA 7.5 through DevTest 8.4)	Yes (LISA 7.5 through DevTest 8.4)	
Web HTTP	Yes (LISA +7.5.2)	Yes	Yes		
REST	Yes	Yes	Yes	Yes	Yes
Web services	Yes	Yes	Yes	Yes	Yes
EJB	Yes (LISA +6.x)	Yes (LISA +6.x)	Yes (LISA +6.x)	Yes	Yes
JCA		Yes (LISA +7.5)	Yes (LISA +7.5)	Yes	
JMS	Yes (LISA +6.x)	Yes (LISA +6.x)	Yes (LISA +6.x)	Yes	Yes
WebSphere MQ	Yes (LISA +6.1)	Yes (LISA +6.1)	Yes (LISA +6.1)	Yes	
webMethods			Yes	Yes	

Feature	2012	2013	2014	2015	2016
		Yes (LISA +7. 1)	Yes (LISA +7. 1)		
TIBCO BW	Yes (LISA +6.1) 1)	Yes (LISA +6. 1)	Yes (LISA +6. 1)	Yes	Yes
CICS	Yes	Yes	Yes	Yes	Yes
SAP JCo		Yes (LISA +7. 1)	Yes (LISA +7. 1)		Yes
SAP IDoc		Yes (LISA +7. 5)	Yes (LISA +7. 5)	Yes (LISA +7.5)	Yes
ERPConnect 3.0			Yes (LISA +7. 5.2)	Yes	Yes
Java				Yes (DevTest +8.2)	Yes (DevTest +8.2)

Baseline Features

The following table indicates the support for various baseline features:

Feature	2012	2013	2014	2015	2016
Web HTTP			Yes (LISA +7.5.2)	Yes	Yes
Web Services	Yes	Yes	Yes	Yes	Yes
REST	Yes	Yes	Yes	Yes	Yes
REST (stateful)		Yes (LISA +7.1)	Yes (LISA +7.1)	Yes	Yes
RMI	Yes	Yes	Yes	Yes	Yes
EJB	Yes (LISA +6.1)	Yes (LISA +6.1)	Yes (LISA +6.1)	Yes	Yes
EJB (stateful)		Yes (LISA +7.1)	Yes (LISA +7.1)	Yes	Yes
JMS	Yes (LISA +6.x)	Yes (LISA +6.x)	Yes (LISA +6.x)	Yes	Yes
WebSphere MQ	Yes	Yes	Yes	Yes	Yes
webMethods		Yes (LISA +7.1)	Yes (LISA +7.1)	Yes	Yes
TIBCO BW and EMS	Yes (LISA +6.1)	Yes (LISA +6.1)	Yes (LISA +6.1)	Yes	Yes

Planning Your DevTest System

In order to successfully install DevTest Solutions, consider the following decision points prior to the installation:

- What supported operating system do you plan to use?
- Are you performing a shared or local installation?
- 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.
- Do you plan to install the Demo Server?
The demo server is supplied separately from the product installer.

Before beginning an installation, the license file (devtestlic.xml) must be available and present on your target system (the Enterprise Dashboard machine).

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

- 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, if practicable.
- 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 hosted within a virtual machine are not recommended for general availability use.

For more information about database requirements, see [Database System Requirements \(see page 57\)](#).

Before you begin planning your DevTest system, review the Release Notes for the release you install. If you are migrating a system, review [Migrating DevTest Solutions \(see page 160\)](#).

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.

When you download the DevTest Solutions installers, you can select one of three products, all of which point to the same installers. [DevTest Components \(see page 69\)](#) depicts graphically these three products with product-specific UIs and processes and the common UI and processes shared by all products. [About DevTest Server Component \(see page 71\)](#)s describes the purpose of each process and where you can find more details.

[Example Relationships in a Distributed System \(see page 72\)](#) introduces major concepts concerning what processes you can run on each host in a distributed system. The [Post-Installation \(see page 96\)](#) section can help you tailor your distributed system; specifically:

- [Running components on different systems \(see page 101\)](#)
- [Calculate simulator instances \(see page 101\)](#)
- [Load and performance server sizing \(see page 102\)](#)

Another aspect of planning is designing naming conventions for your [project directory structure \(see page 104\)](#).

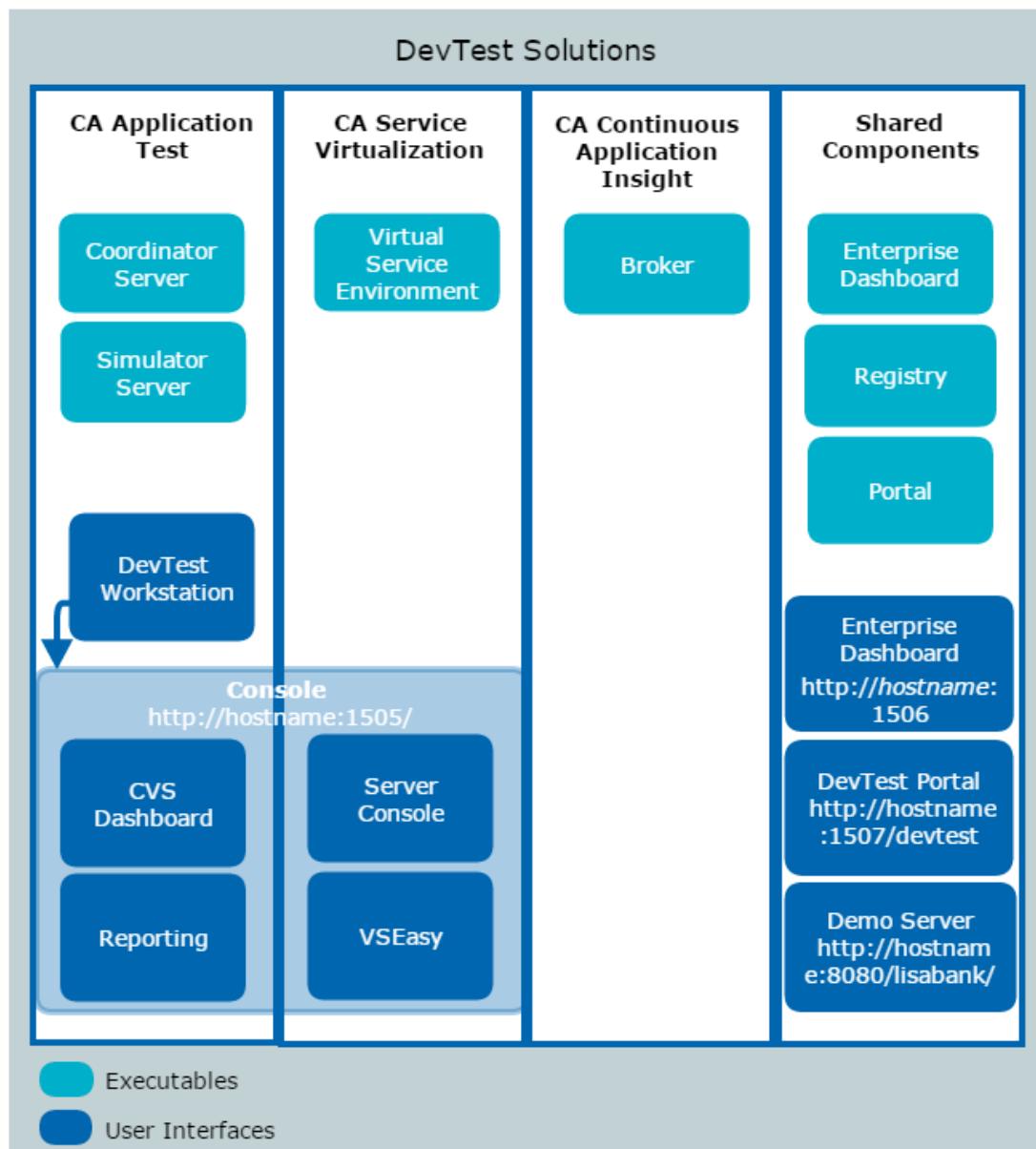
You can use the JRE that comes with DevTest or you can install your own. For more information, see [Using DevTest Workstation with your Java environment \(see page 102\)](#).

DevTest Components

The DevTest Solutions installer installs the following products when you install a DevTest Server (named *Server* in the Setup wizard):

- CA Application Test
- CA Service Virtualization
- CA Continuous Application Insight

These three products each have their own executables (and corresponding services). They are not independent products, however, because they all require the same registry executable. Previously, each product had its own UI. Now, the DevTest Portal is becoming the shared user interface for all products. (Some functionality can be accessed only from the DevTest Workstation or the DevTest Console.)



[About DevTest Server Components \(see page 71\)](#) describes each executable component with where to find more information.

[Start the Server Components \(see page 115\)](#) specifies the order in which to start the executables.

[Log in to the User Interfaces \(see page 120\)](#) includes procedures for accessing each UI.

For the details on setting up and maintaining DevTest Solutions, see [Administering \(see page 1362\)](#).

For product-specific details, see:

- [Using CA Application Test \(see page 318\)](#)
- [Using CA Service Virtualization \(see page 661\)](#)

- [Using CA Continuous Application Insight \(see page 1007\)](#)

About DevTest Server Components

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 \(see page 115\)](#).

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

The referenced pages provide more information about each process supporting DevTest Server.

Process	Reference
Enterprise Dashboard	Use the Enterprise Dashboard (see page 1460)
Registry	Registry (see page 319)
Portal	Using the DevTest Portal (see page 319)
Coordinator	Coordinator Server (see page 321)
Simulator	Simulator Server (see page 322)
Workstation	DevTest Workstation (see page 348)
Virtual Service Environment	Virtualizing a Service (see page 970)
Broker	Start the Broker (see page 1272)

DevTest Process Relationships

The registry is at the center of all DevTest systems. Typically, the installer is run on multiple computers with different selections. Consider the following example:

- Computer 1: Enterprise Dashboard and server components, where the embedded workstation may be unused.
- Computer 2: Server where the registry component links to 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, which can also represent user machines, particularly CA Continuous Application Insight users who need only portal access.
- 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:

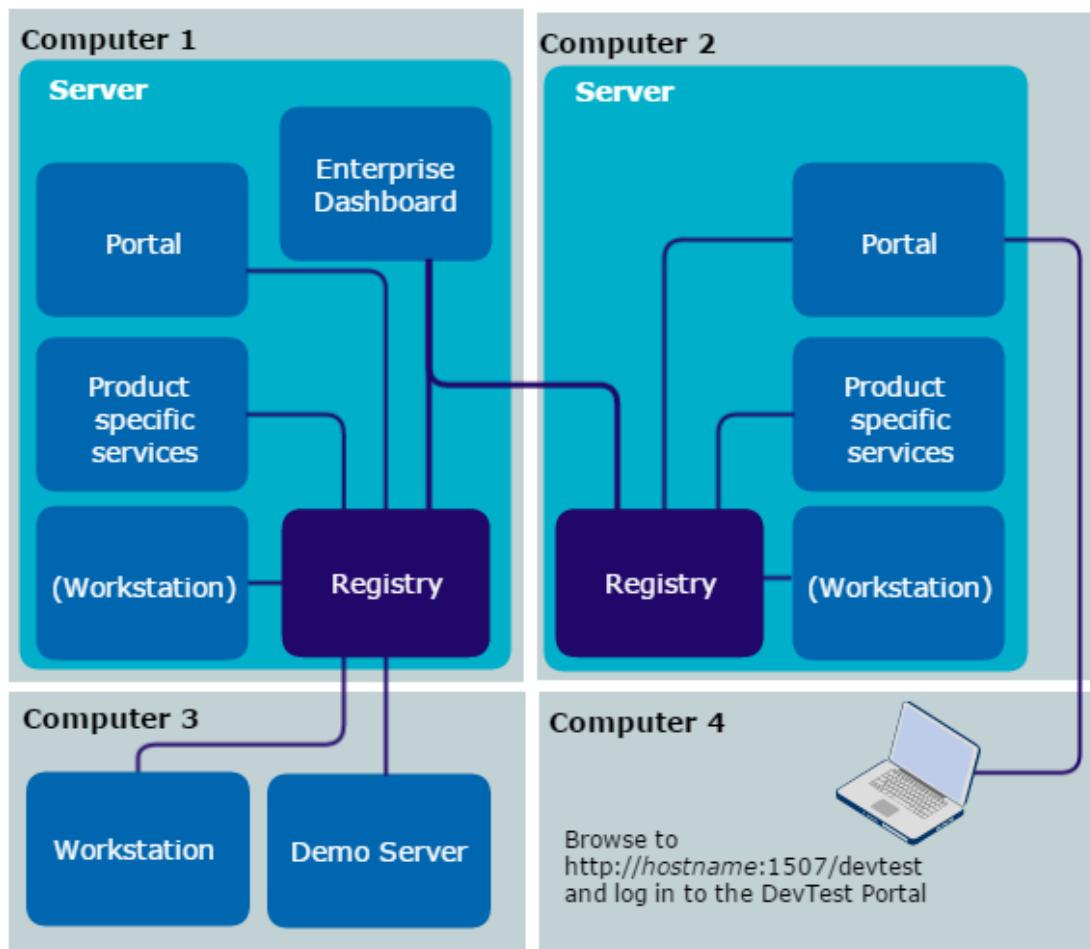
- Each DevTest Solutions system has only one Enterprise Dashboard.



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

- Each Enterprise Dashboard is connected 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 the DevTest Portal by browsing to a DevTest Server on port 1507 where the URL ends with **devtest** (all lower case). The portal can be accessed from any web browser without needing other DevTest Solutions software to be installed on the computer.

The following diagram highlights the relationships among the Enterprise Dashboard, the registries, the workstation, and the Demo Server in a distributed system. The diagram also shows that users with no local DevTest component can browse to the DevTest Portal.



DevTest Solutions Architecture

Contents

- [CA Application Test Architecture \(see page 73\)](#)
- [CA Service Virtualization Architecture \(see page 75\)](#)
- [CAI Architecture \(see page 75\)](#)
- [Enterprise Dashboard Architecture \(see page 76\)](#)

CA Application Test Architecture

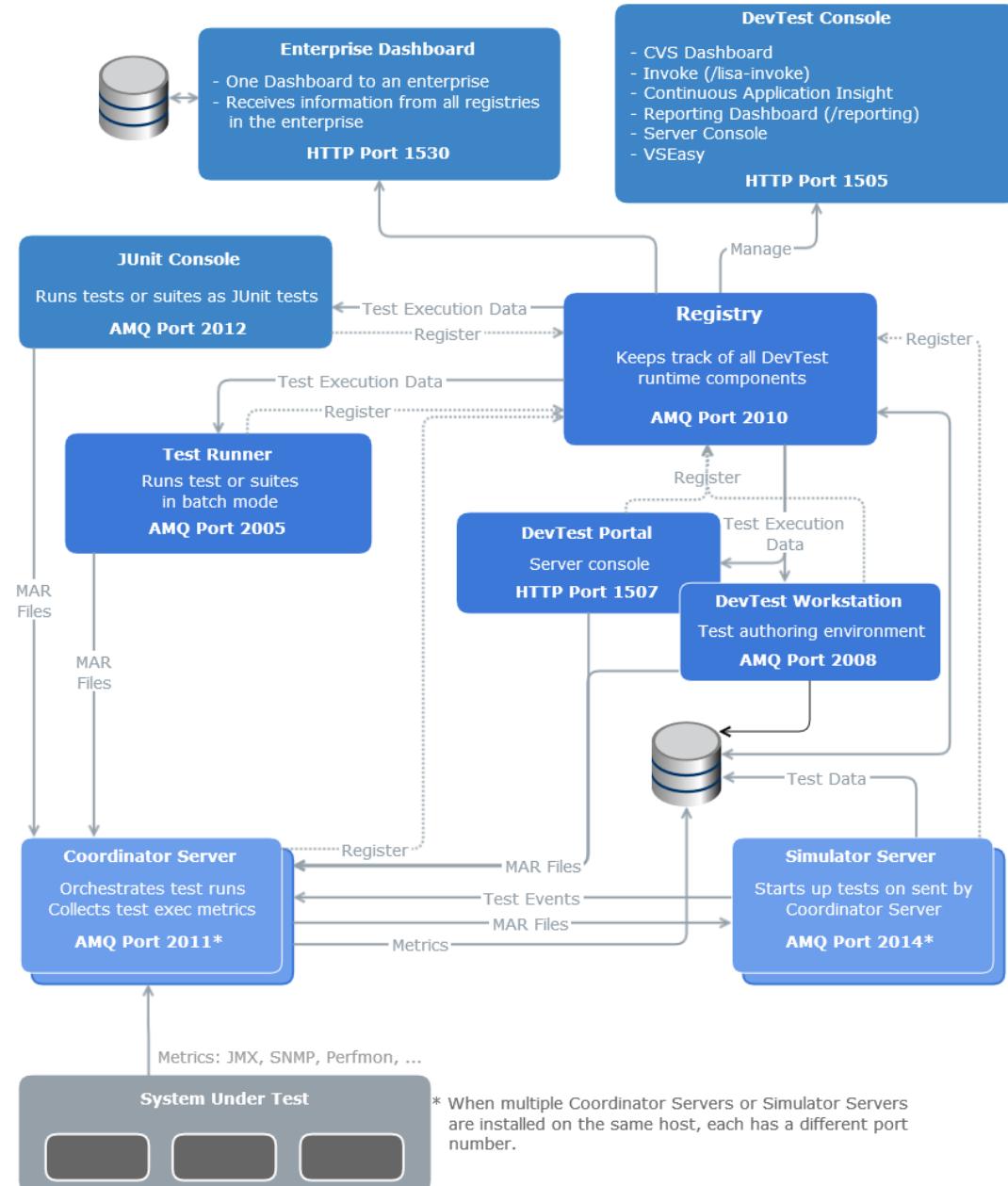
CA Application Test uses the following DevTest Server components:

- Registry
- DevTest Workstation
- DevTest Portal
- Coordinator server

- Simulator server

The 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.

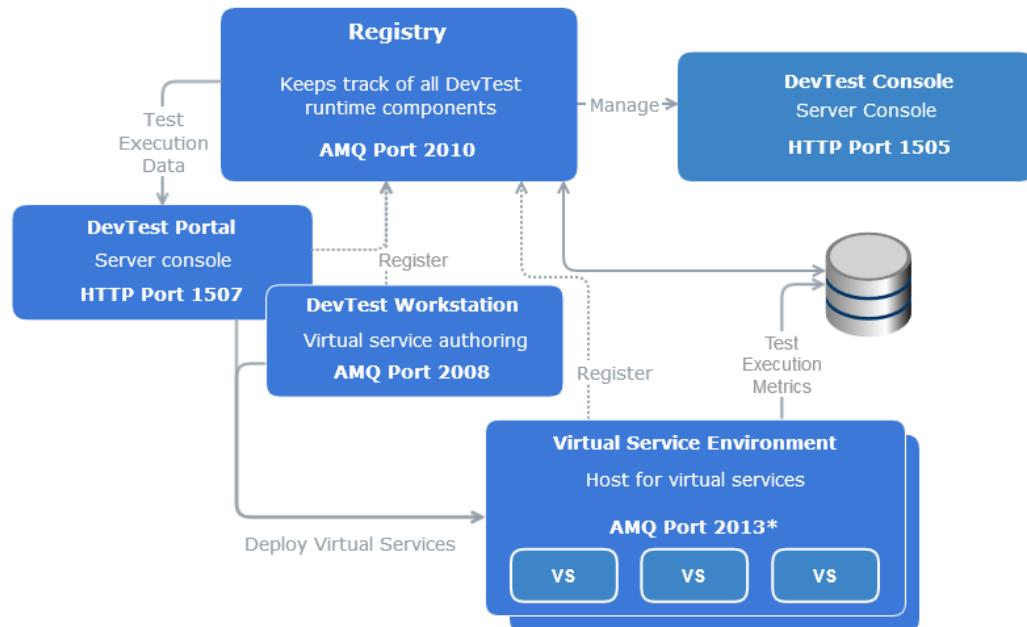
Each simulator connects to and invokes actions on the system under test. A load test results when the virtual users are running in a parallel mode.

An embedded instance of the 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 CA Continuous 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.

CA Service Virtualization Architecture

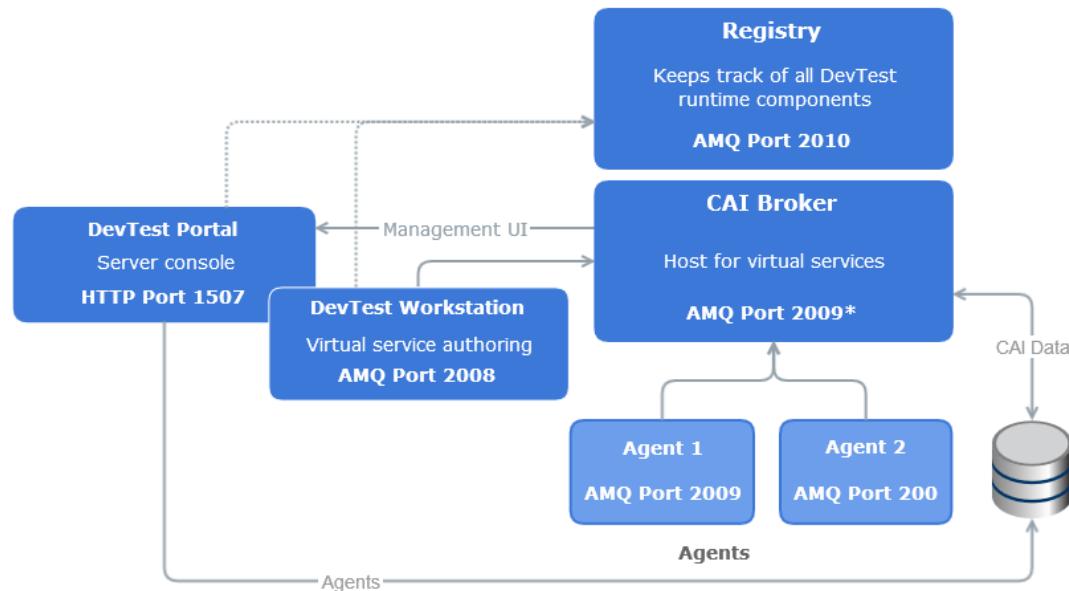
The following diagram shows the CA Service Virtualization architecture.



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

CAI Architecture

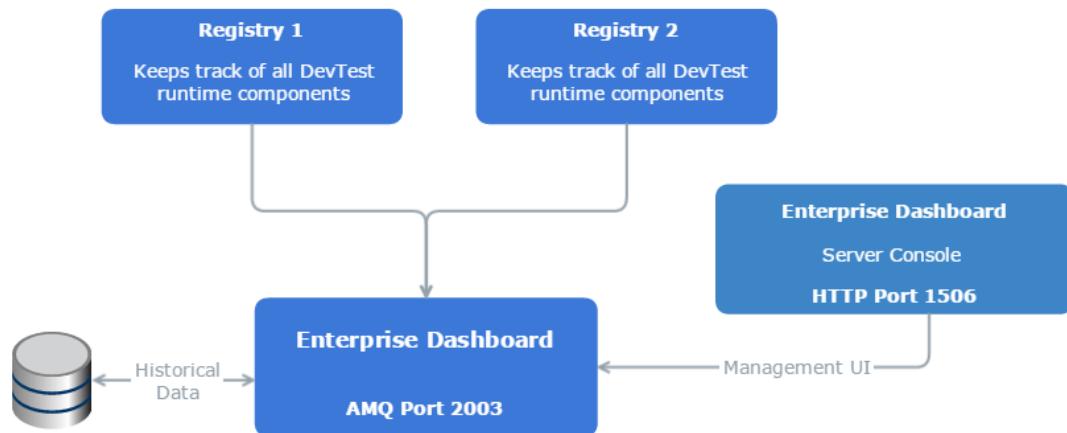
The following diagram shows the CAI architecture.



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

Enterprise Dashboard Architecture

The following diagram shows the Enterprise Dashboard architecture.



More Information:

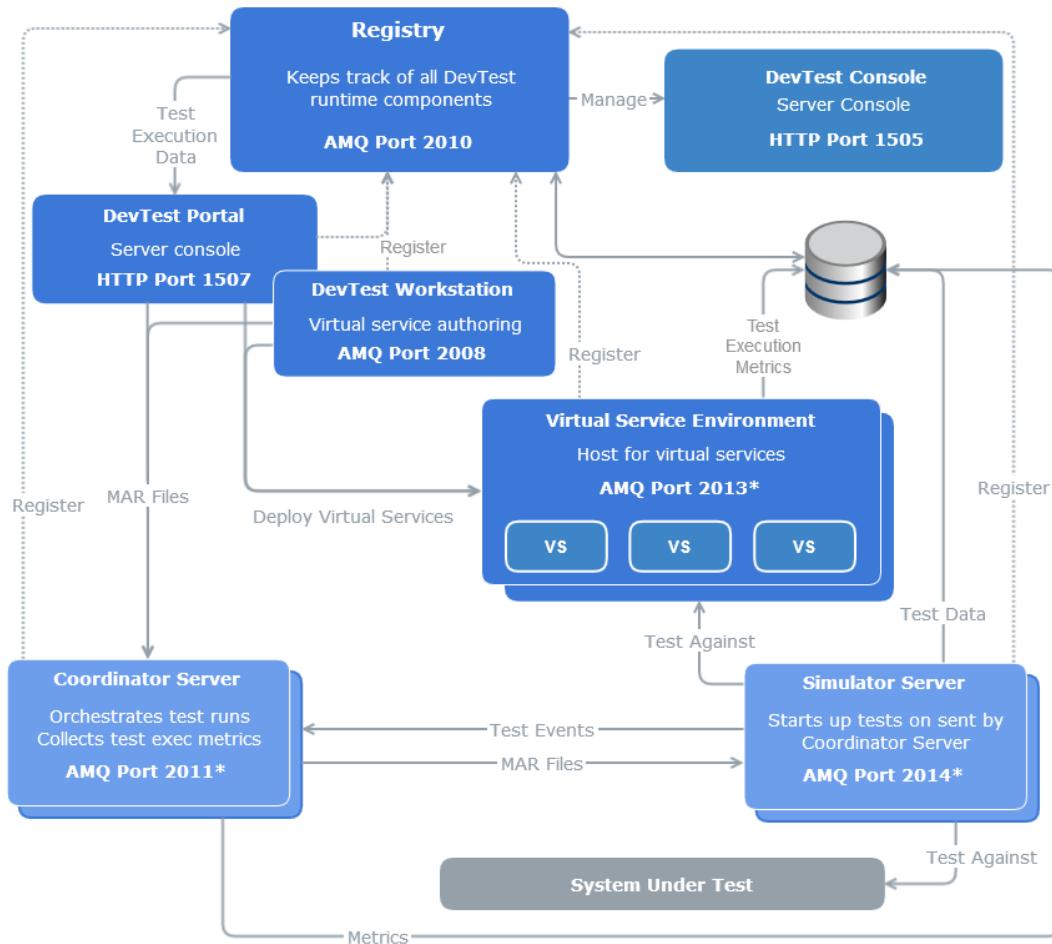
- [About DevTest Server Components \(see page 71\)](#)
- [Data Flow in DevTest Server for CA Application Test and CA Service Virtualization \(see page 78\)](#)

DevTest Server Components

In CA Application Test, the tests are run in the DevTest Server environment. DevTest Workstation connects to the DevTest Server to deploy and monitor tests that were developed in DevTest Workstation.

CA Application Test and CA Service Virtualization use the following DevTest Server components:

- **DevTest Workstation:** An integrated development environment (IDE) where test case assets and virtual service models are created and edited. You can run test cases and models locally in the workstation or you can stage them for a remote execution. 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. For more information, see [DevTest Workstation \(see page 348\)](#).
- **Registry:** A central location for the registration of all DevTest Server and DevTest Workstation components. For more information, see [Registry \(see page 319\)](#).
- **DevTest Console:** The Console includes links to the Server Console (including VSE), CVS Dashboard, and Reporting Dashboard.
- **Coordinator Server:** The coordinator receives the test run information as MAR files, and coordinates the tests that are run on one or more simulator servers. For more information, see [Coordinator Server \(see page 321\)](#).
- **Simulator Server:** The simulator runs the tests under the supervision of the coordinator server. For more information, see [Simulator Server \(see page 322\)](#).
- **VSE:** Used to deploy and run virtual service models. For more information, see [Using CA Service Virtualization \(see page 661\)](#).



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

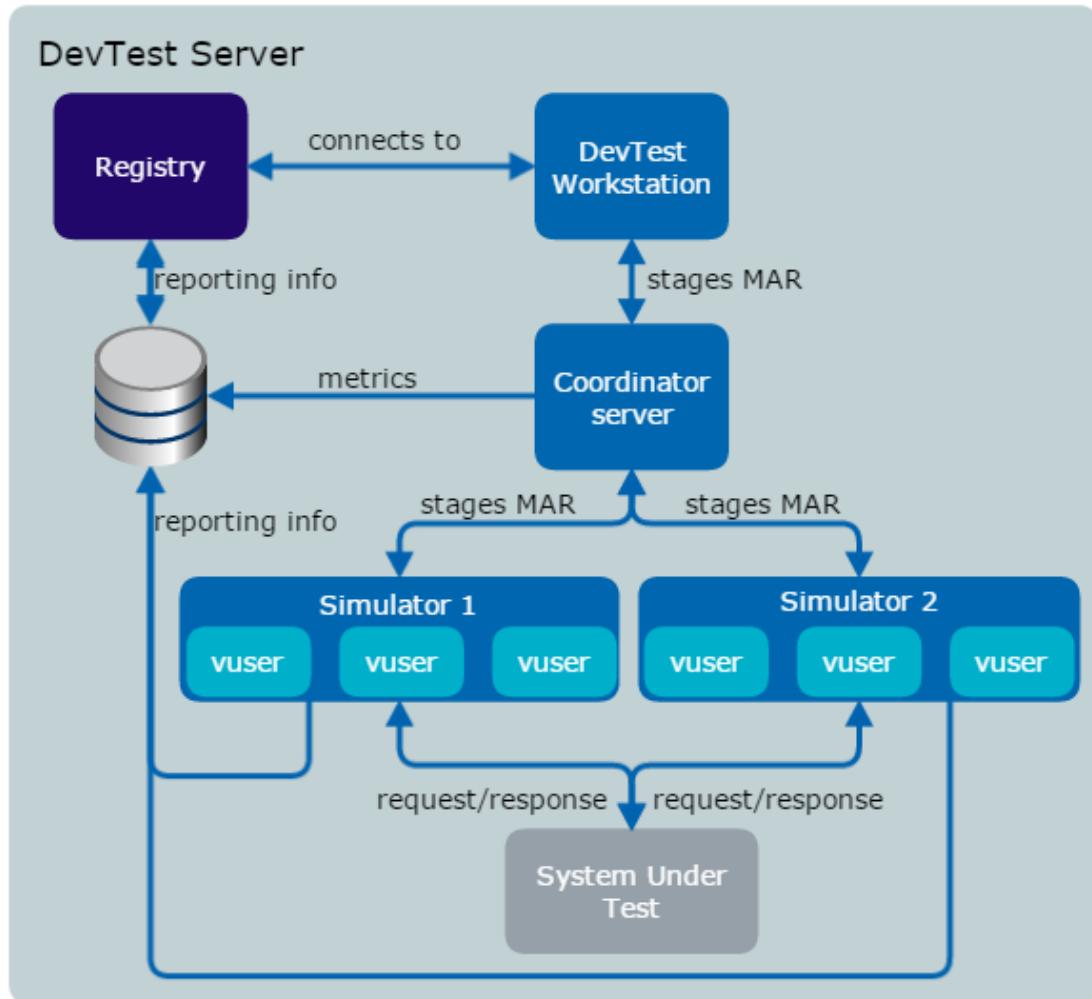
A minimal DevTest Server configuration for CA Application Test and CA Service Virtualization must include at least one of each of these components. There can be as many instances of each type as is needed for a specific testing environment.

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.

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.



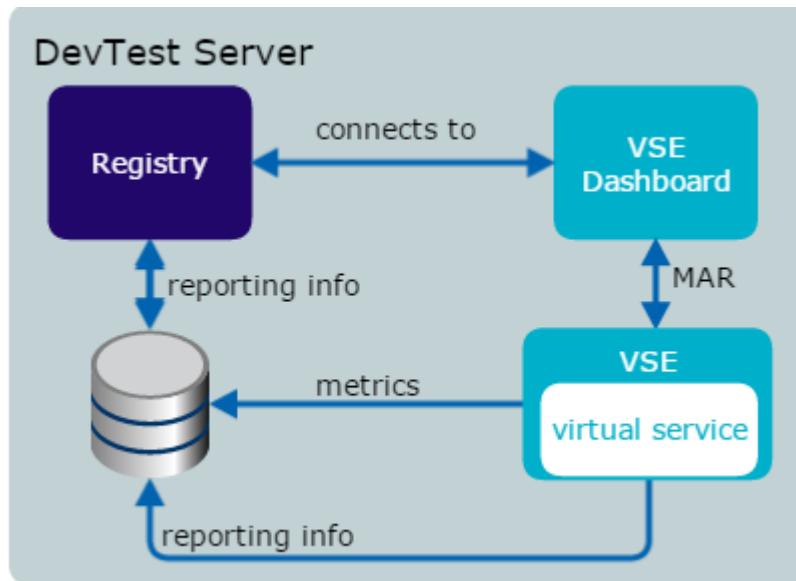
The coordinator server sends test cases to one or more simulator servers in the form of Model Archives (MARs).

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, VSE Dashboard, and database.



Download DevTest Solutions Installers

The Download Center on CA Support Online lets you download one or more platform-specific installers and an optional related file for this release of DevTest Solutions.

First, download the platform-specific installers. After the download is complete, repeat the process to download the Demo Server with examples. You can use the demo server to practice with examples or to work through the tutorials that are included in the CA Application Test documentation.

Follow these steps:

1. Go to support.ca.com (<https://support.ca.com>).
2. Click **Login** and log in with your CA Support Online user name and password.
3. Hover over **Download Center** and select **Download Products**.
The Download Center page opens, with the **All Products** option selected.
4. Type one of the following DevTest product names and the drop-down list displays relevant options.
 - CA Productivity Accelerator for CA Service Virtualization - MULTI-PLATFORM
 - CA Productivity Accelerator for CA Service Virtualization EDITOR LICENSE - MULTI-PLATFORM
 - CA Productivity Accelerator for CA Service Virtualization Foundation 200 - MULTI-PLATFORM
 - CA Service Virtualization for Performance - MULTI-PLATFORM

- CA Service Virtualization Power User - MULTI-PLATFORM
 - CA Continuous Application Insight Power User - MULTI-PLATFORM
 - CA Continuous Application Insight Runtime User - MULTI-PLATFORM
 - CA Application Test Power User - MULTI-PLATFORM
 - CA Application Test Virtual User for Performance Testing - MULTI-PLATFORM
5. Select the product to download. Each of these selections lets you download the DevTest Solutions Installation Wizard for your operating system. With this installer, you can install the DevTest Server, which includes all three products, and the DevTest Workstation.
6. Select the specific release to download.
7. Accept the default values for all other fields.
8. Click **Go**.
A list of product components opens.
9. Select the check box in the **Add to Cart** column for each component to download.
- (Required) The installer for each of your operating systems
 - (Optional) DevTest Demo Server
10. Click **My Download Cart** at the top of the window.
The **My Download Cart** page opens.
11. Enter your email address in the **Checkout** section, and click **Checkout**.
The **Download Method** page opens.
12. Click **Download** and save the file.

DevTest Installation Overview

This installing documentation covers a single, all-inclusive DevTest Solutions installation. There is no breakout of per-product sections. CA Application Test, CA Service Virtualization, and CA Continuous Application Insight are installed together as one solution set.

This section contains the following pages:

- [Installation Options \(see page 82\)](#)
- [How to Install and Set up DevTest Solutions \(see page 84\)](#)
- [How License Activation Works \(see page 85\)](#)

Installation Options

Those users without system administrator privileges can install the Server (DevTest Server), the DevTest Workstation, and the Demo Server. However, only system administrators can install the Server components as Windows services.

Options of Components to Install

The DevTest Solutions Setup Wizard presents you with three dialogs that control the options that are installed on the local host. You can select only one option from each dialog. The default option is shown in italics:

1. Select Components

- *Server* (with Workstation) also known as DevTest Server
- Workstation (only)

2. Enterprise Dashboard

- *New Enterprise Dashboard*
- Existing Enterprise Dashboard Service
This option requires the devtestlic.xml license file to be available.

3. Demo Server

- *Install*
This option requires the Demo Server installation file to be downloaded and present.
- *Do not install*

The installer presents a dialog requesting a configuration type choice. The two configuration types are Shared and Local, where the Shared configuration is suited to multiple users on a single system. The Local configuration is suited to an installation where users may access the installation from other systems or where components are controlled and run on multiple systems (a distributed configuration).

Typical Configurations

Standalone (Single computer that is used to host DevTest components)

If you are installing a new standalone DevTest Solutions on one host, install all three components:

- **Server** (with Workstation)
- **New Enterprise Dashboard**
- **Demo Server**

If multiple users will use the embedded DevTest Workstation on a single computer, select the **Shared** option for Installation Type. For more details, see [Shared Installation Type \(see page 1365\)](#).



Note: If you want to install Component Services and you are an Administrator, use the local install.

Distributed Servers (Multiple computers that are used to host DevTest components)

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

1. For the *first installation*, from a server that meets system requirements, select the following options:

- **Server**
- **New Enterprise Dashboard**, where you browse to the license file, **devtestlic.xml**. The installer validates the license file and its content for signature, version, and expiration.

2. For *more installations of servers*, select the following options:

- **Server**
- **Existing Enterprise Dashboard**



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, select the following options:

- **Workstation**

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

- **Demo Server - Install** (for new users who could benefit from tutorials)

4. For CA Continuous Application Insight users, ensure that a [supported browser \(see page 60\)](#) is installed. All work is done from a web-based portal.

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



Note: 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 to Install and Set up DevTest Solutions

The *Installing* section includes system requirements, conceptual background, and procedures that are required only under certain circumstances. Use the following guidelines to install DevTest Solutions for the first time:

1. [Download DevTest Solutions installers \(see page 80\)](#).
2. If you are installing the Enterprise Dashboard, store your **devtestlic.xml** license file on the target system.



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

3. If you are using the generic UNIX installer, [supply your own JVM \(see page 52\)](#).
4. [Install DevTest Solutions \(see page 105\)](#)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 external databases.
 - [Configure an external database for Registries \(see page 1386\)](#). (DB2, MySQL, Oracle, or SQL Server).
 - [Configure an external database for the Enterprise Dashboard \(see page 1393\)](#). (MySQL, Oracle, or SQL Server)

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.

2. Activate the Enterprise Dashboard and all registries.

- a. If you are upgrading, [configure existing registries \(see page 96\)](#). Registry configuration is automated for releases 8.0.0 and above.

- b. [Activate the registries \(see page 95\)](#).



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

- c. [Verify registry activations \(see page 96\)](#). (New or upgrade)

3. Perform [post-installation \(see page 96\)](#) tasks.
4. (Optional) [Install more workstations \(see page 105\)](#) with the Demo Server for use with each registry.
5. [Verify the DevTest Solutions installation \(see page 114\)](#).
6. To prevent unauthorized use, [change the passwords for standard users \(see page 1430\)](#).
7. [Install the integration tools \(see page 121\)](#) that you need.
8. [Set up the mobile testing environment \(see page 148\)](#).

When the installation is complete, see [Administering \(see page 1362\)](#) for details on setting up ACL-enforced security and honoring your license agreement.

How License Activation Works

One DevTest Solutions license (**devtestlic.xml**) is issued for each enterprise. This file unlocks all functionality of DevTest Solutions. 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 **lisa.enterprisedashboard.service.url** property with the new URL.

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 \(see page 1464\)](#).

Installing and Configuring DevTest Server

This section describes how to install and configure DevTest Server.

- [Install DevTest Server on Windows \(see page 86\)](#)
- [Install DevTest Server on UNIX \(see page 90\)](#)
- [Install DevTest Server on a Mac \(see page 92\)](#)

Install DevTest Server on Windows

To preview installation choices, see [What You Can Install When You Run the Installer \(see page 82\)](#).

Before you start the installation procedure, download the following files from the [Download Center \(see page 80\)](#):

- 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 the 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 \(see page 353\)](#) 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 Destination Directory step opens.
4. 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_8.0. Or, accept the default (C:\Program Files\CA\DevTestSolutions). If you browse to a path and you enter the name of a new folder, the installation wizard creates the folder.
5. Click **Next**.
The Installation Type step opens.
6. Select one of the following options and click **Next**.

- **Local**

Installs all DevTest Solutions components into a single directory on the local computer. By default, all data is stored in this directory, and each user has a personal temp directory. Local is the most common installation type that is used in most environments.

- **Shared**

Used by administrators to install all of the DevTest Solutions components to a shared location where multiple users can log in and can use the DevTest Workstation. All data and temporary files are stored in user-specified directories. Each user has personal data, but they share a common DevTest Solutions installation. With a shared installation, users only need read access to the DevTest Solutions programs directory. This installation type is a good option for a standalone installation.

The Select Components step opens.

7. Ensure that the **Server** check box is selected; an embedded Workstation is installed with the Server. Click **Next**.
The Enterprise Dashboard step opens.
8. 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 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 all registries are to connect to the new Enterprise Dashboard.

- **Existing Enterprise Dashboard Service**

If this is not the first Server that is installed in this network, enter the URL for the existing Enterprise Dashboard. This option specifies that the registry installed with this Server is to connect to the existing Enterprise Dashboard. Replace *localhost* with the appropriate host name:

```
tcp://localhost:2003/EnterpriseDashboard
```



To find the URL for an existing Enterprise Dashboard, navigate to the DevTest home directory and locate the **site.properties** file. The **lisa.enterprisedashboard.service.url** property defines the Enterprise Dashboard URL.

The Demo Server step opens.

9. 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.

10. Click **Next**.

11. If you chose the Local installation type, skip the next step, which applies to a Shared installation type.
12. Specify the data directory, clicking **Next** after each step.
The Select Start Menu Folder step opens.
13. (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.
14. Click **Next**.
The Desktop Icons step opens.
15. (Optional) If you do not want to create desktop icons for DevTest Enterprise Dashboard UI, DevTest Portal UI, and DevTest Workstation, clear the **Create a desktop icon** check box. Click **Next**.
- If you selected a Local installation type and you are an administrator, the Windows Services step opens.
 - If you selected a Local installation type and you are a nonadministrator, skip the next step.
 - If you selected a Shared installation type, skip to the information step.
16. (Optional) Select **Install Services** to create the following Windows services.
- DevTest Broker Service (for CAI)
 - DevTest Coordinator Service
 - DevTest Enterprise Dashboard 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.
17. Click **Next**.
The Select File Associations step opens. All associations are selected by default.
18. Leave all associations selected or clear the file extensions that you do not want to associate with DevTest Solutions.
19. Click **Install** to start the installation.
The Installing step opens. When the installation is finished, the Information step opens.
20. Read the information and click **Next**.
The Completing the DevTest Solutions Setup Wizard step opens.

21. Click **Finish**.

Continue as described in [How to Install DevTest Solutions \(see page 84\)](#).

Install DevTest Server on UNIX

Before you start the installation procedure, download the following files from the [Download Center \(see page 80\)](#):

- 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 the 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_platform_x64.sh -c
```



Note: If you are using the generic UNIX installer, ensure that a [Java virtual machine \(JVM\) \(see page 52\)](#) is on the same computer. The version of the JVM must be 1.7. You can specify a specific JVM by setting the **JAVA_HOME** environment variable. If the installer cannot find a JVM, the installer displays a message and exits.

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.

```
chmod 777 devtest_platform_x64.sh
```

This command gives `rwxrwxrwx` permissions on the file.

3. Run the installer file. For example:

```
./devtest_platform_x64.sh
```

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 Destination Directory step opens.

6. 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.

7. Click **Next**.

The Installation Type step opens.

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

- **Local**

Installs all DevTest Solutions components into a single directory on the local computer. By default, all data is stored in this directory, and each user has a personal temp directory. Local is the most common installation type that is used in most environments.

- **Shared**

Used by administrators to install all of the DevTest Solutions components to a shared location that multiple users from multiple computers can access. All data and temporary files are stored in user-specified directories. Each user has personal data, but they share a common DevTest Solutions installation. With a shared installation, users only need read access to the DevTest Solutions programs directory.

The Select Components step opens.

9. Ensure that the **Server** check box is selected and click **Next**.

The Enterprise Dashboard step opens.

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

- **New Enterprise Dashboard (Specify location of license file)**

If this is the first DevTest Server you are installing, click **Browse**, navigate to the location of the license file, select **devtestlic.xml** and click **Open**. When you click Next, the installer copies the devtestlic.xml file to the installation directory (LISA_HOME) on the local host. The Enterprise Dashboard process is installed in the LISA_HOME/bin directory.

- **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 URL for the existing Enterprise Dashboard (ED).

The Specify Demo Server step opens.

11. 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.

12. Click **Next**.

If you chose the shared installation type, the following steps prompt you to specify the data directory and the temporary files directory.

13. Specify the directories, clicking **Next** after each step.
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 desktop icons for, DevTest Enterprise Dashboard, DevTest Portal UI, and DevTest Workstation, clear the check box.
17. Click **Install** to start the installation.
When the installation finishes, the Information step opens.
18. Read the information and click **Next**.
The Completing the DevTest Solutions Setup Wizard step opens.
19. Click **Finish**.

Continue as described in [How to Install DevTest Solutions \(see page 84\)](#).

Install DevTest Server on a Mac

To preview installation choices, see [What You Can Install When You Run the Installer \(see page 82\)](#).

Before you start the installation procedure, download the following files from the [Download Center \(see page 80\)](#):

- 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 the 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 Destination Directory step opens.

4. 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.

5. Click **Next**.
The Installation Type step opens.

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

- **Local**

Installs all DevTest Solutions components into a single directory on the local computer. By default, all data is stored in this directory, and each user has a personal temp directory. Local is the most common installation type that is used in most environments.

- **Shared**

Used by administrators to install all of the DevTest Solutions components to a shared location where multiple users can log in and can use the DevTest Workstation. All data and temporary files are stored in user-specified directories. Each user has personal data, but they share a common DevTest Solutions installation. With a shared installation, users only need read access to the DevTest Solutions programs directory. This installation type is a good option for a standalone installation.

The Select Components step opens.

7. Ensure that the **Server** check box is selected; an embedded Workstation is installed with the Server. Click **Next**.

The Enterprise Dashboard step opens.

8. 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 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.

- **Existing Enterprise Dashboard Service**

If this is not the first Server that is installed in this network, enter the URL for the existing Enterprise Dashboard. This option specifies that the registry installed with this Server is to connect to the existing Enterprise Dashboard. Replace *localhost* with the appropriate host name:

`tcp://localhost:2003/EnterpriseDashboard`

The Demo Server step opens.

9. 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.

10. Click **Next**.
11. If you chose the Local installation type, skip the next step, which applies to a Shared installation type.
12. Specify the data directory, clicking **Next** after each step.
13. Click **Next**.
The Desktop Icons step opens.
14. (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**.
15. Click **Next**.
The Select File Associations step opens. All associations are selected by default.
16. Leave all associations selected or clear the file extensions that you do not want to associate with DevTest Solutions.
17. Click **Install** to start the installation.
The Installing step opens. When the installation is finished, the Information step opens.
18. Read the information and click **Next**.
The Completing the DevTest Solutions Setup Wizard step opens.
19. Click **Finish**.

Continue as described in [How to Install DevTest Solutions \(see page 84\)](#).



More Information:

- [Activate the Registries \(see page 95\)](#)
- [Verify Registry Activation \(see page 96\)](#)
- [Post-Installation \(see page 96\)](#)
- [Uninstall DevTest Server \(see page 104\)](#)

Activate the Registries

After you install all instances of DevTest Server, you start 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 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 start EnterpriseDashboard.exe.
2. If you have existing registries, [configure the existing registries \(see page 96\)](#).
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.exe 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.

Verify Registry Activation

After starting each registry, wait until the top 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**.
2. Log in.
 To log in, type *admin* in the **Username** field, type *admin* in the **Password** field, and click **Login**.
 The Enterprise Dashboard opens.
3. Examine the registry configurations on the Enterprise Dashboard.
 A list of your registries appears.
4. Verify that a registry from each new and existing DevTest Server displays on the Enterprise Dashboard.



Note: If existing registries are not displayed, [configure existing registries \(see page 96\)](#).

Post-Installation

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

- [Configure Existing Registries \(see page 96\)](#)
- [Using an HTTP/S Proxy Server - DevTest Server \(see page 99\)](#)
- [Running Components on Different Systems \(see page 101\)](#)
- [Calculate Simulator Instances \(see page 101\)](#)
- [Load and Performance Server Sizing \(see page 102\)](#)
- [Using DevTest Workstation with Your Java Environment \(see page 102\)](#)
- [Change the Default Project Home \(see page 103\)](#)
- [Project Directory Structure \(see page 104\)](#)

Configure Existing Registries

Each DevTest Server you install has one registry. Product licensing requires that the registries are configured. The configuration process varies by release:

- **Version 8.0 and above:** The installation process automatically configures new registries.

- **Releases 7.5.x:** Follow the procedure "To configure the registries (DevTest 7.5.x)"
- **Releases before 7.5:** Follow the procedure "To add a registry from a release earlier than DevTest 7.5"

To configure the registries (DevTest 7.5.x):

1. Log on to a computer where a DevTest Server is installed and navigate to the installation directory.
2. Open **site.properties** and locate Section 1 - Enterprise Dashboard. (If the **site.properties** file does not exist, copy **_site.properties** and rename the copy to **site.properties**.)

```
lisa.enterprisedashboard.service.url=tcp://somehost:2003/EnterpriseDashboard
```
3. Uncomment the following line and replace *somehost* with the host name where Enterprise Dashboard is installed.

```
lisa.enterprisedashboard.service.url=tcp://somehost:2003/EnterpriseDashboard
```
4. Save the **site.properties** file, and then close the file.
5. (Optional) If you plan to use DevTest Workstation from a remote computer, you can specify a property that lets you connect to a specific registry automatically.
 - a. Log on to the computer where Workstation is installed.
 - b. Navigate to LISA_HOME.
 - c. Open local.properties and locate Section 2 - Autoconnection.
 - d. Uncomment the following line and replace *somehost* with the host name of the computer where you are logged on.

```
lisaAutoConnect=tcp://somehost:2010/Registry
```
 - e. Save the local.properties file, and then close the file.
6. Log on to the computer with the target registry
7. Start or restart the registry.
 - a. Navigate to the LISA_HOME\bin directory.
 - b. Run registry.exe.
8. Repeat Steps 1-7 for each registry.
9. To verify registry configurations, browse to the Enterprise Dashboard.

```
http://hostname:1506
```



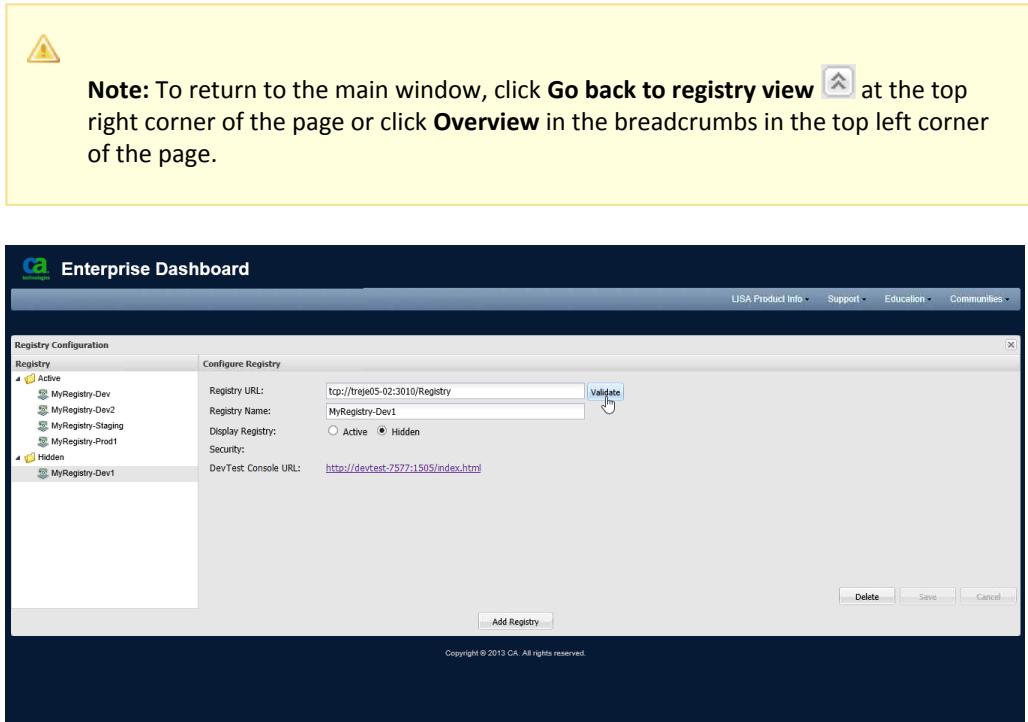
Important! If there is a change to the host name of the system on which you install DevTest Solutions, repeat this procedure.

The **Registry Configuration** window in the **Enterprise Dashboard** lets you add and configure older registries and also delete unused registries that display in the dashboard.

To add a registry from a release earlier than DevTest 7.5

1. Open the Enterprise Dashboard.
2. Click **Options** in the upper right corner of the **Enterprise Dashboard** main window.
3. Select **Configure**.

The Registry Configuration window opens with the Configure Registry fields blank.



Enterprise Dashboard registry configuration

4. Click **Add Registry** in the Registry Configuration window. Click **Yes** to confirm.
5. Complete the following fields:

■ Registry URL

The URL of the registry. This field is required.

Example:

tcp://hostname:2010/Registry

To validate the URL, click **Validate**.

■ Registry Name

The common name or nickname for this registry.

Example:

MyRegistry@*hostname*:2010

- **Display Registry**

Select **Active** to display this registry in the main panel of the dashboard, or **Hidden** to hide this registry. This parameter also controls which folder on the left side contains this registry.

The **Registry Configuration** window also includes the following fields:

- **Security**

Whether [ACL \(see page 1417\)](#) security is enabled or disabled.

- **DevTest Console URL**

The URL of the [Open the DevTest Console \(see page 369\)](#) for this registry. Example:

`http://hostname:1505/index.html`

6. Click **Save** to add the registry to the dashboard.

To validate a registry:

1. Select **Configure** from the **Options** menu.
2. Click the registry that you want to validate from the list of registries in the **Registry** column.
3. Click **Validate**.
The dashboard server queries the selected registry and displays the status, for example, is running.
4. Click **OK**.

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 accepted <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 accepted <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 challenge
## ===== 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

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.

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 \(see page 52\)](#)

Change the Default Project Home

By default, projects are saved in the **LISA_HOME\Projects** directory.

This procedure describes how to change the default location for the 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

The top level of the LISA_HOME directory includes an uninstall application. Use the DevTest Solutions Uninstall wizard to uninstall the DevTest Server components.

Follow these steps:

1. Stop DevTest Solutions.
 - a. Verify that all users have logged off DevTest workstations, the Portal, the DevTest Console, 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.
 - a. If the logs are stored in a custom directory, open the DevTest Workstation and select DevTest Runtime Info from the Help menu, and scroll to lisa.tmpdir. The path is the value for lisa.tmpdir.
 - b. Navigate to the lisatmp_release-number directory. The default location of each release-specific log directory (lisatmp_release-number) is the USER_HOME directory.
 - c. Right-click the directory and select **Delete**. Click **Yes** to the confirmation prompt.
3. If you are uninstalling a Shared installation type, manually delete the lisa.user.properties file. The default location is the USE_HOME directory for the user who installed DevTest.



Important! If you do not delete this file, future installations of a Local installation type will not install correctly.

4. Start the uninstall process.

- Windows: You can start this application from:
 - The Windows Start menu option
 - **DevTest Solutions Uninstaller**
 - The Control Panel, Programs, Programs and Features, Uninstall or change a program window.
- UNIX or Linux: Open DevTest, click uninstall, and select Run.

The DevTest Solutions Uninstall step opens.

5. Click **Next**.

6. To delete folders for the database, hotDeploy, lib/core, and related user preferences, select the **Delete all files** check box.

7. Click **Next**.

If you did not choose to delete all files, a Results of Uninstaller step opens with the list of files that could not be deleted.

8. 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 \(see page 349\)](#). 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 \(see page 105\)](#)
- [Install DevTest Workstation on UNIX \(see page 107\)](#)
- [Install DevTest Workstation on a Mac \(see page 109\)](#)

Install DevTest Workstation on Windows

Before you start the installation procedure, download the following files from the [Download Center \(see page 80\)](#):

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

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 Destination Directory step opens.
4. Specify the folder where you want to install one or more components of the DevTest Solutions. You can use a directory that contains spaces, such as **C:\Program Files\CA\DevTestSolutions**. If you specify a folder that does not exist, the Setup wizard creates it.
5. Click **Next**.
The Installation Type step opens.
6. Select one of the following options and click **Next**.
 - **Local**
Installs all DevTest Solutions components into a single directory on the local computer. By default, all data is stored in this directory, and each user has a personal temp directory. Local is the most common installation type that is used in most environments.
 - **Shared**
Used by administrators to install all of the DevTest Solutions components to a shared location that multiple users from multiple computers can access. All data and temporary files are stored in user-specified directories. Each user has personal data, but they share a common DevTest Solutions installation. With a shared installation, users only need read access to the DevTest Solutions programs directory.
7. Clear the **Server** check box, ensure that the **Workstation** check box is selected, and click **Next**.
If you chose the shared installation type, the following steps prompt you to specify the data directory and the temporary files directory.
8. Specify the directories, clicking **Next** after each step.
The Demo Server step opens.
9. 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.
10. Click **Next**.
The Select Start Menu Folder step opens.

11. 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.
12. 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.
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](#) ([see page 80](#)):

- 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
```



Note: If you are using the generic UNIX installer, ensure that a [Java virtual machine \(JVM\)](#) ([see page 52](#)) is on the same computer. The version of the JVM must be 1.7. You can specify a specific JVM by setting the **JAVA_HOME** environment variable. If the installer cannot find a JVM, the installer displays a message and exits.

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.

```
chmod 777 devtest_platform_x64.sh
```

This gives rwxrwxrwx permissions on the file.

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

```
./devtest_platform_x64.sh
```

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 Destination Directory step opens.

6. Specify the directory where you want to install DevTest Workstation. Do not use a directory that contains spaces. (The default is /opt/CA/DevTest.)

7. Click **Next**.

The Installation Type step opens.

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

- **Local**

Installs all DevTest Solutions components into a single directory on the local computer. By default, all data is stored in this directory, and each user has a personal temp directory. Local is the most common installation type that is used in most environments.

- **Shared**

Used by administrators to install all of the DevTest Solutions components to a shared location that multiple users from multiple computers can access. All data and temporary files are stored in user-specified directories. Each user has personal data, but they share a common DevTest Solutions installation. With a shared installation, users only need read access to the DevTest Solutions programs directory.

The Select Components step opens.

9. Clear the **Server** check box, ensure that the **Workstation** check box is selected, and click **Next**. If you chose the shared installation type, the following steps prompt you to specify the data directory and the temporary files directory.

10. Specify the directories, clicking **Next** after each step.

The Specify 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**.
The Select Additional Tasks step opens.
13. If you do not want to create a desktop icon for DevTest, clear the check box.
14. Click **Install**.
When the installation is finished, 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 a Mac

Before you start the installation procedure, download the following files from the [Download Center](#) (see page 80):

- 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 Destination Directory step opens.
4. 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.
5. Click **Next**.
The Installation Type step opens.
6. Select one of the following options and click **Next**.
 - **Local**
Installs all DevTest Solutions components into a single directory on the local computer. By default, all data is stored in this directory, and each user has a personal temp directory. Local is the most common installation type that is used in most environments.

- **Shared**

Used by administrators to install all of the DevTest Solutions components to a shared location that multiple users from multiple computers can access. All data and temporary files are stored in user-specified directories. Each user has personal data, but they share a common DevTest Solutions installation. With a shared installation, users only need read access to the DevTest Solutions programs directory.

The Select Components step opens.

7. Clear the **Server** check box, ensure that the **Workstation** check box is selected, and 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 DevTest Workstation desktop icon, 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.

When the installation finishes, the Information step opens.

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

The Completing the DevTest Solutions Setup Wizard step opens.

15. Click **Finish**.



More Information:

- [Using an HTTP/S Proxy Server - DevTest Workstation \(see page 111\)](#)
- [Environment Settings \(see page 112\)](#)

- Using DevTest Workstation with Your Java Environment (see page 102).

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 accepted <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 accepted <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 challenge
## ===== 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 OSX 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.



Note: For more information about environment settings, see [Common Properties and Environment Variables \(see page 390\)](#).

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.

Follow these steps:

1. Download the demo server as described in [Download DevTest Installers \(see page 80\)](#).
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 7 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 CA Continuous Application Insight. 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.

Verifying the DevTest Solution Installation

Contents

- [Start DevTest and Log In to UIs \(see page 114\)](#)
- [Start the DevTest Processes or Services \(see page 115\)](#)
- [Log in as the Standard Super User \(see page 118\)](#)
- [Create a User with the Super User Role \(see page 119\)](#)
- [Access the DevTest User Interfaces \(see page 120\)](#)

To verify successful installation of the DevTest Solution, start the DevTest Server, open each user interface and log in.

Start DevTest and Log In to UIs

Follow these steps:

1. Start the server components (see page 115).
2. Prepare to log in to the user interfaces. You can:
 - Log in as the standard DevTest Super User (see page 118).
 - Create a DevTest user with the Super User role (see page 119) for yourself and log in with your own credentials.
3. Access each user interface (see page 120) and log in. Examine the UIs.
4. (Optional) Examine DevTest Server directories (see page).

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: About DevTest Server Components (see page 71) describes the processes.

Start Order Sequence

Start the DevTest Server processes (or services) in the following sequence. (The Start menu shortcuts are shown.)

Follow these steps:

1. Start the **Enterprise Dashboard Server**.
2. Start each **registry**.
3. Start the **Portal**.
4. Start the following components in any order:
 - **Broker**
 - **Coordinator Server**
 - **Simulator Server** that is associated with each coordinator
 - **Virtual Service Environment**
5. If used on the DevTest Server:
 - **Workstation**

- **Demo Server**

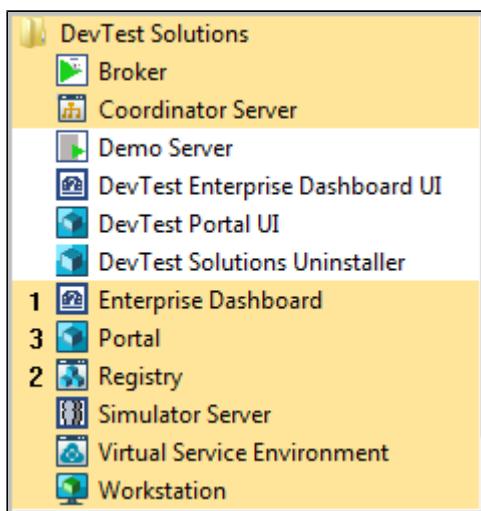


Note: To shut down the server components, use the reverse order.

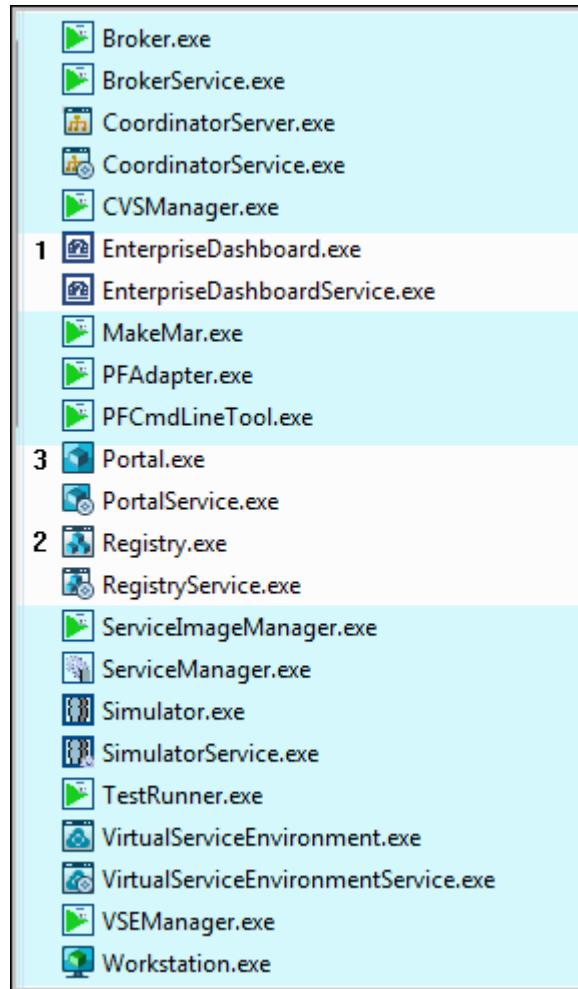
Ways You Can Start DevTest Server

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.

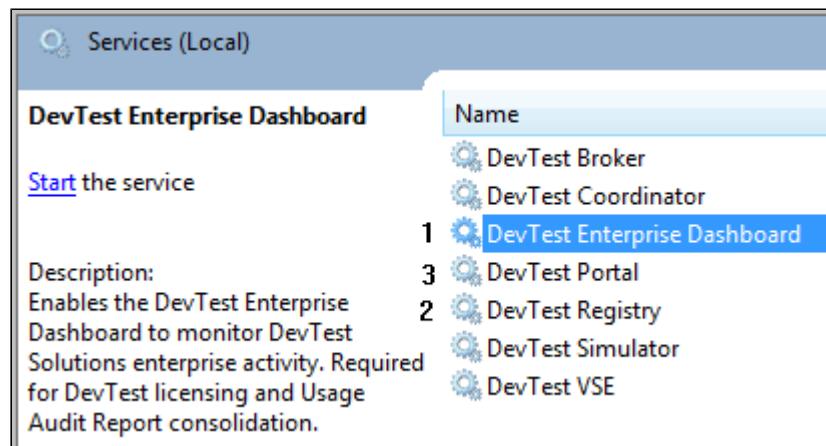


- Go to the **LISA_HOME\bin** folder. Start the executables in start order sequence.

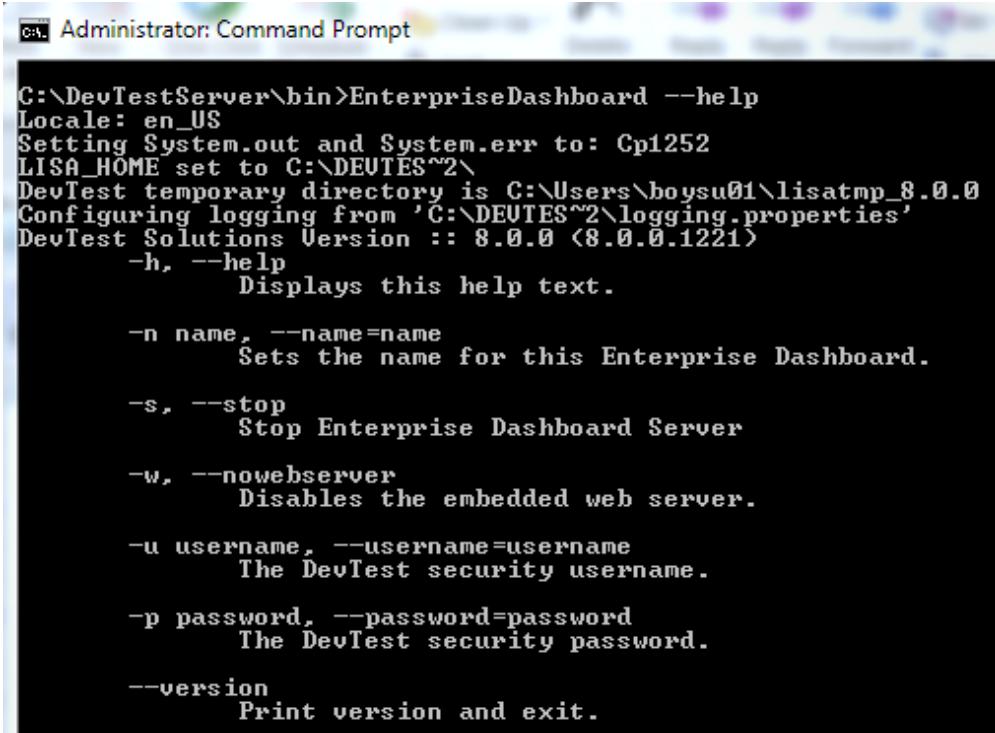


Start DevTest components

- (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**. For example:



```
C:\>DevTestServer\bin>EnterpriseDashboard --help
Locale: en_US
Setting System.out and System.err to: Cp1252
LISA_HOME set to C:\DEUTES^2\
DevTest temporary directory is C:\Users\boysu01\lisatmp_8.0.0
Configuring logging from 'C:\DEUTES^2\logging.properties'
DevTest Solutions Version :: 8.0.0 (8.0.0.1221)
-h, --help
    Displays this help text.

-n name, --name=name
    Sets the name for this Enterprise Dashboard.

-s, --stop
    Stop Enterprise Dashboard Server

-w, --nowebserver
    Disables the embedded web server.

-u username, --username=username
    The DevTest security username.

-p password, --password=password
    The DevTest security password.

--version
    Print version and exit.
```

Log in as the Standard Super User



Important! Access Control List (ACL) (see page 1417) is mandatory. No one can use DevTest Solutions without first logging in with a valid name and password.

Setting up security involves creating role-based user accounts and specifying the ACL with LDAP or with credentials that you define. 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 \(see page 119\)](#). To access the UIs before you configure LDAP, use the standard user that is defined with the Super User role.

When you access the DevTest Workstation, a login dialog opens.

To log in, type **admin** in the **Username** field, type **admin** in the **Password** field, and click **Login**. The DevTest Workstation opens.

When you browse to the 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**.

The DevTest Console presents a similar login dialog. You can log in to that browser with the same standard user credentials.

Create a User with the Super User Role

Grant yourself full access to DevTest Solutions.

Follow these steps:

1. Browse to the Server Console.

`http://localhost:1505`

2. Log in as the standard administrator user.

- a. Enter *admin* for the user name.
- b. Enter *admin* for the password.

3. Expand the Administration pane in the left navigation bar. In the Security area, the Super User provides credentials to authenticate users and grants permissions to access features through role assignments.



4. Create a user account for yourself with a password you will not forget. Assign a Super User role for full access to DevTest Solutions.

- a. At the bottom of the right panel, click **Add User**.
- b. In the **User ID** field, enter a unique ID that is composed of any combination of alphanumeric, hyphen (-), underscore (_), period (.), and ampersand (@) characters.
- c. In the **Password** field, enter a password that is composed of any combination of alphanumeric, hyphen (-), underscore (_), and ampersand (@) characters.
- d. In the **Re-type Password** field, enter the password again.
- e. In the **Name** field, enter the user name (alphanumeric, hyphen (-), underscore (_), and space characters).

- f. In the **Roles for the User** area, select **Super User**.
- g. Click **Add User**.
- h. Click **OK**.

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 DevTest User Interfaces

The Enterprise Dashboard, the registry, and the Portal you are using must be running for all UIs. See [Start the Server Components \(see page 115\)](#).

Access the UIs as described here. Then, [log in as the standard Super User \(see page 118\)](#) or log in with your own credentials, where you [defined yourself with the Super User Role \(see page 119\)](#).

Demo Server

To learn about DevTest Solutions, browse to the demo server on your local host, and follow the tutorials in [Getting Started \(see page 195\)](#).

- Browse to the Demo Server and log in.
`http://localhost:8080/lisabank`
- (Windows) From the Start menu, expand **DevTest Solutions** and select **Demo Server**.

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

Before you use CA Continuous Application Insight, install the DevTest Java agent. Browse to the following URL and log in to the DevTest Portal to access CAI. Specify the host name of a computer with a running registry. See [Using CA Continuous Application Insight \(see page 1007\)](#).

CAI requires the Portal to be running. 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 \(see page 318\)](#) and [Using CA Service Virtualization \(see page 661\)](#).

- 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. Workstation is where you create and edit tests and also stage tests. Some alternative methods include TestRunner and junitlisa ant task, for example. See [Using CA Application Test \(see page 318\)](#). The UI requires a coordinator server and a simulator server to be running.

- (Windows) From the Start menu, expand **DevTest Solutions** and select **Workstation**, specify the registry, and log in.
- Go to `LISA_HOME\bin` and run the `Workstation.exe`.

DevTest Console

The Console includes links to the Server Console (including VSE), CVS Dashboard, and Reporting Dashboard. Specify the host name of a computer with a running registry. For information about using the Administrative tab in the Server Console, see [Administering \(see page 1362\)](#). For other information, see [Using CA Service Virtualization \(see page 661\)](#). The UI for CA Service Virtualization requires VSE to be running.

- Browse to the DevTest Console and log in.
`http://hostname:1505`
- Open Workstation and click **Server Console**.

Installing Integration Tools

This section describes how to install and configure third-party tools that can be used with DevTest.

- [Install Performance Monitor \(Perfmon\) \(see page 121\)](#)
- [Install and Configure SNMP \(see page 122\)](#)
- [Run TCPMon \(see page 124\)](#)
- [Install the HP ALM - Quality Center Plug-in \(see page 125\)](#)
- [Install IBM Rational Quality Manager \(see page 127\)](#)
- [Configure Integration with CA APM \(see page 130\)](#)
- [Set Up the SAP System Landscape Directory \(see page 146\)](#)

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 \(see page 1634\)](#).

Install and Configure SNMP

Contents

- [SNMP support on UNIX \(see page 123\)](#)
- [SNMP support on Windows \(see page 123\)](#)
 - [Install Microsoft SNMP Agent on Windows 7 \(see page 123\)](#)
 - [Install Microsoft SNMP Agent on Earlier Windows Versions \(see page 123\)](#)
 - [Configure Microsoft SNMP Agent \(see page 124\)](#)

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 provide an agent that is able to answer SNMP requests and send traps.

Install Microsoft SNMP Agent on Windows 7

Follow these steps:

1. From the Start menu, click **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 \(see page 318\)](#).

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 the HP ALM - Quality Center Plug-in

The HP ALM - Quality Center plug-in lets you load and run a DevTest test case as a Quality Center test from the HP ALM - Quality Center suite. You can import into and can run DevTest tests from Quality Center. 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 and DevTest callbacks returning from any system under test. DevTest tests are executable in the workflow of Quality Center, and they report back results to maintain the context and status of the testing process.

The HP ALM - Quality Center plug-in only runs on Microsoft Windows. The DevTest Server and test cases must run on Microsoft Windows.

The following software must be installed:

- DevTest 8.2 or later, 32-bit version (including 32-bit JRE and 32-bit browser)



Important! On a 64-bit system, with a 64-bit DevTest or a 64-bit JVM, your test DevTest cases will not run with the ALM plug-in.

- DevTest HP ALM - Quality Center Plug-in
- HP Quality Center 10, HP ALM 11, or HP ALM 12

HP Quality Center Client Side components for v 10: HP Quality Center Connectivity, HP Quality Center Client Registration

HP ALM Client Side components for v 11 or V 12: HP ALM Connectivity, HP ALM Client Registration

- .NET 3.5 Runtime

When you install the plug-in on a computer that has DevTest installed, the **QCRunner.exe** file is added to the **DEVTEST_HOME\bin** directory. **QCRunner.exe** is the COM server that gets registered as part of the install process. DevTest is invoked using the Quality Center VAPI-XP interface. The VAPI-XP script creates an instance of our COM object. That in turn stages the test to be executed and listens for the test results. Finally, the COM object takes the results and updates the Quality Center instance with the results from DevTest.

The following files are also installed:

- Script_Template_js.txt
- Script_Template_vbs.txt

These files contain scripts that are used during the process of setting up DevTest tests in Quality Center.



Note: For information about using the HP ALM - Quality Center plug-in, see *Using CA Application Test*.

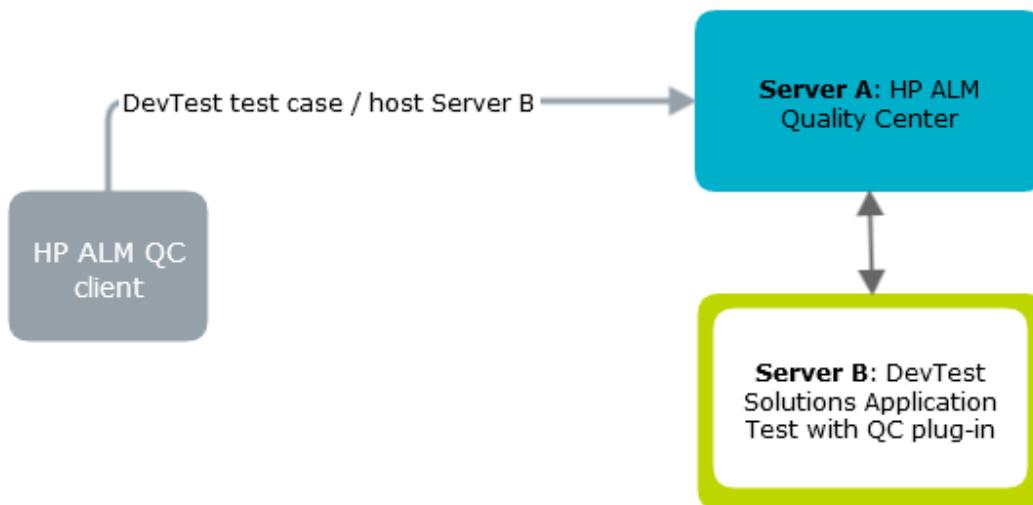
Follow these steps:

1. Ensure that the .NET runtime is installed on the client computer.
2. Install the HP ALM Quality Center Client Side components.
3. Install DevTest.
4. Go to the **DEVTEST_HOME\addons\qc** directory and run the **td_plugin.exe** file.
5. Complete the wizard steps.

Using the HP ALM - Quality Center Client to Run Test Cases

Your Quality Center users can run DevTest test cases without having CA Application Test installed on their machines. The following diagram shows a test case being initiated from a Quality Center client to run on the CA Application Test installed on Server B.

Running a test case from a remote QC client



Install IBM Rational Quality Manager

Contents

- [Implementation \(see page 128\)](#)
- [Install the Adapter UI \(see page 128\)](#)
- [Run Command Line Adapter \(see page 129\)](#)
- [General Usage Workflow \(see page 129\)](#)

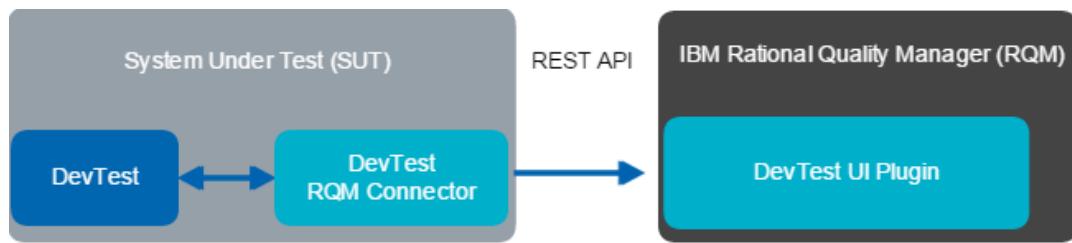
IBM Rational Quality Manager is a web-based centralized test management environment for test planning, workflow control, tracking, and metrics reporting. Rational Quality Manager is extensible in a number of ways including the ability to use "connectors" or plug-ins to bridge between RQM and external systems. The DevTest RQM plug-in lets you reuse or create DevTest test cases and tie them into RQM. The DevTest test cases can then be executed from the RQM interface. Results from test runs are aggregated into the RQM test execution and reporting history.

Implementation

The complete DevTest RQM solution is implemented as two components:

- The DevTest RQM Connector
- The web UI extension.

The web UI allows for the extension of the out-of-the-box UI. This UI also allows the necessary parameters to be passed and displayed between DevTest and RQM. The DevTest RQM Connector is used to respond to work tasks that the RQM schedules. The task information that the UI collects is passed to the connector, and the connector invokes TestRunner with the necessary parameters.



The connector takes advantage of the TestRunner -html switch and uses it to generate the HTML output that is later uploaded back to RQM. After the test completes, the results of the test are uploaded to RQM and associated with the test run. The user can then inspect the results through the RQM interface.

Install the Adapter UI

Follow these steps:

1. Go to the **LISA_HOME\addons** directory and locate the zip file named **rqm-adapter.zip**.
2. Unpack the file and locate the file **com.itko.lisa.integration.ibm.rqm.update.site.zip** in the folder **lisa-adapter-ui**.
3. On the RQM server, unzip the file **com.itko.lisa.integration.ibm.rqm.update.site.zip**.
4. Open the file **com.itko.lisa.integration.ibm.rqm.adapter.web.update.ini** and update the reference to point to the location of the update site.
5. Copy the .ini file that you modified to the following location: **<RQM install root>\server\server\conf\jazz\provision_profiles**.

6. Update the server configuration by using the Rational Quality Manager Server Reset service. The reset utility is found at the following URL:
[https://<hostname>:<portnumber>/jazz/admin?internal#action=com.ibm.team.repository.admin.serverReset|https://%3chostname%3e%3cportnumber%3e]
The service prompts you to log in to the IBM Rational Quality Manager Admin console. The user ID must have administrator privileges.
7. Stop the RQM server.
8. Restart the Rational Quality Manager server.

Run Command Line Adapter

Follow these steps:

1. Go to the **LISA_HOME\addons** directory and locate the zip file named **rqm-adapter.zip**.
2. Unpack the file and locate the child folder **lisa-adapter2.0** in the folder.
3. From a command line, start the adapter. The following parameters are needed:
 - -repository [https://rqmserver:port/jazz|https://%3crqmserver%3e%3cport%3e]
 - [https://%3crqmserver%3e%3cport%3e]-user *userid*
 - -password *password*
 - -adapter *adapter*(com.itko.lisa.integration.ibm.rqm)
 - -adapterName *adapterName* (LISA RQM Adapter)## -LISA_HOME *lisaHomePath* (d\:/lisa)
 - [-projectArea *project area* (Quality%20Manager)]
 - [-sleepTime *sleep time(5)*]

General Usage Workflow

Create a test script in RQM.

1. Associate a DevTest test case, staging document, and optional config document with the test script.
2. Associate one or more test scripts with a test case, and the test case is ready to run.
3. Select to run the test case, select a running adapter, and click **OK**.
4. The server contacts the system under test through the RQM Connector plug-in, which in turn invokes DevTest.
5. Wait for the test to run, and then verify the run results in RQM.

Configure Integration with CA APM

DevTest Solutions is available in two versions, Workstation and Server. The Workstation version is designed to create and verify DevTest test cases and virtual simulations of server components. The Workstation runs in a single Java process (the DevTest Workstation) and is therefore limited in the size of such tests and simulations. The Server version breaks out a number of functions into one or more separate Java processes that can be run on multiple systems, if necessary.

All DevTest Java processes can be instrumented with CA Application Performance Management (APM) Introscope agents. This instrumentation allows generic metrics that describe the state of the process to be reported to the Introscope Enterprise Manager. These metrics include:

- CPU and memory usage
- Garbage collection operations
- Use of sockets and backend databases

In addition, the DevTest TestEvent tracer can instrument the DevTest coordinator to generate a series of metrics describing the execution of DevTest test cases. This instrumentation is available regardless of whether the coordinator is run as a separate process or in DevTest Workstation. DevTest Workstation is an interactive process with a Windows-based UI. Other than DevTest Workstation, all the other DevTest processes can be run either as regular Windows processes or as Windows services. Both kinds of process can be instrumented with Introscope agents.



More Information:

- [Instrumenting DevTest Processes \(see page 130\)](#)
- [Introscope Agent Files \(see page 132\)](#)
- [Metrics Reported by Tracers \(see page 132\)](#)
- [Deriving DevTest Metrics \(see page 134\)](#)
- [Tracer Configuration \(see page 134\)](#)
- [Install Introscope Agent \(see page 136\)](#)
- [Configuring Tracer Logging \(see page 136\)](#)
- [Typical Metrics Reported \(see page 137\)](#)
- [Displaying Metrics \(see page 139\)](#)
- [Reporting \(see page 146\)](#)

Instrumenting DevTest Processes

The **LISA_HOME\bin** directory contains a Windows executable file for each DevTest process. This folder includes separate versions for those processes that can be run as regular Windows processes and as Windows services. The list of processes and their executable file names is:

DevTest Process Name	Process Type	Executable Name
Coordinator	Regular	CoordinatorServer.exe

Coordinator	Service	CoordinatorService.exe
Enterprise Dashboard	Regular	EnterpriseDashboard.exe
Enterprise Dashboard	Service	EnterpriseDashboardService.exe
Portal	Regular	Portal.exe
Portal	Service	PortalService.exe
Registry	Regular	Registry.exe
Registry	Service	RegistryService.exe
Simulator	Regular	Simulator.exe
Simulator	Service	SimulatorService.exe
Virtual Service Environment	Regular	VirtualServiceEnvironment.exe
Virtual Service Environment	Service	VirtualServiceEnvironmentService.exe
Workstation	Regular	Workstation.exe



Note: This list excludes executables in the bin directory that are not relevant for the Introscope instrumentation.

To instrument any DevTest process with an Introscope agent, create a file with the same name as the executable and the extension **.vmoptions**. For example, to instrument the DevTest Workstation, create a **Workstation.vmoptions** file in the **bin** directory under the DevTest Server installation directory. The .vmoptions file contains the following two lines:

```
-javaagent:<AGENT_HOME>/Agent.jar
-Dcom.wily.introscope.agentProfile=<AGENT_HOME>/core/config/IntroscopeAgent.profile
```

<AGENT_HOME> is the path to a DevTest-specific Introscope agent installation. Typically this path is an absolute path, but it can also be a path relative to the current directory in which the DevTest process runs.

To have the Introscope agent report under a name other than the default DevTest Agent, add a line defining the **Java com.wily.introscope.agent.agentName** system property. For example, to call the agent DevTest Workstation 6.0.5.87 Agent, add the following line:

```
-Dcom.wily.introscope.agent.agentName=DevTest Workstation 6.0.5.87 Agent
```

To set other JVM command-line options or system properties as required, add extra lines. Each option must be specified in a separate line.

The Introscope agent installation that you use can be specific to the DevTest installation, or even to the DevTest process. You can create variations in agent configurations by varying contents of the **IntroscopeAgent.profile** in those separate agent installations. Or, to create multiple agent profiles in a single agent installation, modify the **com.wily.introscope.agentProfile** system property value that is defined in each **.vmoptions** file. This approach allows different levels of tracing or logging (or even different Introscope Enterprise Managers) to be used to monitor different DevTest processes.

For example, a site can include a production use of DevTest that uses a coordinator node that is run as a Windows service. This site can also include a test/experimental use that uses a coordinator that is run as a regular Windows process. This coordinator could even be the one that is built into the DevTest Workstation. The production metrics are reported to the production EM, while the test/experimental metrics are reported to the test EM. This can be achieved by using multiple profiles and nominating the correct profile in the relevant .vmoptions file. It can also be achieved by overriding the value of **introscope.agent.enterpriseManager.connectionorder** and **introscope.agent.enterpriseManager.transport.*** properties that are defined in the profile by setting Java system properties of those names in the .vmoptions file. Consult the Introscope Java Agent Guide for the full range of Introscope Agent configuration options.

Introscope Agent Files

The DevTest-specific Introscope agent files are shown in the following table. For more details on the contents of these files and how they can be modified to customize the instrumentation of DevTest, see [Tracer Configuration \(see page 134\)](#).

File	Location relative to <AGENT_HOME>	Contents
Lisa.jar	core\ext	DevTest-specific tracers, name formatters.
Introscope Agent. profile	core\config	DevTest-specific agent profile. By default, references lisa-typical.pbl to define a set of tracers used to instrument DevTest processes.
lisa.pbd	core\config	Tracer and instrumentation point definitions.
lisa-full.pbl	core\config	List of PBD files to be used for "full" DevTest instrumentation. References lisa-toggles-full.pbd .
lisa-toggles- full.pbd	core\config	List of toggles used to turn on specific instrumentation features when instrumenting in "full" mode.
lisa-typical. pbl	core\config	List of PBD files to be used for "typical" DevTest instrumentation. References lisa-toggles-typical.pbd .
lisa-toggles- typical.pbd	core\config	List of toggles used to turn on specific instrumentation features when instrumenting in "typical" mode.

Metrics Reported by Tracers

DevTest tracers generate metrics at one or more of the four levels:

- DevTest
- Test Case
- Simulator
- Test Step

Metrics that are created at the simulator and test step levels are also aggregated to a user-configurable maximum level (test case, by default). If a test case executes on two simulators and it contains two test steps, the "Responses Per Interval" metric is generated for each test step under each simulator. Then (depending on the configuration) the values are aggregated to generate the "Responses Per Interval" metrics for each simulator. Finally (depending on the configuration) the values are aggregated again to generate the "Responses Per Interval" metric for the test case. The highest level to which any metric can be rolled up is the test case level. Thus, the only metrics that are generated at the DevTest level are listed in the following table with a level of DevTest.

Data from all events of relevant type in each agent reporting interval of 15 seconds are combined as shown.

Metric	Lev	Source
	el	
Average Response Time (ms)	Tes t Ste p	Average of "Long Description" value (converted to integer) from all "Step response time" events (id 18)
Responses Per Interval	Tes t Ste p	Count of all "Step response time" events (id 18)
Errors Per Interval	Tes t Ste p	Count of all "Step error" events (id 20).
Failures Per Interval	Tes t Ste p	Count of all "Abort" (id 50) and "Cycle failed" (id 13) events.
Tests Running	Tes t e	Count of test cases running at the end of agent reporting interval. Count is incremented for each "Test started" event (id 4) and decremented for each "Test Case ended" event (id 5).
Virtual Users Running	Sim	Count of virtual users running at the end of agent reporting interval. Count is updated incremented for each "Cycle started" event (id 11) and decremented for each or "Cycle ending" event (id 24).
Test Runner Errors Per Interval	Dev Tes t	Count of all "Cycle runtime error" events (id 25)
Staging Errors Per Interval	Dev Tes t	Count of all "Model definition error" events (id 23)

The Errors Per Interval metrics for any test step, simulator, or test case are reported only when the first error event is detected. The same is true for Failures Per Interval.

If only a few, short test cases run, the Tests Running and Virtual Users Running metrics can continue to report 0 for each agent reporting cycle. This happens when both start and stop events occur in the same 15-second period, and the count at the end of the period is 0.

When the metrics are being aggregated, the lower-level metric can be created in one agent time reporting period. The aggregated metrics can be created in the following period. Occasionally, there are points where lower level and rolled up metrics within a time period do not agree in value. This situation is unavoidable.

Deriving DevTest Metrics

The DevTest-specific Test Event tracer is a "method" tracer. A method tracer means that it contains code that is inserted immediately before and after a specific method call. The Test Event tracer is designed to instrument the method that is the constructor of the BaseCoordinator class. After the constructor completes, the tracer creates a Test Event Listener and registers this listener with the coordinator. The listener then receives each TestEvent object of the types that were shown previously. The listener extracts relevant information from these TestEvent objects and sends that information to the Introscope Enterprise Manager in the form of Introscope metric values.

The DevTest Workstation screen shot that follows shows the TestEvents generated by one run of a test case named "broken," which contains an error. To understand which of the Introscope metrics are created, look up each value in the **Event** column, in [Metrics Reported by Tracers \(see page 132\)](#).

Timestamp	Event	Simulator	Instance	Short Info	Long Info
2012-02-27 11:02:14,797	Test ended	local	0/0	65349165136333...	316365122095633...
2012-02-27 11:02:14,033	Cycle ending	local	0/0	65349165136333...	316365122095633... Signal to stop test
2012-02-27 11:02:14,033	Abort	local	0/0	65349165136333...	316365122095633... Client version: 1.2
2012-02-27 11:02:14,483	Stop error	local	0/0	65349165136333...	Unknown protocol: http://www.introscope.com/
2012-02-27 11:02:16,311	Cycle started	local	0/0	65349165136333...	316365122095633...
2012-02-27 11:02:16,300	Test started	N/A	0/0	65349165136333...	N/A

Sample test event metrics

Tracer Configuration

All DevTest tracing can be disabled by removing all references to DevTest-specific .pbl files (**lisa-full.pbl** or **lisa-typical.pbl**) in the agent profile (IntroscopeAgent.profile by default). You can also delete **Lisa.jar**. However, if there are any references remain to the tracers defined in **lisa.pbd**, the agent fails to start correctly. This failure is due to an attempt to use a tracer that cannot be found in any agent extension.

The Test Event tracer can be disabled by commenting out the "TurnOn: LisaTestEventTracing" directive in the active DevTest toggles file (**lisa-toggles-full.pbd** or **lisa-toggles-typical.pbd**).

You can use a number of other parameters to limit the number of metrics that are reported. These parameters are defined in the **lisa.pbd** file. You can control the following parameters:

- **minMetricLevel:** Parameter that controls the lowest level at which metrics are reported. The parameter can take any of the following values:
 - Lisa
 - TestCase (Default Value)
 - Simulator
 - TestStep
- **maxRollupLevel:** Parameter that controls the maximum level to which metrics can be aggregated. The parameter can take any of the following values:
 - TestCase (Default Value)
 - Simulator
 - TestStep.

Metrics that are generated below the current setting of **minMetricLevel** are aggregated until they reach **minMetricLevel**. Those metrics are then reported from that level up to **maxRollupLevel**, unless **minMetricLevel** is above the **maxRollupLevel**. If **minMetricLevel** is above the **maxRollupLevel**, metrics are not reported at all. Any metrics that are generated at a level above the **maxRollupLevel** are reported only at the generated level, not at the aggregated level.

In addition, six more parameters allow the Test Event tracer to restrict metrics to selected combinations of test cases, simulators, or test steps. You can configure the tracer to include or exclude based on the name of the test case, simulator, or test step. You can match the name against regular expressions or a pair of regular expressions. All names that match the inclusion regular expression and that do not match the exclusion regular expression are selected to report metrics. If the inclusion regular expression is not defined, all names are included. If the exclusion regular expression is not defined, no names are excluded.

By default, all possible test cases, simulators, and test steps are included. Only the internal test steps named "abort" and "end" are excluded. If the regular expression contains certain special characters (for example, the | symbol that is used to specify alternative patterns), the entire regular expression must be in double quotation marks. The quotation marks are optional otherwise. The following shows the default inclusion and exclusion patterns that are defined in **lisa.pbd**.

```
SetTracerParameter: LisaCoordinatorTracer includeTestCasesRegExp ""
SetTracerParameter: LisaCoordinatorTracer excludeTestCasesRegExp ""
SetTracerParameter: LisaCoordinatorTracer includeSimulatorsRegExp ""
SetTracerParameter: LisaCoordinatorTracer excludeSimulatorsRegExp ""
SetTracerParameter: LisaCoordinatorTracer includeTestStepsRegExp ""
SetTracerParameter: LisaCoordinatorTracer excludeTestStepsRegExp "abort|end"
```

You can configure the metric path under which metrics are reported. The DevTest level part of the metric path is the last value that is specified in the **TraceOneMethodWithParametersIfFlagged** directive for the [set the init variable for your book] method. This defaults to "DevTest". For the remaining levels, the following parameters define the parts of the metric path for the levels Test Case, Simulator, and Test Step:

- **pathComponentForTestCase**
- **pathComponentForSimulator**
- **pathComponentForTestStep**

Avoid the following character usage in the values for these parameters:

- The colon character (:) is not valid in metric paths.
- Metric paths cannot begin or end with the metric path node separator (|).
- Metric paths cannot contain two adjacent metric path node separators (||).

An underscore (_) replaces each occurrence of a colon or vertical bar character. Leading and trailing spaces are removed from these names. The value "Unknown" replaces any name with a null value or an empty string.

The default values for these parameters are applied if the parameter is commented out or is an empty string. Each value must contain the relevant placeholder to indicate the place at which the name of the relevant level is inserted into the metric path. The following shows the default values defined in **lisa.pbd**:

```
SetTracerParameter: LisaCoordinatorTracer pathComponentForTestCase "Test Case|{TestCase}"
SetTracerParameter: LisaCoordinatorTracer pathComponentForSimulator "Simulator|{Simulator}"
SetTracerParameter: LisaCoordinatorTracer pathComponentForTestStep "Step|{TestStep}"
```

Install Introscope Agent

You can install the Introscope agent for DevTest by extracting the contents of the distribution file **CALISAIntegrationNoInstaller9.1.1.0.zip** (on Windows) or **CALISAIntegrationNoInstaller9.1.1.0.tar** (on UNIX/Linux).

To avoid profile file name conflicts, extract this file to a directory that does not contain any of the other Introscope Java agent distributions. Then create one or more vmoptions files to instrument the set of DevTest processes to instrument. The only DevTest processes that report DevTest-specific metrics are the DevTest Workstation and the coordinator. You can instrument other processes. However, the other processes report a minimal set of metrics. These metrics are typically limited to CPU usage, memory usage, and agent and JVM identity.

Configuring Tracer Logging

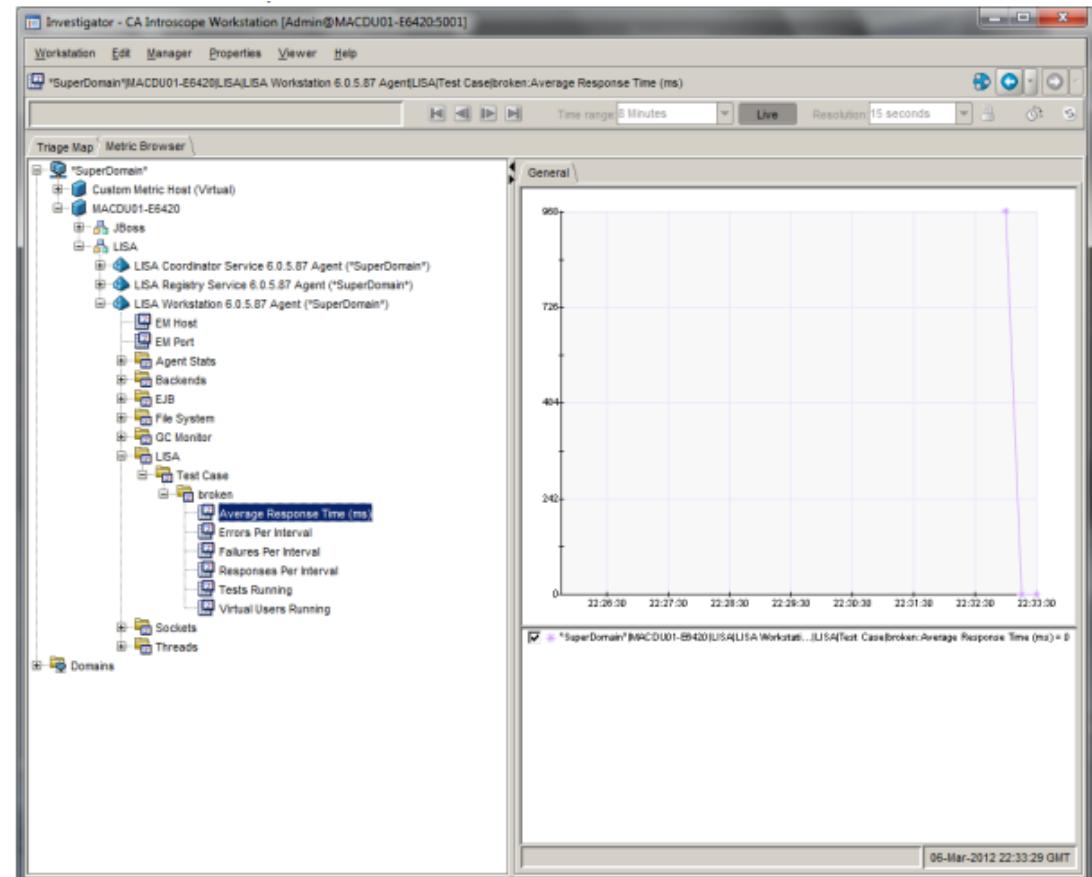
To write more information about the operation of the DevTest-specific tracer to the agent log file, add the following lines to the agent profile:

```
log4j.additivity.IntroscopeAgent.LisaCoordinatorTracer=false
log4j.logger.IntroscopeAgent.LisaCoordinatorTracer=DEBUG, logfile
```

If the agent logging level is set to DEBUG, these lines are not required. However, logging at that level writes a large amount of information to the agent log. This amount of information can make it difficult to find the lines from the DevTest tracer.

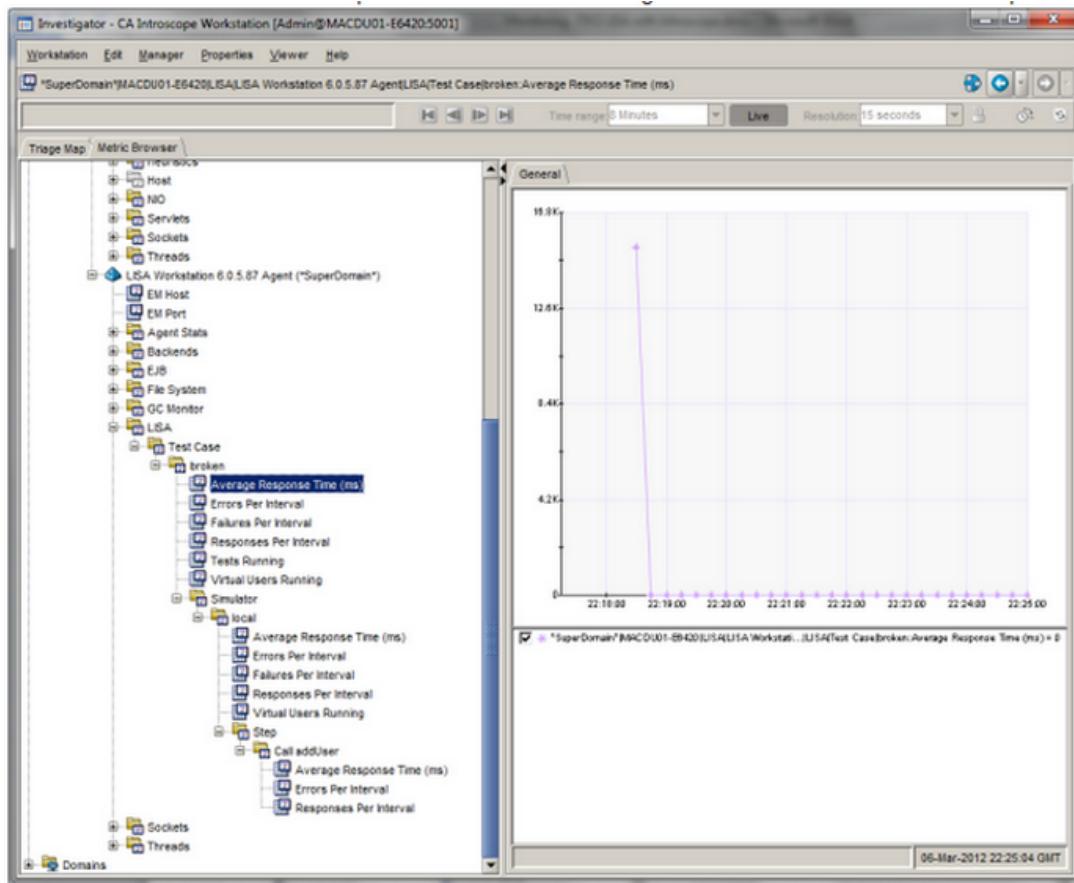
Typical Metrics Reported

The following graphic shows typical sets of metrics that the Test Event tracer reports. In the following example, the simple test case named "broken" was executed. This test case contains only one test step that contains a configuration error. This error causes the test case to fail. The failure causes the "Errors Per Interval" and "Failures Per Interval" metrics to be reported as shown.



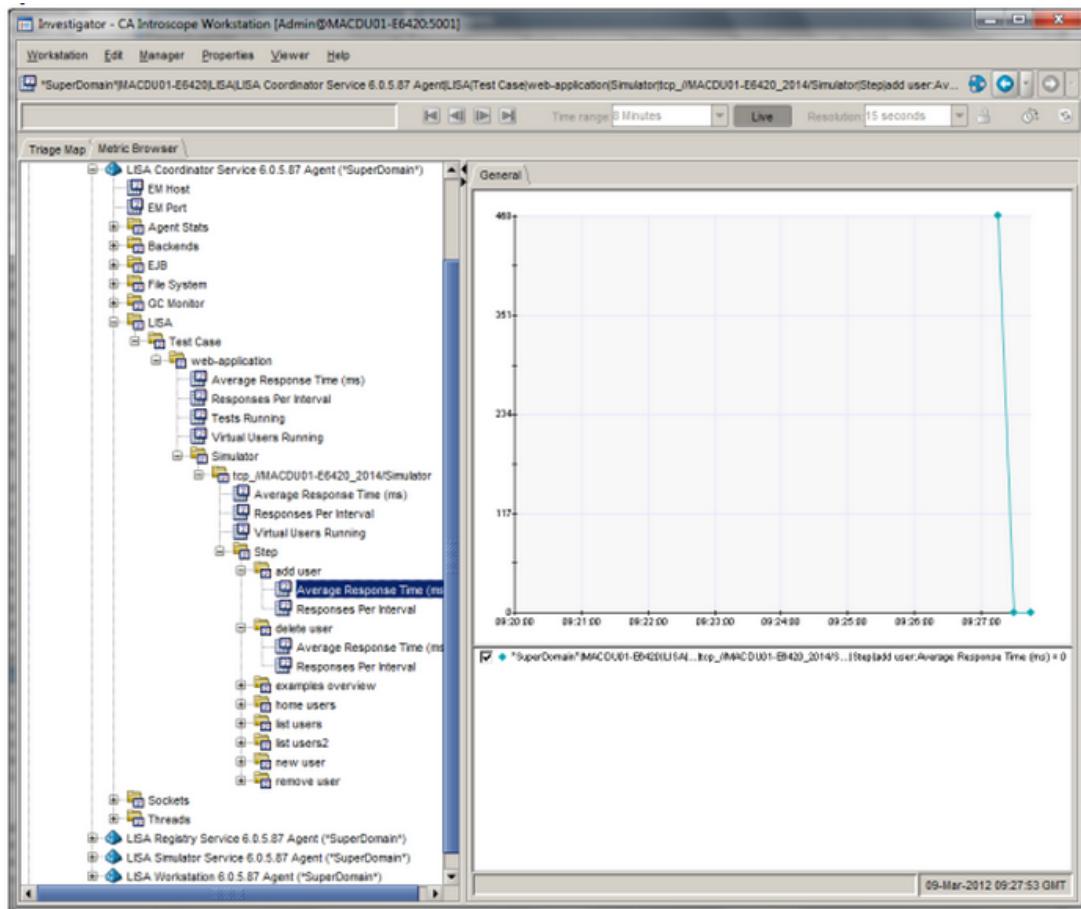
Typical sets of metrics reported by DevTest Test Event tracer

The following graphic shows the same test case that was run after the **minMetricLevel** parameter was changed from the default of TestCase to TestStep. Metrics are generated for simulator and test step levels.



Test case run after the minMetricLevel parameter is changed from TestCase to TestStep

The graphic that follows shows a more complex test case that contains eight test steps and does not generate any errors when run. In addition, the test has been staged using separate coordinator and simulator processes rather than the processes built into the DevTest Workstation. This window shows the additional test step nodes, but no "Errors Per Interval" or "Failures Per Interval" metrics. The metrics are reported under a Coordinator Service Agent node (whose name has been customized to include the DevTest version number). Also notice that because the simulator name contained colon characters that are invalid in metric path names, the colons were replaced by underscores.



Complex DevTest test case run with CA Application Performance Management

Displaying Metrics

The Introscope Enterprise manager ships with a DevTest-specific management module that contains metrics groupings, alerts, dashboards, and reports relevant to monitoring the DevTest environment. Two dashboards are provided:

- [Overview Dashboard \(see page 139\)](#)
 - [Test Case Overview Dashboard \(see page 142\)](#)
 - [Simulator Overview Dashboard \(see page 143\)](#)
 - [Test Step Overview Dashboard \(see page 144\)](#)
 - [System Under Test Overview Dashboard \(see page 145\)](#)

Overview Dashboard

The **Overview** dashboard gives a high-level overview of all instrumented processes and of the tests being run in the DevTest installation.

The Tests section of the **Overview** dashboard includes four simple alert icons:

- **Failures**

Set to caution if any test has reported a failure in the last reporting interval. Set to danger if any test has reported two or more failures in the last reporting interval.

- **Errors**

Set to caution if any test has reported an error in the last reporting interval. Set to danger if any test has reported two or more errors in the last reporting interval.

- **Runner Errors**

Set to caution if the Test Runner Errors Per Interval metric has a value of 1 for any reporting interval. Set to danger if it has a value of 2 or higher for any reporting interval.

- **Staging Errors**

Set to caution if the Staging Errors Per Interval metric has a value of 1 for any reporting interval. Set to danger if it has a value of 2 or higher for any reporting interval.

This section also includes the following elements:

- **Overall alert**

A summary alert that is set to caution if any of the preceding four alerts is set to caution. This alert is set to danger if any of the preceding three alerts is set to danger.

- **Test Average Response Time (ms) graph**

Shows up to ten Average Response Time (ms) metric graphs for test case, simulator, and test steps. The ten graphs that are chosen have the top ten metric values in the time period displayed.

- **Test Responses Per Interval Graph**

Shows up to ten Responses Per Interval metrics graphs for test case, simulator, and test steps. The ten graphs that are chosen have the top ten metric values in the time period displayed.

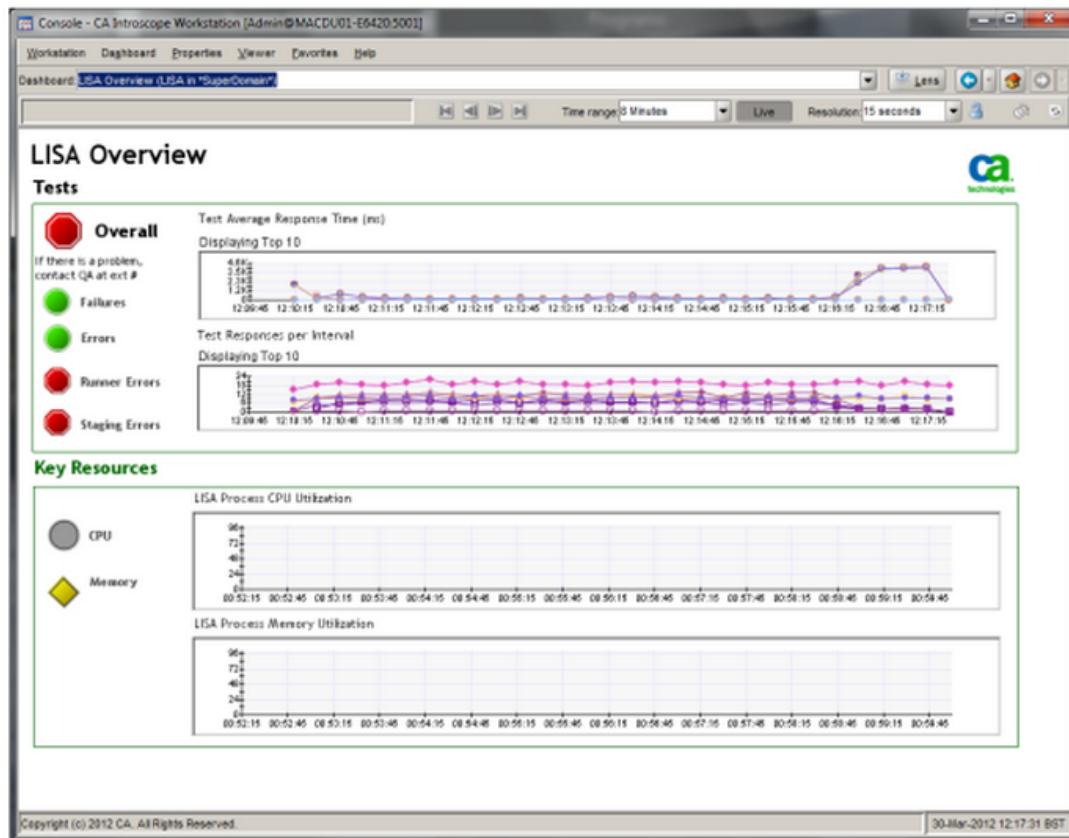
- **LISA Process CPU Utilization Graph**

Shows a graph of **CPU:Utilization Percent** (process) metrics for all instrumented processes.

- **LISA Process Memory Utilization Graph**

Shows a graph of the **GC Heap:Bytes In Use** metrics for all instrumented processes.

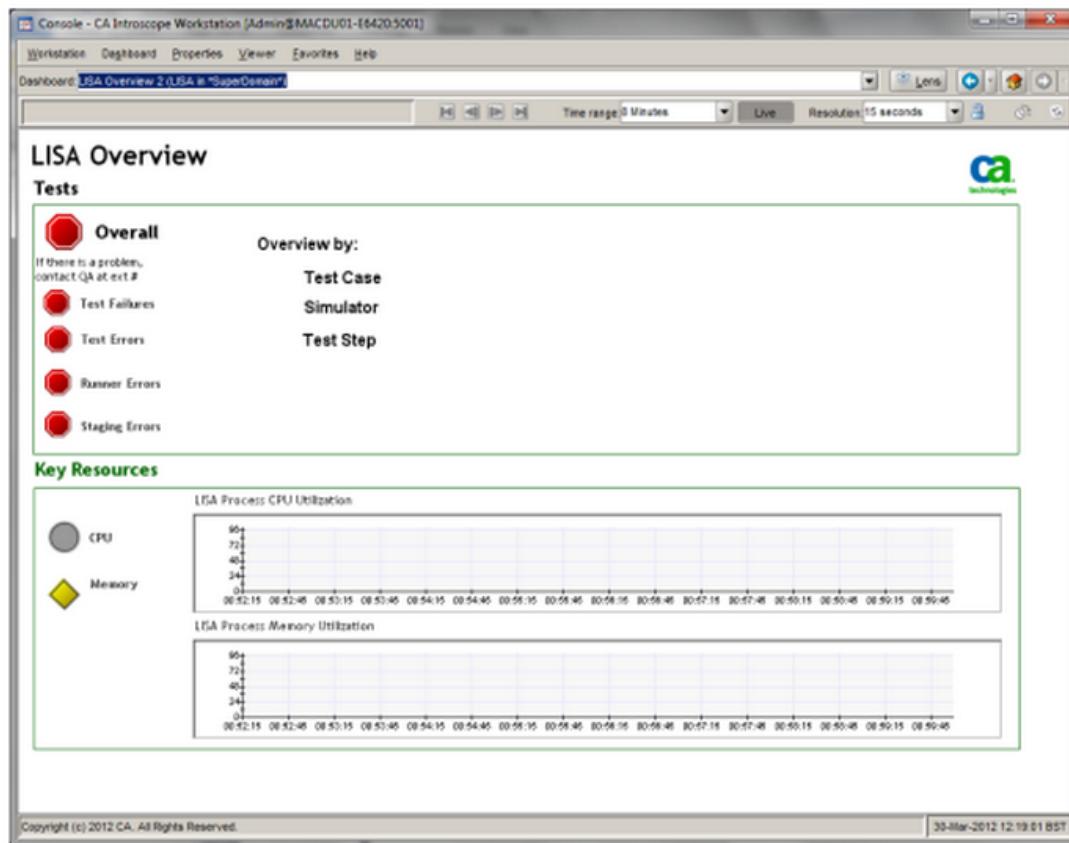
The following graphic shows the **LISA Overview** Dashboard.



Alternate view of the LISA Overview dashboard

An alternate version of the **Overview** dashboard replaces the graphs in the top left panel with links to three new dashboards. These dashboards provide overviews of the following metrics:

- Test case metrics
- Simulator metrics
- Test step metrics



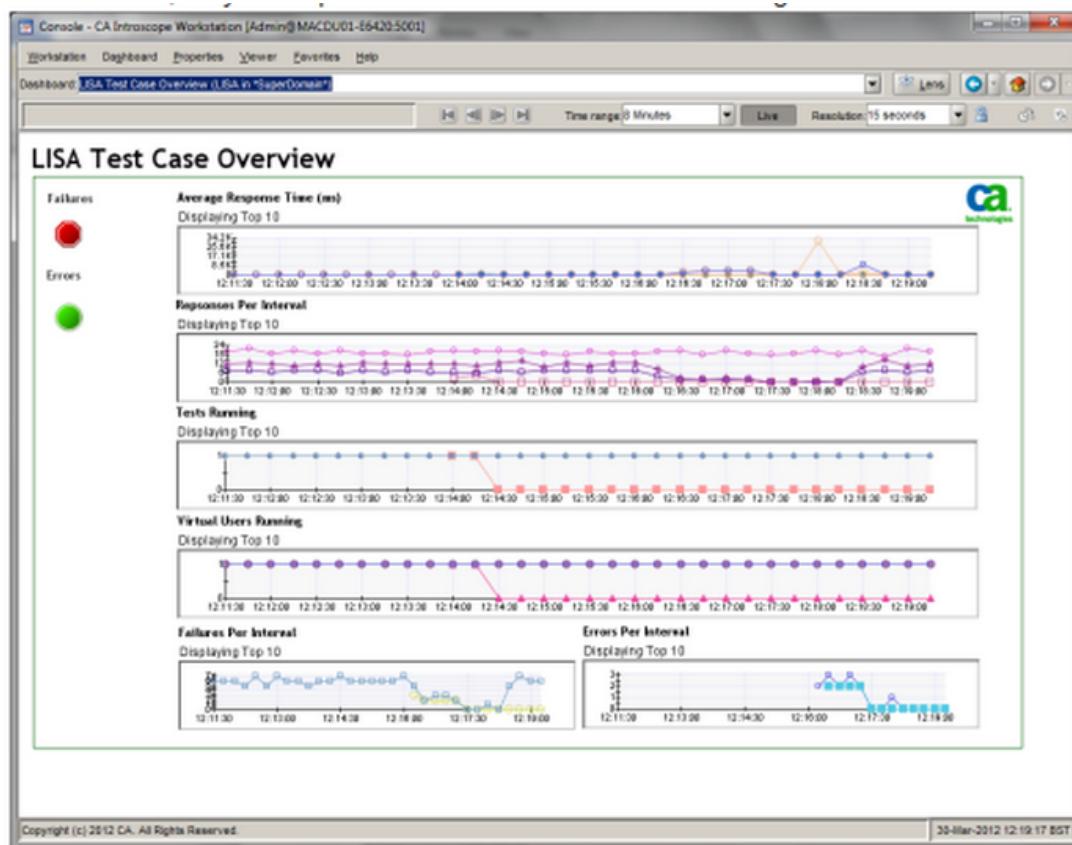
Test Case Overview dashboard

Test Case Overview Dashboard

The **Test Case Overview** dashboard shows graphs for the following metrics:

- Average Response Time (ms)
- Responses Per Interval
- Tests Running
- Virtual Users Running
- Failures Per Interval
- Errors Per Interval

In each case, only the top ten metrics are shown.



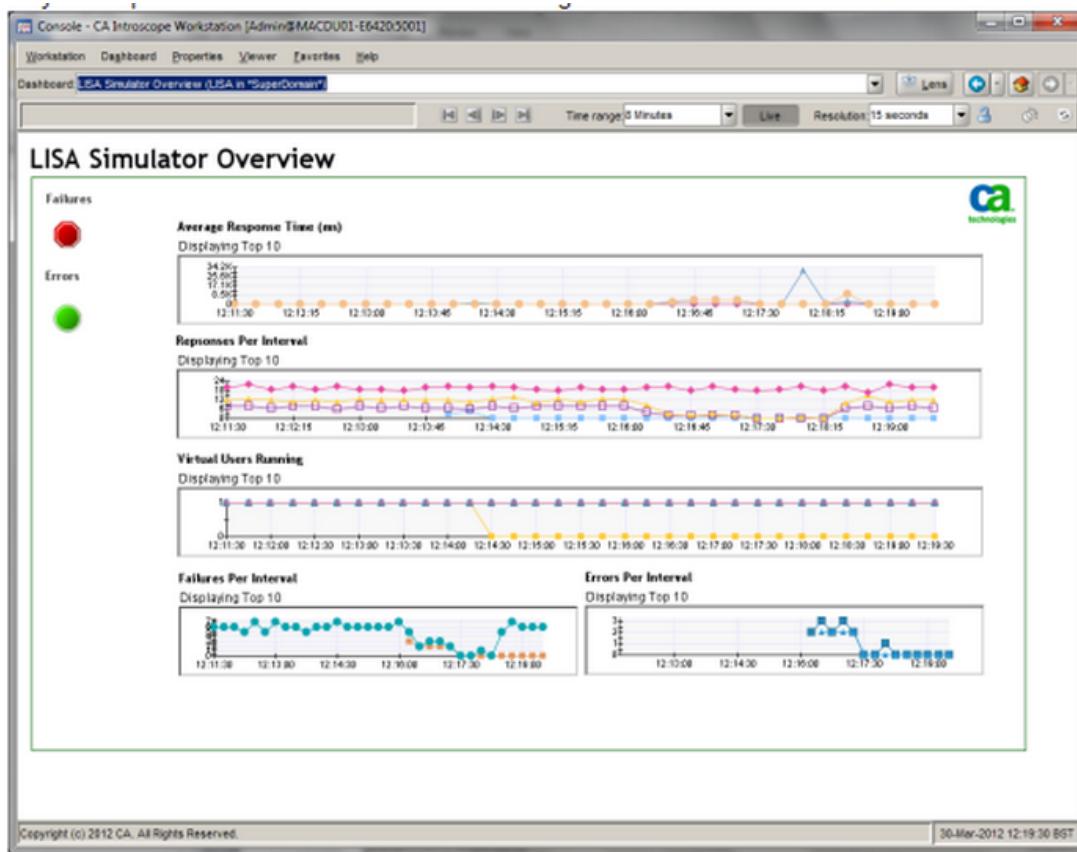
Test Case Overview Dashboard

Simulator Overview Dashboard

The **Simulator Overview** dashboard shows graphs for the following metrics:

- Average Response Time (ms)
- Responses Per Interval
- Virtual Users Running
- Failures Per Interval
- Errors Per Interval

In each case, only the top ten metrics are shown.



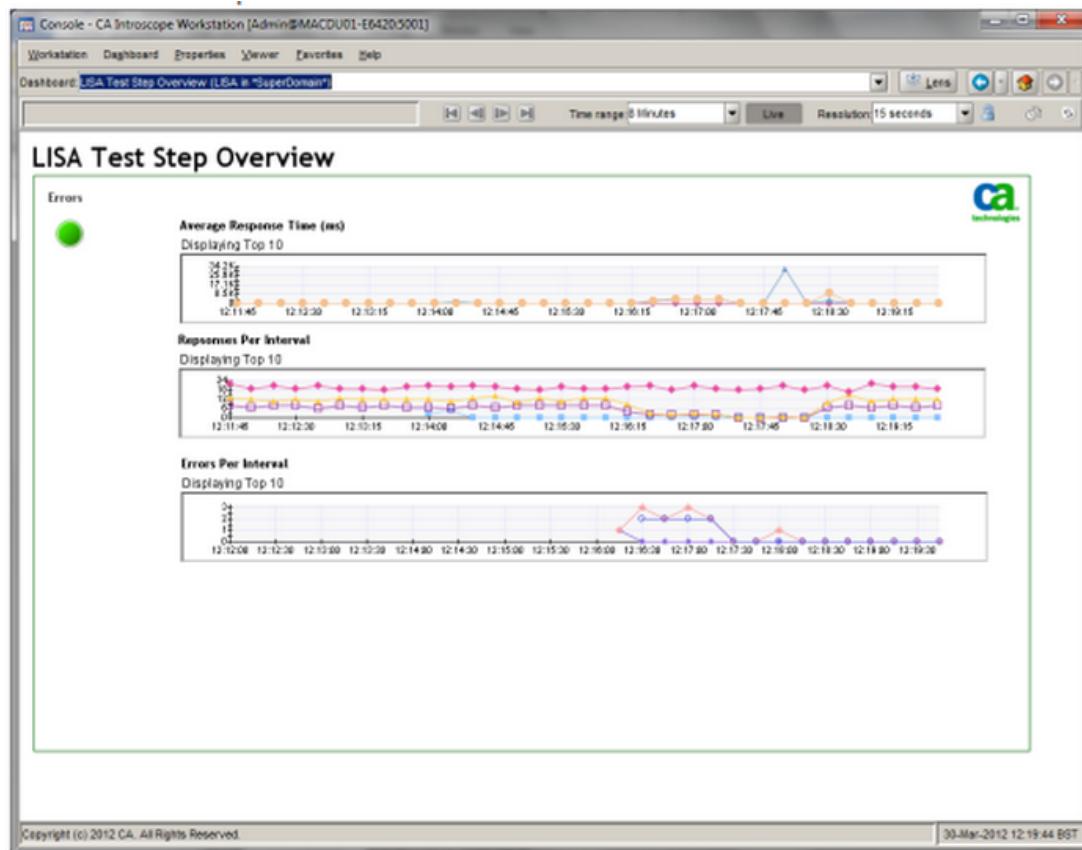
LISA Simulator Overview dashboard

Test Step Overview Dashboard

The **Test Step Overview** dashboard shows graphs for the following metrics:

- Average Response Time (ms)
- Responses Per Interval
- Errors Per Interval

In each case, only the top ten metrics are shown.



LISA Test Step Overview dashboard

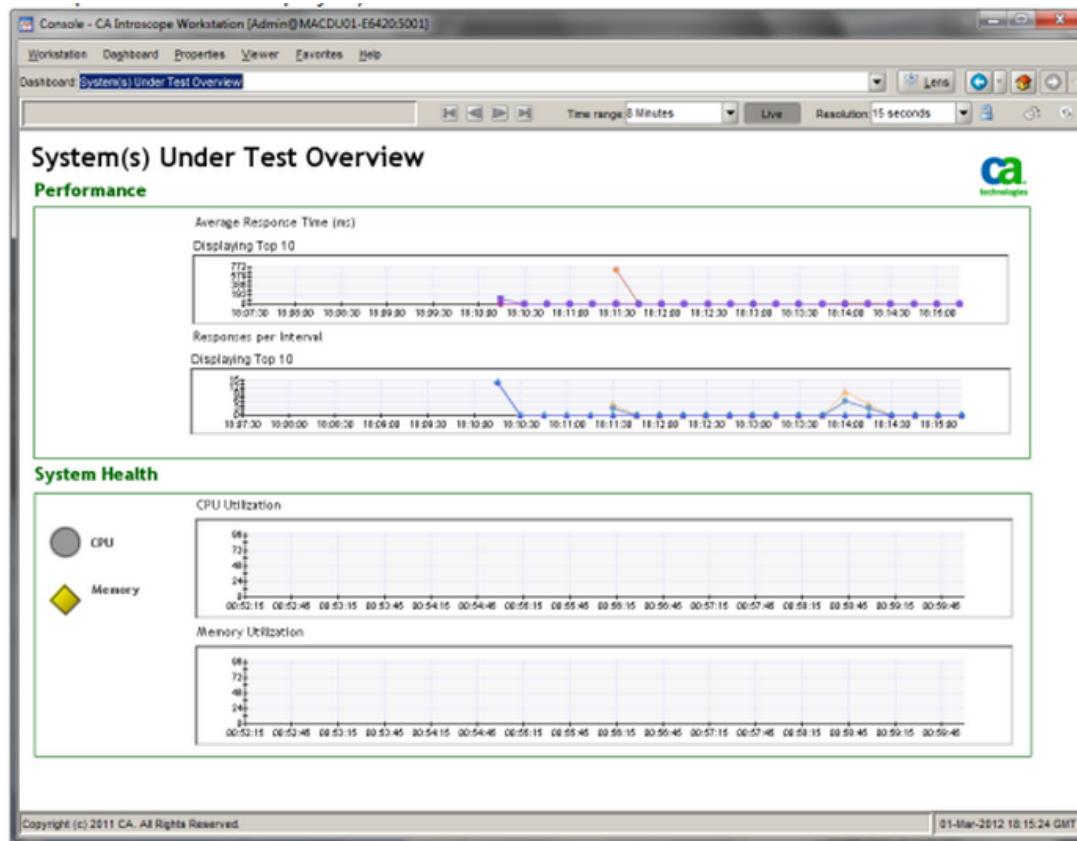
System Under Test Overview Dashboard

The System(s) Under Test Overview gives a high-level overview of systems that are executing transactions that are sent from DevTest tests. Identifying the systems that DevTest is testing is difficult. You can edit the metric groupings to define which of the systems that Introscope instruments are also systems that DevTest is testing. By default, the grouping contains the demo server.

The **System(s) Under Test** dashboard displays metrics from systems under test by DevTest. To define which agents instrument systems under test, edit the Metric Grouping Agent Expression for all metric groups that are defined in the Management Module with names starting with "Systems Under Test". By default this expression is set to `(.)|(.)|JBoss LISA Demo Server(.*)`, which selects the demo server.

The dashboard is divided into two sections. The top section shows metrics on the top ten front-end Average Response Time (ms) and Responses Per Interval metrics from all application servers under test.

The bottom section shows CPU and memory utilization for all application servers under test. This section also contains two alert icons that indicate alerts relating to CPU and memory utilization.



System(s) Under Test Overview Performance dashboard

Reporting

The management module contains a report named DevTest Report. This report contains a series of graphs describing the state of the systems under test by DevTest. The report also includes test metrics that DevTest gathers from the execution of those tests. You can duplicate and refine the report to focus it upon specific tests. To focus on specific tests, modify the duplicate metric grouping expressions to select only those metrics relevant to the specific report. You can also drop individual graphs from the duplicate report.

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 \(see page 147\)](#)
- [Import the DevTest SLD XML File \(see page 147\)](#)

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, **V7.5**, 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, **V7.5**, 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 \(see page 1448\)](#).
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 [Property Files \(see page 392\)](#).
All property files are human readable java name=value entries.

Setting Up the Mobile Testing Environment

Contents

- [Supported Operating Systems for Mobile Testing \(see page 148\)](#)
- [Supported Mobile Operating Systems \(see page 148\)](#)
- [Hardware Requirements for Mobile Testing \(see page 149\)](#)

This page describes the system requirements for testing mobile applications in DevTest Solutions.

Supported Operating Systems for Mobile Testing

The following operating systems are supported:

- **Mac OSX 10.8 - Mountain Lion**
VM Image is supported.
- **Mac OSX 10.9 - Mavericks**
VM Image is supported.
- **Windows 7 or Later**
Only remote iOS tests are supported. Local iOS tests are not supported.

Supported Mobile Operating Systems

The following mobile operating systems are supported:

- iOS 8.1 and later
- Android 4.2-5.1
- Native, Hybrid, and Web

Hardware Requirements for Mobile Testing

The following hardware is required to support mobile testing:

CPU

- Intel Core i5 2.0 GHZ or AMD equivalent

Memory

- Cloud or Remote Mobile Tests: 4GB
- Local Simulators: 8GB



More Information:

- [Preinstallation Steps for Mobile Testing \(see page 149\)](#)
- [Using Genymotion \(see page 154\)](#)

Preinstallation Steps for Mobile Testing

You must perform the following tasks before you can start testing mobile applications with DevTest Solutions.

Download the Latest Version of Android Studio

Android Studio provides the tools that are used to develop Android applications. DevTest Solutions requires the Android SDK Manager (included in Android Studio) if you are performing testing on Android devices. Mobile Testing only supports the Android SDK portion of the Android Studio bundle.



Note: The minimum version that Mobile Testing supports is SDK Platform 19 rev 3.

Download the latest version here:

<http://developer.android.com/sdk/index.html#download>

Define ANDROID_HOME

For your Android Virtual Devices (AVDs) to function properly, you must define an ANDROID_HOME property.

Follow these steps:

1. In DevTest Workstation, click **System, Edit Properties**.
The System Properties file opens.

2. Include the following edits:

```
ANDROID_HOME= <The fully qualified path to your Android Studio Bundle  
installation>
```

For example, on Windows:

```
ANDROID_HOME= C:\Android Studio\sdk
```

3. Click **Save**.

4. Close the System Properties window.

Configure the Android SDK Manager

Once you download and install Android Studio, you must install additional APIs and tools within the Android SDK Manager that are used to develop Android applications.

Install Additional APIs

These APIs must be installed before using the Android SDK Manager, as the initial installation lacks essential tools that you need to perform tests on Android devices.

Follow these steps:

1. Open Android Studio: from the installation directory, open a file named Android Studio (Mac) or studio64.exe (Windows).



Make sure that you set the JAVA_HOME property in your computer settings to point to the JDK location.

Android Studio opens.

2. Select **Configure, SDK Manager** to open the Android SDK Manager.
3. In the Packages window, install Android 4.2.2 (API 17) or greater.

Install Android SDK Build-tools

The Android SDK bundle contains a component called Build-tools. Mobile testing requires at least version 19.0.1, 19.1.0, or 20.0.0.

If these versions are not installed with your ADT bundle, you can encounter an error message when creating a mobile asset in DevTest Workstation:

Cannot find the aapt command. Please install Android SDK Build-tools Rev 20.

Or

Cannot find the zipalign command. Please install Android SDK Build-tools Rev 20.

If you encounter this error message, update your Android SDK Build-tools version.

Follow these steps:

1. Open the Android SDK Manager from Android Studio (**Select Configure, SDK Manager**).
2. Under the Tools folder, ensure that 19.0.1, 19.1.0, or 20.0.0 is installed and selected. Version 20.0.0 is the optimal version to use.

Enable USB Debugging (Windows Only)

USB Debugging Mode can be enabled in Android after connecting your device directly to a computer with a USB cable. Enabling this mode lets you facilitate a connection between an Android device and the Android SDK.

USB Debugging Mode can be found in your device's Developer Options under the Settings menu.

Create an Android Virtual Device

Use the Android AVD Manager to create one or more Android Virtual Devices (AVDs).

You can create as many AVDs as you would like to use with the Android SDK Manager. To effectively test your app, you should create an AVD that models each device type for which you have designed your application to support.

Follow these steps:

1. Open the Android SDK Manager.
2. Select **Tools, Manage AVDs**.
3. Click **Create**.
The **Create New Android Virtual Device (AVD)** page opens.
4. Complete the following fields:
 - AVD Name**
Name of the Android Virtual Device that to create. Use this name in the Android AVD field when creating an Emulator Session Asset.

■ Device

Select the type of device to test.

■ Target

Select the target for your AVD.

■ CPU/ABI

Specifies the underlying CPU emulator type that the AVD is made for. Most mobile devices use ARM.

You must specify a system image for the API level of your AVDs. In the Android SDK Manager, select a System Image for each API version that you install.

■ Memory Options

Set the RAM value to 512 and the VM Heap value to 32.

■ Emulator Options

Clear the **Use Host GPU** check box. Increase the window size if this option is not initially visible.

5. Accept the default values for the remaining fields, or customize them for your AVD.

6. Click **OK**.

Your new device is added to the list of devices in the Android Virtual Device Manager.

Install Xcode Version 6.2 (Mac Only)

Mobile Testing requires Xcode version 6.2 on Macs (later versions are not supported). Xcode is available free from the Mac App Store.

Follow these steps:

1. Go to <https://developer.apple.com/support/xcode>.
2. Download and install Xcode 6.2.



Running iOS 7 tests from Xcode 6 is not supported. Xcode 5 must be installed to run iOS 7 tests.

Prepare IPA Signing (Mac Only)

To test iOS apps on devices, you must get SSL certificates and provision your device for testing. This can be done at the Apple Developer Member Center.

You can then provision your device with the mobile provision file to identify the apps that are signed with a certificate. *Provisioning* is the process of preparing and configuring an app to launch on devices and use app services.

Adding the mobile provision file to an .ipa file (through IPA signing) ensures that the signatures on the certificate match the signatures of those who can run the apps on their devices.

Mobile Testing can sign iOS .ipa files before deploying them to your test device. Some configuration is required for it to function properly.

Follow these steps:

1. Log in to the [Apple Developer Member Center](https://developer.apple.com/membercenter/) (<https://developer.apple.com/membercenter/>).
2. Click **Certificates, Identifiers, & Profiles**.
3. Connect your iOS device with a USB cable.
4. Under iOS Apps (on the left), click **Devices**.
5. To add your device, click the + icon.



Note: You can also add your device through iTunes.

6. Confirm that your device is included in the list of iOS devices
7. Under Certificates, click **Development**.
8. Click the certificate in the list.
9. Click **Download**, and save it to your hard drive.
10. Install this certificate into Keychain Access.
11. Under Provisioning Profiles, click **Development**.
12. Select the iOS Team Provisioning Profile in the list.
13. Download this file and save it on your hard drive.
The file name is similar to this example: `iOS_Team_Provisioning_Profile_.mobileprovision`.
14. Open DevTest Solutions.
15. In the `lisa.properties` property file, set the following project properties:

```
MOBILE_PROVISION = path to iOS_Team_Provisioning_Profile_.mobileprovision
IOS_CERTIFICATE = name of the certificate as displayed in Keychain Access (not the certificate file name itself)
```



Note: You do not need to type the exact string, as the codesign utility can match it. You can use a name such as "iPhone Developer" and that would be enough to be a unique name. However, if you have other iOS certificates, it is good practice to use the entire certificate name.

If a developer profile is made available from another team member, you can import it into XCode.

Follow these steps:

1. Click **XCode, Preferences, Accounts**.
2. Click the gear icon in the bottom left panel and select **Import Accounts**.
3. Select the file and enter the password.

Click [here](https://developer.apple.com/library/ios/recipes/xcode_help-accounts_preferences/articles/import_signing_assets.html) (https://developer.apple.com/library/ios/recipes/xcode_help-accounts_preferences/articles/import_signing_assets.html) for more information from the Apple Development site.

Enable UI Automation (Mac Only)

UIAutomation should be enabled in iOS 8 devices. Enabling this mode lets Appium detect instruments.

The UIAutomation option can be found under Settings, Developer, Enable UI Automation.

Using Genymotion

Contents

- [Managing Genymotion Devices \(see page 155\)](#)

Genymotion is another emulator that uses VirtualBox as an Android emulator. Genymotion is faster and provides overall better performance than the Android emulator provided with the Android Software Development Kit (ASDK). This tool requires additional setup.

Follow these steps:

1. Download and install both VirtualBox and Genymotion from www.genymotion.com (<http://www.genymotion.com>).



Note: Installing Genymotion requires you to create a Genymotion user account, but the basic download is free. The Windows platform has a combined Virtualbox /Genymotion installer.

2. Add your virtual devices.
 - a. Start Genymotion.
The first time you start Genymotion, you are prompted to add a virtual device.
 - b. Click Yes.
The Create a New Virtual Device window opens.

- c. Click Connect and enter your Genymotion user ID and password to view available devices.
A list of available virtual devices displays. You can filter the list of devices by Android version or device model at the top of the page.
- d. Select the device you want to add, then click Next.
- e. Review the details for the selected device.



Note: The name of the device is the name you enter in the asset dialog in DevTest Solutions. You can change the default Genymotion name to a simpler name.

- f. Click Next.
The download of your virtual device begins. These download images are typically around 200MB in size.
- g. Click Finish when the download is complete.
- h. To add another virtual device, click Add in the Genymotion main window and repeat the previous steps.

3. Create a config file and asset in DevTest for your new virtual device.
 - a. Create a standard Simulator Session asset for an Android virtual device. For more information, see [Mobile Assets \(see page 1519\)](#).
For more information, see [Mobile Tutorial 2 - Record an Android Test Case \(see page 292\)](#).
 - b. In the AVD field, select your Genymotion device.

Managing Genymotion Devices

You can manage your Genymotion devices with the VBoxManage tool that is installed with VirtualBox.

The following is a list of useful command-line commands:

▪ **vboxmanage list avd**

Lists all available virtual devices. The output from this command resembles the following:

```
Nexus 7 - 4.3 - API 18 - 1280x800" {144161dd-750e-4fff-8d46-0da8bc0c226b}
GalaxyNexus4.2.2-API17" {d740a4fa-df15-4768-aeel-ffebfb883dc1}
```

Each line is a pair of identifiers. The first identifier is the name you gave the device when you created it. The second identifier is the UID. You can use either identifier when passing devices as targets on the command line. The UID does not need the braces when you are using it on the command line.

- **vboxmanage list runningvms**

Lists the virtual devices that are running.

You can perform the same function with the ADK command, **\$ANDROID_HOME/platform-tools/adb devices**. The ADK command lists both Genymotion and ADK emulated devices.

The output from this command resembles the following:

```
emulator-5554    device
192.168.0.56:5555    device
```

ADK devices are prefixed with "emulator." Genymotion devices begin with the IP address. The next number is the port that the device is using.

- **player --vm-name <UID or name>**

Navigate to the path where the device is installed, and use this command to start a virtual device. You can also start a device from the Genymotion interface.

Define VBOXMANAGE_CMD

If you install VirtualBox in a non-default location, define the VBOXMANAGE_CMD property. This property lets DevTest find the vboxmanage tool, so your Genymotion simulators function efficiently.

You can define this property in three ways:

- Define an environment variable in Windows
- Edit the [lisa.properties file \(see page 1638\)](#) (to affect all projects)
- Add a property to the project property file in the Properties Editor (to affect a single project)

Define an Environment Variable in Windows

In the System Properties for your machine, open the Environment Variables dialog and add the following command:

- **Windows**

```
VBOXMANAGE_CMD=c:\program files\oracle\virtualbox\vboxmanage.exe
export VBOXMANAGE_CMD=/usr/bin/vboxmanage
```

- **Mac**

```
export VBOXMANAGE_CMD=/usr/bin/vboxmanage
```

Edit the LISA Property File

Editing the [lisa.properties file \(see page 1638\)](#) affects all of the projects that are defined in DevTest Workstation.

Follow these steps:

1. Select **System, Edit LISA Properties** from the main menu.
2. To define the VBOXMANAGE_CMD in the system file, add the following text:

```
VBOXMANAGE_CMD=c:\program files\oracle\virtualbox\vboxmanage.exe
```

3. Save the file.

Add a Property to the Project Property File

Adding a property to the property file in the **Properties Editor** affects only that project.

Follow these steps:

1. In the **Project** panel, double-click your **project** in the **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:
`VBOXMANAGE_CMD`
4. In the **Value** field, type:
`c:\program files\oracle\virtualbox\vboxmanage.exe`



5. From the main toolbar, click  **Save**.

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 5.1.7.

Follow these steps:

1. Consult the install4j documentation for detailed information about unattended installations.
2. Consult the DevTest [installation instructions \(see page 105\)](#) 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 following example, and changing the values as necessary.
4. To initiate a silent installation, execute the following command (the example assumes a Windows system):

```
devtest_win_x86.exe -q -varfile response.varfile
```

Sample response file

```
#install4j response file for DevTest Solutions 8.0.1 (build 8.0.1.371)
#Tue Jan 06 07:48:21 CST 2015
licenseXmlFile=C:\\\\Users\\\\rhoan02\\\\Downloads\\\\devtestlic.xml
sys.programGroupDisabled$Boolean=false
sys.installationDir=C:\\\\Program Files (x86)\\\\CA\\\\DevTest801371
installDemoServer=No
sys.languageId=en
installationType$Integer=0
steAssociation$Boolean=true
stgAssociation$Boolean=true
sys.programGroupName=DevTest Solutions
vsmAssociation$Boolean=true
installWorkstation=Yes
autostartServices$Boolean=false
installServices$Boolean=true
caAgreementChoice=2
installServer=Yes
sys.programGroupAllUsers$Boolean=true
installEnterpriseDashboard=Yes
tstAssociation$Boolean=true
createDesktopLinkAction$Boolean=true
vsiAssociation$Boolean=true
sys.adminRights$Boolean=true
```

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.



Note: Docker support was added in DevTest Solutions 8.3. You must have installed the Linux version of 8.3 or later.

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 above and that Docker and the Linux version of DevTest 8.3 or later are both successfully installed.
2. Change to the **docker** subfolder of the DevTest installation.
3. 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.

4. 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 DevTestPortal container.

- **devtest/demoserver**

Approximate size: 508.6 MB

This image can be used to run a standalone DevTestDemo 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, broker containers, or both. 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 **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
```

Migrating DevTest Solutions

This section describes how to migrate to DevTest Solutions from an earlier version.

Contents

Key Points to Understand for DevTest 8.X

- **Product Name - Rebranding**

Beginning with version 8.0, the product names have changed as follows:

Current Product Name	Formerly Known As
CA LISA	DevTest Solutions
CA Service Virtualization	CA LISA Service Virtualization (Virtual Services Environment)

CA Application Test	CA LISA Test
CA Continuous Application Insight	CA LISA Pathfinder

The LISA prefix was removed from the product names, as well as from some prominent file and component names. For example, LISA Workstation is now DevTest Workstation and LISAWorkstation.exe is now Workstation.exe. However, not all references to LISA have been removed from the product.

- **Licensing**

The licensing scheme changed in DevTest Solutions 8.0. Previous licenses do not function with DevTest Solutions 8.X. All 8.X licenses are file based. Internet and Local License Server licenses are no longer supported. For more information, see [Licensing \(see page 164\)](#).

- **Enterprise Dashboard**

- Enterprise Dashboard manages the license in DevTest Solutions 8.X, and is now required for any 8.X installation. The installation wizard for DevTest 8.X now installs the Enterprise Dashboard, rather than requiring a separate installation.
- If you have an older version of the Enterprise Dashboard (7.X), you must install the 8.X Enterprise Dashboard.
- Every release of Enterprise Dashboard starting with 8.x is backward compatible with the 8.x registries. You can upgrade Enterprise Dashboard to 8.4 and run with an 8.1 registry.
- The license key for DevTest 8.X is stored in a file named **devtestlic.xml**. The installation wizard prompts you for the license location and then copies the license file to the LISA_HOME directory on the server where your Enterprise Dashboard is installed. The license file is not required on any other server or workstation. For more information, see [Enterprise Dashboard \(see page 169\)](#)

- **DevTest Portal**

DevTest Solutions includes the DevTest Portal, a web-based application that is intended to become the primary user interface for DevTest Solutions. It provides capabilities for some of the most-used workflows for DevTest products. Over time, CA will enhance the functionality of the DevTest Portal and eventually sunset the other interfaces in the DevTest product line, including DevTest Workstation. For more information, see [DevTest Portal \(see page 196\)](#).

- **Access Control (ACL)**

ACL (role-based access control) is enabled in DevTest Solutions 8.X, and is therefore required. If you have not previously used ACL, you are required to do so in DevTest 8.X. For more information, see [Access Control \(ACL\) \(see page 1417\)](#).

- **Databases**

The reuse of an existing database schema or the import of data from a previous release is not supported in DevTest Solutions 8.X. You must configure DevTest to use a clean enterprise database schema.

If you require access to previous data or reports, you must maintain an instance of the appropriate registry and the associated database. Please contact your account manager to obtain a different license for the legacy information.

Supported Upgrade Paths

If you plan to migrate from a release prior to CA LISA 6.x, the upgrade to DevTest Solutions 8.0 is a two-step process. You must first upgrade to CA LISA 6 before upgrading to DevTest Solutions 8.0. For more information about migrating from versions prior to 6.x, see the [Migration Guide for CA LISA 7.0](https://support.ca.com/cadocs/7/CA%20LISA%207%200%202-ENU/Bookshelf_Files/HTML/Migration/index.htm) (https://support.ca.com/cadocs/7/CA%20LISA%207%200%202-ENU/Bookshelf_Files/HTML/Migration/index.htm)



Important! CA LISA 6 is no longer available through the normal CA download mechanisms. Contact your CA Account Manager to discuss your plans to migrate to DevTest Solutions 8.0 and for assistance obtaining a CA LISA 6 download.

Migration System Requirements

For a detailed list of requirements, see [System Requirements \(see page 50\)](#).



Important! You must fully understand the DevTest Solutions installation process before you proceed with a migration.

Upgrading to DevTest Solutions 8.X

The *Migrating DevTest Solutions* section is targeted toward an audience who has studied the DevTest 8.X product documentation *and* obtained DevTest 8.X training from CA.

Mastery of previous versions expedites the DevTest 8.X learning curve. However, only proper DevTest 8.X education can prepare you to enjoy a smooth transition and realize the maximum value.

We highly recommend engaging professional services to assist with the DevTest 8.X upgrade. Contact your account executive for more details.

The Upgrade Process

Both major and minor version upgrades follow the same best practices.

To upgrade to DevTest 8.X, complete the following steps:

1. Verify the [system requirements \(see page 50\)](#) for DevTest 8.X.
2. Review the release notes for all major releases from your current version up to and including DevTest 8.X.

3. Backup Data

Even though you install DevTest 8.X to a new location, it is prudent to back up the following to a remote location:

- The entire existing LISA folder.

This covers all of the following, which are important files in your installation:

- Property Files (local, site, lisa)
- Virtual Services (if stored locally)
- Test Cases and Suites (if stored locally)
- HotDeploy Folder
- vseDeploy Folder

If you have stored your existing tests and virtual services in a source control repository, CA recommends making a copy of those assets. Once the copy is created, you can use the new copy with DevTest 8.X.

4. Download the DevTest Solutions 8.X installers.

For step-by-step instructions, see [Download DevTest Solutions Installers \(see page 80\)](#).

5. Shut down LISA Windows services.

Shut down the following component services if your existing version runs as Windows services. This applies to components that are installed remotely and connect to the registry.

- LISA Registry Service
- LISA Coordinator Service
- LISA Simulator Service
- LISA VSE Service

6. Install and configure DevTest 8.X.

For step-by-step instructions, see [How to Install and Set up DevTest Solutions \(see page 84\)](#).



Important! Install DevTest 8.X into a new directory: for example, **C:\DevTest-8.0**. Once your new installation is validated, you can then delete your previous installation. Do not delete the earlier installation if you want to preserve your legacy reports or CA Continuous Application Insight paths, which are not upgradable.

7. Review the remainder of *Migrating DevTest Solutions* for information about upgrade tasks specific to DevTest 8.X and the version that you are upgrading from.

8. Perform the appropriate upgrade tasks for your environment.

Clearing Browser Cache between Upgrades



Important! When migrating from one release to another, always clear your browser cache before starting your new instance of DevTest for the first time. This requirement applies to every user that uses web applications on the registry.

Failure to clear the browser cache can create an exception in some cases, depending on your individual browser settings. If you used a previous version of LISA (hitting - web portals) before upgrading, it is possible for the older versions of the client-side pages for the portal to be cached. If the older, cached versions of the client-side pages are used when a request is made, the server is unable to use the older version.

This scenario creates an exception that can manifest itself in the Registry.log. For example:

```
2013-06-21 15:09:30,947Z (09:09) [qtp969344010-65] WARN serverconsole - Exception while dispatching incoming RPC call
```

```
com.google.gwt.user.client.rpc.SerializationException: Type 'com.itko.lisa.serverconsole.shared.model.network.PerformanceStatisticModel' was not assignable to 'com.google.gwt.user.client.rpc.IsSerializable' and did not have a custom field serializer. For security purposes, this type will not be serialized.: instance =
```

If you receive this type of Registry.log message after an upgrade, clear your browser cache as your first attempt to eliminate this problem.

Licensing

Contact your account executive to coordinate your new license. You must be current with your maintenance to be eligible for a new release.

- Network-based licensing and Local License Server (LLS) are no longer applicable in DevTest Solutions 8.0. A new file-based DevTest license file (**devtestlic.xml**) is required.
- If requested, CA can provide a new license for your legacy registry so that you can continue to access your legacy reports and CA Continuous Application Insight paths. LISA 4, 5, 6, or 7 are left active on the license.
- Upgrades are often evaluated in parallel with the current version, which can require temporary duplicate licenses until the new version is generally available.

User Preferences

User preferences are not upgraded.

Configuration

In DevTest Solutions 8.0, the `_local.properties` template was reformatted and contains new properties. When migrating from an earlier version of LISA to DevTest Solutions 8.0, do not copy the `local.properties` or `site.properties` files from your legacy installation.

local.properties

After installing DevTest Solutions, start with the new _local.properties template rather than copying the existing local.properties from your legacy installation.

1. Create a copy of the **_local.properties** file in the new DevTest 8.0 installation directory.
2. Rename the copy to **local.properties** (remove the leading underscore '_').
3. Copy *only* the properties that you set in your legacy **local.properties** file into the appropriate section of the new property file.

site.properties

After installing DevTest Solutions, start with the new _site.properties template rather than copying the existing site.properties from your legacy installation.

1. Create a copy of the **_site.properties** file in the new DevTest 8.0 installation directory.
2. Rename the copy to **site.properties** (remove the leading underscore '_').
3. Copy *only* the properties that you set in your legacy **site.properties** file into the appropriate section of the new property file.
4. When configuring to a DevTest supported external database, such as Oracle, DB2, MySQL, or SQL Server:
 - a. Uncomment the following properties for that external database in the appropriate section.
For example:
lisadb.pool.common.driverClass=oracle.jdbc.driver.OracleDriver
lisadb.pool.common.url=jdbc:oracle:thin:@HOST:1521:SID
 - b. Update the **lisadb.pool.common.url** property with the appropriate hostname, port, and SID values.
 - c. Update the following user and password properties with the correct values for accessing the database:
lisadb.pool.common.user
lisadb.pool.common.password
 - d. Comment out the following two properties for the Derby database:
#lisadb.pool.common.driverClass=org.apache.derby.jdbc.ClientDriver
#lisadb.pool.common.url=jdbc://localhost:1528/database/lisa.db;create=true
 - e. Set the following property to false:
lisadb.internal.enabled=false



Note: If you are using the Derby database, the property changes listed in the previous step are not required. Review [Database System Requirements \(see page 57\)](#). The Derby database is only adequate for small deployments that do not require load and performance testing. For all other scenarios, configure DevTest to use an external database.

Depending on the database you are using, you must also install the appropriate JDBC driver. For more information, see [External Registry Database Configuration \(see page 1386\)](#).

For more information about database setup and configuration, see in [Database Administration \(see page 1385\)](#).

Reporting Upgrades

CA does not support reporting database upgrades. CA can issue a registry-only license that provides access to legacy reports, if necessary. Contact your account executive for more information.

Database

Registry Database

Before you upgrade, be sure to shut down the registry and then back up the database.

Release 8.1 added the following requirement for Oracle databases: For the initial creation of the database, the Oracle user must have the **CREATE VIEW** system privilege.

DevTest makes any database changes included in the new version to the database the first time the registry is started.

If you are using the included Apache Derby database, installing in a new DevTest directory preserves your existing data in the Derby database in the old installation folder.

In an enterprise installation of DevTest, use one of the supported enterprise-class databases instead. Create a DevTest database schema and configure the new installation to use the new database schema. The easiest approach is to let DevTest modify the database schema during the first startup. The database user that is configured needs schema-owner rights or CRUD on most schema objects.

DevTest, when run in a distributed configuration, depends upon any 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 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.



Note: Access Control List (ACL) entries for user IDs, passwords, and permissions that were created in a previous release must be manually re-entered through the Server Console.

Enterprise Dashboard Database

The Enterprise Dashboard uses a different database than the registry.

If you want to use an external database for the Enterprise Dashboard after upgrading to DevTest Solutions 8.0, you must manually [configure the external database \(see page 1393\)](#).

MySQL JAR File

If you are upgrading from release 8.0.2 and are using MySQL as the database, the MySQL JAR file is removed during the upgrade procedure.

Be sure to add the MySQL JAR file after the upgrade.

CA Continuous Application Insight Upgrades

CAI supports upgrading to release 8.4 from the following releases: 7.5, 7.5.1, 7.5.2, 8.0, 8.0.1, 8.0.2, 8.1, 8.2, and 8.3.

Contents

- [rules.properties and rules.xml \(see page 167\)](#)
- [InsightAgent.jar \(see page 168\)](#)
- [Database Considerations \(see page 168\)](#)
- [Automate Automation \(see page 169\)](#)

rules.properties and rules.xml

In release 7.5, the **rules.properties** file was changed to an XML file named **rules.xml**.

The following list describes what happens to the **rules.properties** file during upgrades from releases earlier than 7.5:

- If the **rules.properties** file exists, the file is migrated to **rules.xml** only if it is not empty.
- If the **rules.properties** file exists, the file is renamed to **rules.properties.org**.
- For the broker, if the **rules.properties** file does not exist or is empty, a **rules.xml** file with minimal data is created.
- For the broker, a **rules.xml.sample** file is created. This file serves as a reference of all the internally maintained properties and their default values.
- The following directives have been deprecated: **virtualize protocol** and **virtualize transform**. These directives are not migrated.

In release 8.0, the location of the **rules.xml** file changed from the **LISA_HOME\bin** directory to the **LISA_HOME** directory.

InsightAgent.jar

In release 8.0, the **InsightAgent.jar** file was added as a replacement for the **LisaAgent2.jar** file.

The **LisaAgent2.jar** file is still included for backward compatibility.

Database Considerations

The database schema for CAI has changed in recent releases. The following list describes the main changes:

Release	Schema Changes
8.0	Added the LISA_FRAME_CATEGORY table. Added the LISA_MANUAL_CASE table.. Updated the LISA_TRANSACTION_FRAME table.
8.0.1	Updated the LISA_TRANSACTION_FRAME table.
8.0.2	Updated various tables to support internationalization.
8.1	Created four views. Updated the LISA_TRANSACTION_FRAME table.
8.3	Updated the LISA_TRANSACTION_FRAME table.

When you upgrade to this release, the database schema is automatically updated and the existing data is maintained. However, be sure to review the following information before you upgrade:

- If you are upgrading from release 7.5, 7.5.1, or 7.5.2, open the **rules.xml** file and try to locate the following properties. If the properties exist and the values are set to false, delete the properties or change the values to true. Otherwise, the database schema is not automatically updated.

```
<broker>
  <property key="lisa.broker.auto.migrate.schema" value="false" comment="Whether to automatically create or upgrade the schema"/>
  <property key="lisa.broker.schema.v2" value="false" comment="Use v2 of the PF schema"/>
</broker>
```

- If you have a large database, the upgrade can take a long time.
- You cannot use CAI until the upgrade finishes.
- If the broker is using the default embedded Derby database and the database is large, increase the memory of the broker. For more information, see [Administering \(see page 1362\)](#).



Note: The default embedded Derby database is adequate only for small deployments that do not require load and performance testing, and is not supported as an enterprise class database.

You can manually upgrade the database by using the SQL scripts that are embedded in the **LisaAgent.jar** file. These scripts are also available in the **LISA_HOME\database\upgrade** directory.

Follow these steps:

1. Obtain the SQL scripts by running the following command:

```
java -jar LisaAgent.jar -upgradeddl <oracle|sqlserver|mysql|derby|db2> <output-directory>
```

2. Connect to the database.

3. Run the following SQL statement to get the schema version:

```
select SCHEMA_VERSION from LISA_AGENT_VERSION
```

4. Run the SQL scripts for your database in order, starting with the version that follows the current version. For example, if the schema version is 1 and the database is Oracle, the first script that you run is **oracle_002_cai.sql**.

Automate Automation

You can leverage CA Continuous Application Insight to “automate automation.” This function can be valuable for rapidly regenerating regression test suites and virtual services. CAI is particularly adroit at the following tasks for web services and EJBs:

- Regenerating regression test suites
- Regenerating virtual services
- Regenerating data sets (XML-based request/response pair only)

The supported platforms are WLC 10.3, WAS 7.0, and JBoss 4.2.3+.

Contact your account executive for more information about how CAI can transform your upgrade experience.

Continuous Validation Service

Continuous Validation Service (CVS) Monitors are not upgraded and must be re-created.

Enterprise Dashboard

Enterprise dashboard is now required by a DevTest Solutions 8.0 install. When you run the installation wizard, you are prompted to either install the dashboard or enter the location of an existing dashboard. The dashboard is no longer a separate install and license.

New data gathering mechanisms were introduced to the Enterprise Dashboard for 7.5 registries. Due to these new mechanisms, you are no longer able to view detailed registry data for LISA registry versions older than version 7.5. To view the detailed data in the Registry Details Window, you must first upgrade your LISA registries to 7.5. Once the registry is upgraded, you must also update a new **lisa.enterprise.dashboard.url** property for each registry that you want to view in the dashboard.

The **lisa.enterprise.dashboard.url** property is required for a registry to send data to the Enterprise Dashboard. This property does not affect the ability to run either the Enterprise Dashboard or the registry. If a registry has the value set, data for that registry is sent to the URL that is specified by the property. If this property is not set for a registry, no data for that registry is sent.

For more information about configuring a registry, see [Configure Existing Registries \(see page 96\)](#).

Upgrading through Major Versions

CA has historically recommended upgrading through each major version of LISA. For example, upgrading from LISA 5 to LISA 7 would include opening assets in LISA 6 before opening them in LISA 7.

Because this process is potentially time consuming, we only recommend this process for legacy assets that do not migrate directly to LISA 7, unless otherwise specified in this section.



Note: Newer versions of LISA will have features that older versions do not. Therefore, if you create and/or update any LISA asset and then open it in an older version of LISA, you may experience unexpected results.

Upgrading DevTest Workstation

Contents

- [Project Panel Changes \(see page 170\)](#)
- [JDBC Virtualization Approach \(see page 171\)](#)
- [New and Deprecated JMS Components \(see page 171\)](#)
- [New and Deprecated WebSphere MQ Components \(see page 172\)](#)
- [Test Invocation \(see page 173\)](#)
- [The Negative Testing Companion \(see page 173\)](#)
- [Model Archive \(MAR\) Architecture and Staging Tests \(see page 173\)](#)
- [Web Services Testing \(see page 174\)](#)
- [Web 2.0 Testing \(see page 174\)](#)
- [Metric Name Changes in LISA 5 \(see page 175\)](#)
- [Test Timeouts \(see page 175\)](#)
- [Subprocess Migration \(see page 175\)](#)

Project Panel Changes

In release 7.5.1, the following changes were made to the **Project** panel in DevTest Workstation:

- The **AuditDocs**, **StagingDocs**, **Subprocesses**, and **Suites** folders are now subfolders of the **Tests** folder.
- The **VServices** folder is now named **VirtualServices**.

- The **VirtualServices** folder appears before the **Tests** folder.

If you open a project that was created in an earlier release, the project continues to use the old folder structure. If the project is from a release earlier than 5.0, the project is converted to the folder structure from releases 5.0 through 7.5.0.

The **LISA_HOME\examples** project uses the old folder structure.

JDBC Virtualization Approach

To perform JDBC virtualization in DevTest 8.0, use the DevTest Java Agent.

DevTest 8.0 still includes driver-based JDBC virtualization. However, we plan to remove this approach in a future release.

New and Deprecated JMS Components

LISA 7.5 introduced a new approach for doing messaging-related tasks. This approach builds on the asset framework that was introduced in the same release.

As part of the new approach, the following test steps were added:

- **JMS Send Receive**
- **JMS VSE Listen**
- **JMS VSE Live Invocation**
- **JMS VSE Respond**

One goal of these changes is to make configuration easier.

For example, the **JMS Send Receive** step includes a **Send Destination** field and a **Receive Destination** field. You use these fields to specify the request queue and the response queue. All the logic that is necessary to connect to the queues is encapsulated in the selection.

You can switch between different JMS providers by changing assets, rather than by having to configure a different type of step (**TIBCO Direct JMS**, **IBM WebSphere MQ**, and so on).

The **JMS Send Receive** step has built-in correlation schemes: **JMS Correlation ID**, **JMS Message ID to Correlation ID**, and **JMS Payload**. **JMS Payload** is a generic correlation scheme that can pull a correlation ID out of the message payload and can itself be extended to support custom payload and correlation formats.

Another goal is to make connection sharing easier. The new steps provide fine-grained control over how JMS connection assets are shared and reused. A particular JMS connection, session, queue, or other JMS asset can be reused for the duration of a test model's execution, reused by all instances of a multi-VU test, or reused globally by all tests running in the same simulator.

The following messaging steps are still available. However, we plan to remove these steps in a future release.

- **JMS Messaging (JNDI)**

- **WebLogic JMS (JNDI)**
- **JCAPS Messaging (Native)**
- **JCAPS Messaging (JNDI)**
- **Oracle OC4J (JNDI)**
- **TIBCO EMS Messaging**
- **TIBCO Direct JMS**
- **SonicMQ Messaging (Native)**
- **SonicMQ Messaging (JNDI)**
- **IBM WebSphere MQ (JMS mode only)**

In addition to the test steps, a new VSE transport protocol was added:

- **JMS**

This protocol comes with a new recorder and a new set of VSE steps, mentioned earlier. It supports any JMS platform that can be configured with assets, including both the JNDI and direct versions of TIBCO, SonicMQ, and so on. The new JMS VSE protocol supports a new type of recording for TIBCO platforms, TIBCO Monitor recording, which allows recording without the use of proxy queues. Release 8.0.2 added the ability to generate a virtual service model that operates on live queues instead of proxy queues.

The old **Standard JMS** transport protocol is still included. However, we plan to remove this protocol in a future release.

The new **JMS** transport protocol is backwards compatible with service images that were recorded with the old **Standard JMS** transport protocol.

New and Deprecated WebSphere MQ Components

DevTest 8.1 added WebSphere MQ versions of the JMS steps and protocols that are described in the preceding section.

The following test steps were added:

- **IBM MQ Native Send Receive**
- **IBM MQ Native VSE Listen**
- **IBM MQ Native VSE Respond**
- **IBM MQ Native VSE Live Invocation**

The **IBM WebSphere MQ** step is still available. However, we plan to remove this step in a future release.

In addition to the test steps, the **IBM MQ Native** transport protocol was added.

The old **IBM MQ Series** transport protocol is still included. However, we plan to remove this protocol in a future release.

Test Invocation

junitlisa Ant Task

The Ant task for invoking LISA tests during a build was updated in LISA 7.0. Ensure that you update the ant **build.xml** file to match the updated one shipped with the examples in LISA 7. Specifically, ensure that you add the new reference to **lisa-modules.jar** and **_misc-modules.jar**:

```
<!-- pull in our custom ant tasks -->
<taskdef resource="AntTasks.properties">
  <classpath>
    <fileset dir="${LISA_HOME}/bin">
      <include name="lisa-modules.jar"/>      <include name="lisa-core.jar"/>
      <include name="lisa-annotations.jar"/>
    </fileset>
    <fileset dir="${LISA_HOME}/lib">
      <include name="_misc-modules.jar"/>          <include name=
      _misc-core.jar"/>
      <include name=_misc-annotations.jar"/>
      <include name="emma.jar"/> <!
      -- only needed if lisa classes are instrumented, ignored otherwise -->
    </fileset>
  </classpath>
</taskdef>
```

Stand-Alone / Execute on Simulators

See [Running Test Cases and Suites \(see page 533\)](#).

The Negative Testing Companion

Release 7.0 replaced the Negative Testing Companion functionality with Audit Documents. Remove Negative Test Companions from legacy tests and replace them with a reference to a negative testing Audit Document at runtime.

Use an Audit Document such as **LISA_HOME\examples\AuditDocs\main_all_should_fail.aud** in a suite with your test case. This audit document looks for the "cycle failed" event, and considers the test case to have passed if that event occurs. This behavior is basically the opposite of the default audit document behavior.

Model Archive (MAR) Architecture and Staging Tests

CA introduced Model Archives (MARs) in LISA 6 for staging tests and suites.

See [Working with Model Archives \(see page 524\)](#) for details about how tests are staged in DevTest.

External File References and MARs

DevTest properties identify the external files in MARs, which are not resolved until runtime. The default optimized MAR file that is created to stage the test does not include actual files like some of the legacy staging mechanisms. This absence of actual files can cause **FileNotFoundException** errors. This concept frequently applies to Steps and Data Sets, but can affect any DevTest component that references an external file.

The upgrade process is as follows:

1. Instead of staging directly, create a .mari file.
2. Open the .mari file.
3. Move the external files that are needed for staging into the "include" list.
4. Save the .mari.
5. Now you can right-click, stage the .mari file, and the external files are included.

Persisting Test Results to the File System and MARs

Test cases often write result files to {{LISA_PROJ_ROOT}}/Data. This behavior works in the ITR, but causes the results to be lost when staged. The results are lost because LISA stages projects in a temporary area as of 6.0.

To work around this problem, we recommend the following steps:

1. Identify an external location to write result files to, and reference it with a property. For example, {{RESULT_DIR}}, in the project configuration.
2. Modify the steps that write results out to write to this external location instead.

Write to Delimited File Step

The "Write to Delimited File" step has been enhanced as of 5.0 to support I18N and alternate line endings. Tests that require specific line break characters must explicitly set them using the drop-down list in the "Write to Delimited File" step.

Web Services Testing

Legacy Web Services steps are deprecated. These steps are supported in LISA 7.0, but they are scheduled for removal from DevTest Solutions 8.0. Consider using CA Continuous Application Insight to regenerate your Web Services tests.

Web 2.0 Testing

Legacy Web 2.0 tests are deprecated. These steps are supported in LISA 7.0, but they are scheduled for removal from DevTest Solutions 8.0. Consider using CA Continuous Application Insight to regenerate your Web Services tests.

Metric Name Changes in LISA 5

Some metric names were changed in LISA 5 and could affect staging documents and custom reports. We recommend re-creating staging documents in LISA 7.1 and engaging professional services to support upgrading custom components respectively.

Test Timeouts

A common solution to test timeouts is to set the property lisa.net.timeout.ms (`lisa.net.timeout.ms =60000`) in **local.properties**.

Subprocess Migration

A common solution to test timeouts is to set the property lisa.net.timeout.ms (`lisa.net.timeout.ms =60000`) in **local.properties**.

Upgrading VSE

Contents

- [Upgrading Models \(see page 175\)](#)
- [Upgrading Service Images \(see page 175\)](#)
- [New Service Image Editor \(see page 176\)](#)
- [Known Migration Issues \(see page 178\)](#)

Upgrading Models

Copy models to the `\VServices` directory of the appropriate DevTest project if they are not already in a DevTest project.

Some customers have created "more flexible" models by manually doing things to the XStream'd Request and Response representations that you get in that style. Changes to the VSE Request and Response classes can break models that include directly manipulating the XML form of these objects. See the *Using the SDK* for API history details. Contact your account executive if you require assistance evaluating your models.

Upgrading Service Images

Export LISA 4.x and 5.x service images from your current database and import them into DevTest.

Service images created in LISA 5.0 or later versions should run correctly in DevTest 7.5 and later.

Beginning with LISA 6.0, service images are no longer stored in a database. To use LISA 5.0 service images in LISA 6.0, export them using LISA 5.0. Then use the Import item on the project trees context menu in LISA 6.0 to import the service images.

See [Creating Service Images \(see page 759\)](#) for information about how to export legacy service images.

New Service Image Editor

Release 8.0 introduced a new web-based editor for service images. The new editor is included in the DevTest Portal.

This page summarizes the differences between the old editor and the new editor.

Feature Comparison of Service Image Editors

The following table lists which service image editor features are available in the DevTest Portal and DevTest Workstation.

Feature	DevTest Portal	DevTest Workstation
Basic and advanced views	Yes	No
Open service images from multiple projects	Yes	No
Stateless Transactions view	Yes	Yes
Conversation view	Yes	Yes
Zoom in and out of Conversation view	Yes	No
Import request/response pairs into existing service image	Yes	No
Reorder or move transactions	Yes (not currently available for conversational transactions)	Yes
Cut, copy, and paste transactions	Yes (copy and paste)	Yes
View request and response data at the same time	Yes	No
Default transaction or Meta	Default plus signature definition	Meta
Comprehensive search with autofill and highlighting of results	Yes	No
Add notes to signatures	Yes	No
Add labels to specific transactions	Yes	No
Find and replace	Yes (preview feature)	Yes
Find a match by request	Yes	No
Search for regular expressions in a response	Yes	No
Identification of magic strings and dates	Yes	Yes
Regenerate magic strings and date variables for all transactions	No	Yes
Change the navigation tolerance for a transaction	No	Yes
Apply a match script	No	Yes
Response views	Yes (binary and DOM views not currently available)	Yes

Meta Transactions

In the old editor, the term *Meta transaction* represented a combination of the following concepts:

- The matching algorithm for top-level transactions
- The response to send when a match is not found in the specific transactions

In the new editor, the term *Meta transaction* is not used.

The matching algorithm for top-level transactions is automatically set to **Signature**. The **Match style** field does not appear in the new editor.

The response to send when a match is not found in the specific transactions is called the *default response*.

Request Data Changes

The old editor contains a panel named **Request Data**. The **Request Data** panel has a set of tabs. The following graphic shows the **Arguments** tab:

Name	Name in Session	Comparison Operator	Value	Magic String	Date Pattern	Case Sensitive	Is Numeric
username			DemoFirst	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>
password			pass	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>

The columns in the **Arguments** tab of the old editor appear in different locations in the new editor.

The following columns are located in the **Request Data Arguments** panel:

- **Name**
- **Operator**
- **Value**
- **Magic String**

The following columns are located in the **Signature Definition** tab:

- **Name**
- **Date Pattern**
- **Case Sensitive**
- **Is Numeric**

The **Name in Session** column is not available in the new editor.

Save Button

The new editor does not include a **Save** button. When a field that you update loses focus, the changes are saved.

Known Migration Issues

CICS Virtual Services does not upgrade to 7.0 or later from previous versions.

Historically, customized and manually created models are less likely than product-generated ones to migrate with issues.

8.0-Specific Upgrade Information

Contents

- [Pre-Release 6 to 8.0 Upgrade \(see page 178\)](#)
- [6.x or Later to 8.0 Upgrade \(see page 178\)](#)

Pre-Release 6 to 8.0 Upgrade

If you are planning to migrate from a release prior to CA LISA 6.X, the upgrade to DevTest Solutions 8.0 is a two-step process. You must first upgrade to CA LISA 6 before upgrading to DevTest Solutions 8.0. For more information about migrating from versions prior to 6.X, see the [Migration Guide for CA LISA 7.0](https://support.ca.com/cadocs/7/CA%20LISA%207%200%202-ENU/Bookshelf_Files/HTML/Migration/index.htm) (https://support.ca.com/cadocs/7/CA%20LISA%207%200%202-ENU/Bookshelf_Files/HTML/Migration/index.htm).



Important! CA LISA 6 is no longer available through the normal CA download mechanisms. Contact your CA Account Manager to discuss your plans to migrate to DevTest Solutions 8.0 and for assistance obtaining a CA LISA 6 download.

6.x or Later to 8.0 Upgrade

This section describes the specific considerations for planning a migration from LISA 6.x or later to DevTest Solutions 8.0.

Licensing

The licensing scheme changed in DevTest Solutions 8.0. Previous licenses do not function with DevTest Solutions 8.0. All licenses in 8.0 are file based. Internet and Local License Server licenses are no longer supported. For more information, see [Installing \(see page 50\)](#).

Database

The reuse of an existing database schema or the import of data from a previous release is not supported. You must configure DevTest to use a clean enterprise database schema.

If you require access to previous data or reports, you must maintain an instance of the appropriate registry and the associated database.

Access Control (ACL)

ACL (role-based access control) is enabled in DevTest Solutions 8.0, and is therefore required. If you have not previously used ACL, you are required to do so in DevTest Solutions 8.0.

If you have previously used ACL, refer to [Access Control \(ACL\) \(see page 1417\)](#) for more information about role changes in 8.0 and re-enter your user data accordingly. There is no direct migration path from the previously defined roles to the new roles.

Assets

You can import all test and virtual service assets from CA LISA 6.x and later directly into DevTest Solutions 8.0.

To import assets, open a copy of the project in DevTest Solutions 8.0.



Note: Be sure to open a copy of the project in DevTest 8.0 so that the original project remains intact and can still be opened in an earlier release.

You cannot directly import assets from a release of CA LISA earlier than 6.x. For more information, see the [Migration Guide for CA LISA 7.0](#) (https://support.ca.com/cadocs/7/CA%20LISA%207%200%202-ENU/Bookshelf_Files/HTML/Migration/index.htm).

Customization

The migration of any extension to DevTest Solutions that is not provided as part of the product remains the responsibility of the customer. The migration of an installation containing extensions that were developed for previous releases is not supported.

Be sure to plan and consult with the appropriate persons with relevant domain knowledge regarding the customization to ensure that they are prepared for their migration.

The verification and correction of any scripting or scripted steps remain the responsibility of the customer, as does any other code that is not provided as a part of the standard DevTest Solutions distribution.

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 targeted toward an audience who has studied the DevTest 8.0 product documentation *and* obtained DevTest 8.0 training from CA. Mastery of previous LISA versions expedites the DevTest 8.0 learning curve. However, only proper DevTest 8.0 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 8.0 upgrade. Contact your account executive for more details.



Note: All SDK components must be compiled with Java 7 to run in DevTest 8.0.

This guide is organized by component. It provides the relevant history of each component, starting with LISA 4.0 or the LISA 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 version 8.0 and update your code as required.

SDK Documentation

The following SDK documentation is available:

- [Using the SDK \(see page 1212\)](#)
- JavaDocs for DevTest Solutions are available in the **doc** folder of your installation directory.

Java Version Notes

Once DevTest 8.0 is successfully installed and configured, complete the following steps to upgrade your SDK components:

- DevTest 8.0 is supported in Java 7. All SDK components must be compiled with Java 7. Most relevant changes to the core Java language are backwards compatible.
- 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

Contents

- [4.0 to 4.5 \(see page 182\)](#)
- [4.5 to 4.6 \(see page 182\)](#)
- [4.6 to 5.0 \(see page 182\)](#)
- [5.0 to 6.0 \(see page 182\)](#)
- [6.0 to 6.1 \(see page 182\)](#)
- [6.1 to 7.0 \(see page 182\)](#)
- [7.0 to 7.1 \(see page 182\)](#)
- [7.1 to 7.5 \(see page 182\)](#)
- [7.5 to 8.X \(see page 183\)](#)

SDK Assertion components extend **com.itko.lisa.test.Assertion**.

4.0 to 4.5

No changes that affect SDK components.

4.5 to 4.6

No changes that affect SDK components.

4.6 to 5.0

- **com.itko.lisa.test.Assertion** now also implements the **com.itko.lisa.model.IWriteXML** interface, which affects how data is persisted. Few SDK components need to override the framework level behavior. For more information, see the JavaDocs in the **doc** folder of your installation directory.
- Added a reference to the **com.itko.lisa.test.TestCase** class, the test case on which the assertion operates, and corresponding getters and setters **getTc()** and **setTc(TestCase tc)**.
- Added **getEvaluatedILongMsg()**, which provides a more user-friendly error message when the assertion fails.
- Added global Assertion support with corresponding methods:
 - **public void markAssertionAsGlobal(boolean isGlobal)**
 - **public boolean isGlobalAssertion()**
 - **public boolean isScopeLocal()**
 - **public boolean isScopeGlobal()**

5.0 to 6.0

No changes that affect SDK components.

6.0 to 6.1

com.itko.lisa.test.Assertion now also implements the **com.itko.lisa.test.NameGenerator** interface, which controls how default component names are generated. This change does not affect the legacy SDK components. For more information, see the JavaDocs in the **doc** folder of your installation directory.

6.1 to 7.0

No changes that affect SDK components.

7.0 to 7.1

No changes that affect SDK components.

7.1 to 7.5

No changes that affect SDK components.

7.5 to 8.X

No changes that affect SDK components.

Companion Updates

Contents

- [4.0 to 4.5 \(see page 183\)](#)
- [4.5 to 4.6 \(see page 183\)](#)
- [4.6 to 5.0 \(see page 183\)](#)
- [5.0 to 6.0 \(see page 183\)](#)
- [6.0 to 6.1 \(see page 184\)](#)
- [6.1 to 7.0 \(see page 184\)](#)
- [7.0 to 7.1 \(see page 184\)](#)
- [7.1 to 7.5 \(see page 184\)](#)
- [7.5 to 8.X \(see page 184\)](#)

SDK Companion components extend **com.itko.lisa.test.SimpleCompanion**.

4.0 to 4.5

No changes that affect SDK components.

4.5 to 4.6

No changes that affect SDK components.

4.6 to 5.0

SimpleCompanion now extends **com.itko.lisa.test.CompanionBase** instead of implementing the **com.itko.lisa.test.CompanionInterface** directly. This update does not directly affect SDK components, although the following methods were added to the underlying interfaces.

- **public Object getCustomParams()**
- **public void writeSubXML(PrintWriter ps)**
- **static public void writeSubXML(PrintWriter ps, ParameterList params)**

5.0 to 6.0

- Added **public void setParameters (ParameterList parameters)** method. This method assigns the parameters, which can contain custom parameters and parameters that are injected by LISA. For example, {@code type}, which indicates the classname of the companion.
- Added **protected ParameterList removeDuplicates (ParameterList plist)** method, which is a helper method to scrub duplicate Parameter values. This method is called by **setParameters()**.

6.0 to 6.1

No changes that affect SDK components.

6.1 to 7.0

No changes that affect SDK components.

7.0 to 7.1

No changes that affect SDK components.

7.1 to 7.5

No changes that affect SDK components.

7.5 to 8.X

No changes that affect SDK components.

Filter Updates

Contents

- [4.0 to 4.5 \(see page 184\)](#)
- [4.5 to 4.6 \(see page 184\)](#)
- [4.6 to 5.0 \(see page 184\)](#)
- [5.0 to 6.0 \(see page 185\)](#)
- [6.0 to 6.1 \(see page 185\)](#)
- [6.1 to 7.0 \(see page 185\)](#)
- [7.0 to 7.1 \(see page 185\)](#)
- [7.1 to 7.5 \(see page 185\)](#)
- [7.5 to 8.X \(see page 185\)](#)

SDK Filter components extend **com.itko.lisa.test.FilterBaseImpl**.

4.0 to 4.5

No changes that affect SDK components.

4.5 to 4.6

No changes that affect SDK components.

4.6 to 5.0

- The underlying **FilterInterface** now implements **IWriteXML** and added the following methods to support persisting data:

- **static public void writeXML(PrintWriter pw, ParameterList pl)**
- **public void writeXML(PrintWriter ps)**
- **static public void writeXMLHeader(PrintWriter ps, String type, String valueToFilterPropKey)**
- **protected void writeSubXML(PrintWriter ps)**
- **static public void writeXMLFooter(PrintWriter ps)**
- Replaced **public Collection<FilterNodeConnection> getFilterNodeConnections(ParameterList params)** with **public void, gatherFilterStepConnections(String stepName, ParameterList params, Collection<StepConnection> stepConnections)** method. This method returns a **Collection** of the ways that a filter can attempt to pass execution to a specific node. This change is unlikely to affect SDK components.
- Added support for global filters, which includes the following methods:
 - **public boolean isScopeLocal()**
 - **public boolean isScopeGlobal()**

5.0 to 6.0

No changes that affect SDK components.

6.0 to 6.1

Added implementation of the **com.itko.lisa.test.NameGenerator** interface, which controls how default component names are generated. This change does not affect the legacy SDK components. For more information, see the JavaDocs in the **doc** folder of your installation directory.

6.1 to 7.0

No changes that affect SDK components.

7.0 to 7.1

No changes that affect SDK components.

7.1 to 7.5

No changes that affect SDK components.

7.5 to 8.X

No changes that affect SDK components.

Step Updates

Contents

- [4.0 to 4.5 \(see page 186\)](#)
- [4.5 to 4.6 \(see page 186\)](#)
- [4.6 to 5.0 \(see page 186\)](#)
- [5.0 to 6.0 \(see page 186\)](#)
- [6.0 to 6.1 \(see page 186\)](#)
- [6.1 to 7.0 \(see page 186\)](#)
- [7.0 to 7.1 \(see page 186\)](#)
- [7.1 to 7.5 \(see page 186\)](#)
- [7.5 to 8.X \(see page 187\)](#)

Simple SDK Steps or Nodes implement the **com.itko.lisa.test.CustJavaNodeInterface**.

4.0 to 4.5

No changes that affect SDK components.

4.5 to 4.6

No changes that affect SDK components.

4.6 to 5.0

No changes that affect SDK components.

5.0 to 6.0

No changes that affect SDK components.

6.0 to 6.1

No changes that affect SDK components.

6.1 to 7.0

No changes that affect SDK components.

7.0 to 7.1

No changes that affect SDK components.

7.1 to 7.5

No changes that affect SDK components.

7.5 to 8.X

No changes that affect SDK components.

Data Protocol Updates

Contents

- [5.0 to 5.0.25 \(see page 187\)](#)
- [5.0 to 6.0 \(see page 187\)](#)
- [6.0 to 6.1 \(see page 187\)](#)
- [6.1 to 7.0 \(see page 187\)](#)
- [7.0 to 7.1 \(see page 187\)](#)
- [7.1 to 7.5 \(see page 188\)](#)
- [7.5 to 8.X \(see page 188\)](#)

Data Protocols became extensible in LISA 5.0 and extend **com.itko.lisa.vse.stateful.protocol.DataProtocol**.

5.0 to 5.0.25

LISA 5.0.25 introduced the **com.itko.lisa.vse.stateful.common.DataProtocolFilter** to support more complex data configuration requirements. Models that were created in LISA before 5.0.25 do not open in LISA 7 and must be re-created.

5.0 to 6.0

Added public void **updateUnknownStatelessResponse(Response response)**, which is used by subclasses to update the given default response for unknown stateless requests.

6.0 to 6.1

Added the **com.itko.lisa.vse.stateful.protocol.DataProtocolConfiguration** object with getters and setters: **public DataProtocolConfiguration getConfig()** and **protected void setConfig(DataProtocolConfiguration config)**. For more information, see the JavaDocs in the **doc** folder of your installation directory.

6.1 to 7.0

Added **com.itko.lisa.common.LisaComponentConfig**, which is extended by type-specific configuration objects that are used to persist unparsed and parsed. That is, data that has been processed by **TestExec.parseInState()** for all SDK components. **com.itko.lisa.vse.stateful.protocol.DataProtocolConfiguration** still exists and should only be used for **DataProtocols**.

7.0 to 7.1

No changes that affect SDK components.

7.1 to 7.5

No changes that affect SDK components.

7.5 to 8.X

No changes that affect SDK components.

Transport Protocol Updates

Contents

- [5.0 to 6.0 \(see page 188\)](#)
- [6.0 to 6.1 \(see page 188\)](#)
- [6.1 to 7.0 \(see page 188\)](#)
- [7.0 to 7.1 \(see page 188\)](#)
- [7.1 to 7.5 \(see page 189\)](#)
- [7.5 to 8.X \(see page 189\)](#)

Extensible Transport Protocols were introduced in LISA 5.0 and extend **com.itko.lisa.vse.stateful.protocol.TransportProtocol**.

5.0 to 6.0

Transport Protocols became extensible by the SDK.

6.0 to 6.1

- Added the **com.itko.lisa.vse.stateful.protocol.DataProtocolConfiguration** object, which can also be used with Transport Protocols, with getters and setters: **public DataProtocolConfiguration getConfig()** and **protected void setConfig(DataProtocolConfiguration config)**. For more information, see the JavaDocs in the **doc** folder of your installation directory.
- Added protected **TestExec getTestExec()** to provide consistent access to **TestExec**.
- Added public **RecorderSelector.SmartWizardStep getSmartWizardStep(RecordingWizard wizard, WizardPhase phase)**

6.1 to 7.0

Added **com.itko.lisa.common.LisaComponentConfig** which is extended by type-specific **TransportProtocol** configuration objects that are used to persist unparsed and parsed. That is, data that has been processed by **TestExec.parseInState()**.

7.0 to 7.1

No changes that affect SDK components.

7.1 to 7.5

No changes that affect SDK components.

7.5 to 8.X

No changes that affect SDK components.

Body Carrier Updates

Contents

- [6.0.7 \(see page 189\)](#)
- [6.0 to 6.1 \(see page 189\)](#)
- [6.1 to 7.0 \(see page 189\)](#)
- [7.0 to 7.1 \(see page 190\)](#)
- [7.1 to 7.5 \(see page 190\)](#)
- [7.5 to 8.X \(see page 190\)](#)

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
- com.itko.lisa.vse.stateful.model.TransientResponse.

6.0.7

- Deprecated **public String getBodyText()**, replaced by **getBodyAsString()**.
- Deprecated **public void setBodyText(String text)**, replaced by **setBody(String body)**.
- Deprecated **public byte[] getBodyBytes()**, replaced by **getBodyAsByteArray()**.
- Deprecated **public void setBodyBytes(byte[] body)**, replaced by **setBody(byte[] body)**.

6.0 to 6.1

No changes that affect SDK components.

6.1 to 7.0

Added the **com.itko.lisa.asset.vse.IBodyCarrier** interface.

Implemented a more informative **toString()** method.

7.0 to 7.1

No changes that affect SDK components.

7.1 to 7.5

No changes that affect SDK components.

7.5 to 8.X

No changes that affect SDK components.

Request Updates

Contents

- [5.0 to 6.0 \(see page 190\)](#)
- [6.0 to 6.1 \(see page 190\)](#)
- [6.1 to 7.0 \(see page 190\)](#)
- [7.0 to 7.1 \(see page 190\)](#)
- [7.1 to 7.5 \(see page 190\)](#)
- [7.5 to 8.X \(see page 191\)](#)

com.itko.lisa.vse.stateful.model.Request represents the request side of a VSEtransaction.

5.0 to 6.0

LISA 5.0 persisted Service Images to a database. LISA 6.0 and later persists Service Images to the file system. This change resulted in a large impact to the Request API, but not in areas that affect SDK components. For more information, see the JavaDocs in the **doc** folder of your installation directory.

6.0 to 6.1

See [Body Carrier \(see page 189\)](#) for details about setting and accessing the Request body.

6.1 to 7.0

No changes that affect SDK components.

7.0 to 7.1

No changes that affect SDK components.

7.1 to 7.5

No changes that affect SDK components.

7.5 to 8.X

No changes that affect SDK components.

Response Updates

Contents

- [5.0 to 6.0 \(see page 191\)](#)
- [6.0 to 6.1 \(see page 191\)](#)
- [6.1 to 7.0 \(see page 191\)](#)
- [7.0 to 7.1 \(see page 191\)](#)
- [7.1 to 7.5 \(see page 191\)](#)
- [7.5 to 8.X \(see page 191\)](#)

com.itko.lisa.vse.stateful.model.Response represents the response side of a VSE transaction during recording.

5.0 to 6.0

LISA 5.0 persisted Service Images to a database. LISA 6.0 and later persists Service Images to the file system. This change resulted in a large impact to the Response API, but not in areas that affect SDK components. For more information, see the JavaDocs in the **doc** folder of your installation directory.

6.0 to 6.1

See [Body Carrier \(see page 189\)](#) for details about setting and accessing the Response body.

6.1 to 7.0

No changes that affect SDK components.

7.0 to 7.1

No changes that affect SDK components.

7.1 to 7.5

No changes that affect SDK components.

7.5 to 8.X

No changes that affect SDK components.

Transient Response Updates

Contents

- [5.0 to 6.0 \(see page 192\)](#)
- [6.0 to 6.1 \(see page 192\)](#)
- [6.1 to 7.0 \(see page 192\)](#)
- [7.0 to 7.1 \(see page 192\)](#)
- [7.1 to 7.5 \(see page 192\)](#)
- [7.5 to 8.X \(see page 192\)](#)

com.itko.lisa.vse.stateful.model.TransientResponse represents the response side of a VSEtransaction during playback.

5.0 to 6.0

- LISA 5.0 persisted Service Images to a database. LISA 6.0 and later persists Service Images to the file system. This change resulted in a large impact to the Response API, but not in areas that affect SDK components. For more information, see the JavaDocs in the **doc** folder of your installation directory.
- **public TransientResponse(Response r)** was replaced by **com.itko.lisa.vse.stateful.model.Response.createTransientCopy()**.

6.0 to 6.1

See [Body Carrier \(see page 189\)](#) for details about setting and accessing the TransientResponse body.

6.1 to 7.0

See [Body Carrier \(see page 189\)](#) for details about setting and accessing the TransientResponse body.

7.0 to 7.1

No changes that affect SDK components.

7.1 to 7.5

No changes that affect SDK components.

7.5 to 8.X

No changes that affect SDK components.

TCP Delimiter Updates

Contents

- [6.0 to 6.1 \(see page 193\)](#)
- [6.1 to 7.0 \(see page 193\)](#)
- [7.0 to 7.1 \(see page 193\)](#)
- [7.1 to 7.5 \(see page 193\)](#)
- [7.5 to 8.X \(see page 193\)](#)

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.

6.0 to 6.1

No changes that affect SDK components.

6.1 to 7.0

No changes that affect SDK components.

7.0 to 7.1

No changes that affect SDK components.

7.1 to 7.5

No changes that affect SDK components.

7.5 to 8.X

No changes that affect SDK components.

API Notes

Contents

- [5.0 to 6.0 \(see page 193\)](#)
- [6.1 to 7.0 \(see page 193\)](#)
- [7.0 to 7.1 \(see page 194\)](#)
- [7.1 to 7.5 \(see page 194\)](#)
- [7.5.1 \(see page 194\)](#)
- [7.5.1 to 8.X \(see page 194\)](#)

5.0 to 6.0

Changed the signature in **com.itko.lisa.vse.stateful.DelayedResponder** from **invokeLater(uniqueId, action, delayTime)** to **invokeLater(testExec, action, delayTime)**.

6.1 to 7.0

- Removed the following methods from **com.itko.util.StrUtil**. **ModuleCore.resource.get()**, **ModuleVSE.resource.get()**, **Module*.resource.get()**.. methods. Replace these methods.
 - **public static String get(String key, Object[] args)**
 - **public static String get(String key)**
- Removed the following methods from **com.itko.util.Strings**. **ModuleCore.resource.get()**, **ModuleVSE.resource.get()**, **Module*.resource.get()**.. methods. Replace these methods.
 - **public static String get(String key)**

7.0 to 7.1

No changes that affect SDK components.

7.1 to 7.5

No changes that affect SDK components.

7.5.1

Added a new **getListenersCount()** method that produces a count of listeners.

Originally, this function was performed by the **getListeners()** API method, but this method was deprecated in 6.0. If you are using the **getListeners()** method, update your tests to use the new **getListenersCount()** method.

7.5.1 to 8.X

No changes that affect SDK components.

Getting Started

This section contains an overview of the DevTest Portal, tutorials for each functional area of the product suite, and a glossary.

- [DevTest Portal \(see page 196\)](#)
- [CA Application Test Tutorials \(see page 204\)](#)
- [Mobile Testing Tutorials \(see page 287\)](#)
- [CA Service Virtualization Tutorial \(see page 296\)](#)
- [CA Continuous Application Insight Tutorial \(see page 306\)](#)
- [Glossary \(see page 309\)](#)

DevTest Solutions is a solution that is made up 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.

CA Continuous Application Insight

- End-to-End Transaction Discovery: Enables thorough analysis by breaking down architectural and structural complexity of applications.

- Automating your Automation: Eliminates manual work by automating virtual service and test-case.
- Point-and-Click Defect Tickets: Capture detailed business transactions for the identified session and build virtual environments for easy defect reproduction.

DevTest Portal

The DevTest Portal is a web-based application that is intended to become the primary user interface for DevTest Solutions. The portal provides capabilities for some of the most-used workflows for DevTest products. Over time, CA will enhance the functionality of the DevTest Portal and eventually sunset the other interfaces in the LISA product line, including DevTest Workstation.

This page describes the available DevTest Portal functionality. For a quick summary, see [DevTest Portal Functionality \(see page 48\)](#).

This page also describes the **All Resources** window and how to work with projects.

Contents

Open the DevTest Portal

You open the DevTest Portal from a web browser.



Note: For information about the server components that must be running, see [Start the DevTest Processes or Services \(see page 115\)](#).

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.

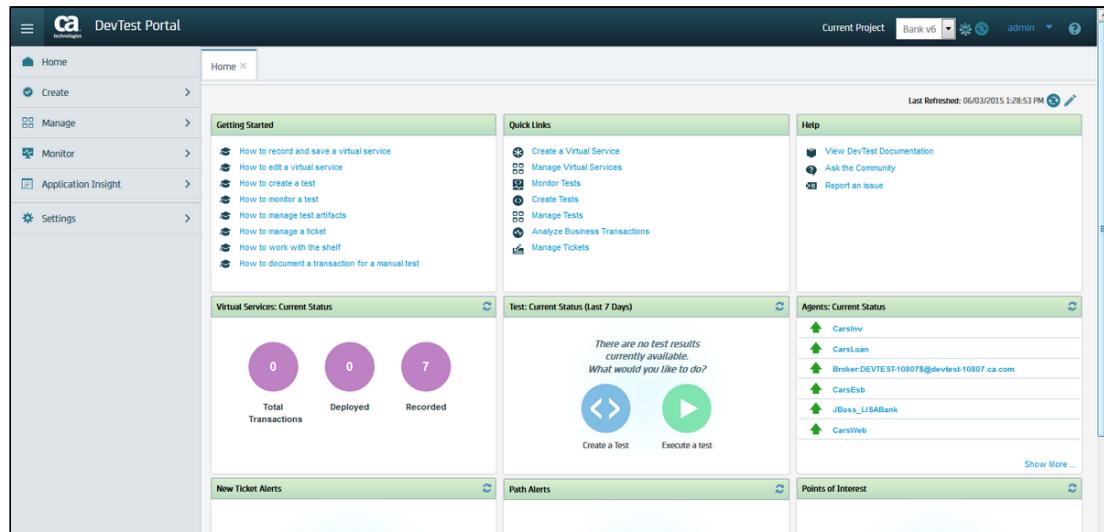
Follow these steps:

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 the DevTest Portal

When you log in, the 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.

Some menu items redirect you to the DevTest Console.

The navigation menu contains the following sections:

- **Home**

Display the home page.

- **Create**

Create API tests and virtual services.



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.

- **API Test**

Opens: API Test window

- **Virtualize Website (HTTP/S)**
Opens: Virtualize Website (HTTP/S) window
- **Virtualize JDBC**
Opens: Virtualize JDBC window
- **Virtualize using RR pairs**
Opens: Virtualize using RR Pairs window
- **Copybook Bundle**
Opens: Create New Bundle window
- **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**
Opens: All Resources window
- **API Tests**
Opens: Window that lets you edit and run API tests.
- **Tests**
Opens: Window that displays test case information and lets you run the test case
- **Test Suites**
Opens: Window that displays test suite information and lets you run the test suite
- **Virtual Services**
Opens: Window that lets you view and edit a virtual service
- **Copybook Bundles**
Opens: Window that lets you view and edit a copybook bundle
- **Monitor**
Monitor tests and virtual services.
 - **Monitoring Tests**
Opens: Monitoring Tests window
 - **Virtual Services**
Redirects: DevTest Console, Server Console, DevTest Network. Click **VSE** in the network graph.
 - **Virtual Service Environments**
Opens: VSE window
- **Application Insight**
Analyze and document transactions with CAI. You can also manage tickets.

- **Analyze Transactions**
Opens: Analyze Transactions window
- **Document Transactions**
Opens: Document Transactions window
- **Manage Tickets**
Opens: Manage Tickets window
- **Settings**
Configure agents and access control.
 - **Agents**
Opens: Agents window
 - **Mainframe Agents**
Redirects: DevTest Console, Server Console, DevTest Network
 - **Access Control**
Redirects: DevTest Console, Server Console. Expand the Administration tab to display Security options.
 - **Reporting**
Redirects: DevTest Console, Reporting

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 the 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 CAI database.

- **Path Alerts**

Displays a list of paths with exception flags. Selecting a path navigates you to the **Analyze Transactions** window.

- **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 **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

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 panel. Each resource type opens in a new tab.

Manage Projects Window

You can manage projects from the DevTest Portal.

To add a project:

1. Click **Manage Projects**  .
The **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 **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 **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 **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 into a project:

1. Click Manage Projects  .
The **Projects** window appears.
2. Click **Upload MAR**  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.

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

From the DevTest Portal home page, click the down arrow beside your user name to access User Preferences.

The DevTest Portal provides tips for new users on how to perform key tasks. You can enable or disable the help tips.

Follow these steps:

1. Open the DevTest Portal.
2. Select **Preferences**.
The Preferences dialog opens.
3. To enable the help tips, select the **Enable help for first-time user** check box and click **OK**.
4. 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.

The DevTest Portal lets you select the default language of the portal and the default project.



When using a browser with an automatic translation capability, you must set the language in the Preferences dialog.

Follow these steps:

1. Open the DevTest Portal.
2. Select **Preferences**.
The Preferences dialog opens.
3. To choose a language, select a language from the **Language** drop-down list and click **OK**.

The DevTest Portal lets you set a default project.

Follow these steps:

1. Open the DevTest Portal.
2. Select **Preferences**.
The Preferences dialog opens.
3. Select a project from the **Default Project** drop-down list and click **OK**.
The default is to use the last saved project as the project default.

Enable and Disable Preview Features

From the Sandbox, you can view all the features that are in a preview phase for this release. You enable or disable the preview features from the **Features** portion of **The Sandbox** window. Use the **Enabled** switch to change between disabled and enabled preview mode. The **Enabled** switch is gray when the feature is disabled and blue when enabled.

Follow these steps:

1. From the DevTest Portal, click the down arrow to the right of your user name and select **The Sandbox**.
2. To enable a feature, click the right side of the gray **Enabled** switch.
The **Enabled** switch turns blue and the feature is enabled.
3. To disable a feature, click the left side of the blue **Enabled** switch.
The **Enabled** switch turns gray and the feature is disabled.

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 \(see page 50\)](#).

This section contains the following pages:

- [Tutorial 1 - Projects, Test Cases, and Properties \(see page 205\)](#)
- [Tutorial 2 - Data Sets \(see page 210\)](#)
- [Tutorial 3 - Filters and Assertions \(see page 215\)](#)
- [Tutorial 4 - Manipulate Java Objects \(POJOs\) \(see page 223\)](#)
- [Tutorial 5 - Run a Demo Server Web Application \(see page 232\)](#)
- [Tutorial 6 - Test a Website \(see page 238\)](#)
- [Tutorial 7 - Test an Enterprise JavaBean \(EJB\) \(see page 253\)](#)
- [Tutorial 8 - Test a Web Service \(see page 264\)](#)
- [Tutorial 9 - Examine and Test a Database \(see page 270\)](#)
- [Tutorial 10 - Stage a Quick Test \(see page 282\)](#)

Tutorial 1 - Projects, Test Cases, and Properties

Contents

- [Step 1 - Start DevTest Workstation \(see page 206\)](#)
- [Step 2 - Create a Project \(see page 206\)](#)
- [Step 3 - Create a Test Case \(see page 206\)](#)
- [Step 4 - Add a Property to the Project Configuration \(see page 207\)](#)
- [Step 5 - Add a Test Step \(see page 207\)](#)
- [Step 6 - Add a Log Message \(see page 208\)](#)
- [Step 7 - Add a Second Log Message \(see page 208\)](#)
- [Step 8 - Run the My Output Log Message Step \(see page 209\)](#)
- [Step 9 - Observe Property Values \(see page 209\)](#)
- [Step 10 - Run the Output Log Message Step \(see page 209\)](#)
- [Review \(see page 210\)](#)

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 \(see page 309\)](#).

Step 1 - Start DevTest Workstation

Follow these steps:

1. Start the registry.
 - If your computer has DevTest Server installed:
 - a. Start the registry by clicking **Start Menu, All Programs, DevTest Solutions, EnterpriseDashboard**. Wait until the "Enterprise Dashboard started" message appears.
 - b. Start the registry by clicking **Start Menu, All Programs, DevTest Solutions, Registry**.
 - If your computer has DevTest Workstation installed, use a registry that is running on another computer.
2. Click **Start, All Programs, DevTest Solutions, Workstation**.
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.

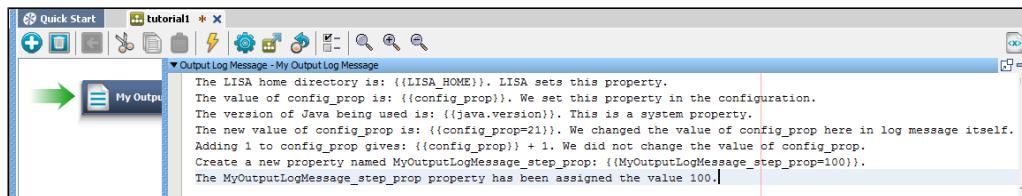
The syntax for a property is **{{property_name}}**.

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 configuration.
The version of Java being used is: {{java.version}}. This is a system property.
The new value of config_prop is: {{config_prop=21}}. We changed the value of config_prop here in log message itself.
Adding 1 to config_prop gives: {{config_prop}} + 1. We did not change the value of config_prop.
Create a new property named MyOutputLogMessage_step_prop: {{MyOutputLogMessage_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.

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 \(see page 211\)](#)
- [Step 2 - Create a Test Case \(see page 211\)](#)
- [Step 3 - Add a Property to the Project Configuration \(see page 212\)](#)
- [Step 4 - Add a Test Step for Output Log Message \(see page 212\)](#)
- [Step 5 - Create Another Output Log Message Step \(see page 213\)](#)
- [Step 6 - Execute the Test \(see page 213\)](#)
- [Step 7 - Add the Data Set \(see page 214\)](#)
- [Step 8 - Change the Data Set Behavior \(see page 214\)](#)
- [Tutorial 2 - Review \(see page 215\)](#)

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 \(see page 216\)](#)
- [Step 2 - Change Action of Test Step \(see page 216\)](#)
- [Step 3 - Add an Assertion \(see page 217\)](#)
- [Step 4 - Test the Assertion \(see page 219\)](#)
- [Step 5 - Add a Filter \(see page 221\)](#)
- [Step 6 - Test the Filter \(see page 221\)](#)
- [Tutorial 3 - Review \(see page 223\)](#)

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 \(see page 405\)](#) and [Assertions \(see page 419\)](#).

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

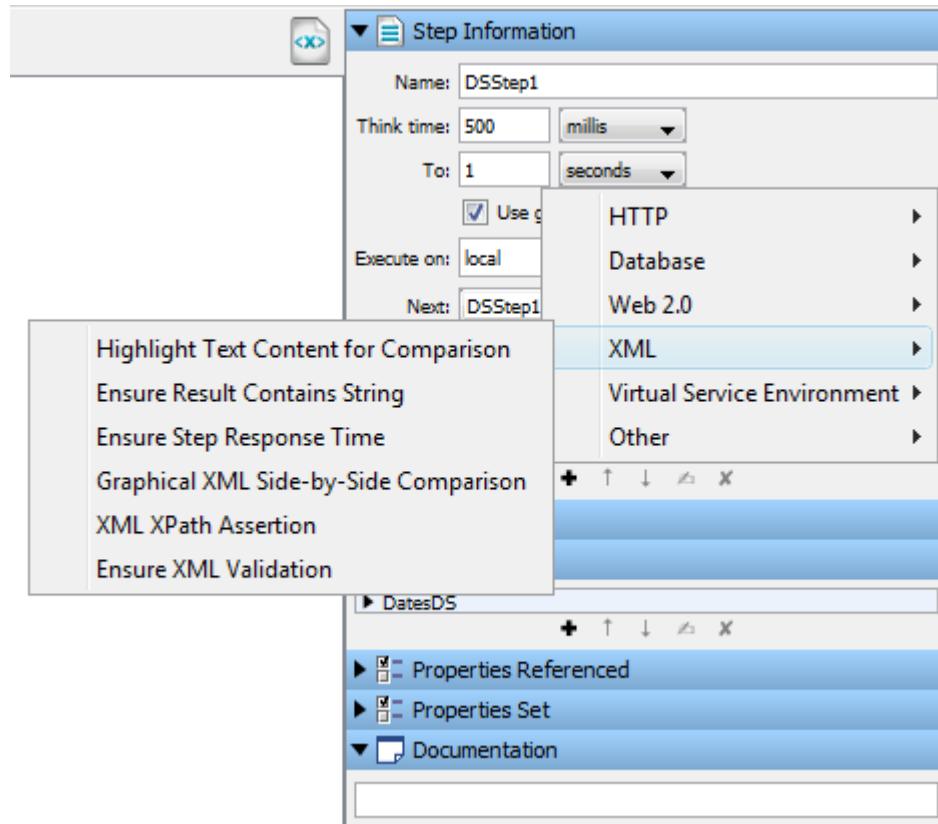
You can add various types of assertions to a test case. In this procedure, you add an XML assertion with the name **Ensure Result Contains String**.

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.

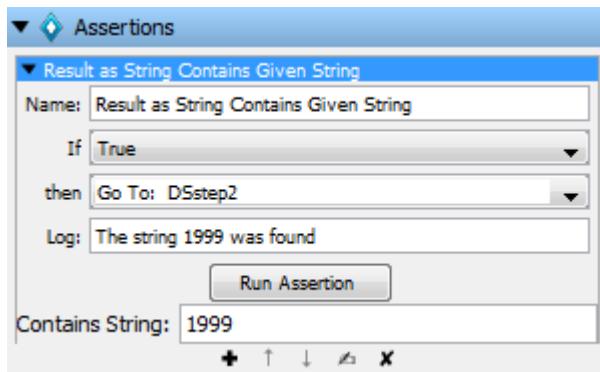
Follow these steps:

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

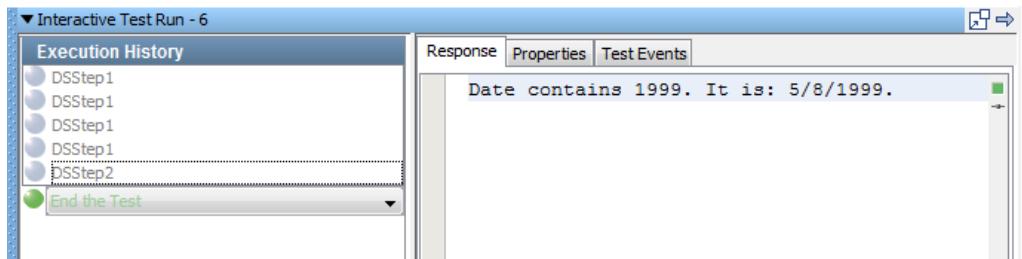
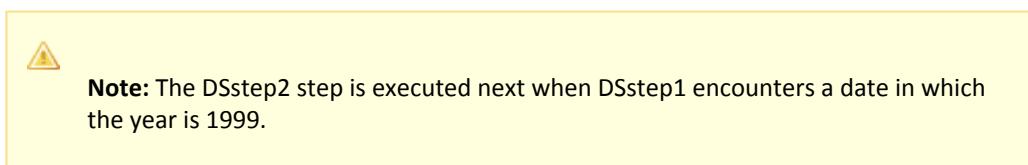
7. Click **Save**.

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.



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.

Execution History			
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...	

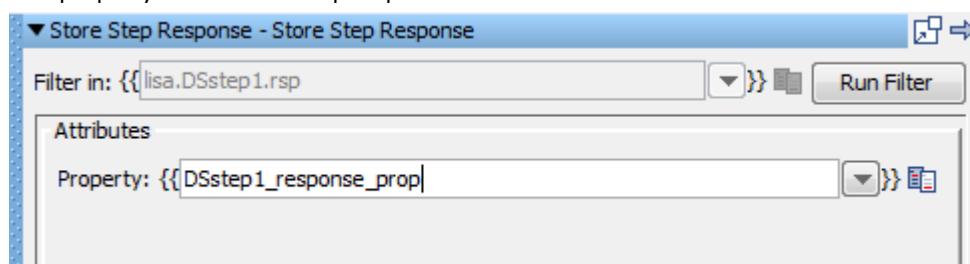
Screenshot of ITR Test Events tab for Tutorial 3

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:

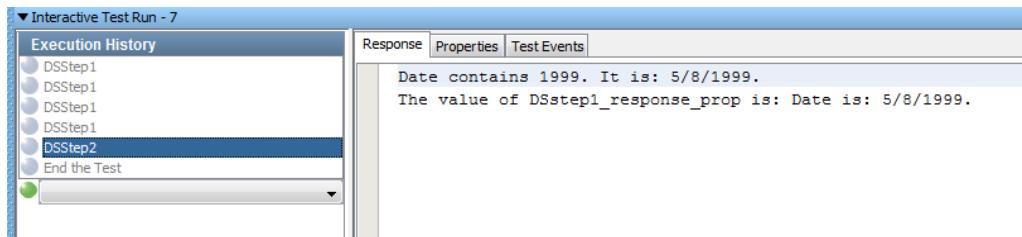
The value of DSstep1_response_prop is: {{DSstep1_response_prop}}.
7. Click **Save**.

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.

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.

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 Developers Kit (SDK). See [Using the SDK \(see page 1212\)](#) for more information.

Tutorial 4 - Manipulate Java Objects (POJOs)

Contents

- [Step 1 - Create a Test Case \(see page 224\)](#)
- [Step 2 - Create a Dynamic Java Execution Test Step \(see page 224\)](#)
- [Step 3 - Make a Call on the Java Object \(see page 226\)](#)
- [Step 4 - Add an Inline Filter \(see page 230\)](#)
- [Step 5 - Verify the Property Created \(see page 231\)](#)
- [Tutorial 4 - Review \(see page 232\)](#)

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

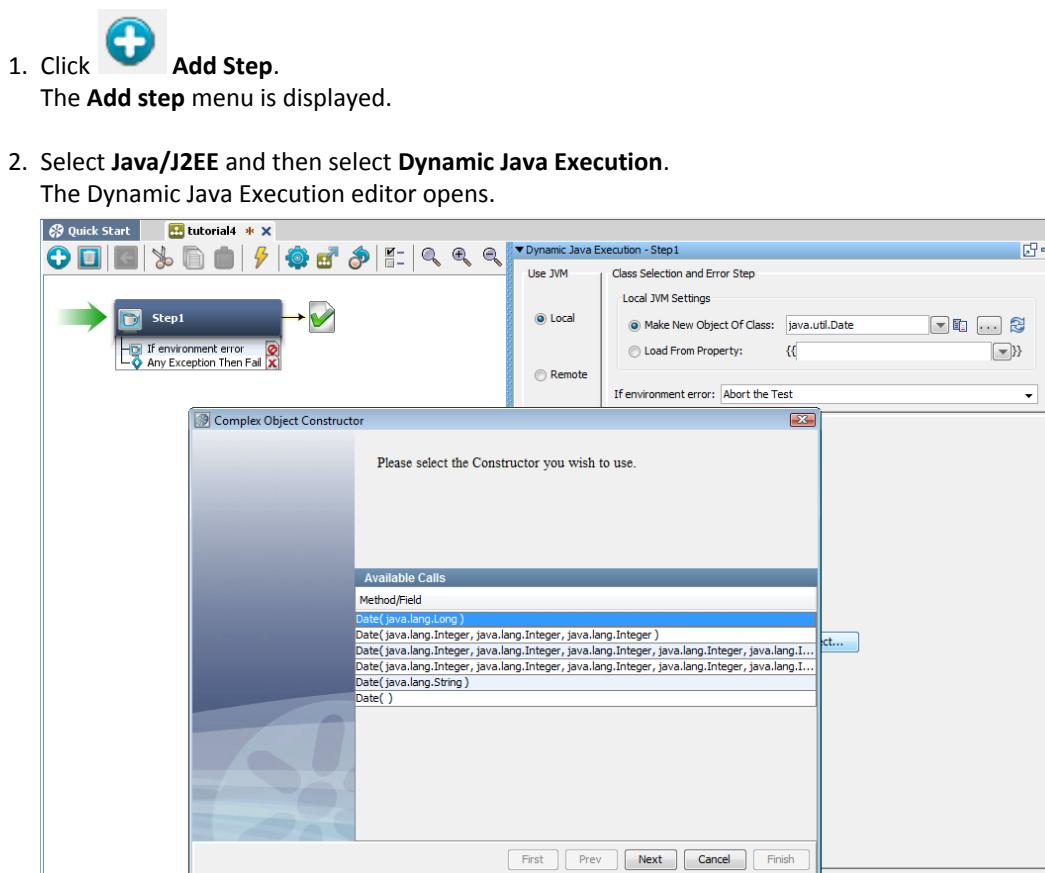
To create a test case:

- 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:



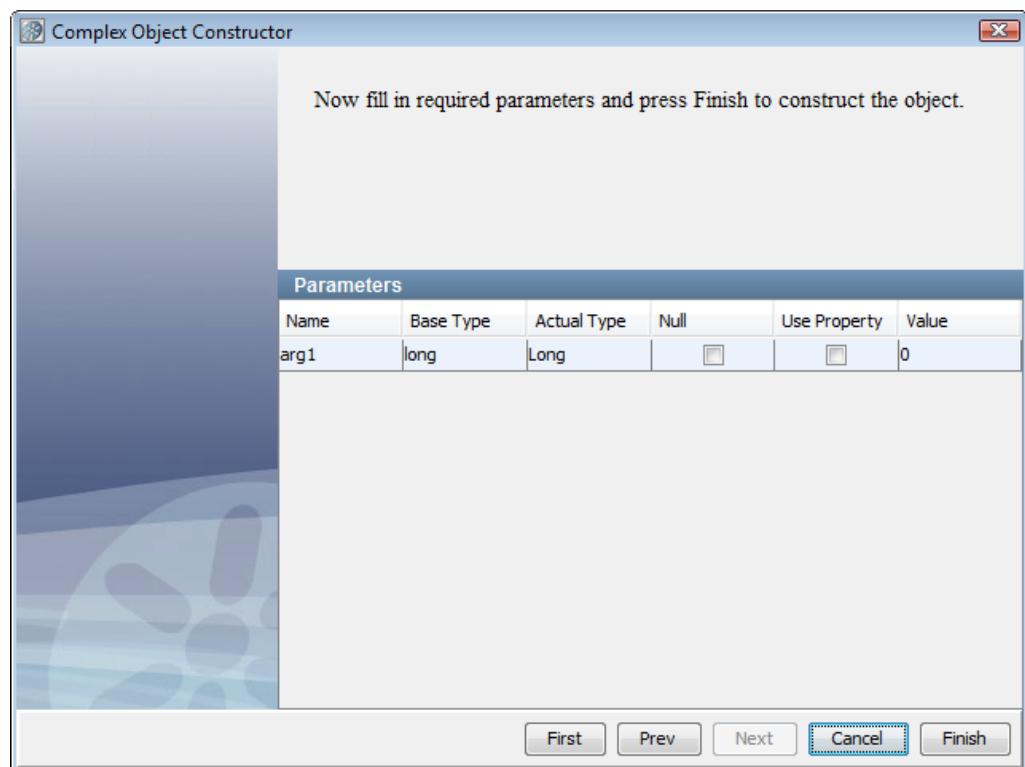
Screenshot of Dynamic Java Execution editor for Tutorial 4

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.

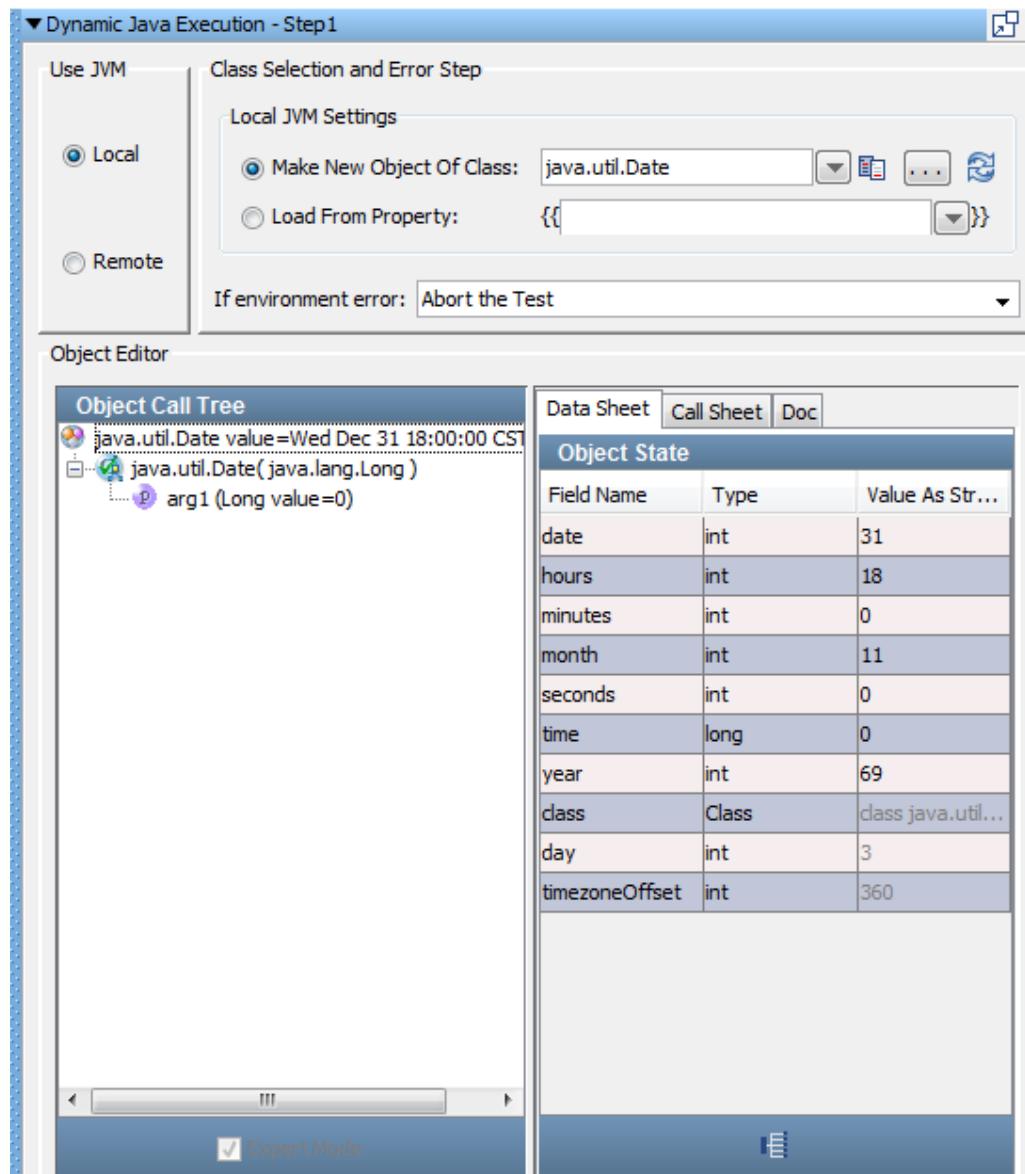


Screenshot of Complex Object Constructor

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.

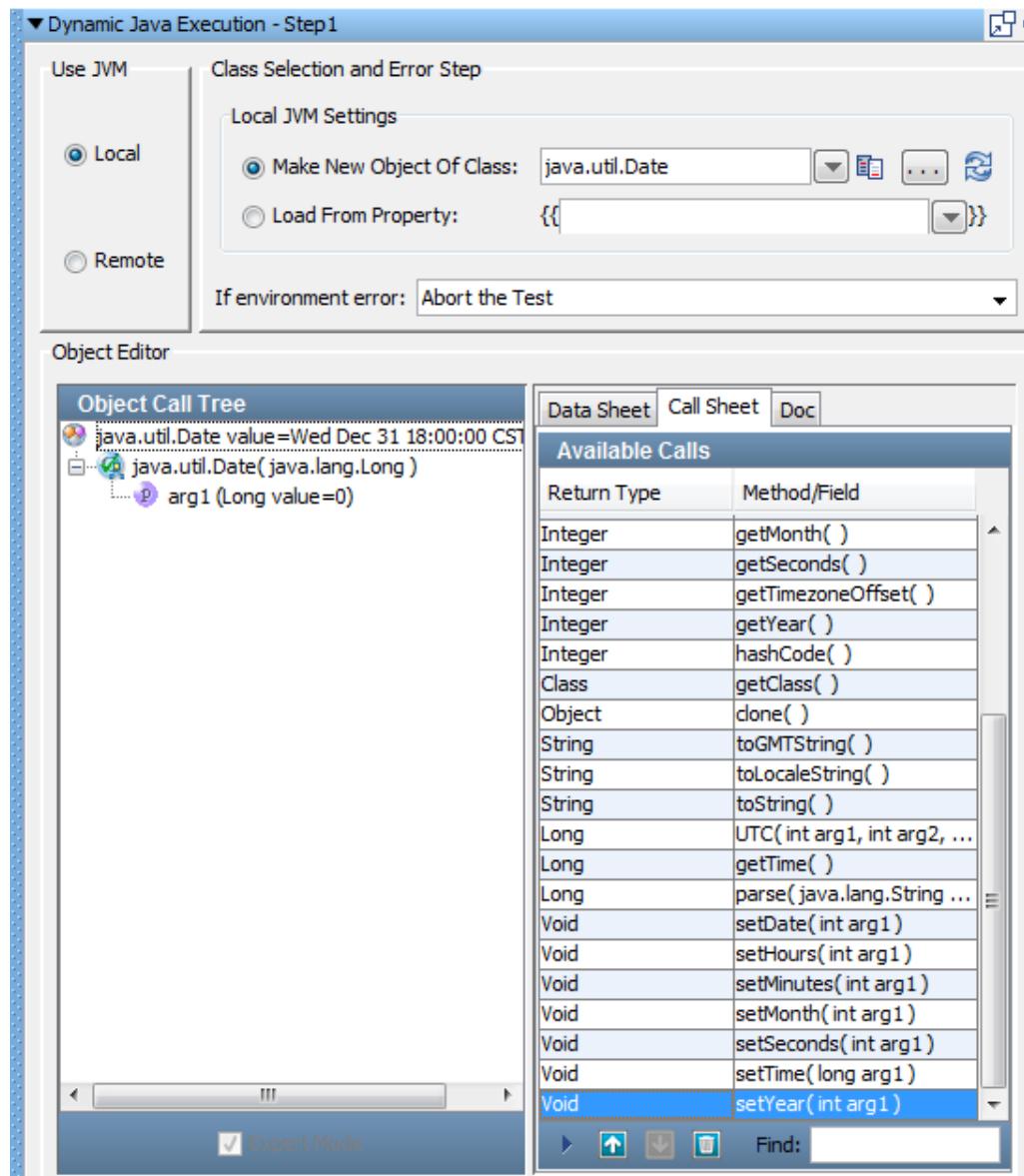
The contents of the right panel vary depending on what is selected in the left panel.

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**.

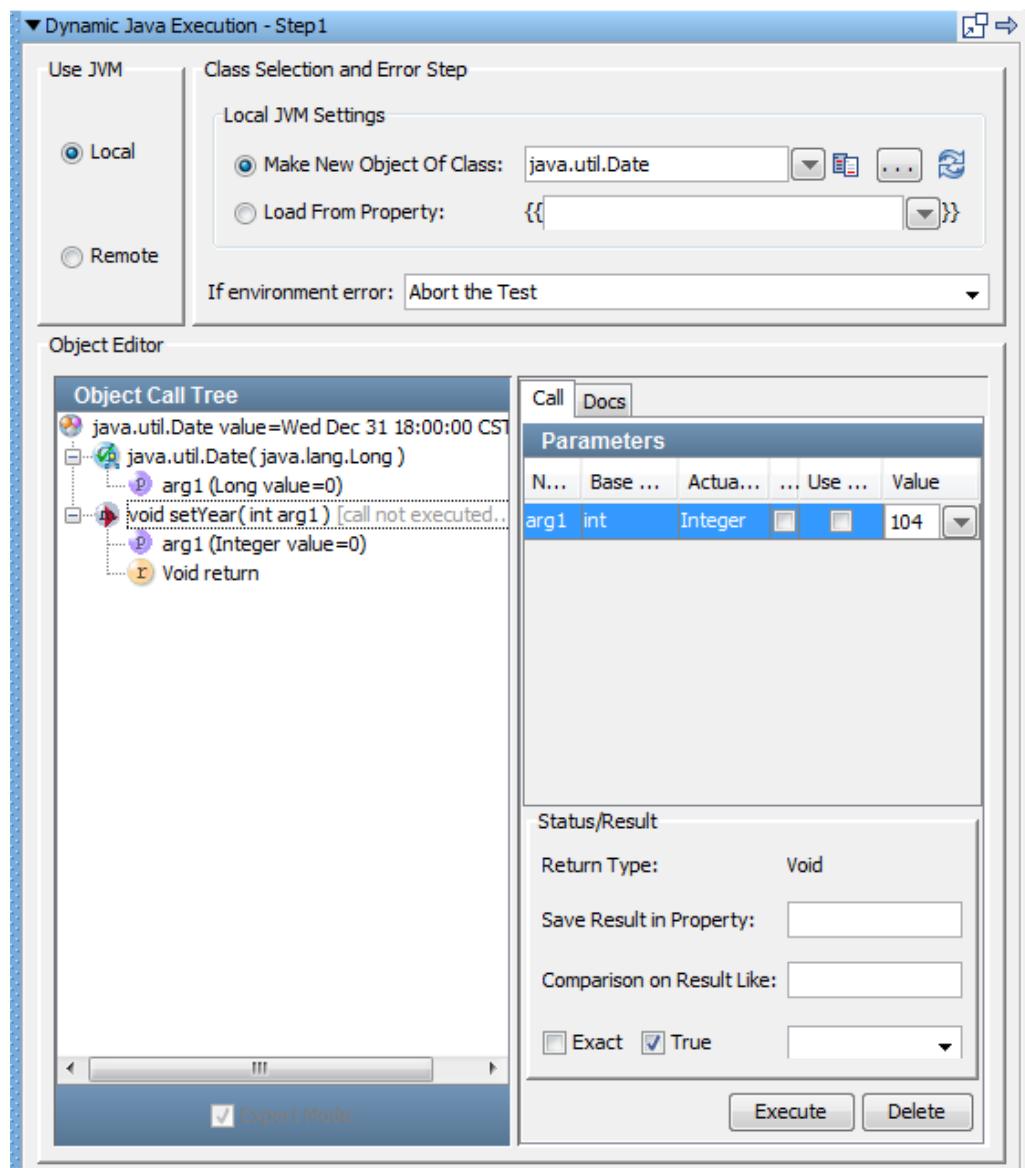


Screenshot of the Call Sheet tab for the Dynamic Java Execution step.

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.

The screenshot shows the DevTest Complex Object Editor interface. On the left, the 'Object Call Tree' panel displays a hierarchical structure of method calls. At the top is a java.util.Date object with a value of 'Fri Dec 31 18:00:00 CST 2011'. Below it is a call to 'java.util.Date(java.lang.Long)' with an argument 'arg1 (Long value=0)'. This is followed by a call to 'void setYear(int arg1)' with an argument 'arg1 (Integer value=104)'. At the bottom of the tree is a call to 'java.lang.String toString()'. To the right of the tree is a 'Data Sheet' tab, which is currently selected. The 'Object State' section contains a table with the following data:

Field Name	Type	Value As String
date	int	31
hours	int	18
minutes	int	0
month	int	11
seconds	int	0
time	long	1104537600000
year	int	104
class	Class	class java.util....
day	int	5
timezoneOffset	int	360

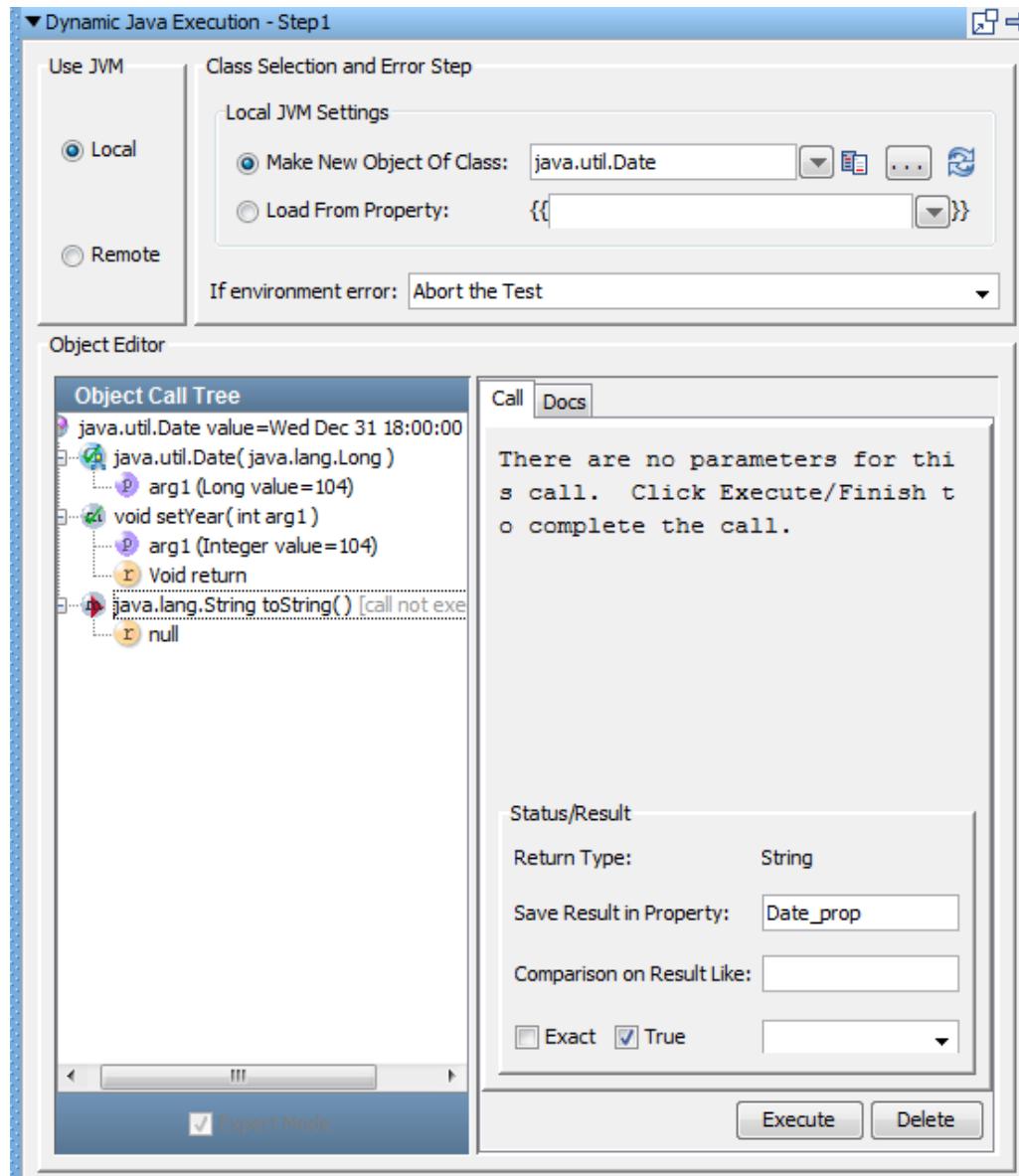
Screenshot of Dynamic Java Execution step COE Data Sheet tab for Tutorial 4

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

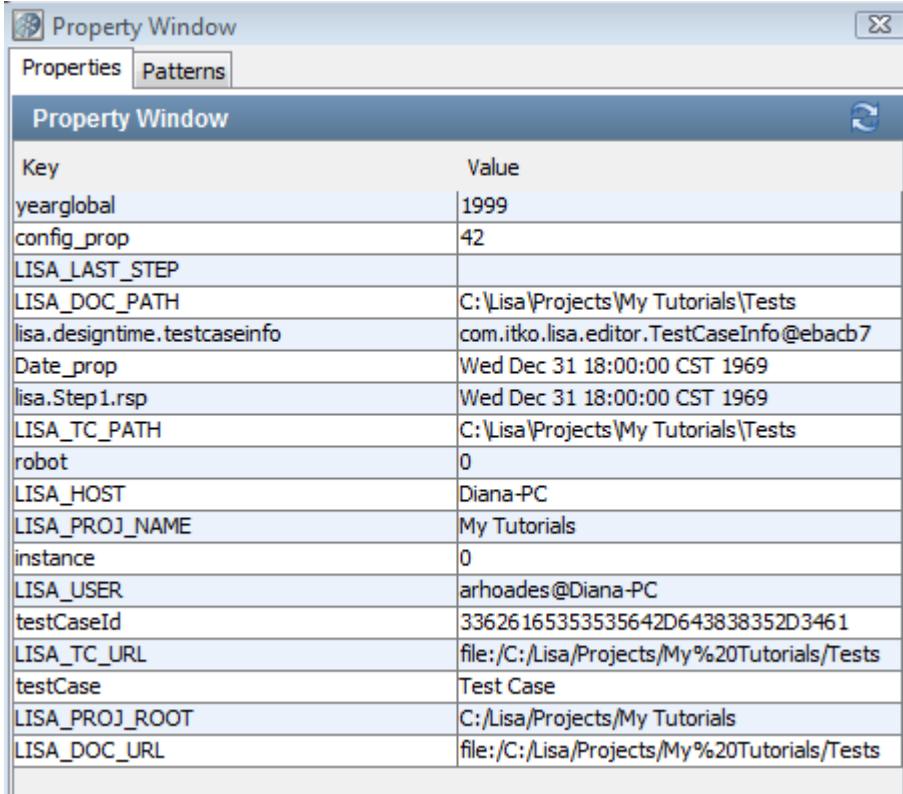
You can display the **Property Window** to verify that the **Date_prop** property was created.

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.



The screenshot shows the 'Property Window' dialog box. It has tabs for 'Properties' and 'Patterns', with 'Properties' selected. A refresh button is located at the top right of the main area. The table lists properties and their values:

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 \(see page 233\)](#)
- [Step 2 - Log in to the Web Application \(see page 234\)](#)
- [Step 3 - Create an Account \(see page 235\)](#)
- [Step 4 - Close an Account \(see page 237\)](#)

- [Step 5 - Log Out from the Web Application \(see page 238\)](#)
- [Tutorial 5 - Review \(see page 238\)](#)

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.



The screenshot shows the LISAfinancial Online login page. At the top right are links for Help, Log In, Home, Location Finder, Contact Us, and Printable. A "Welcome!" message is displayed. On the left, there's a sidebar with the date (Thursday, October 9, 2014), a MyMoney™ Login link, and fields for Name and Password, followed by a Login button. Below this is a "Tested By iTKO LISA™" logo. The main content area features a banner with the text "Sound financial footing for the things that matter". It also includes sections for "New at LISAfinancial" (links to New Flexible-Interest Loan Plans, Just added to MyMoney™: All-in-One Planning Dashboards, and Q1 2012 Financial Results for LISAfinancial) and "Market Rates" (a table showing rates for Mortgage ARM loans). At the bottom, there are links for Home, About Lisa Bank, ABA/Routing Number, Careers, Press, Security and Privacy, and a copyright notice. To the right are logos for Equal Housing Lender and NCUA.

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**.

The screenshot shows the LISAfinancial Online MyMoney Home page. At the top right, there are links for 'Help | Log Out'. Below that, the user 'lisa simpson (lisa_simpson)' is logged in. The main content area has a blue header 'MyMoney Home >'. Underneath it, a section titled 'Accounts' displays a table with columns: Account Number, Account Type, Name, Current Balance, and Available Balance. The table shows 'No Accounts' and a 'Total' row with '\$0.00' for both current and available balance. To the right of the table is a bar chart titled 'Stock Tracker' showing quarterly performance for East, West, and North regions. The chart has four bars per quarter, with values ranging from 0 to 90. Below the chart is a section titled 'MyMoney Tools' with links to Mortgage Planner, Retirement Planner, Online Transactions, Fund Allocation, and Identity Security.

Account Number	Account Type	Name	Current Balance	Available Balance
No Accounts				
Total			\$0.00	\$0.00

MyMoney Tools

- [Mortgage Planner](#)
- [Retirement Planner](#)
- [Online Transactions](#)
- [Fund Allocation](#)
- [Identity Security](#)

Stock Tracker

Region	1st Qtr	2nd Qtr	3rd Qtr	4th Qtr
East	25	30	15	20
West	35	40	30	35
North	45	48	42	45

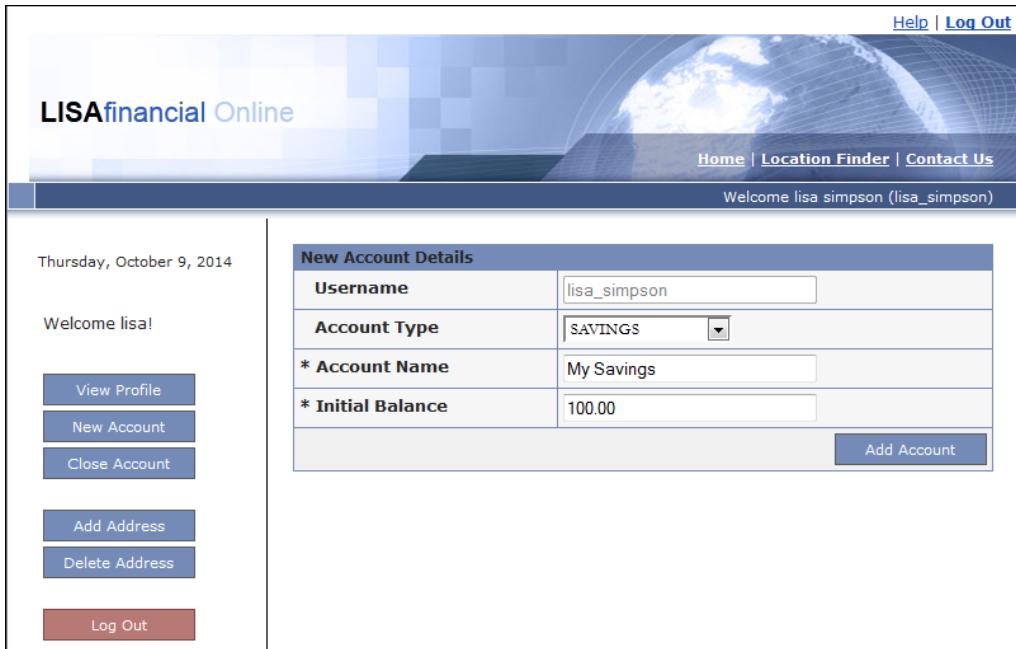
LISA Bank welcome page

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**.



LISAfinancial Online

Home | Location Finder | Contact Us

Welcome lisa simpson (lisa_simpson)

Thursday, October 9, 2014

Welcome lisa!

[View Profile](#)

[New Account](#)

[Close Account](#)

[Add Address](#)

[Delete Address](#)

[Log Out](#)

New Account Details	
Username	<input type="text" value="lisa_simpson"/>
Account Type	<input type="text" value="SAVINGS"/> <input checked="" type="checkbox"/>
* Account Name	<input type="text" value="My Savings"/>
* Initial Balance	<input type="text" value="100.00"/>
Add Account	

LISA Bank new account page

The new savings account is added to the **Accounts** section.



LISAfinancial Online

Home | Location Finder | Contact Us

Welcome lisa simpson (lisa_simpson)

Thursday, October 9, 2014

Welcome lisa!

[View Profile](#)

[New Account](#)

[Close Account](#)

[Add Address](#)

[Delete Address](#)

[Log Out](#)

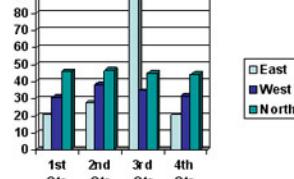
MyMoney Home >

Accounts

Account Number	Account Type	Name	Current Balance	Available Balance
6947696618	SAVINGS	My Savings	\$100.00	\$100.00
Total			\$100.00	\$100.00

MyMoney Tools

- [Mortgage Planner](#)
- [Retirement Planner](#)
- [Online Transactions](#)
- [Fund Allocation](#)
- [Identity Security](#)



Quarter	East	West	North
1st Qtr	30	25	45
2nd Qtr	35	20	48
3rd Qtr	32	30	88
4th Qtr	20	35	42

LISA Bank with new savings account added

6. To create two more accounts: Checking and Auto_Loan, repeat the preceding steps. 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.

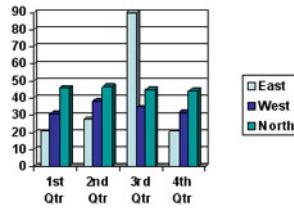


The screenshot shows the LISAfinancial Online MyMoney Home screen. At the top right, there are links for 'Help | Log Out'. Below that, the user 'lisa simpson (lisa_simpson)' is logged in. The main content area has a header 'MyMoney Home > Accounts'. It displays a table of accounts with columns for Account Number, Account Type, Name, Current Balance, and Available Balance. The table includes three rows of accounts and a summary row 'Total'. To the left, a sidebar lists options like View Profile, New Account, Close Account, Add Address, Delete Address, and Log Out. Below the sidebar, it says 'Tested By ITKO LISA™'.

Account Number	Account Type	Name	Current Balance	Available Balance
6947696618	SAVINGS	My Savings	\$100.00	\$100.00
7100234497	CHECKING	My Checking	\$600.00	\$600.00
7123992082	AUTO_LOAN	My Car Loan	\$10,000.00	\$10,000.00
Total			\$10,700.00	\$10,700.00

MyMoney Tools

- [Mortgage Planner](#)
- [Retirement Planner](#)
- [Online Transactions](#)
- [Fund Allocation](#)
- [Identity Security](#)



A bar chart comparing three regions (East, West, North) across four quarters (1st Qtr, 2nd Qtr, 3rd Qtr, 4th Qtr). The Y-axis ranges from 0 to 90. The legend indicates: East (light blue), West (dark blue), and North (teal). The chart shows significant fluctuations, particularly in the 3rd quarter where the North region reaches nearly 90.

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**.

The screenshot shows the LISAfinancial Online interface. At the top right, there are links for 'Help | Log Out'. Below that, the user 'lisa simpson (lisa_simpson)' is logged in. The main content area has a left sidebar with a date ('Thursday, October 9, 2014') and several buttons: 'View Profile', 'New Account', 'Close Account', 'Add Address', 'Delete Address', and 'Log Out'. The right side shows a 'Select Account' section where '6947696618 (My Savings)' is selected, and a 'Select Account' button. Below it is an 'Account Details' table with the following data:

Account Details	
Username	lisa_simpson
Account ID	6947696618
Account Type	SAVINGS
Account Name	My Savings
Balance	100.0

At the bottom right of the details table is a 'Confirm Delete' button.

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:

1. 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 \(see page 239\)](#)
- [Tutorial 6 - Part B - Run the Test Case \(see page 247\)](#)
- [Tutorial 6 - Part C - Modify Request Test Steps \(see page 248\)](#)

Tutorial 6 - Part A - Record the Test Case

Contents

- [Step 1 - Create a Test Case \(see page 239\)](#)
- [Step 2 - Start the Web Recorder \(see page 239\)](#)
- [Step 3 - Record the LISA Bank Application \(see page 240\)](#)
- [Step 4 - Stop the Web Recorder \(see page 244\)](#)

With the web recorder, create a test case that contains HTTP/HTML Request steps.

Step 1 - Create a Test Case

Follow these steps:

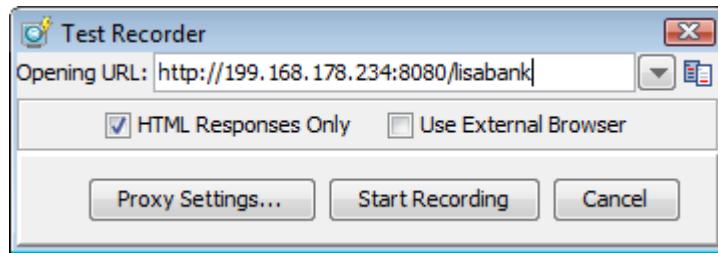
1. In the My Tutorials project, create a test case with the name **tutorial6a** in the **Tests** subfolder.
The model editor opens.

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 of Test Recorder dialog for Tutorial 6

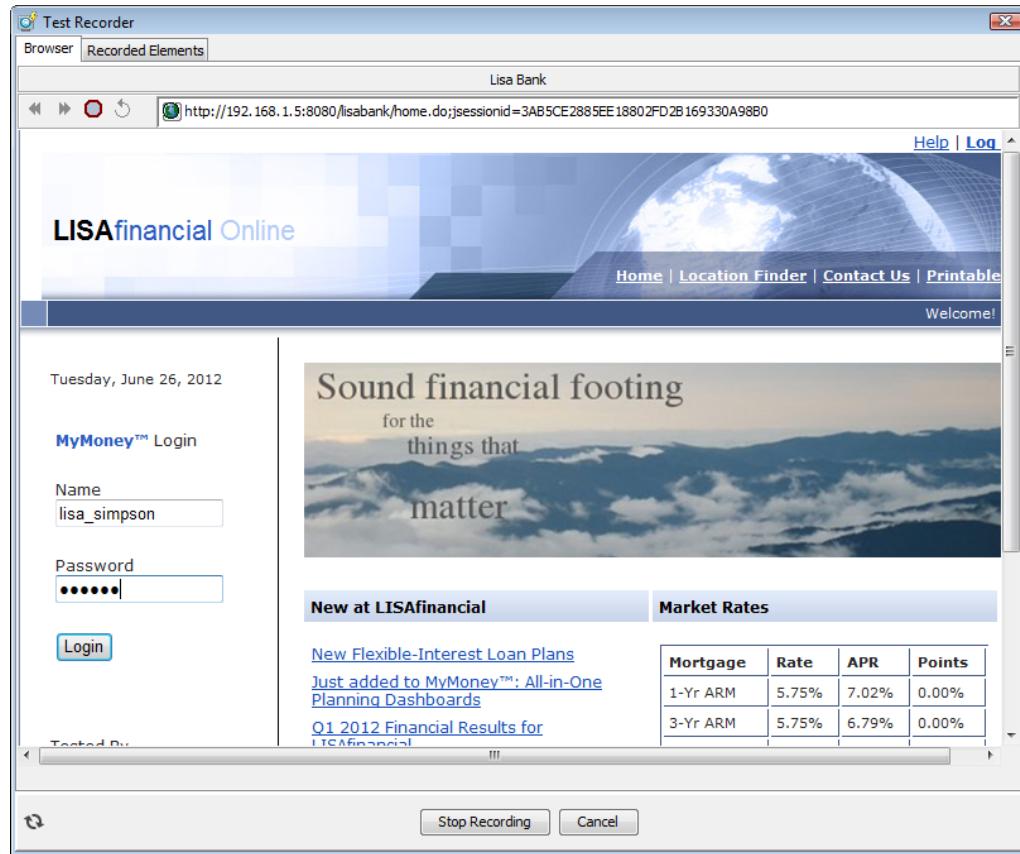
3. Click **Start Recording**.
The **Test Recorder** window opens. The **Test Recorder** window contains two tabs: **Browser** and **Recorded Elements**. The login page of the LISA Bank application appears in the **Browser** tab.

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.

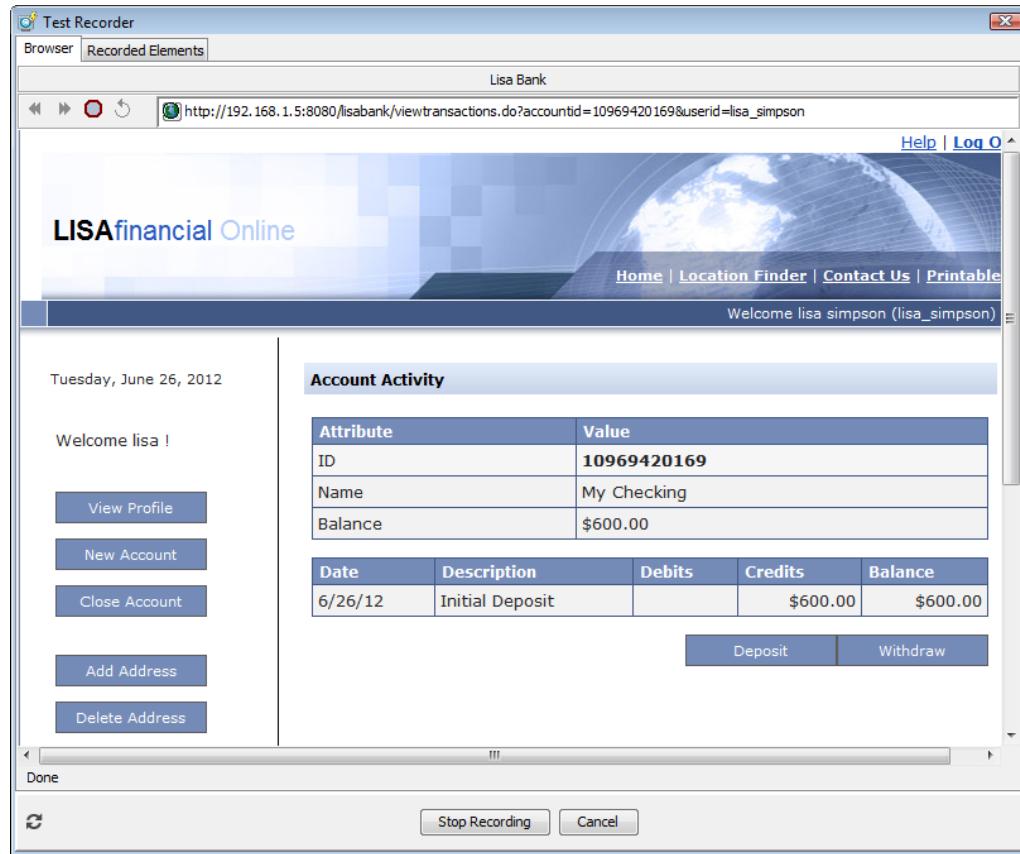
Follow these steps:

1. In the **Name** field, enter **lisa_simpson**. In the **Password** field, enter **golisa**.



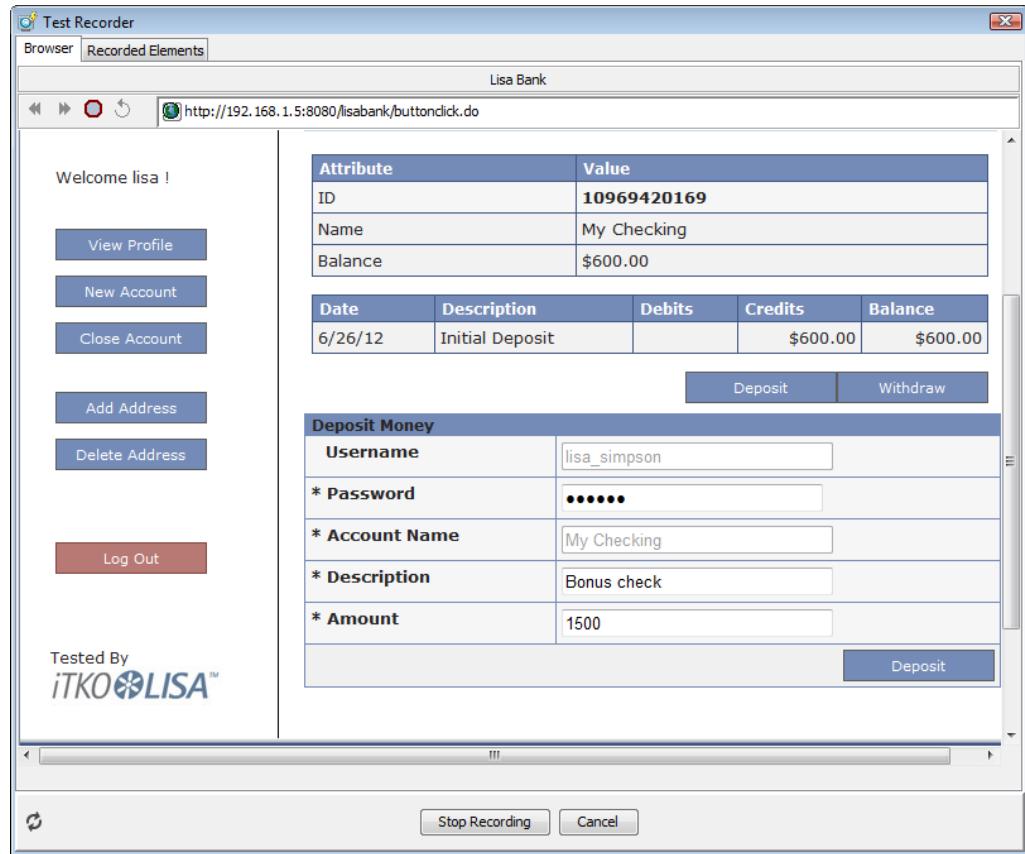
LISA Bank login screen for Tutorial 6

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**.



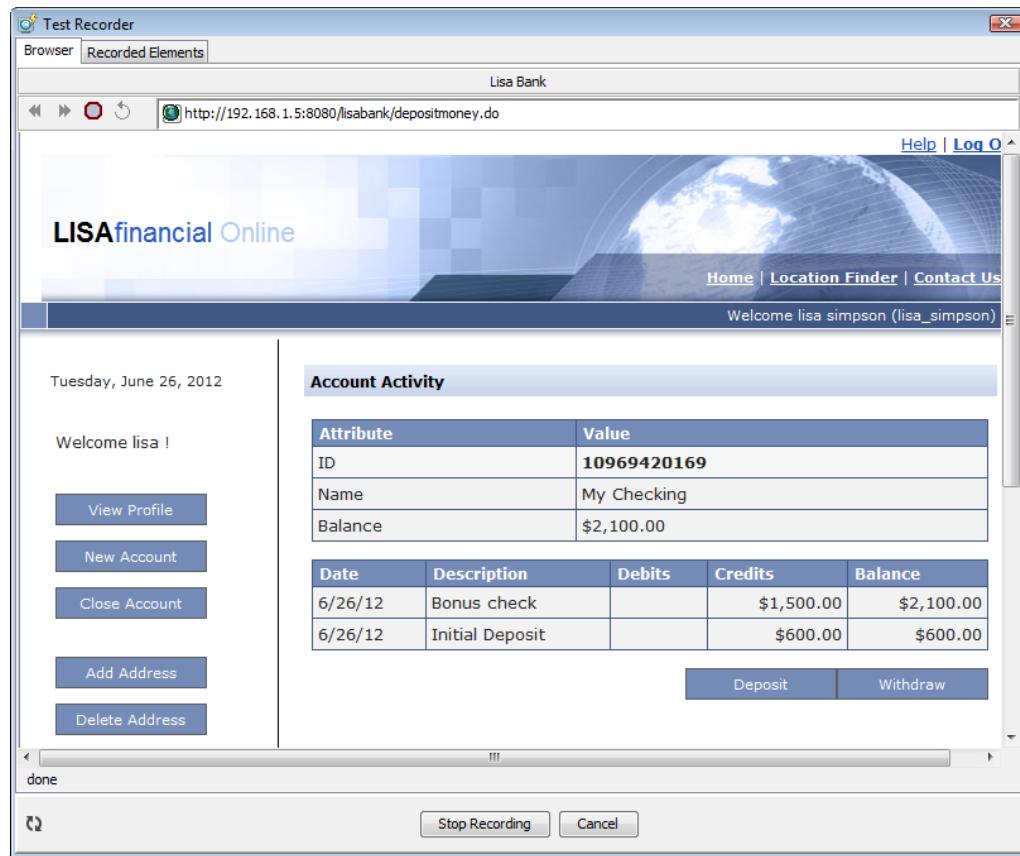
LISA Bank deposit screen for Tutorial 6

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

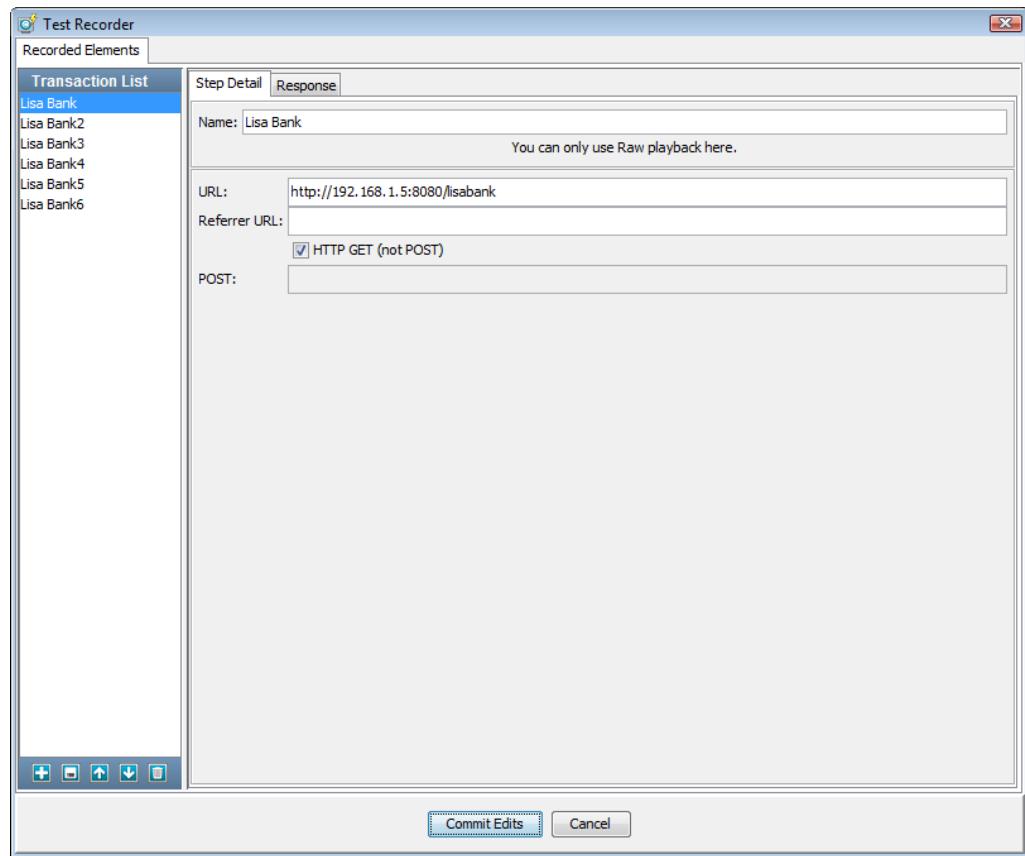
- From the left navigation pane, click **Log Out**.

Step 4 - Stop the Web Recorder

After you stop recording, you can view details about the transactions.

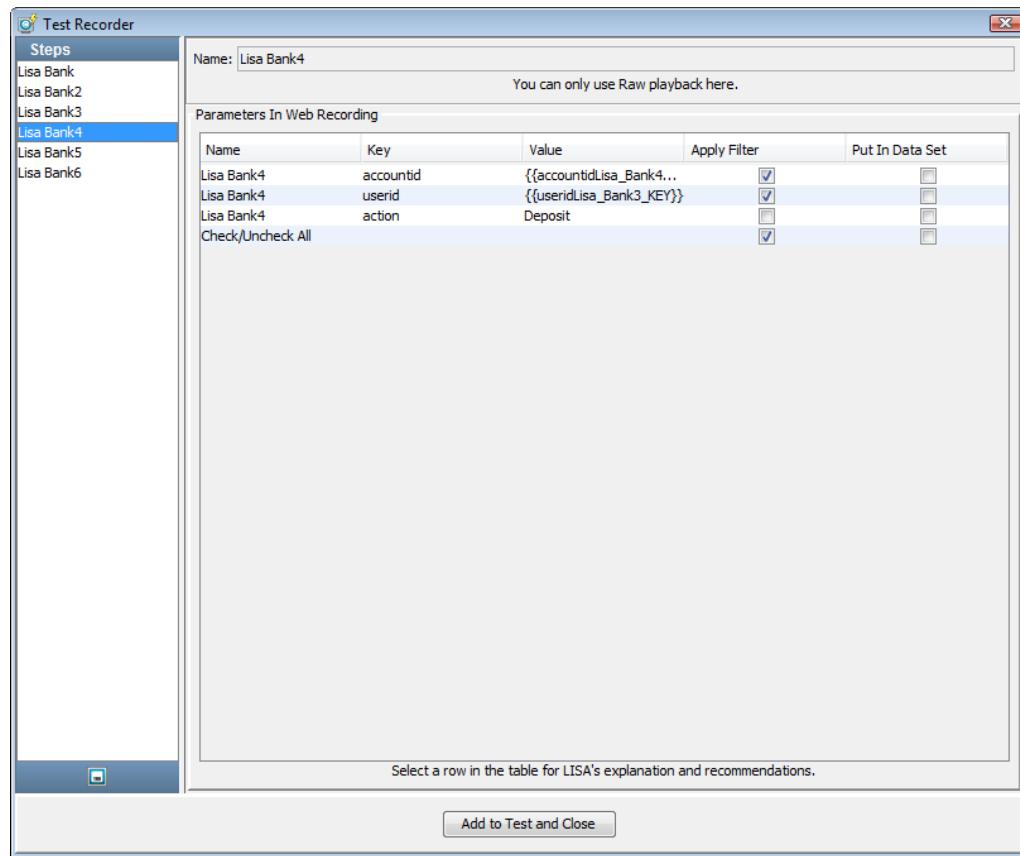
Follow these steps:

- 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.



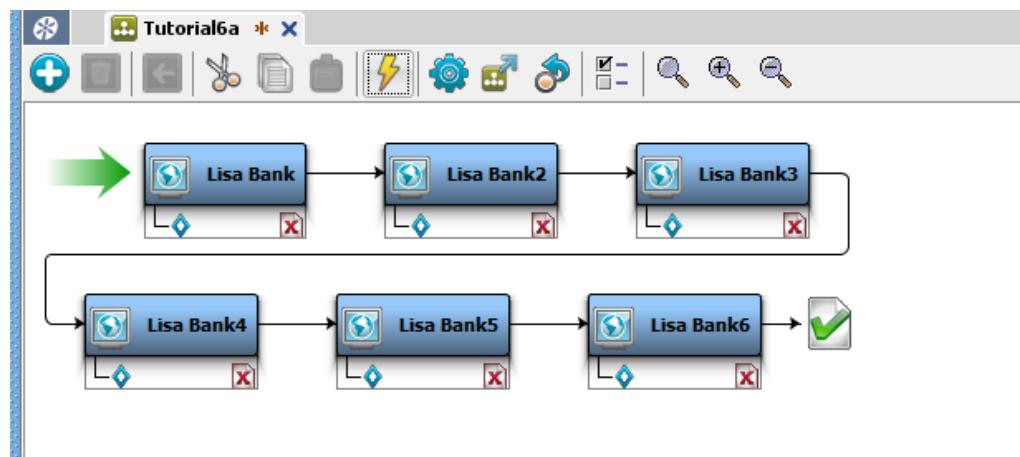
Screenshot of Test Recorder, Step Detail tab for Tutorial 6

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

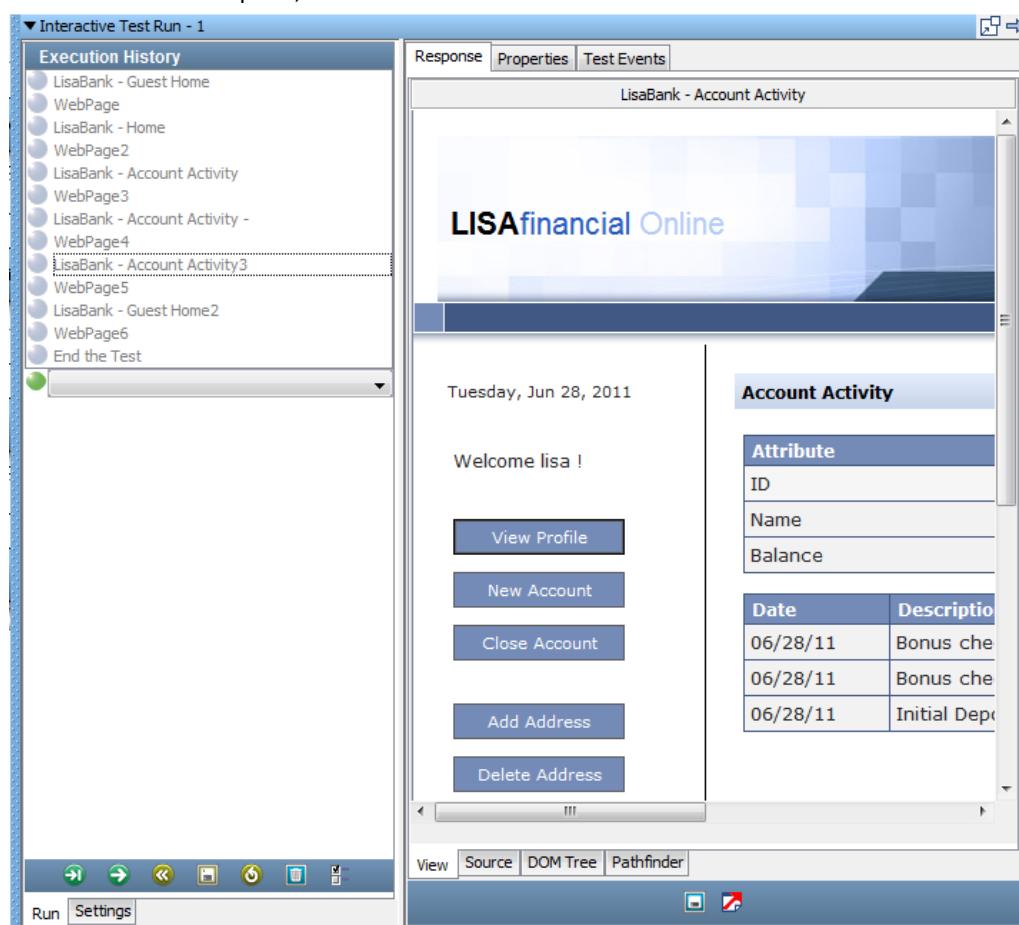
In this section, continue from Part A to run the saved test case in the Interactive Test Run (ITR) utility and view the results.

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 \(see page 248\)](#)
- [Step 2 - Add a Data Set \(see page 248\)](#)
- [Step 3 - Modify the POST Parameters for the Recorded Deposit \(see page 249\)](#)
- [Step 4 - Stage the Test Case \(see page 250\)](#)
- [Step 5 - View the New Deposits in LISA Bank \(see page 252\)](#)
- [Tutorial 6 - Review \(see page 253\)](#)

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**.

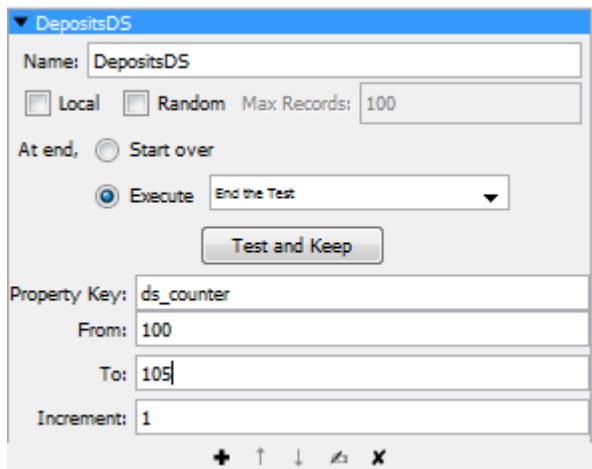
Step 2 - Add a Data Set

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.

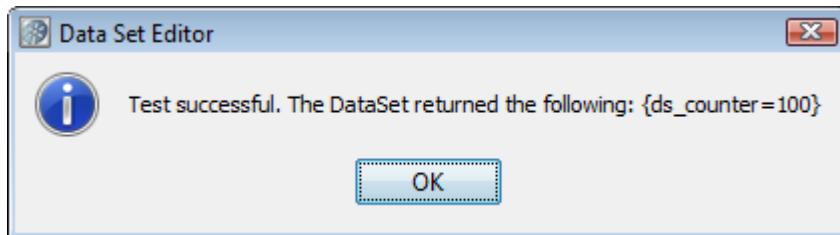
Follow these steps:

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

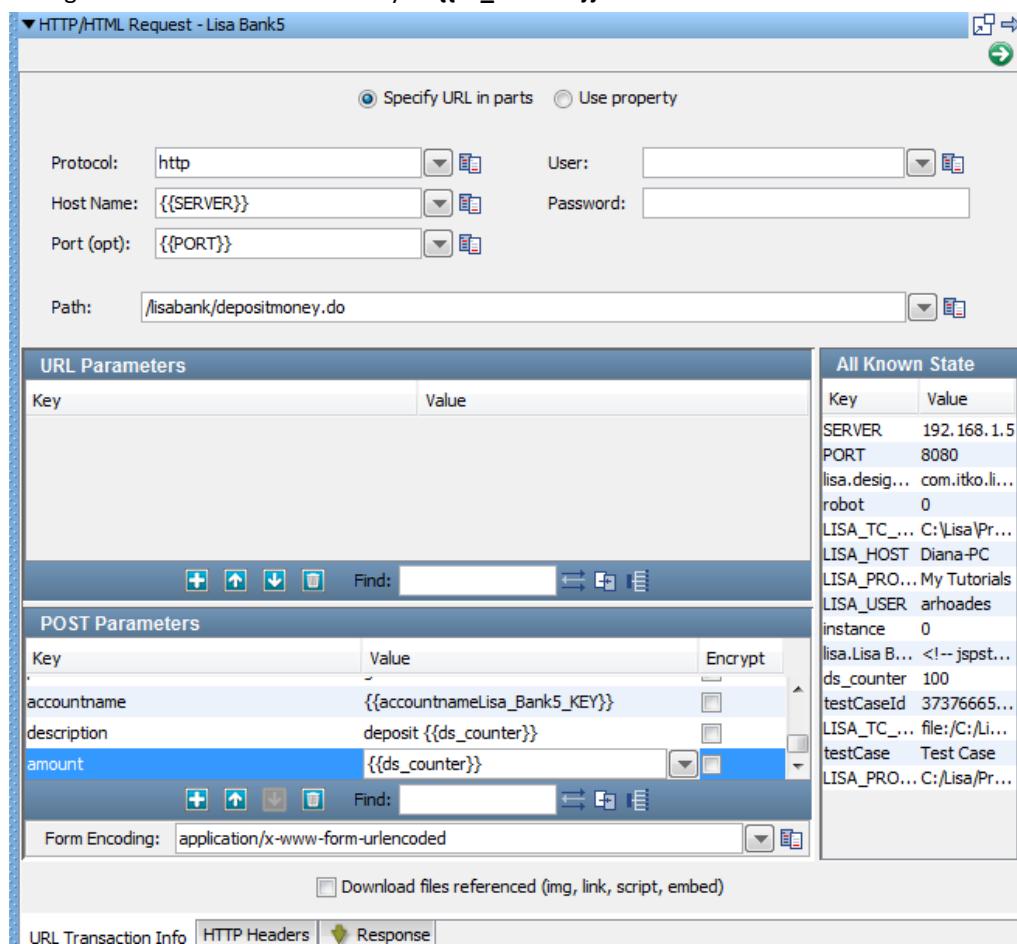
7. Click **OK**.

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}}**.

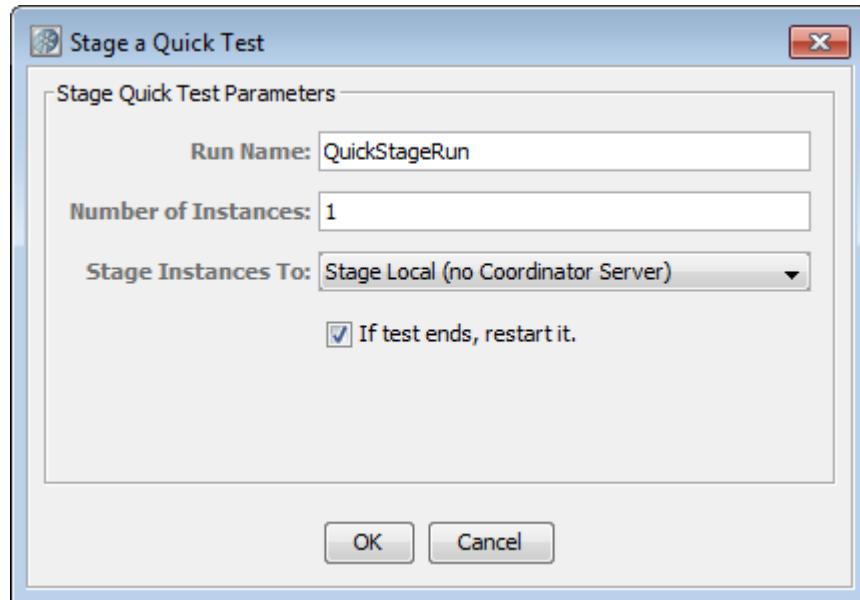


Screenshot of modifying Post parameters for Tutorial 6

4. Save the test case.

Step 4 - Stage the Test Case**To stage (or run) a Quick Test:**

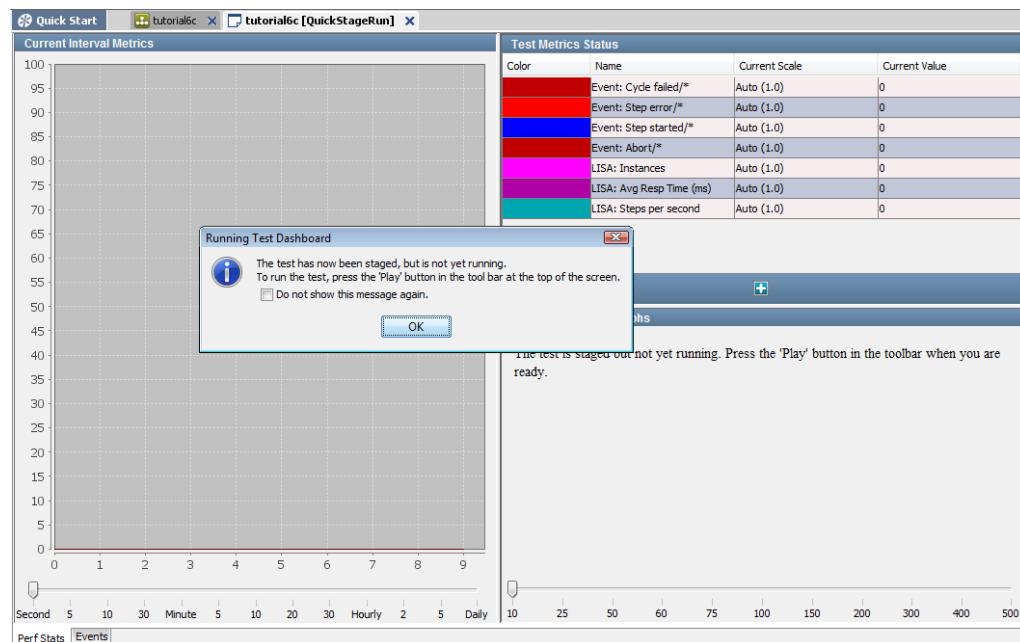
1. From the toolbar, click  **Stage a quick test**.
2. In the **Stage a Quick Test** window, ensure that **If test ends, restart it** is selected.



Stage a Quick Test window

3. Click **OK**.

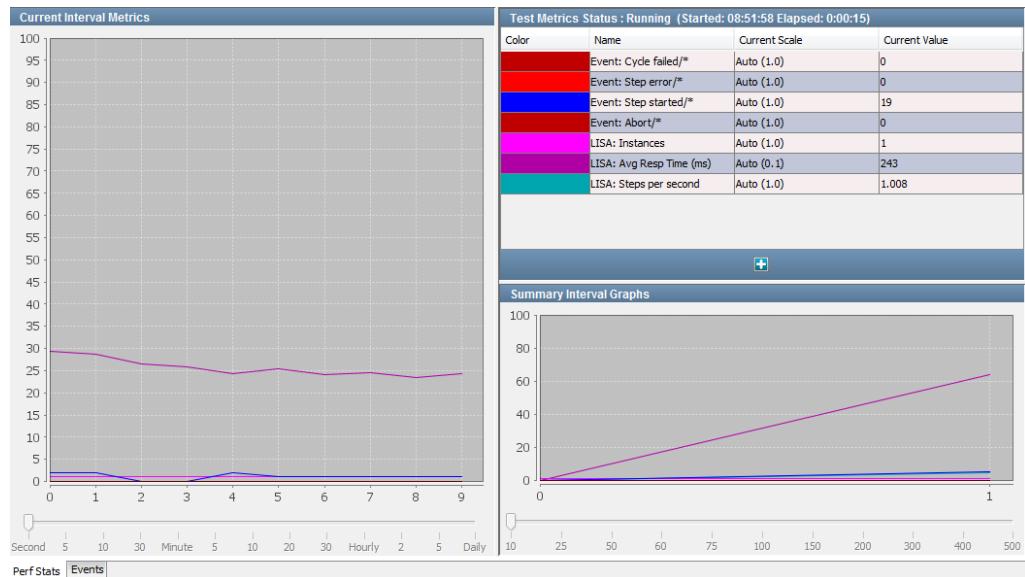
The **Test Monitor** opens, but the test has not been started yet.



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 `go!lisa`.
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.

The screenshot shows a web application interface for LISA financial Online. At the top right, there are links for 'Help | Log Out'. Below that, a banner says 'LISAfinancial Online' with a globe graphic. The main menu includes 'Home | Location Finder | Contact Us | Printable'. A welcome message 'Welcome lisa simpson (lisa_simpson)' is displayed. On the left, a sidebar shows the date 'Tuesday, June 26, 2012' and a list of account management options: 'View Profile', 'New Account', 'Close Account', 'Add Address', 'Delete Address', and 'Log Out'. Below this is a 'Tested By' section with the iTKO LISA logo. The main content area is titled 'Account Activity'. It contains two tables: one for account details (ID: 13060025435, Name: My Checking, Balance: \$2,715.00) and another for transaction history from 6/26/12. The transaction table has columns for Date, Description, Debits, Credits, and Balance. The final balance shown is \$600.00. At the bottom right of the main content area are 'Deposit' and 'Withdraw' buttons.

Attribute	Value
ID	13060025435
Name	My Checking
Balance	\$2,715.00

Date	Description	Debits	Credits	Balance
6/26/12	deposit 105		\$105.00	\$2,715.00
6/26/12	deposit 104		\$104.00	\$2,610.00
6/26/12	deposit 103		\$103.00	\$2,506.00
6/26/12	deposit 102		\$102.00	\$2,403.00
6/26/12	deposit 101		\$101.00	\$2,301.00
6/26/12	deposit 100		\$100.00	\$2,200.00
6/26/12	Bonus Check		\$1,500.00	\$2,100.00
6/26/12	Initial Deposit		\$600.00	\$600.00

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 (see page 254)
- Step 2 - Create a Configuration (see page 254)
- Step 3 - Add an EJB Test Step (see page 255)
- Step 4 - Connect to the Server (see page 255)
- Step 5 - Locate the EJB Interface (see page 256)
- Step 6 - Configure the EJB (see page 257)
- Step 7 - Add an Assertion (see page 259)

- [Step 8 - Verify the Method Execution \(see page 261\)](#)
- [Step 9 - Add Another EJB Test Step \(see page 262\)](#)
- [Tutorial 7 - Review \(see page 264\)](#)

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:

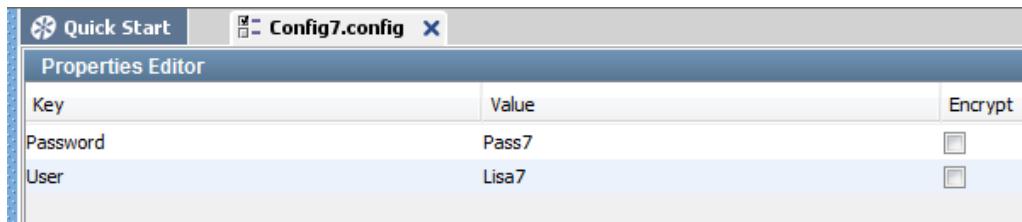
1. In the **Project** pane, right-click the **Tests** folder and select **Create New Test Case**.
2. Set the file name to **tutorial7**.
3. Click **Save**.

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.

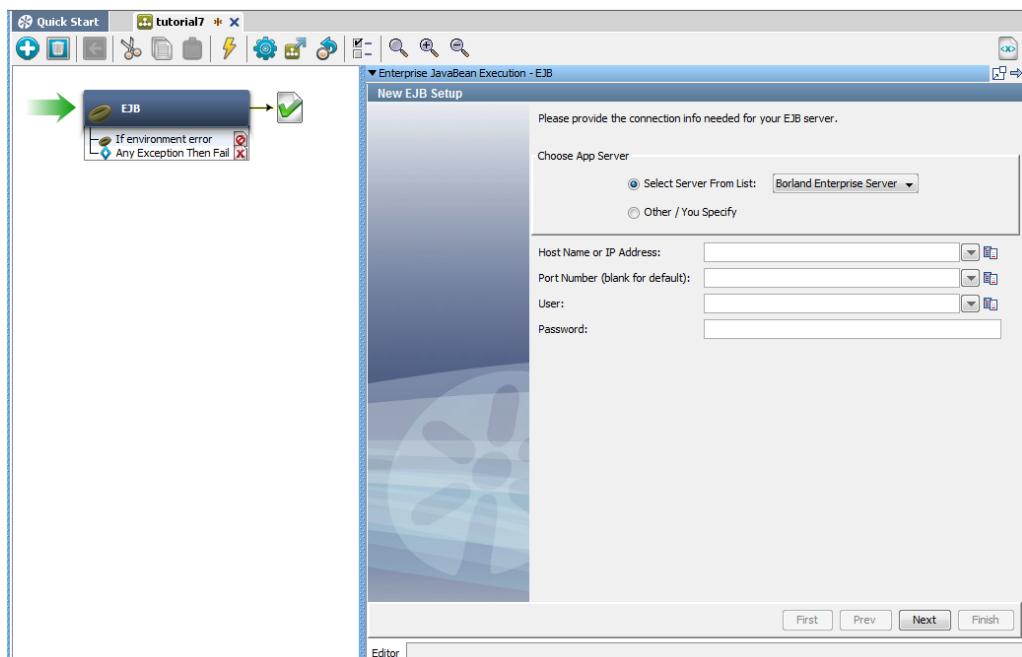
Follow these steps:

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.

Follow these steps:

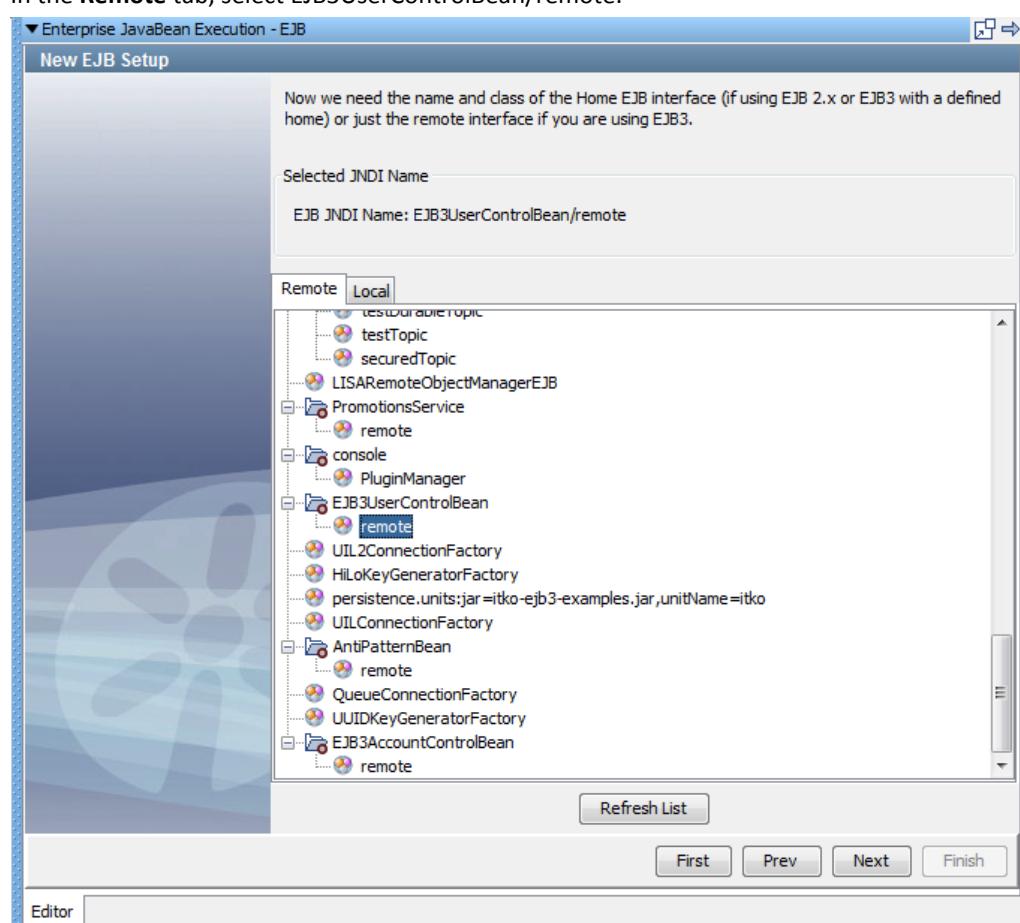
1. From the **Select Server From List** drop-down list, select **JBoss 3.2/4.0**.
2. In the **Host Name or IP Address** field, enter **localhost**.
3. Click **Next**.
The list of JNDI names is retrieved from the EJB server.

Step 5 - Locate the EJB Interface

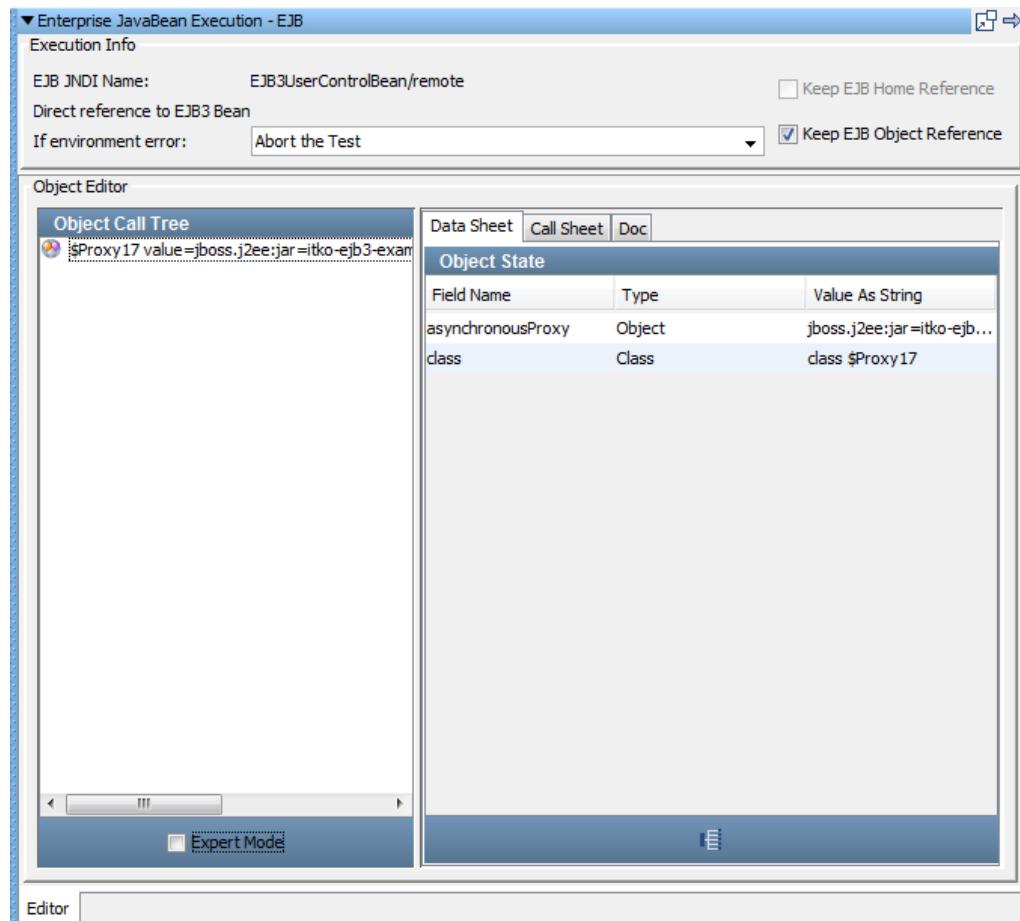
The **New EJB Setup** wizard prompts you to specify the name of the EJB interface.

Follow these steps:

1. In the **Remote** tab, select **EJB3UserControlBean/remote**.



2. Click **Next**.
The Complex Object Editor opens.

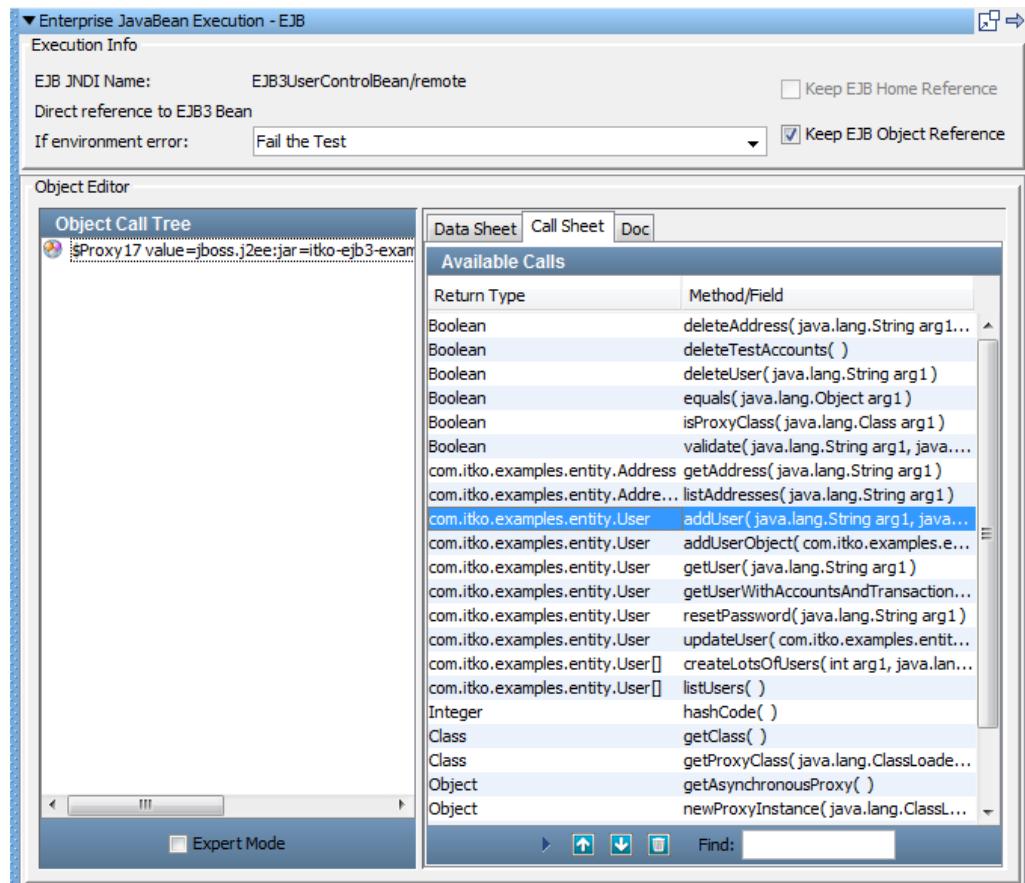


Step 6 - Configure the EJB

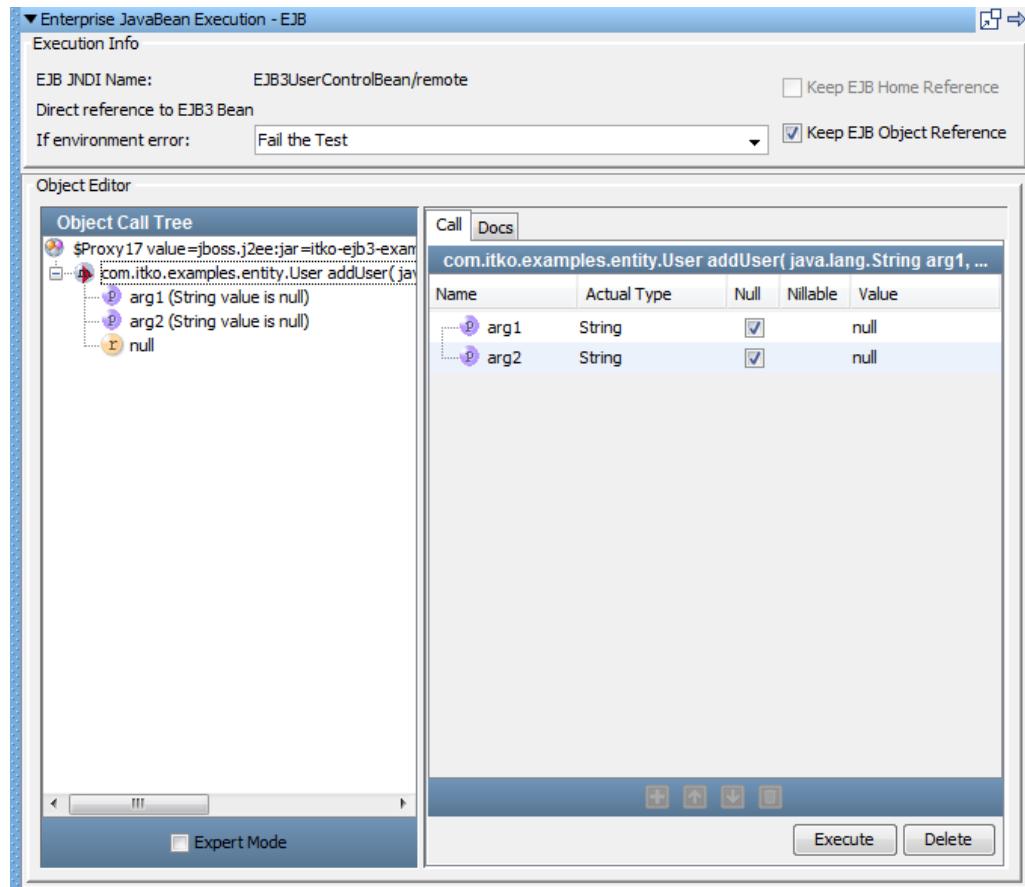
Follow these steps:

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.

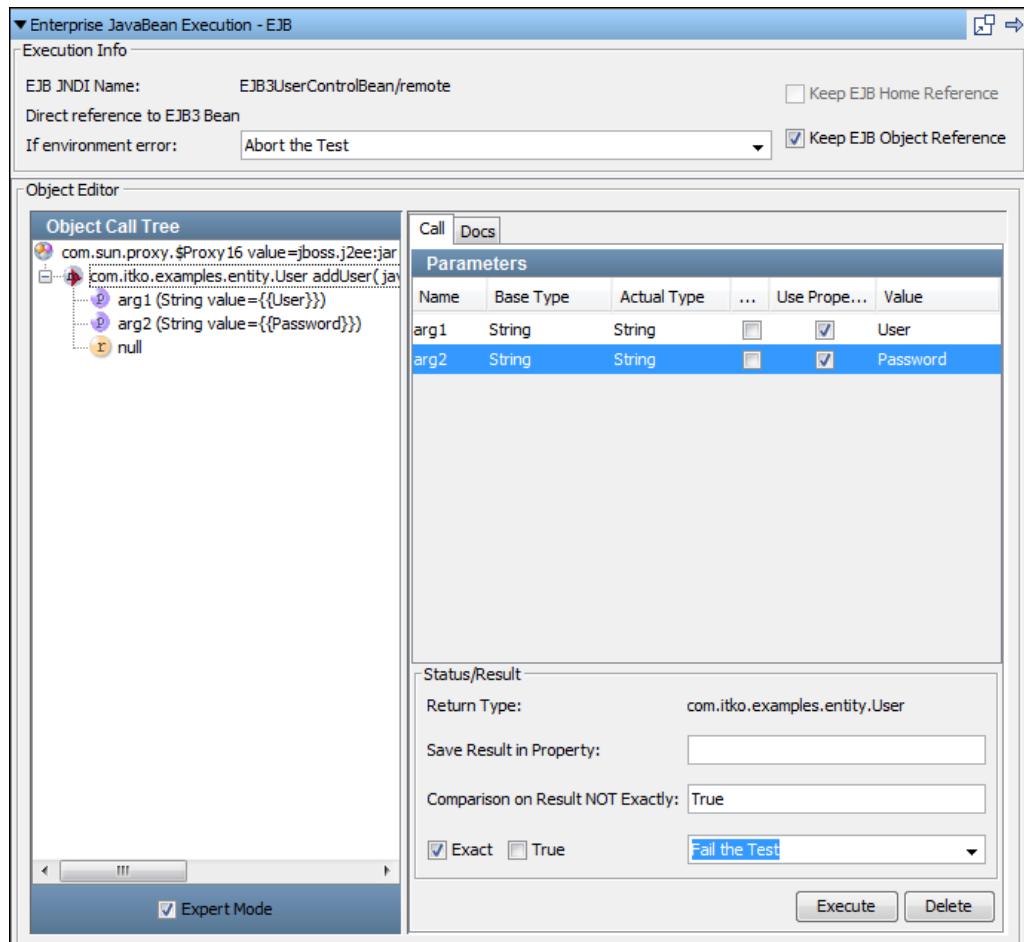
Step 7 - Add an Assertion

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**.

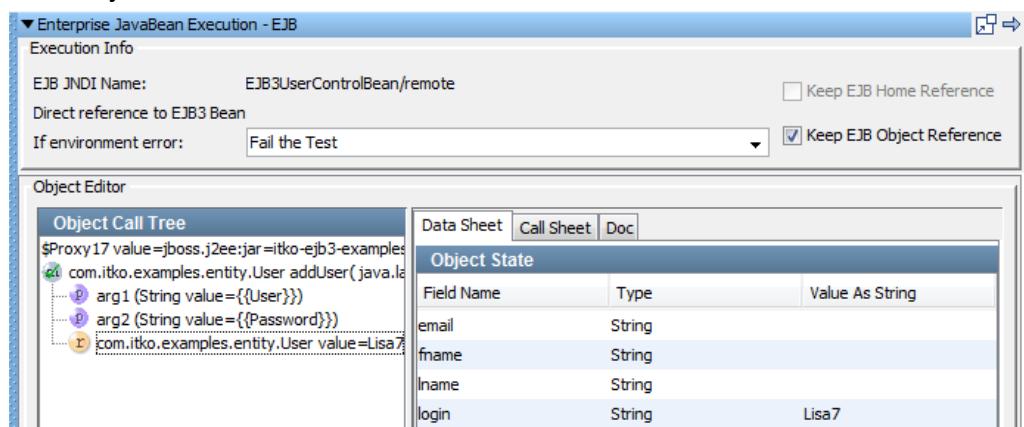
7. From the **Exact** drop-down, select **Fail the Test**.

8. Click **Execute**.



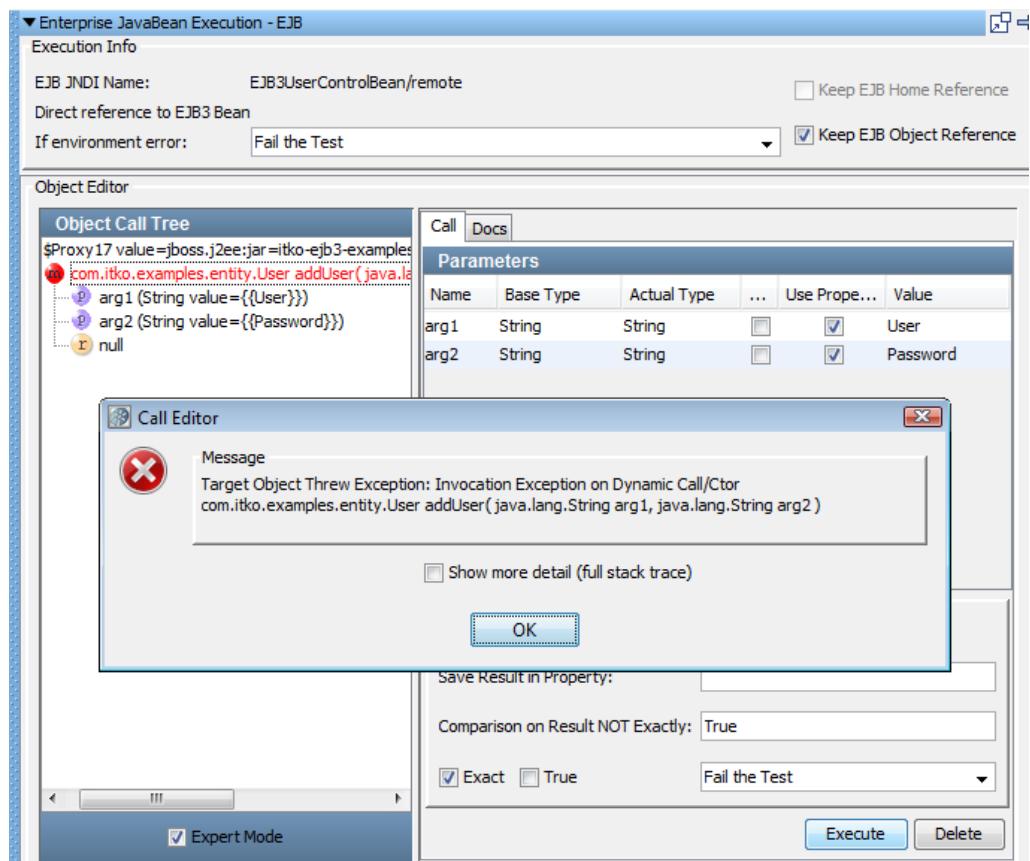
The parameters to the method are displayed in the **Object Call Tree**, next to the **Input**

 **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.

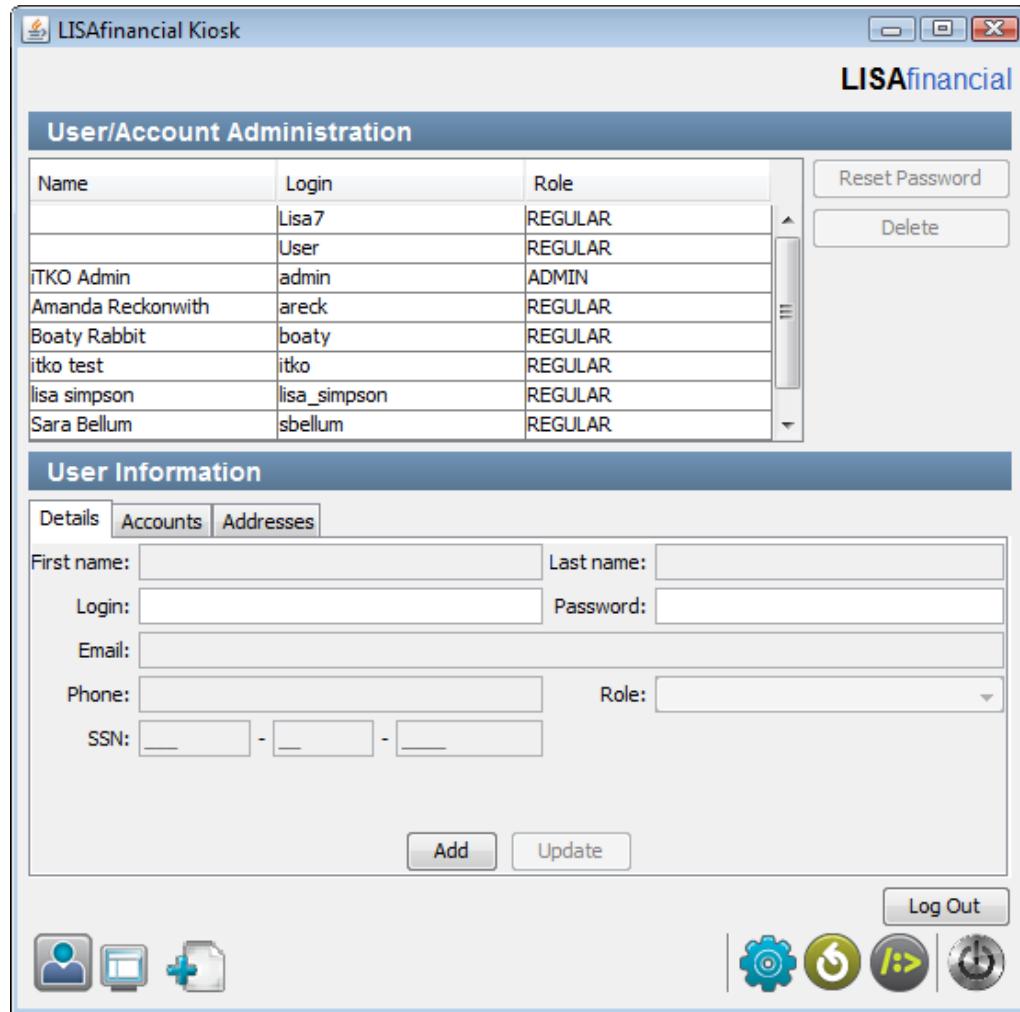


Step 8 - Verify the Method Execution

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.

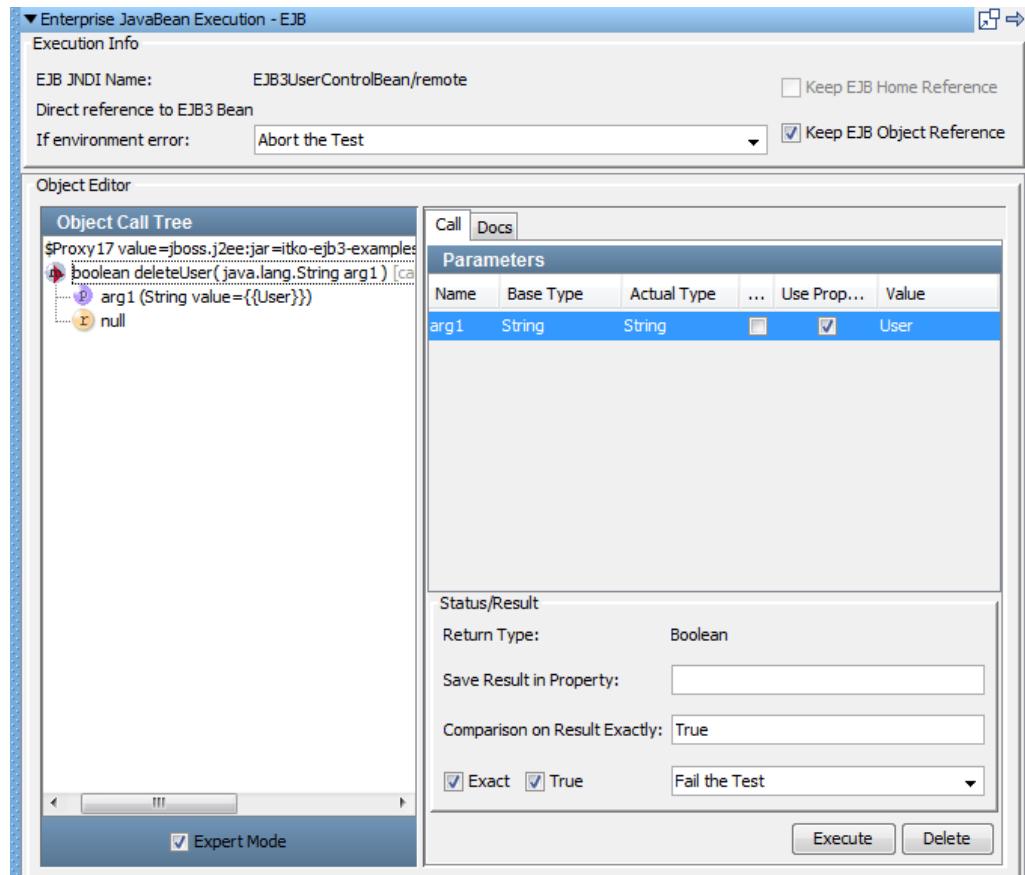


Step 9 - Add Another EJB Test Step

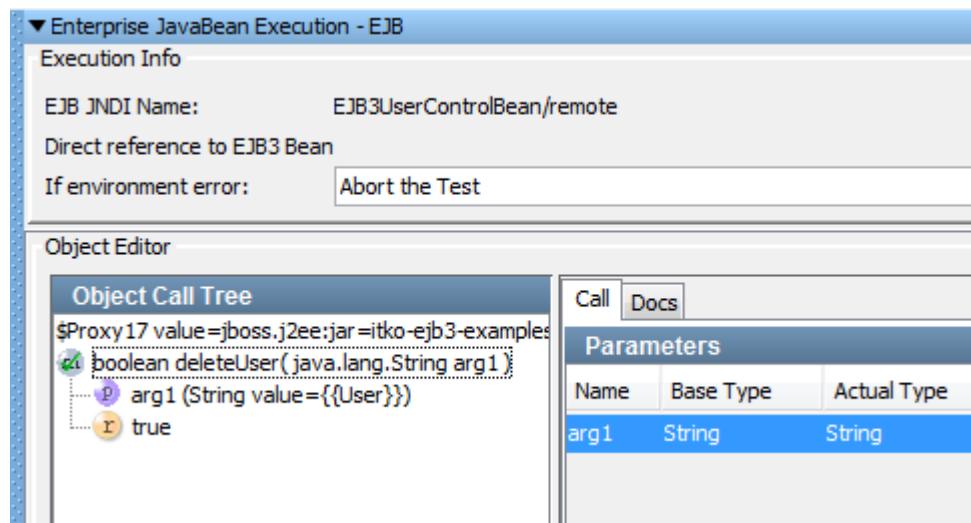
Now try the preceding steps again to invoke the deleteUser method.

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.



- Click **Execute** to execute this method and get results.



The return  is true, indicating the user has been deleted.

- 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 \(see page 264\)](#)
- [Step 2 - Add a Web Service Execution \(XML\) Test Step \(see page 265\)](#)
- [Step 3 - Create a Web Service Client \(see page 266\)](#)
- [Step 4 - Execute the Web Service Request \(see page 268\)](#)
- [Step 5 - View the Request and Response \(see page 268\)](#)
- [Tutorial 8 - Review \(see page 270\)](#)

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**.

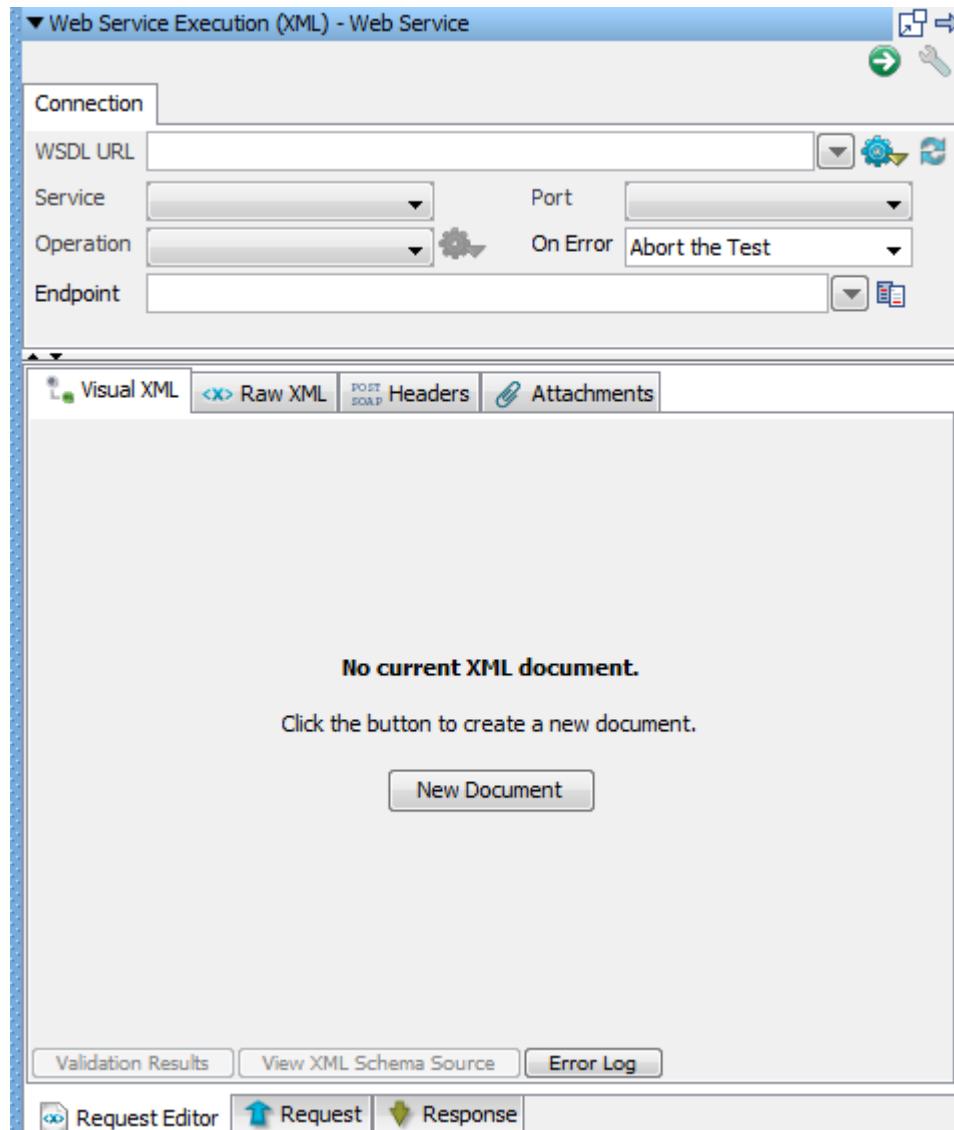
3. Click **Save**.
4. In the **Project** panel, right-click on project.config and select **Make active**.

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 WSOPORT 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.

Node	Type	Occurs	Nil	Nillable	Value
soapenv:Envelope	Envelope	1..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
soapenv:Body	Body	1..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
ns1:addUser			<input type="checkbox"/>		
encodingStyle			<input type="checkbox"/>		http://sche...
login	string		<input type="checkbox"/>	<input checked="" type="checkbox"/>	{{=[:...}}
cleartextPassword	string		<input type="checkbox"/>	<input checked="" type="checkbox"/>	{{=[:...}}

Screenshot of Visual XML editor for Tutorial 8

6. Save the test case.

Step 4 - Execute the Web Service Request

Follow these steps:

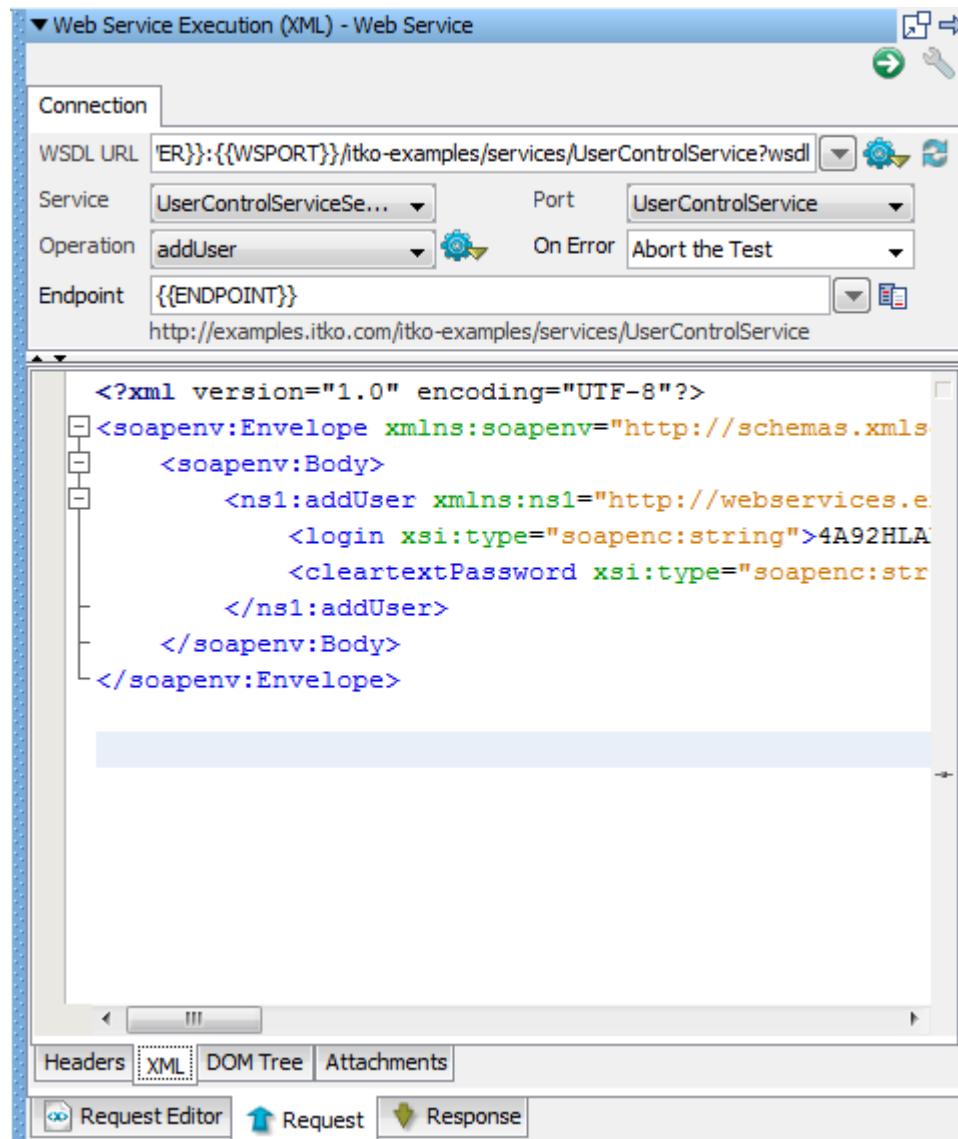
1. Click  **Execute WS Request.**
The test is executed.

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.

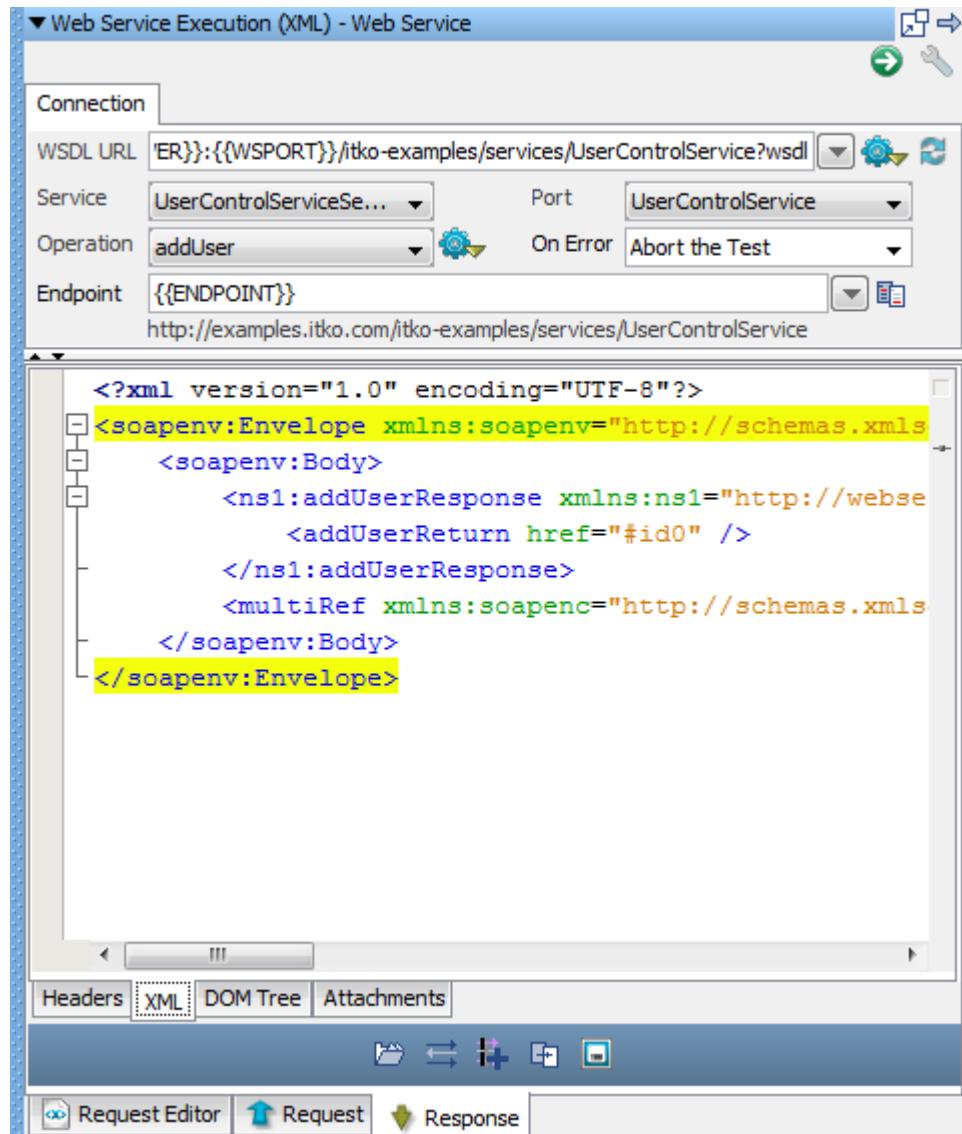
Follow these steps:

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 \(see page 271\)](#)
- [Step 2 - Add Database Properties to the Configuration \(see page 272\)](#)
- [Step 3 - Add a SQL Database Execution \(JDBC\) Test Step \(see page 272\)](#)
- [Step 4 - Connect to the Database \(see page 273\)](#)
- [Step 5 - Execute a SQL Query \(see page 274\)](#)
- [Step 6 - Add an Assertion \(see page 275\)](#)
- [Step 7 - Run the Test Case \(see page 277\)](#)
- [Step 8 - Change the Assertion \(see page 278\)](#)
- [Step 9 - Add a Filter \(see page 279\)](#)
- [Step 10 - Test the Filter and Assertion \(see page 280\)](#)
- [Tutorial 9 - Review \(see page 281\)](#)

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	<input type="checkbox"/>
DBConnect	jdbc:derby://localhost:1529/lisa-demo-server.db	<input type="checkbox"/>
DBUserID	sa	<input type="checkbox"/>
DBPwd	sa	<input type="checkbox"/>

4. Click **Save**.

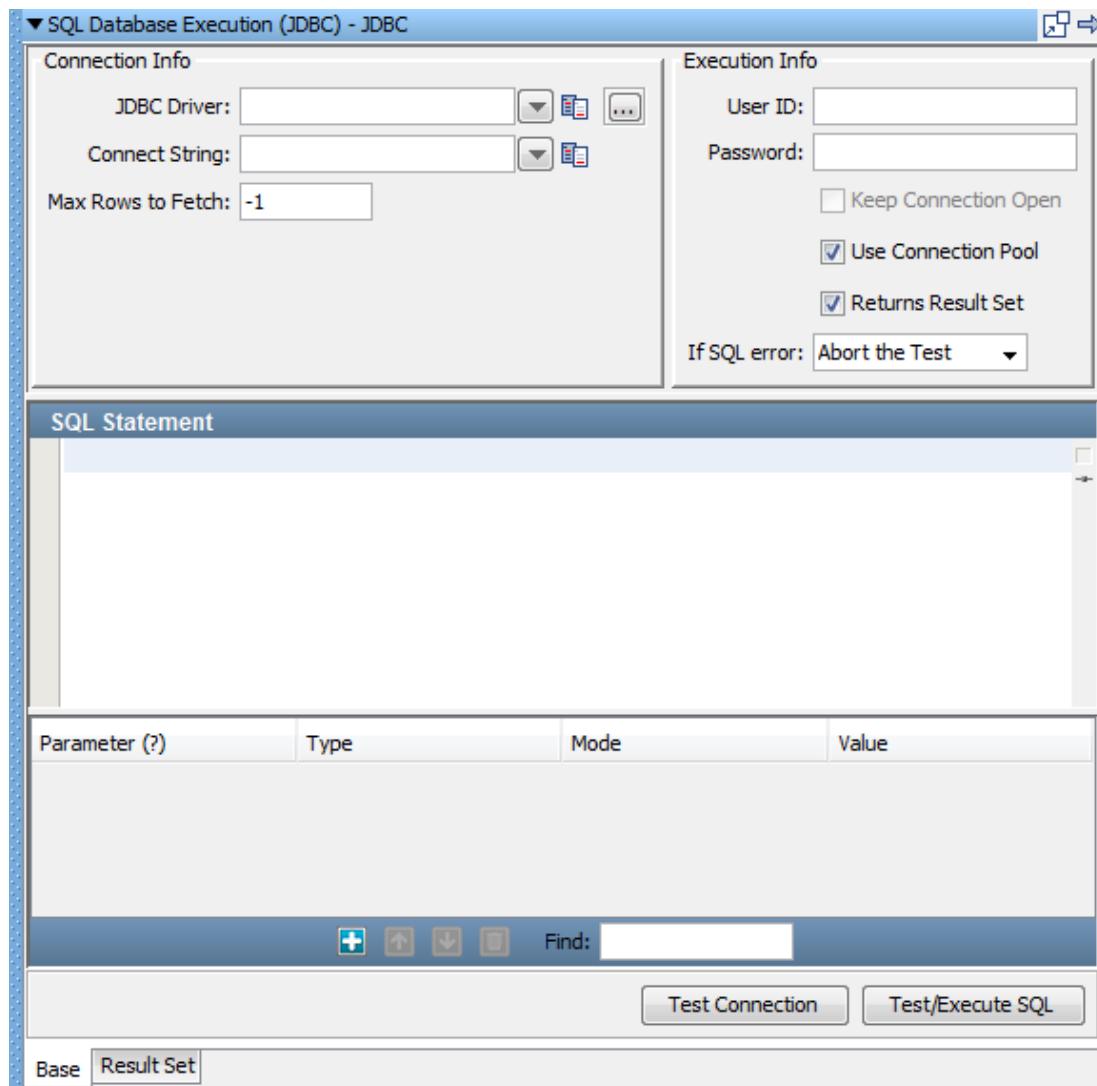
Step 3 - Add a SQL Database Execution (JDBC) Test Step

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

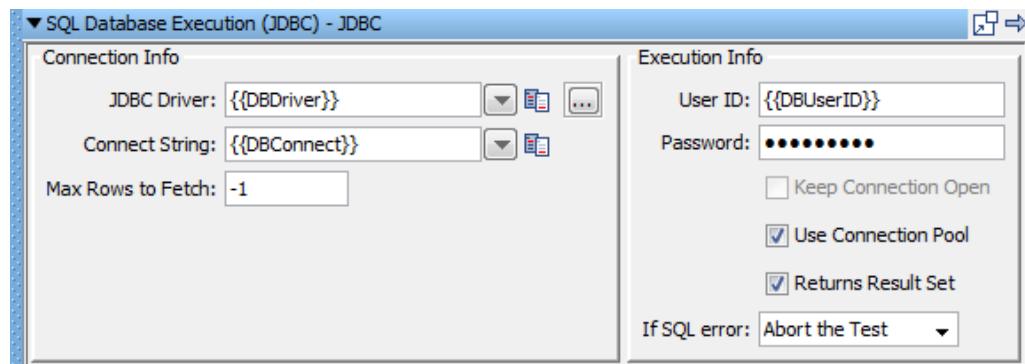
To provide the connection information, use the properties that you added to the project.config configuration.

Follow these steps:

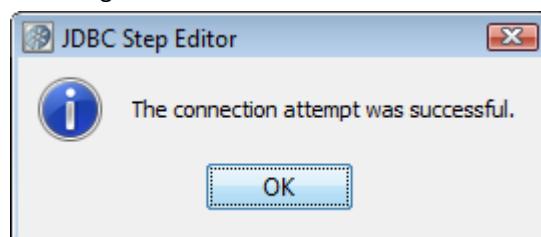
1. Enter the following values in the **Connection Info** and **Execution Info** areas of the step editor. Notice that when you enter the password, the value is masked.
 - **JDBC Driver**
{{DBDriver}}
 - **Connect String**
{{DBConnect}}

- **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**.

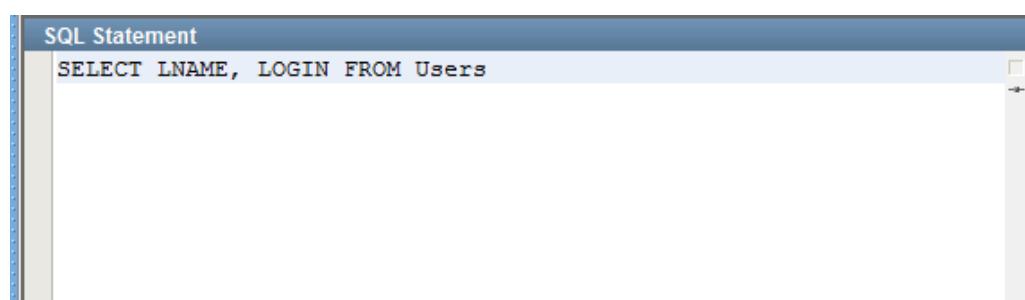
Step 5 - Execute a SQL Query

Now specify and run a SQL statement that retrieves data from the Users table.

Follow these steps:

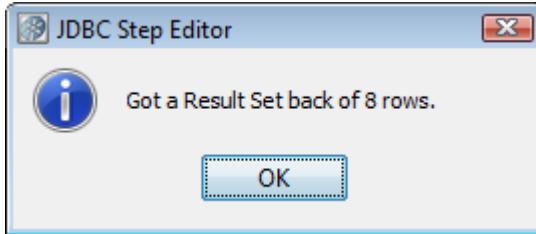
1. In the **SQL Statement** pane, enter the following statement:

```
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.

▼ SQL Database Execution (JDBC) - JDBC	
Result Set	
LNAME	LOGIN
Admin	User
Bellum	sbellum
Piece	wpiece
Reckonwith	areck
Rabbit	boaty
test	itko
simpson	lisa_simpson

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.

▼ SQL Database Execution (JDBC) - JDBC

Result Set

LNAME	LOGIN
	Lisa7
	webapp-1591155383
	webapp-274934681
	webapp-1046745658
Admin	admin
Bellum	sbellum
Piece	wpiece
Reckonwith	areck
Rabbit	boaty
test	itko
simpson	lisa_simpson

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.

The screenshot shows a 'Result Set' table with columns 'LNAME' and 'LOGIN'. The rows contain data for various users: Admin (User, admin), Bellum (User, sbellum), Piece (User, wpiece), Reckonwith, Rabbit, test, and simpson. A modal dialog titled 'Generate JDBC Result Set Value Assertion' is open over the table. It contains a dropdown menu with the option 'Fail the Test' selected. At the bottom are 'OK' and 'Cancel' buttons.

LNAME	LOGIN
	User
Admin	admin
Bellum	sbellum
Piece	wpiece
Reckonwith	
Rabbit	
test	
simpson	

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.

LNAME	LOGIN
Admin	User
Bellum	sbellum
Piece	wpiece
Reckonwith	areck
Rabbit	boaty
test	itko
simpson	lisa_simson

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:

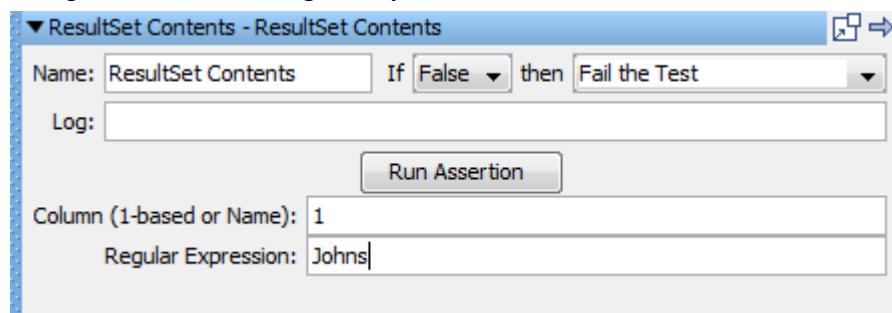
1. In the model editor, click the JDBC SELECT Users test step.
2. Open the **Assertions** tab in the **Element Tree**.

Screenshot of 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.

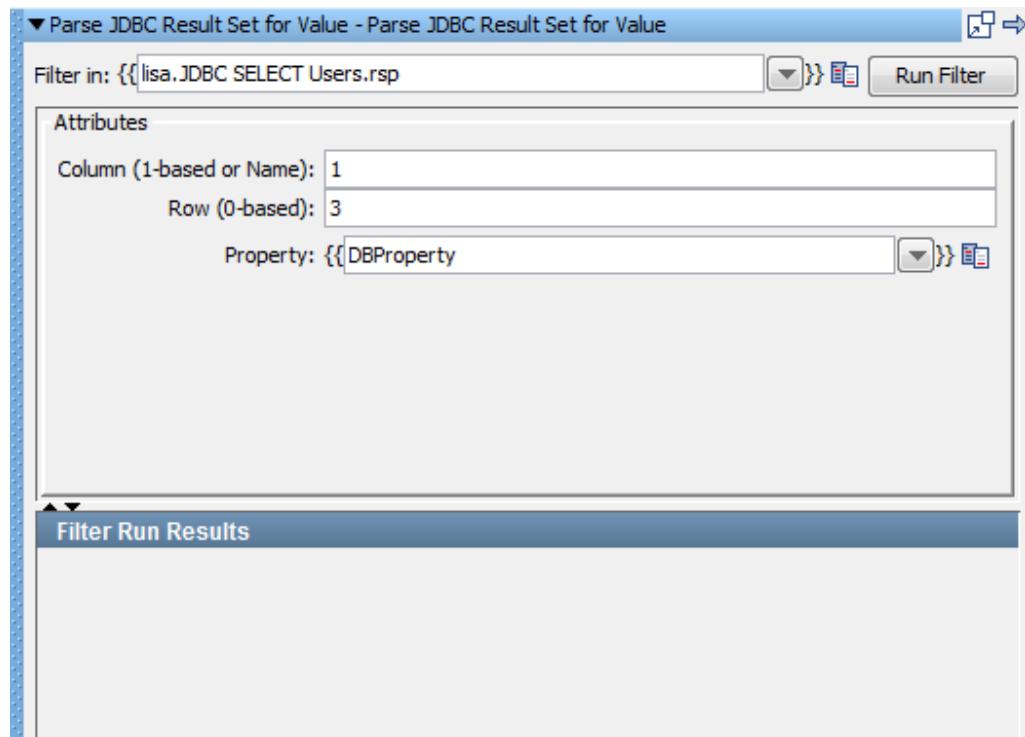
6. Retract the ITR tray.

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*.



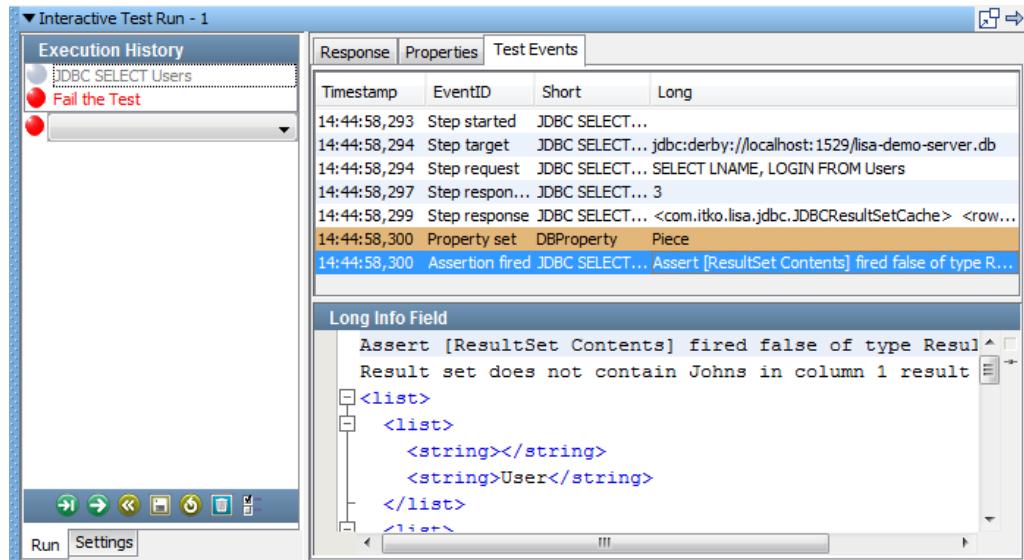
Screenshot for Parse JDBC Result Sset for Value dialog for Tutorial 9

8. Click **Save**.

Step 10 - Test the Filter and Assertion

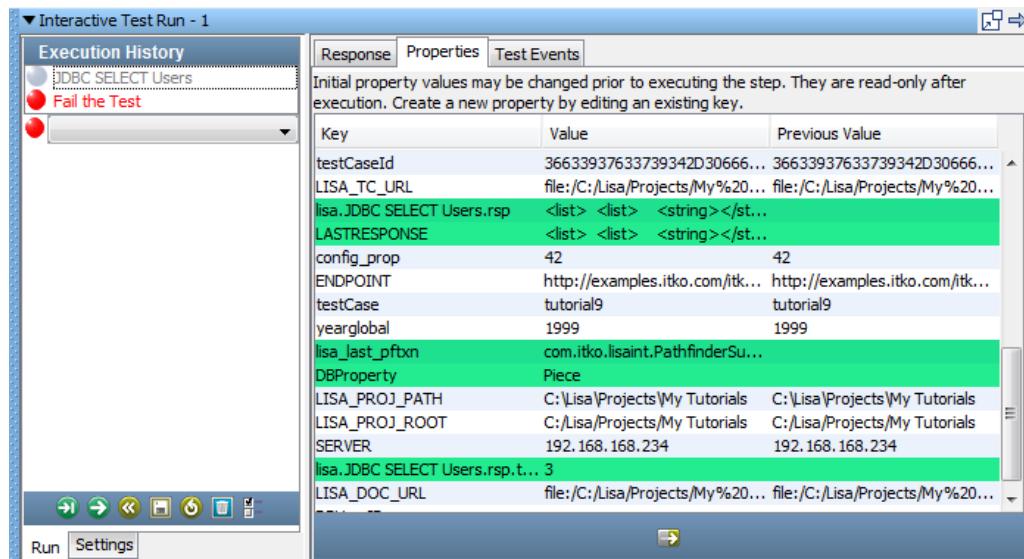
Follow these steps:

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 \(see page 282\)](#)
- [Step 2 - Review the Test Case \(see page 282\)](#)
- [Step 2 - Part A - Run a Quick Test \(see page 283\)](#)
- [Step 2 - Part B - View Generated Reports \(see page 286\)](#)
- [Tutorial 10 - Review \(see page 287\)](#)

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:

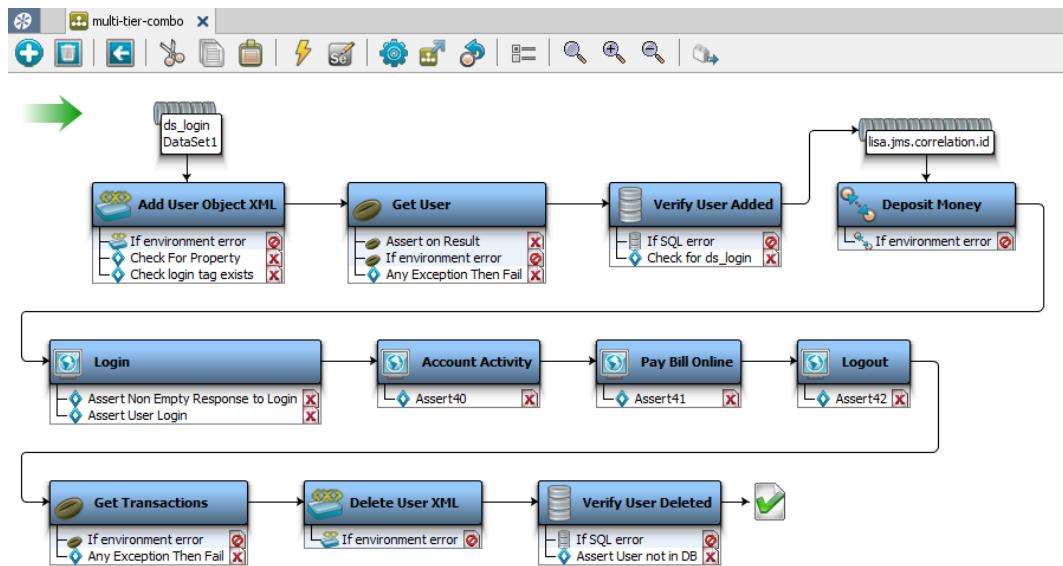
1. Select **File, Open, Test Case, File System** from the main menu.
2. Navigate to the LISA_HOME\examples\Tests folder.
3. Select multi-tier-combo and click **Open**.
The multi-tier-combo test case opens in the model editor.

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.



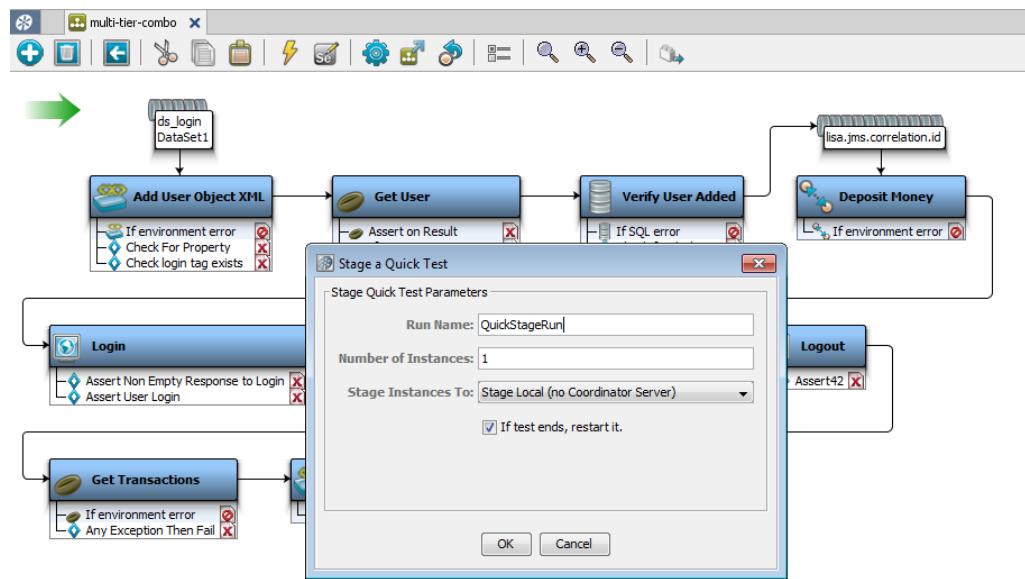
Screenshot of multi-tier-combo test case for Tutorial 9

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:

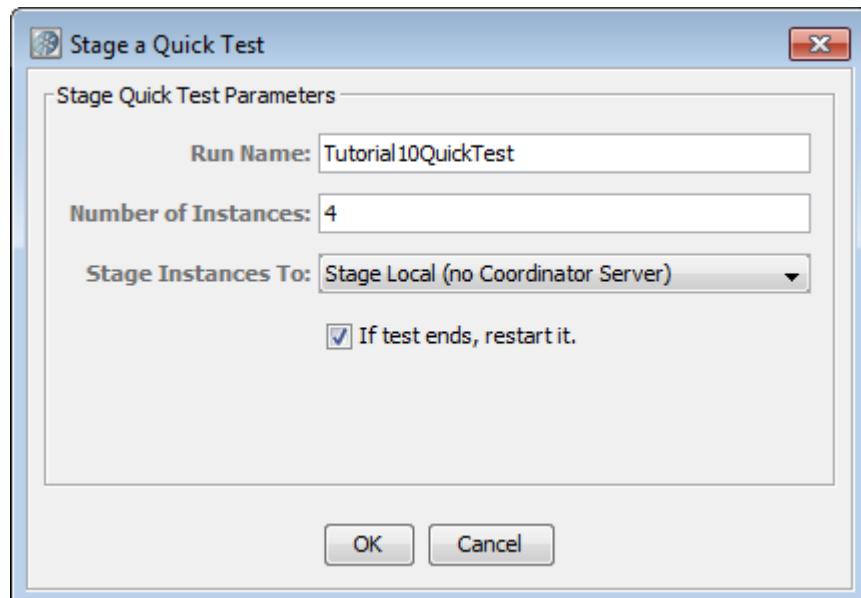
1. From the menu bar, click  **Stage a Quick Test** on the test case toolbar.
To stage a quick test, the example test case can be open in the model editor. Or, you can right-click on the test in the **Project** panel and can enter the parameters to stage a quick test from there.



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.**

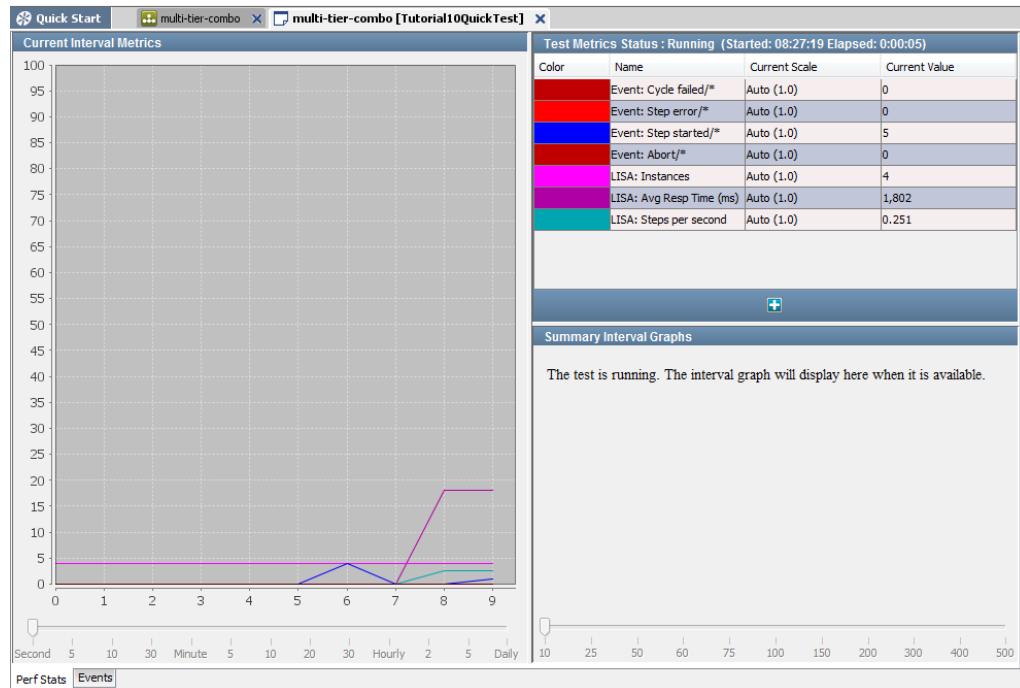


3. Click **OK**.

The **Test Run** window opens, but the test has not started yet.

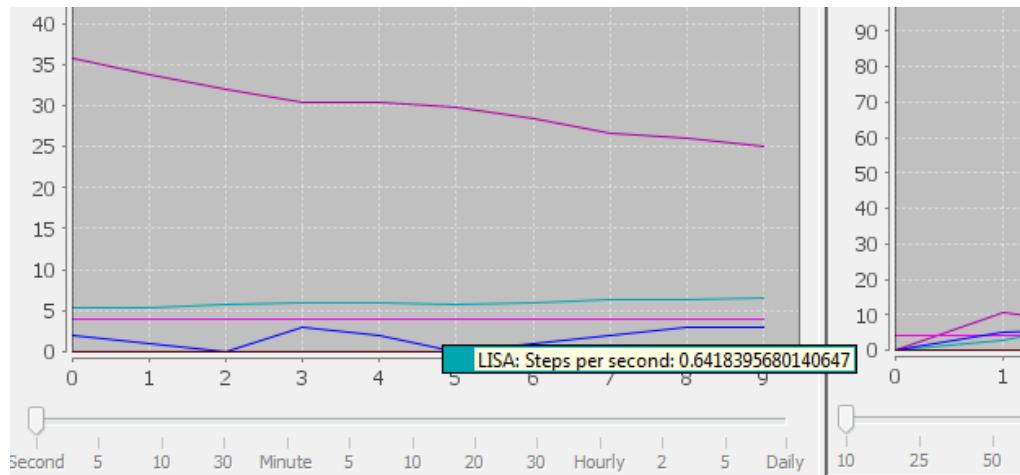


4. To start the test running, click from the main toolbar. The test begins, and the graph immediately plots results.



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.

Test Events						
Timestamp	Event	Simulator	Instance	Short Info	Long Info	
2011-08-24 08:30:14,208	Cycle ending	local	3/4	33646230386666382...	Signaled to stop test	
2011-08-24 08:30:14,205	Cycle started	local	3/4	33646230386666382...	Signaled to stop test	
2011-08-24 08:30:14,204	Cycle ending	local	3/3	63373534646437662...	Signaled to stop test	
2011-08-24 08:30:14,200	Cycle started	local	3/3	63373534646437662...	Signaled to stop test	
2011-08-24 08:30:14,186	Cycle ending	local	3/2	6564336646131302...	Signaled to stop test	
2011-08-24 08:30:08,813	Cycle started	local	0/3	37326661336161632...	Signaled to stop test	
2011-08-24 08:30:08,802	Cycle ending	local	0/2	66333631333066352...	Signaled to stop test	
2011-08-24 08:29:58,686	Cycle started	local	1/3	3636266616637642...	Signaled to stop test	
2011-08-24 08:29:58,669	Cycle ending	local	1/2	62326566656262312...	Signaled to stop test	
2011-08-24 08:29:49,418	Cycle started	local	2/3	38313633626131332...	Signaled to stop test	
2011-08-24 08:29:49,408	Cycle ending	local	2/2	38353061623466622...	Signaled to stop test	
2011-08-24 08:29:21,145	Cycle started	local	1/2	62326566556252312...	Signaled to stop test	
2011-08-24 08:29:21,106	Cycle ending	local	1/1	38626437346438662...	Signaled to stop test	
2011-08-24 08:29:19,388	Cycle started	local	0/2	66333631333066352...	Signaled to stop test	
2011-08-24 08:29:19,381	Cycle ending	local	0/1	36303636373132662...	Signaled to stop test	
2011-08-24 08:29:18,643	Cycle started	local	3/2	6564336646131302...	Signaled to stop test	
2011-08-24 08:29:18,627	Cycle ending	local	3/1	33616264346331392...	Signaled to stop test	
2011-08-24 08:29:10,236	Cycle started	local	2/2	38353061623466622...	Signaled to stop test	
2011-08-24 08:28:31,687	Cycle started	local	1/1	6234643538393352...	Signaled to stop test	
2011-08-24 08:28:31,676	Cycle ending	local	1/0	35313839653361332...	Signaled to stop test	
2011-08-24 08:28:30,525	Cycle started	local	0/1	36303636373132662...	Signaled to stop test	
2011-08-24 08:28:30,518	Cycle ending	local	0/0	39623863346431322...	Signaled to stop test	
2011-08-24 08:28:27,594	Cycle started	local	3/1	33616264346331392...	Signaled to stop test	
2011-08-24 08:28:27,565	Cycle ending	local	3/0	38663362643239312...	Signaled to stop test	
2011-08-24 08:28:26,180	Cycle started	local	2/1	6234643538393352...	Signaled to stop test	
2011-08-24 08:28:26,167	Cycle ending	local	2/0	3331626253235392...	Signaled to stop test	
2011-08-24 08:27:19,731	Cycle started	local	2/0	3331626253235392...	Signaled to stop test	
2011-08-24 08:27:19,729	Cycle started	local	0/0	39623863346431322...	Signaled to stop test	
2011-08-24 08:27:19,730	Cycle started	local	1/0	35313839653361332...	Signaled to stop test	
2011-08-24 08:27:19,729	Cycle started	local	3/0	38663362643239312...	Signaled to stop test	
2011-08-24 08:27:19,706	Test started	N/A	0/0	33366264383131302...	N/A	

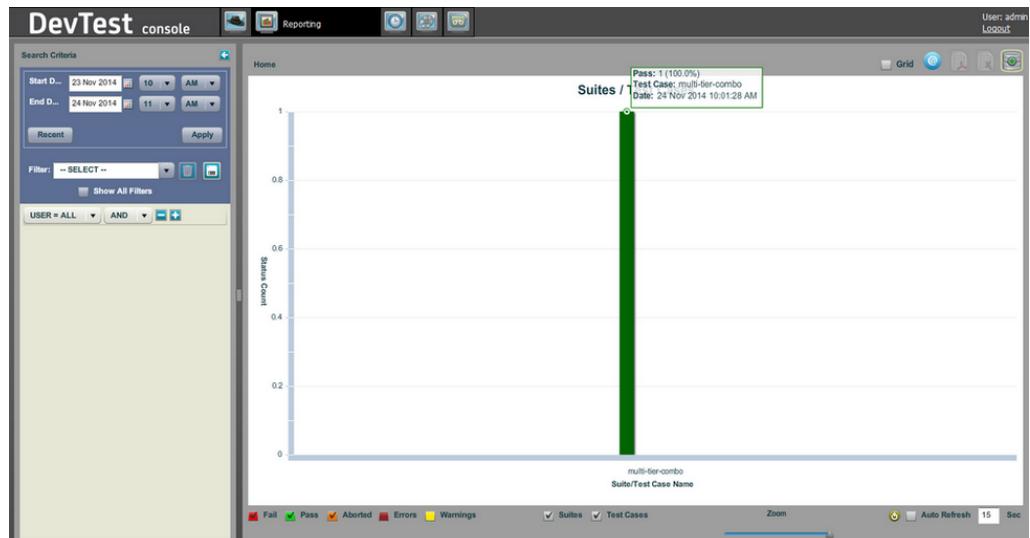
Screenshot of Events Tab after running Quick Test

Step 2 - Part B - View Generated Reports

DevTest provides a report viewer to view reports.

Follow these steps:

- Click **View, Reporting Console** from the main menu, or click **Reports** on the toolbar. The **Report Viewer** opens.



Screenshot of Report Viewer for Tutorial 10

2. To see only recent activity, click **Recent**.

The graph in the previous graphic shows that all of the tests in this test case passed.

You can right-click the graph for different menus. For more information about reports, see [Reports \(see page 598\)](#).

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.

Mobile Testing Tutorials

This section contains tutorials that illustrate various aspects of mobile testing.

Prerequisites for each tutorial

- You have installed DevTest Workstation and entered your DevTest Solutions license credentials.
- You have completed all of the tasks that are described in [Setting Up the Mobile Testing Environment \(see page 148\)](#).
- For *Tutorial 1 - Record an iOS Test Case*, download the [UICatalog.app.zip](https://www.dropbox.com/s/h2njkxmqm1o1k6m/UICatalog.app.zip?dl=0) (<https://www.dropbox.com/s/h2njkxmqm1o1k6m/UICatalog.app.zip?dl=0>) tutorial application to your local computer and unzip it.
- For *Tutorial 2 - Record an Android Test Case*, download the [ApiDemos.apk](https://support.ca.com/cadocs/0/CA%20DevTest%20Solutions%208%200-ENU/Bookshelf_Files/Mobile/ApiDemos.apk) (https://support.ca.com/cadocs/0/CA%20DevTest%20Solutions%208%200-ENU/Bookshelf_Files/Mobile/ApiDemos.apk) tutorial application to your local computer.

This section contains the following pages:

- [Tutorial 1 - Record an iOS Test Case \(see page 287\)](#)
- [Tutorial 2 - Record an Android Test Case \(see page 292\)](#)

Tutorial 1 - Record an iOS Test Case

Contents

- [Step 1 - Start DevTest Workstation \(see page 288\)](#)
- [Step 2 - Create a Project \(see page 288\)](#)
- [Step 3 - Create a Config File for an iOS Simulator \(see page 289\)](#)
- [Step 4 - Create an iOS Simulator Asset \(see page 289\)](#)

- [Step 5 - Record a Test Case \(see page 290\)](#)
- [Step 6 - Run the Test Case in the ITR \(see page 291\)](#)

This tutorial illustrates each of the steps that are required to record a test case using an iOS simulator.

In this tutorial, you perform the following tasks:

- Create an asset
- Record a mobile test case
- Run the test case in the ITR

Step 1 - Start DevTest Workstation

Follow these steps:

1. Start the registry.
 - For DevTest Server:
 - a. Click **Start, All Programs, DevTest Solutions, Registry**.
 - b. Wait until the DevTest Registry Ready message appears.
 - For DevTest Workstation, use a registry that is running on another computer.
2. Click **Start, All Programs, DevTest Solutions, DevTest Workstation**.

The Set DevTest Registry dialog opens.

3. Select a registry and click **OK**.

Step 2 - Create a Project

The project that you create holds all the test case example files that are required for the mobile tutorials.



Note: If you created the MyMobileTutorials project in a previous tutorial, skip to [Step 3, Create a Config File for an iOS Simulator](#).

Follow these steps:

1. From the DevTest Workstation main menu, select **File, New, Project**.

The Create New Project dialog appears.

2. In the Project Name field, type **MyMobileTutorials**.

3. Accept the default setting to create the project in the LISA_HOME directory.
4. Click **Create**.

The MyMobileTutorials project is created.

Step 3 - Create a Config File for an iOS Simulator

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 create a configuration file that contains asset information specific to using an iOS simulator.

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: **MobileiOSimulator**.
3. Click **OK**.
The MobileiOSimulator config file opens in the properties editor.
4. Right-click the MobileiOSimulator config file in the **Project** panel and select **Make Active**.

Step 4 - Create an iOS Simulator Asset

A Simulator Session asset that is associated with a specific config lets you specify the mobile simulator that is used for testing.

Follow these steps:

1. Open the MobileiOSimulator configuration file that you created in Step 3 - [Create a Config File for an iOS Simulator \(see page 289\)](#) (if it is not already open).
2. In the [Asset Browser \(see page 399\)](#), click **Add**  at the bottom of the pane.
3. Click **Mobile Session**.
The Mobile Session dialog displays.



Note: If you have already defined values for any of these fields, you can select the value from the list for that field. To add or remove values from the list, select **Maintain List**.

4. Enter the following fields:

- **Name**

Enter **iOSSimulator**.

- **Description**

Enter **iOS Simulator for use with Mobile Tutorial 1**.

■ **Platform**

Select **iOS** from the drop-down menu.

■ **Application**

Navigate to and select the **UICatalog.app** file that you downloaded.



Note: All iOS apps must be code signed and provisioned to launch on a device. See [Application Signing \(see page 655\)](#) for more information.

■ **Family**

Select **iPhone only** from the drop-down menu.

■ **Target**

Select **Simulator** from the drop-down menu.

■ **Simulator**

Navigate to and select the iOS simulator on your machine.

■ **iOS Version**

Select 6.1.

5. (Optional) To verify the asset (see page 404) before saving, click .
6. Click **OK**.
The new iOS Simulator asset appears in the Asset Browser.
7. Click Save.

Step 5 - Record a Test Case

Follow these steps:

1. Ensure the MobileiOSSimulator config file is active.
2. Create a test case named **Tutorial_iOS_Simulator**:
 - a. Right-click the **Tests** folder in the **Project** panel and click **Create New Test Case**.
 - b. In the dialog, browse to the directory where you plan to store the test case.
 - c. Enter the name of the new test case.
 - d. Click **Save**.
 - e. Click **Create steps by recording or templating** .
3. Click **Record Test Case for User Interface, Mobile Recorder**.
The mobile test recorder and the mobile simulator windows open.

4. If you defined multiple mobile assets, select an asset to connect to using the **Choose Mobile asset** drop-down list, then click **OK**.
5. Click **Start Recording** at the bottom of the recorder window.
You are now recording.
6. Perform the actions that you want to record for your test case in the recorder window.



Note: Do not perform these actions in the mobile simulator directly. All actions that are performed in the recorder window are sent to the simulator automatically.

7. Click **Stop Recording** when you have captured all of the actions for your test. The recorder window and the mobile simulator close. Your new test case is populated with test steps that represent the mobile actions that are captured during the recording. Each test step represents the actions that you performed on a specific screen in the application.
8. To view the details of each step:
 - a. Click the test step to review.
 - b. Click **Mobile testing step** in the element tree on the right to expand the details for the step.
The Mobile Testing Step tab opens and shows a screenshot of the test application. The Actions section at the top of the tab shows the individual actions that are performed in the test step.
 - c. To view the screenshot associated with a specific action, click the action in the Actions section.
For more information about modifying a recorded test step, see [Modify Mobile Test Steps \(see page 1868\)](#).
For more information about adding assertions to your test step, see [Add an Assertion to a Mobile Test Step \(see page 430\)](#).

Step 6 - Run the Test Case in the ITR

Follow these steps:

1. Open the test case from **Step 5 - Record a Test Case** (if it is not already open).



2. Click **ITR** on the toolbar.



Note: Select whether to start a new ITR run or open a previous ITR run (if you have run the ITR before).

The Interactive Test Run window opens.

3. Click **Execute**  at the bottom of the ITR window.

The mobile simulator window opens and DevTest runs the test steps. The mobile application is visible on the simulator while the test is running.

When the test is complete, a message opens indicating the test is complete. The simulator window closes.

4. Click **OK**.

For more information about using the ITR, see [Running Test Cases and Suites \(see page 533\)](#).

Tutorial 2 - Record an Android Test Case

Contents

- [Step 1 - Start DevTest Workstation \(see page 292\)](#)
- [Step 2 - Create a Project \(see page 293\)](#)
- [Step 3 - Create a Config File for an Android Simulator \(see page 293\)](#)
- [Step 4 - Create an Android Emulator Asset \(see page 293\)](#)
- [Step 5 - Record a Test Case \(see page 294\)](#)
- [Step 6 - Run the Test Case in the ITR \(see page 295\)](#)

This tutorial illustrates each of the steps that are required to record a test case using an Android simulator.

In this tutorial, you perform the following tasks:

- Create an asset
- Record a mobile test case
- Run the test case in the ITR

Step 1 - Start DevTest Workstation

Follow these steps:

1. Start the registry.
 - For DevTest Server:
 - a. Click **Start, All Programs, DevTest Solutions, Registry**.
 - b. Wait until the DevTest Registry Ready message appears.
 - For DevTest Workstation, use a registry that is running on another computer.
2. Click **Start, All Programs, DevTest Solutions, DevTest Workstation**.

The Set DevTest Registry dialog opens.

3. Select a registry and click **OK**.

Step 2 - Create a Project

The project that you create holds all the test case example files that are required for the mobile tutorials.



Note: If you created the MyMobileTutorials project in a previous tutorial, skip to **Step 3, Create a Config File for an Android Simulator**.

Follow these steps:

1. From the DevTest Workstation main menu, select **File, New, Project**.

The Create New Project dialog appears.

2. In the Project Name field, type **MyMobileTutorials**.
3. Accept the default setting to create the project in the LISA_HOME directory.
4. Click **Create**.

The MyMobileTutorials project is created.

Step 3 - Create a Config File for an Android Simulator

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 create a configuration file that contains asset information specific to using an iOS simulator.

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: **MobileAndroidSimulator**.
3. Click **OK**.
The MobileAndroidSimulator config file opens in the properties editor.
4. Right-click the MobileAndroidSimulator config file in the **Project** panel and select **Make Active**.

Step 4 - Create an Android Emulator Asset

A Simulator Session asset that is associated with a specific config lets you specify the mobile simulator that is used for testing.

Follow these steps:

1. Open the MobileAndroidSimulator configuration file that you created in Step 3 - [Create a Config File for an Android Simulator \(see page 293\)](#) (if it is not already open).
2. In the [Asset Browser \(see page 399\)](#), click **Add**  at the bottom of the pane.
3. Click **Mobile Session**.
The Mobile Session dialog displays.
4. Define the following fields:
 - **Name**
Enter **AndroidSimulator**.
 - **Description**
Enter **Android Simulator for use with Mobile Tutorial 2**.
 - **Platform**
Select **Android** from the drop-down menu.
 - **Application**
Navigate to and select the **ApiDemos.apk** file that you downloaded.
 - **Target**
Select **Emulator** from the drop-down menu.
 - **AVD**
Enter the name of the Android AVD that you defined when [configuring mobile testing \(see page 293\)](#).
 - **SDK Version**
Select **4.2.2**.
5. Click  to [verify the asset \(see page 404\)](#).
6. Click **OK**.
The new Android Simulator asset appears in the Asset Browser.
7. Click Save.

Step 5 - Record a Test Case

Follow these steps:

1. Ensure the MobileAndroidSimulator config file is active.
2. Create a test case named **Tutorial_Android_Simulator**.
 - a. Right-click the **Tests** folder in the **Project** panel and click **Create New Test Case**.
 - b. In the dialog, browse to the directory where you plan to store the test case.

- c. Enter the name of the new test case.
- d. Click **Save**.

3. Click **Create steps by recording or templating** .
4. Click **Record Test Case for User Interface, Mobile Recorder**.
The mobile test recorder and the mobile simulator windows open.
5. If you defined multiple mobile assets, select an asset to connect to using the **Choose Mobile asset** drop-down list, then click **OK**.
6. Click **Start Recording** at the bottom of the recorder window.
You are now recording.
7. Perform the actions that you want to record for your test case in the recorder window.



Note: Do not perform these actions in the mobile simulator directly. All actions that are performed in the recorder window are sent to the simulator automatically.

8. Click **Stop Recording** when you have captured all of the actions for your test.
The recorder window and the mobile simulator close. Your new test case is populated with test steps that represent the mobile actions that are captured during the recording. Each test step represents the actions that you performed on a specific screen in the application.
9. To view the details of each step:
 - a. Click the test step to review.
 - b. Click **Mobile testing step** in the element tree on the right to expand the details for the step.
The Mobile Testing Step tab opens and shows a screenshot of the test application. The Actions section at the top of the tab shows the individual actions that are performed in the test step.
 - c. To view the screenshot associated with a specific action, click the action in the Actions section.
For more information about modifying a recorded test step, see [Modify Mobile Test Steps \(see page 1868\)](#).
For more information about adding assertions to your test step, see [Add an Assertion to a Mobile Test Step \(see page 430\)](#).

Step 6 - Run the Test Case in the ITR

Follow these steps:

1. Open the test case from **Step 5 - Record a Test Case** (if it is not already open).



2. Click **ITR**  on the toolbar.



Note: Select whether to start a new ITR run or open a previous ITR run (if you have run the ITR before).

The Interactive Test Run window opens.

3. Click **Execute**  at the bottom of the ITR window.

The mobile simulator window opens and DevTest runs the test steps. The mobile application is visible on the simulator while the test is running.

When the test is complete, a message opens indicating the test is complete. The simulator window closes.

4. Click **OK**.

For more information about using the ITR, see [Running Test Cases and Suites \(see page 533\)](#).

CA Service Virtualization Tutorial

In this 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 \(see page 296\)](#)
- [Process for Creating and Testing a VSI \(see page 297\)](#)

Prerequisites

The following steps are prerequisites to completing this tutorial:

- DevTest Workstation and VSE are installed.

- You have reviewed the following pages:
 - [Glossary \(see page 309\)](#)
 - [Using CA Service Virtualization \(see page 661\)](#)
 - [Understanding CA Service Virtualization \(see page 677\)](#)

Process for Creating and Testing a VSI

After you complete the prerequisites, complete the following steps:

- [Step 1 - Start DevTest Workstation \(see page 297\)](#)
- [Step 2 - Start VSE \(see page 298\)](#)
- [Step 3 - Start Demo Server \(see page 298\)](#)
- [Step 4 - Run a Test Case \(see page 298\)](#)
- [Step 5 - Create a Configuration File \(see page 299\)](#)
- [Step 6 - Activate the Configuration File \(see page 300\)](#)
- [Step 7 - Configure the VSE Recorder \(see page 300\)](#)
- [Step 8 - Record the Test Case \(see page 302\)](#)
- [Step 9 - Deploy the Virtual Service Model \(see page 304\)](#)
- [Step 10 - Test Against the Virtual Service Model \(see page 305\)](#)
- [Review the Tutorial \(see page 306\)](#)

Step 1 - Start DevTest Workstation

Follow these steps:

1. Ensure that the registry is running.
 - If your computer has DevTest Server installed:
 - Start the Enterprise Dashboard by clicking **Start Menu, All Programs, DevTest Solutions, EnterpriseDashboard**. Wait until the "Enterprise Dashboard started" message appears.
 - Start the registry by clicking **Start Menu, All Programs, DevTest Solutions, Registry**.
 - If your computer has DevTest Workstation installed, use a registry that is running on another computer.

2. Click **Start, All Programs, DevTest Solutions, Workstation**.
The **Set DevTest Registry** dialog opens.
3. Select a registry and click **OK**.
4. The **Login** dialog opens. Enter a valid username and password and click **Login**.
5. Continue with "[Step 2 - Start VSE \(see page 298\)](#)".

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 \(see page 298\)](#)".

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. Click **Start, All Programs, DevTest Solutions, DevTest Demo Server**.
When the "Started in XXs:YYms" line appears, the demo server has started.
2. Continue with "[Step 4 - Run a Test Case \(see page 298\)](#)".

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 \(see page 299\)](#)".

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 (see page 300)".

[Next Step \(see page 300\).](#)

Step 6 - Activate Config File

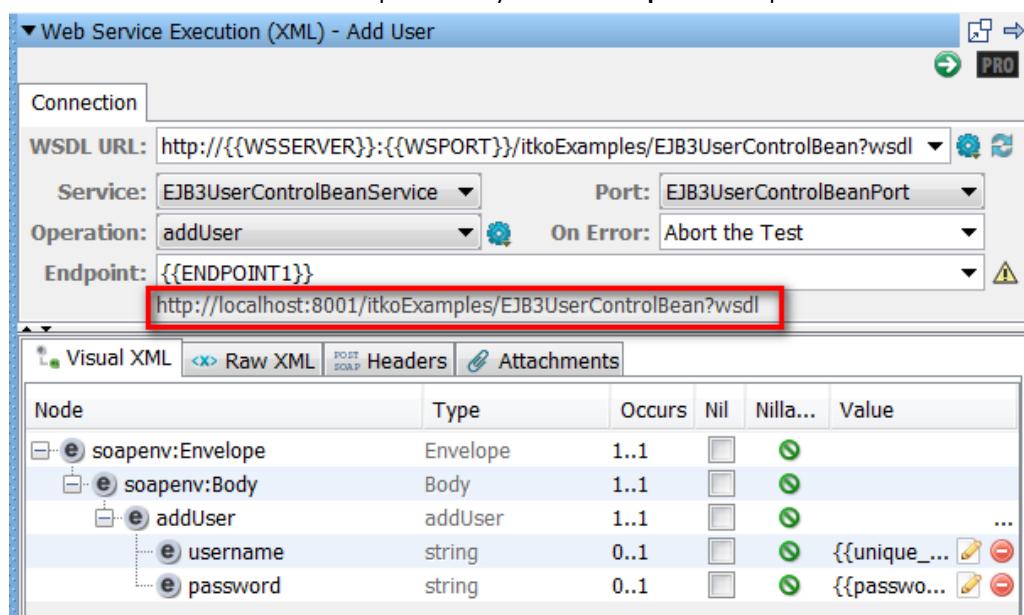
For DevTest to use the newly created config file as the primary or active config file, activate it.

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

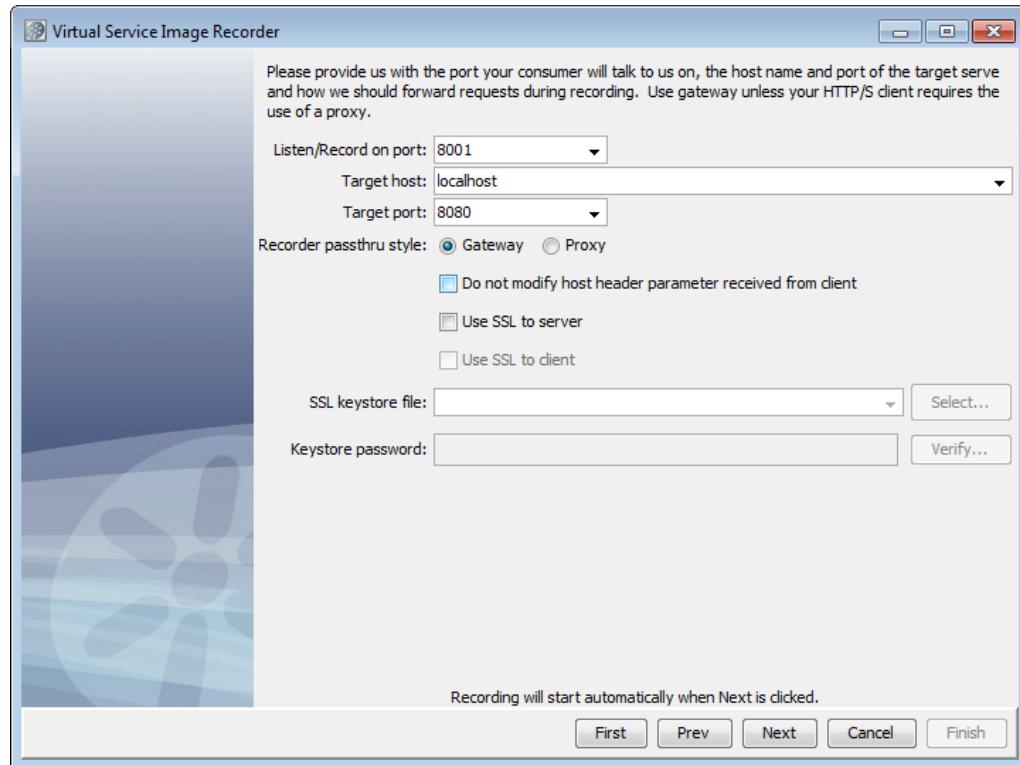
3. Continue with "Step 7 - Configure the VSE Recorder (see page 300)".

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 \(see page 302\)](#)".

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.

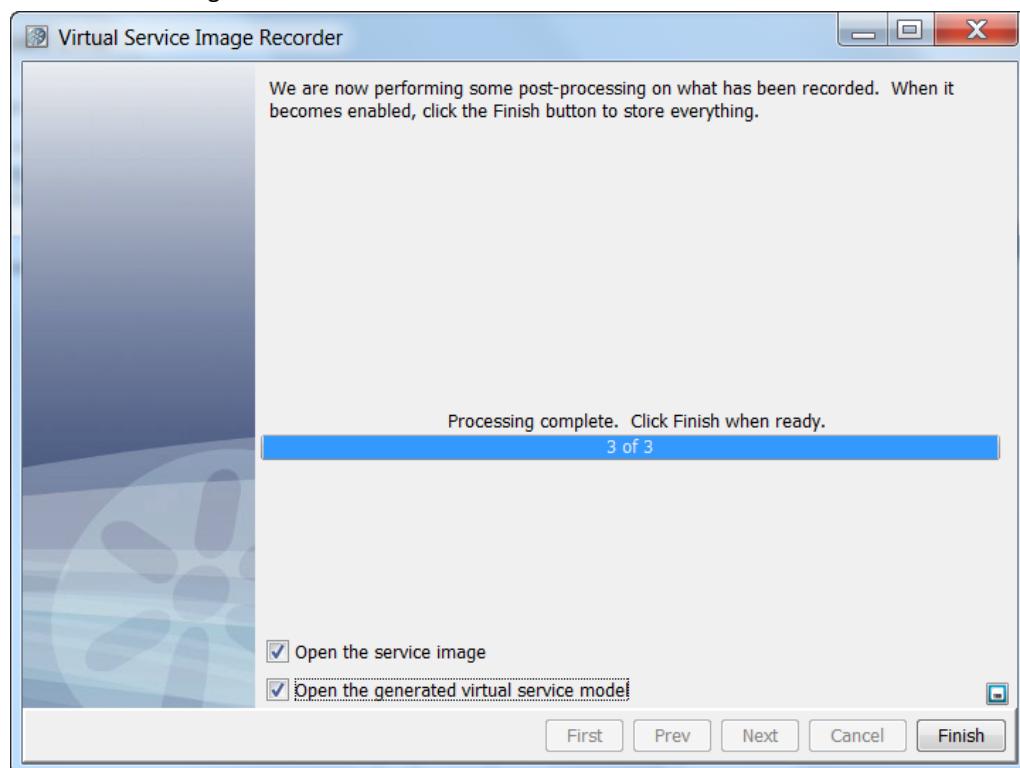
4. Click **Next**.

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 \(see page 304\)](#)".

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 **Server Console**, which provides the VSE UI for managing virtual services.
3. Click VSE in the left navigation pane.
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 VSE dashboard.
The virtual service is deployed to VSE.
5. To return to the VSE Console, click the DevTest Console tab in your browser.

The VSETutorial virtual service model appears in the **Services** tab.

The screenshot shows the DevTest Workstation interface with the 'Services' tab selected in the VSE Console. The table lists one service:

Name	Resource / Type	Status	Up-Time	Txn Count	Model Behavior	Group	Errors
VSETutorial	8001 : http://tkoExamples/EJB3UserControlBean	Running	0:03:55	0	Most Efficient		

Below the table, the 'Virtual Service Details' section for VSETutorial shows the following configuration:

Model Name :	VSETutorial	Config Name :	C:\Users\lhoan02\lisatmp_8.0.0\\ads\6563BD784F0A11E48EA55C260A7ECB81\examples\Configs\VSRecorder.config
Execution mode :	Most Efficient	Auto-Restart :	<input checked="" type="checkbox"/>
Last Start :	Oct 08, 2014 11:44:41	Last End :	
Transaction Count :	0	Error Count :	0
Current bxs/s :	0	Peak bxs/s :	0
Capacity :	1	Think Scale :	100
Group Tag :			

Screenshot of the VSETutorial virtual service model appears in the Services tab of the VSE Console.

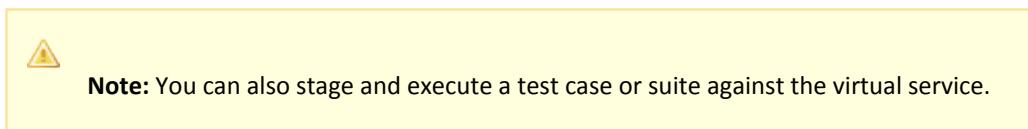
The virtual service is started and ready to process transactions, which are tracked in the **Txn Count** column. The service initially shows that it has processed 0 transactions.

6. Run the test case and return to this window.
7. Continue with "[Step 10 - Test Against the VSM \(see page 305\)](#)".

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.



Name	Resource / Type	Status	Up-Time	Txn Count	Model Behavior	Group	Errors
VSETutorial	8001 : http://itkoExamples/EJB3UserControlBean	Running	0:09:40	3	Most Efficient		

Virtual Service Details : VSETutorial : Virtual HTTP/S Listener	
Model Name :	VSETutorial
Execution mode :	Most Efficient
Last Start :	Oct 08, 2014 11:44:41
Transaction Count :	3
Current txns :	0
Capacity :	1
Group Tag :	
Config Name :	C:\Users\rhoan02\lisatmp_8.0.0\lads6563BD784FDA11E48EA55C260A7ECB81\examples\Configs\VSSRecorder.config
Auto-Restart :	<input checked="" type="checkbox"/>
Last End :	
Error Count :	0
Peak txns :	1
Think Scale :	100

VSE Console Services tab with VSETutorial service showing 3 transactions.

3. 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 VSE Dashboard after running the test case with 10 instances, you see the transaction count is 33.

Name	Resource / Type	Status	Up-Time	Txn Count
VSETutorial	8001 : http://itkoExamples/EJB3UserControlBean	Running	0:12:45	33

VSE Console Services tab with VSETutorial service showing 33 transactions

7. Continue with "[Review the Tutorial \(see page 306\)](#)".

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

CA Continuous 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 the DevTest Portal.

Prerequisites

- The following components are running: Enterprise Dashboard Server, registry, demo server, broker, and portal.

Contents

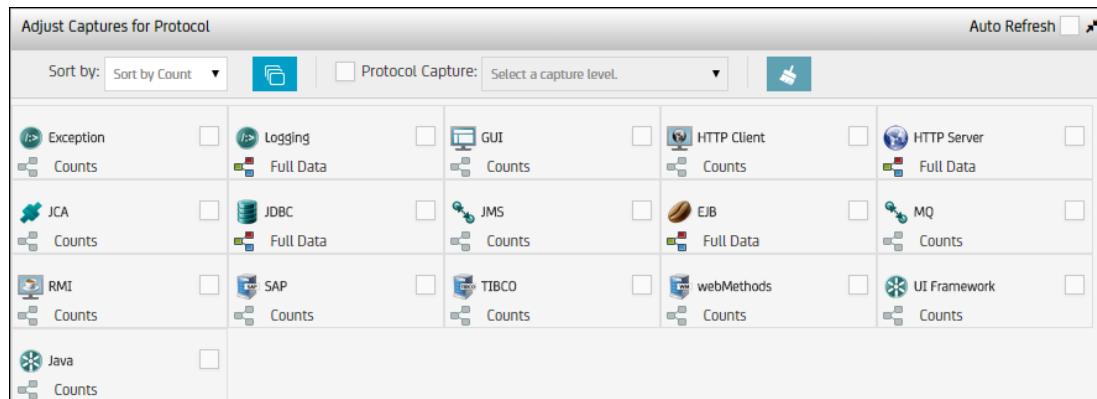
- [Step 1 - Increase the Default Capture Levels \(see page 306\)](#)
- [Step 2 - Generate Transactions from the Demo Application \(see page 307\)](#)
- [Step 3 - View the Transactions in the Analyze Transactions Window \(see page 308\)](#)

Step 1 - Increase the Default Capture Levels

Each protocol that the DevTest Java Agent can capture has a capture level.

The default capture level is **Counts**. To view the transactions 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 the 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**
6. To change a capture level, select the check box of the protocol and select **Full Data** from the capture level drop-down list.

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. The left side contains buttons for various actions that you can perform.
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 lets you do the following tasks:

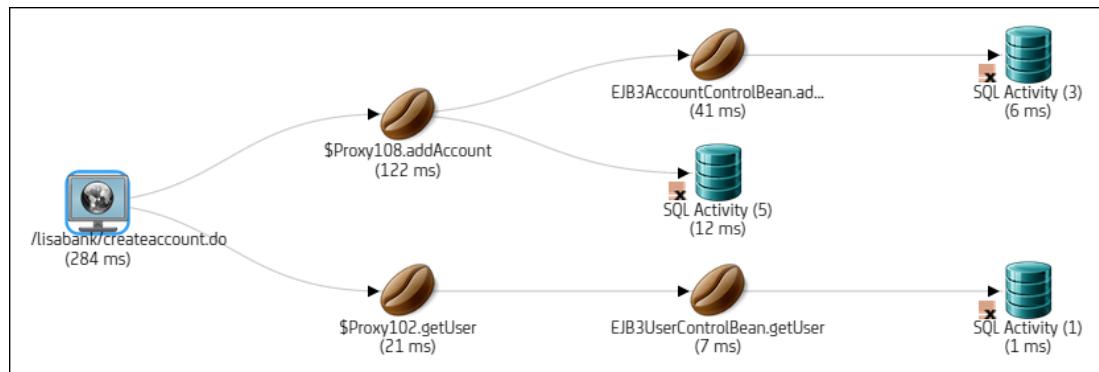
- View transactions
- Search and filter transactions
- View component details

The following graphic shows a set of transactions in the **Analyze Transactions** window. The transactions are displayed in the **List** view. The transactions appear in reverse chronological order.

Name	Start Time	Wall Time	CPU Time	Agent	Actions
/lisabank/buttonclick.do	09/01/2015 2:57:40 PM 574	99 ms	109 ms	JBoss_LISABank	
/lisabank/createaccount.do	09/01/2015 2:57:36 PM 645	284 ms	218 ms	JBoss_LISABank	
/lisabank/buttonclick.do	09/01/2015 2:57:20 PM 242	262 ms	234 ms	JBoss_LISABank	
/lisabank/login.do	09/01/2015 2:57:07 PM 432	916 ms	670 ms	JBoss_LISABank	
/lisabank/home.do	09/01/2015 2:56:48 PM 118	1334 ms	1014 ms	JBoss_LISABank	
/lisabank/	09/01/2015 2:56:46 PM 989	1122 ms	1029 ms	JBoss_LISABank	
/lisabank/	09/01/2015 2:56:46 PM 951	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.

Follow these steps:

1. Return to the DevTest Portal.
2. Select **Application Insight, Analyze Transactions** from the left navigation menu.
The **Analyze Transactions** window opens.
3. Locate the transaction that has the name **/lisabank/createaccount.do**.
4. Click the button in the **Actions** column.
The transactions details 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 of the tabs.
7. Close the dialog.

Glossary

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 \(see page 419\)](#).

asset

An *asset* is a set of configuration properties that are grouped into a logical unit. For more information, see [Assets \(see page 398\)](#).

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 \(see page 510\)](#).

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 \(see page 441\)](#).

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 model 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 \(see page 393\)](#).

Continuous Service Validation (CVS) Dashboard

The *Continuous Validation Service (CVS) Dashboard* lets you schedule test cases and test suites to run regularly, over an extended time period. For more information, see [Continuous Validation Service \(CVS\) \(see page 598\)](#).

conversation tree

A *conversation tree* is a set of linked nodes that represent conversation paths for the stateful transactions in a virtual service image. 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 \(see page 661\)](#).

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 \(see page 321\)](#).

data protocol

A *data protocol* is also known as a data handler. In CA Service Virtualization, it is responsible for handling the parsing of requests. Some transport protocols allow (or require) a data protocol to which the job of creating requests is delegated. As a result, the protocol has to know the request payload. For more information, see [Using Data Protocols \(see page 863\)](#).

data set

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. Data sets provide a mechanism to introduce external test data into a test case or virtual service model. 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 \(see page 433\)](#).

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 \(see page 969\)](#).

event

An *event* is a message about an action that has occurred. You can configure events at the test case or virtual service model level. For more information, see [Understanding Events \(see page 513\)](#).

filter

A *filter* is an element that runs before and after a step. A filter gives you the opportunity to process the data in the result, or store values in properties. Global filters apply to each step in a test case or virtual service model. For more information, see [Filters \(see page 405\)](#).

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 the VSE Console.

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 [Using the Interactive Test Run \(ITR\) Utility \(see page 534\)](#).

lab

A *lab* is a logical container for one or more lab members. For more information, see [Labs and Lab Members \(see page 585\)](#).

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 \(see page 682\)](#).

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 \(see page 682\)](#).

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 \(see page 690\)](#).

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 \(see page 516\)](#).

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 \(MARs\) \(see page 524\)](#).

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 \(MARs\) \(see page 524\)](#).

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 \(see page 689\)](#).

network graph

The network graph is an area of the Server Console that displays a graphical representation of the DevTest Cloud Manager and the associated labs. For more information, see [Start a Lab \(see page 594\)](#).

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 Java Agent captured. For more information, see [Using CA Continuous Application Insight v8.0.2 \(see page 1007\)](#).

path graph

A *path graph* contains a graphical representation of a path and its frames. For more information, see [Path Graph \(see page 1030\)](#).

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 \(see page 351\)](#).

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 \(see page 381\)](#).

quick test

The *quick test* feature lets you run a test case with minimal setup. For more information, see [Stage a Quick Test \(see page 543\)](#).

registry

The *registry* provides a central location for the registration of all DevTest Server and DevTest Workstation components. For more information, see [Registry \(see page 319\)](#).

service image (SI)

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. One way to create a service image is by using the Virtual Service Image Recorder. Service images are stored in a project. A service image is also referred to as a *virtual service image* (VSI). For more information, see [Service Images \(see page 679\)](#).

simulator

A *simulator* runs the tests under the supervision of the coordinator server. For more information, see [Simulator Server \(see page 322\)](#).

staging document

A *staging document* contains information about how to run a test case. For more information, see [Building Staging Documents \(see page 495\)](#).

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 \(see page 489\)](#).

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 \(see page 370\)](#).

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 \(see page 473\)](#).

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 \(see page 516\)](#).

think time

Think time is how long a test case waits before executing a test step. For more information, see [Add a Test Step \(example\) \(see page 474\)](#) and [Staging Document Editor \(see page 496\) - Base Tab](#).

transaction frame

A *transaction frame* encapsulates data about a method call that the DevTest Java Agent or a CAI Agent Light intercepted. For more information, see [Business Transactions and Transaction Frames \(see page 1011\)](#).

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 Model \(VSM\) \(see page 678\)](#).

Virtual Service Environment (VSE)

The *Virtual Service Environment (VSE)* is a DevTest Server application that you use to deploy and run virtual service models. VSE is also known as CA Service Virtualization. For more information, see [Using CA Service Virtualization](#).

Using

Using CA Application Test

- [Using CA Application Test \(see page 318\)](#)
- [Using the DevTest Portal \(see page 324\)](#)
- [Using the Workstation and Console \(see page 348\)](#)

Using CA Service Virtualization

- [CA Service Virtualization \(see page 661\)](#)
- [Installation and Configuration \(see page 666\)](#)
- [Understanding CA Service Virtualization \(see page 677\)](#)
- [Using the DevTest Portal \(see page 693\)](#)
- [Using the Workstation and Console \(see page 758\)](#)
- [VSE Manager \(see page 995\)](#)
- [VSE Commands \(see page 999\)](#)
- [Java Agent VSE Properties \(see page 1006\)](#)

Using CA Continuous Application Insight

- [Getting Started with CAI \(see page 1008\)](#)
- [Configuring Agents \(see page 1013\)](#)
- [Analyzing Business Transactions \(see page 1025\)](#)
- [Using the Shelf \(see page 1061\)](#)
- [Creating and Managing Tickets \(see page 1066\)](#)
- [Working with Defects \(see page 1059\)](#)

- [Creating Virtual Services \(see page 1077\)](#)
- [Creating Baselines \(see page 1110\)](#)
- [Documenting Transactions for Testing \(see page 1156\)](#)
- [Native MQ CAI \(see page 1165\)](#)
- [CAI Command-Line Tool \(see page 1174\)](#)
- [VS Traffic to CA Application Insight Companion \(see page 1182\)](#)
- [CAI Agent Light \(see page 1183\)](#)

Using the SDK

- [Using the SDK \(see page 1212\)](#)
- [Integrating Components \(see page 1214\)](#)
- [Testing Integrated Components \(see page 1219\)](#)
- [Extending DevTest Solutions \(see page 1222\)](#)
- [Extending Test Steps \(see page 1228\)](#)
- [Extending Assertions \(see page 1235\)](#)
- [Extending Filters \(see page 1238\)](#)
- [Custom Reports \(see page 1241\)](#)
- [Custom Report Metrics \(see page 1242\)](#)
- [Custom Companions \(see page 1243\)](#)
- [Using Hooks \(see page 1245\)](#)
- [Custom Data Sets \(see page 1246\)](#)
- [Java .NET Bridge \(see page 1249\)](#)

Using CA Application Test

This section describes how to use CA Application Test to build and run test cases and suites.

- [The Registry \(see page 319\)](#)
- [Coordinator Server \(see page 321\)](#)
- [Simulator Server \(see page 322\)](#)
- [Command-Line Utilities \(see page 324\)](#)



Note: The DevTest Portal is a web-based application that is intended to become the primary user interface for DevTest Solutions. The Portal provides capabilities for some of the most-used workflows for DevTest products. Over time, CA will enhance the functionality of the DevTest Portal and eventually sunset the other interfaces in the LISA product line, including DevTest Workstation. For a quick summary of the functionality available in the Portal, see [DevTest Portal Functionality \(see page 48\)](#).

- [Using the DevTest Portal with CA Application Test \(see page 324\)](#) provides detailed information about using the Portal to build and run test cases and suites for supported features.
- [Using the Workstation and Console with CA Application Test \(see page 348\)](#) provides detailed information about using the DevTest Workstation and DevTest Console to build and run test cases and suites for features that have not yet migrated to the Portal.

The Registry

Contents

- [Start the Registry \(see page 320\)](#)
- [Create a Named Registry \(see page 320\)](#)
- [Change the Registry \(see page 320\)](#)

The registry provides a central location for the registration of all DevTest Server and DevTest Workstation components.

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 CA Continuous Application Insight. The common JMS provider that is used for the component-to-component communication is started and run in the registry process. The broker for DevTest-deployed Java agents is also in the registry.

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 \(see page 1458\)](#), which lets you monitor the test cases, simulators, coordinators, and virtual environments for a test suite.

The registry is associated with at least one [lab \(see page 585\)](#), named the Default lab. If you create a coordinator, simulator, or VSE server without specifying a lab, then the server belongs to the Default lab.

Start the 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
- Click **Start Menu, All Programs, DevTest, Registry**.
- 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**.

Coordinator Server

Contents

- [Create a Coordinator Server \(see page 321\)](#)
- [Monitor Coordinator Servers \(see page 322\)](#)

Coordinator servers receive the test run information as documents, and coordinate the tests that run on one or more [simulator servers \(see page 322\)](#). 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 add the coordinator to a named [lab \(see page 585\)](#) by using the **-l** option. If you do not specify the **-l** option, then the coordinator is added to the Default lab.

You can display the version number by using the **--version** option.

For information, see [Running Server Components as Services \(see page 1370\)](#).

Monitor Coordinator Servers

If DevTest Workstation is attached to the associated registry, then a running coordinator server appears in the [Registry Monitor \(see page 1459\)](#).

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 network graph of the [Server Console \(see page 1455\)](#).

To monitor coordinator servers from the Server Console:

1. Select **View, Server Console**, from the main menu of DevTest Workstation.
2. Click the coordinator server in the network graph.
A details window appears.

Simulator Server

Contents

- [Create a Simulator Server \(see page 323\)](#)
- [Monitor Simulator Servers \(see page 323\)](#)

Simulator servers run the tests under the supervision of the [coordinator server \(see page 321\)](#).

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 add the simulator to a named [lab \(see page 585\)](#) by using the **-l** option. If you do not specify the **-l** option, then the simulator is added to the Default lab.

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 \(see page 1362\)](#).

For information about running the simulator as a service, see [Running Server Components as Services \(see page 1370\)](#).

Monitor Simulator Servers

If DevTest Workstation is attached to the associated registry, then a running coordinator server appears in the [Registry Monitor \(see page 1458\)](#).

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 network graph of the [Server Console \(see page 1455\)](#).

To monitor simulator servers from the Server Console:

1. Select **View, Server Console** from the main menu of DevTest Workstation.
2. Click the simulator in the network graph.
A details window appears.

Command-Line Utilities

The **LISA_HOME\bin** directory contains the following command-line utilities:

- **CAI Command-Line Tool**

The **PFCmdLineTool** command lets you perform various CA Continuous Application Insight tasks from the command line. For more information, see [CAI Command-Line Tool \(see page 1174\)](#).

- **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 \(see page 344\)](#).

- **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 \(see page 531\)](#).

- **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 \(see page 1001\)](#).

- **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 \(see page 1452\)](#).

- **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 \(see page 568\)](#).

- **VSE Manager**

VSE Manager is used for managing virtual service environments. For more information, see [VSE Manager Commands \(see page 999\)](#).

Using the DevTest Portal with CA Application Test

The DevTest Portal is a web-based application that is intended to become the primary user interface for DevTest Solutions. The Portal provides capabilities for some of the most-used workflows for DevTest products. Over time, CA will enhance the functionality of the DevTest Portal and eventually sunset the other interfaces in the LISA product line, including DevTest Workstation. This section provides detailed information about using the Portal to build and run test cases and suites for supported features.

- For a quick summary of the functionality available in the Portal, see [DevTest Portal Functionality \(see page 48\)](#).
- [Using the Workstation and Console with CA Application Test \(see page 348\)](#) provides detailed information about using the DevTest Workstation and DevTest Console to build and run test cases and suites for features that have not yet migrated to the Portal.

Opening the DevTest Portal

You open the DevTest Portal from a web browser.



Note: For information about the server components that must be running, see [Start the DevTest Processes or Services \(see page 115\)](#).

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**.

Enabling and Disabling Tool Tips

The DevTest Portal provides tips for new users on how to perform key tasks such as searching for and creating artifacts. The following graphic displays tips for entering search criteria:



Screen capture of help tip.

You can enable or disable the help tips.

Follow these steps:

1. Open the DevTestPortal.
2. Click the down arrow on the top, right of the portal, and select **Preferences**.
The Preferences dialog opens.
3. To enable help, select **Enable help for first time user**.
4. To disable help, clear **Enable help for first time user**.
5. Click **OK**.



More Information:

- [Create an API Test \(see page 326\)](#)
- [Monitor Tests \(see page 330\)](#)
- [Manage Test Artifacts \(see page 346\)](#)

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 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.
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. 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`
4. Enter a short description of the test in the **Description** field.



More Information:

- [Create a Test by Importing R/R Pairs from a Test \(see page 327\)](#)
- [Create a Test by Importing R/R Pairs from a File \(see page 327\)](#)

- [Create a Test Manually \(see page 328\)](#)
- [Edit an API Test \(see page 329\)](#)

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:

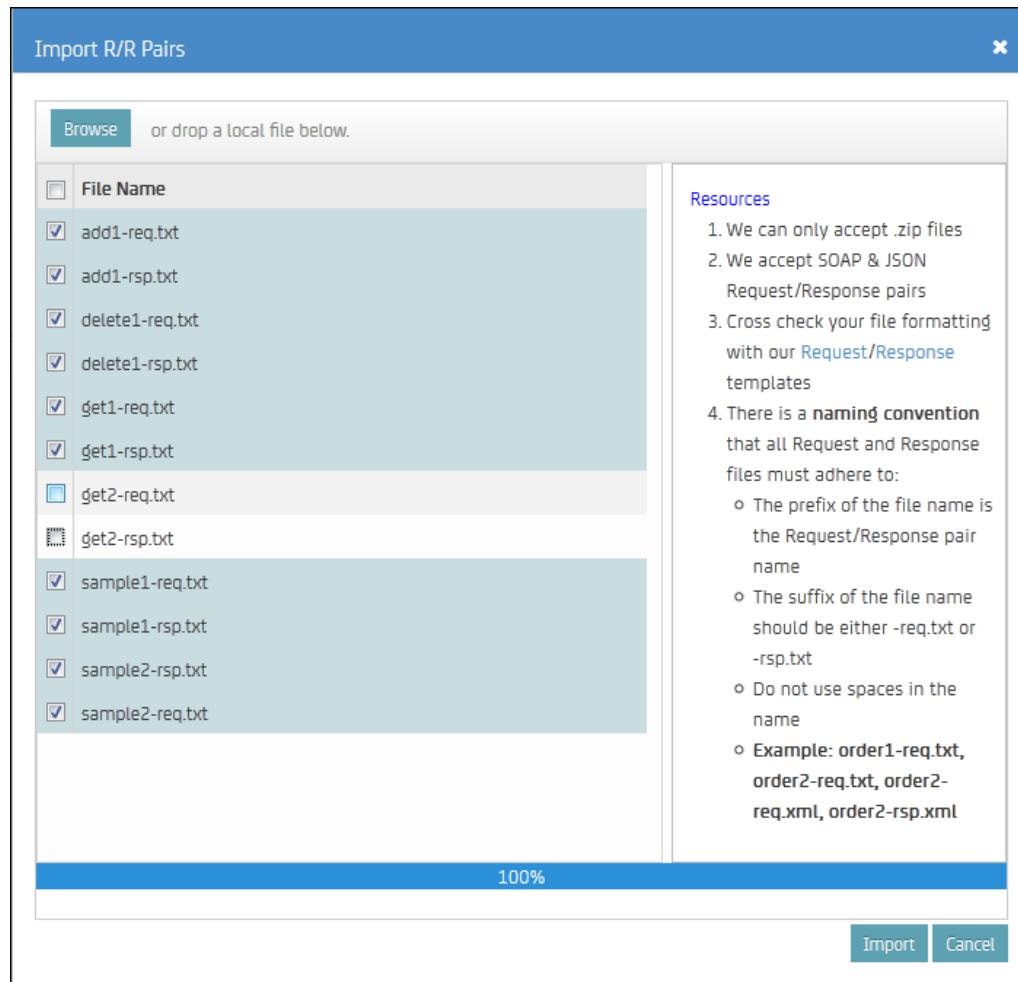
1. From the **R/R Pair Editor** toolbar, click **Import R/R pairs from existing tests** . The **Load Existing API Tests** window opens.
2. The list shows all the projects available on the resource server and the API tests available under each project.
If you have not created an API test in the DevTest Console, 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.

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**.

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, DELETE, or HEAD) 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**.

The screenshot shows the DevTest Portal's Test Editor interface. On the left, there is a list of R/R pairs: add1, delete1, get1, get2, sample1, and sample2. The 'get1' item is selected. In the center, there are two main sections: 'Request' and 'Assertion on Response'. The 'Request' section contains the following XML code:

```

<?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>
        <getUser
            xmlns="http://ejb3.examples.itko.com/">
            <username>${username}</username>
            <password>${password}</password>
        </getUser>
    </soapenv:Body>
</soapenv:Envelope>

```

The 'Assertion on Response' section contains the following XML code:

```

<env:Envelope
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"><env:Header></env:Header>
<env:Body><ns2:getUserResponse
    xmlns:ns2="http://ejb3.examples.itko.com/">
    <email></email><login>webapp-2081255679</login>
    <newFlag>true</newFlag><phone></phone>
    <pwd>${pwd}</pwd><role>${role}</role></ns2:getUserResponse></env:Body>
</env:Envelope>

```

At the bottom of the editor, there are three buttons: 'Create', 'Create and Run', and 'Reset'.

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.

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

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.

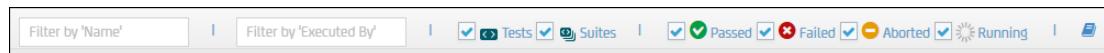
- **Executed By**

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.

To apply your filter, click **Apply**.

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**



To save your filter with a meaningful name, click **Save filter options** and enter a filter name. Then click **OK**.

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 screenshot shows the 'Summary' view of the 'Tests' pane. At the top, there are six colored boxes displaying counts: 23 Passed (green), 5 Failed (red), 11 Aborted (orange), 0 Running (blue), 25 Errors (gray), and 0 Warnings (yellow). Below this is a table titled 'Suites/Tests' with columns: Name, Status, Errors/Warnings, Start Time, Duration, Executed By, and Description. The table lists several entries, some with '+' or '-' icons before their names, indicating expandable/collapsible status. The first entry is 'FastAllTestsSuite' with a green checkmark icon. The second is 'multi-tier-combo' with a green checkmark icon. The third is 'multi-tier-combo' with an orange minus sign icon. The fourth is 'FastAllTestsSuite' with a red minus sign icon. The fifth is 'firsttest20141002-155225' with an orange minus sign icon. The table includes a search bar at the top and various filter checkboxes at the bottom.

Monitor Tests and Suites window

The following fields are displayed:

▪ Name

Displays the name of the test case or suite. A suite name is preceded by a plus sign or minus sign, 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, 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.

▪ Description

Displays the description of the test case or test suite.

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 Cycles results are displayed, showing the status of each cycle.

Time Stamp	Errors/Warnings	Status	VUser	Cycle Number	Duration
Thu Sep 04 13:15:17 CDT 2014	2	✓	0	0	4.9 s
Thu Sep 04 13:15:17 CDT 2014	2	✓	1	0	5.0 s
Thu Sep 04 13:15:17 CDT 2014	2	✓	2	0	5.0 s

The following fields are displayed for each cycle:

- **Time Stamp**

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 , or 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.

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 DevTest Portal with request and response

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.

DevTest Solutions - 8.4

Test - multi-tier-combo

Status	Passed	0	Failed	0	Aborted	0	Errors	0	Warnings
--------	--------	---	--------	---	---------	---	--------	---	----------

Status: ✓ Runs: test_default_run_name
End Time: 1/7/2015, 14:37:40

Cycle Details Cycle Num: 0 Status: Passed VUsers: 0 Start Time: 1/7/2015, 14:36:32 Duration: 1m 07.0s

Work Flow Expand All | Collapse All

Filter by 'Name' | Steps Quiet Steps Fired Assertions Evaluated Assertions Filters Data Sets Companions | Passed Failed Aborted

- > ds_login
- > DataSet1
- > Add User Object XML - adds a new customer to the bank from a XML dataset - the customer's login is verified in the return..

Request Response 1/7/2015, 14:36:32 1967ms 2682bytes

Request Response

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema">
3   <soapenv:Body>
4     <addUserObject xmlns="http://ejb3.examples.itko.com/">
5       <userObject xmlns="">
6         <accounts>
7           <balance>101.01</balance>
8           <name>Basic Checking</name>
9         </accounts>
10        <balance>101.01</balance>
11        <name>Orange Savings</name>
12      </userObject>
13    </addUserObject>
14  </soapenv:Body>
15 </soapenv:Envelope>
```

Properties

XML XPaths Filter

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.

View a Screenshot of a Test Step

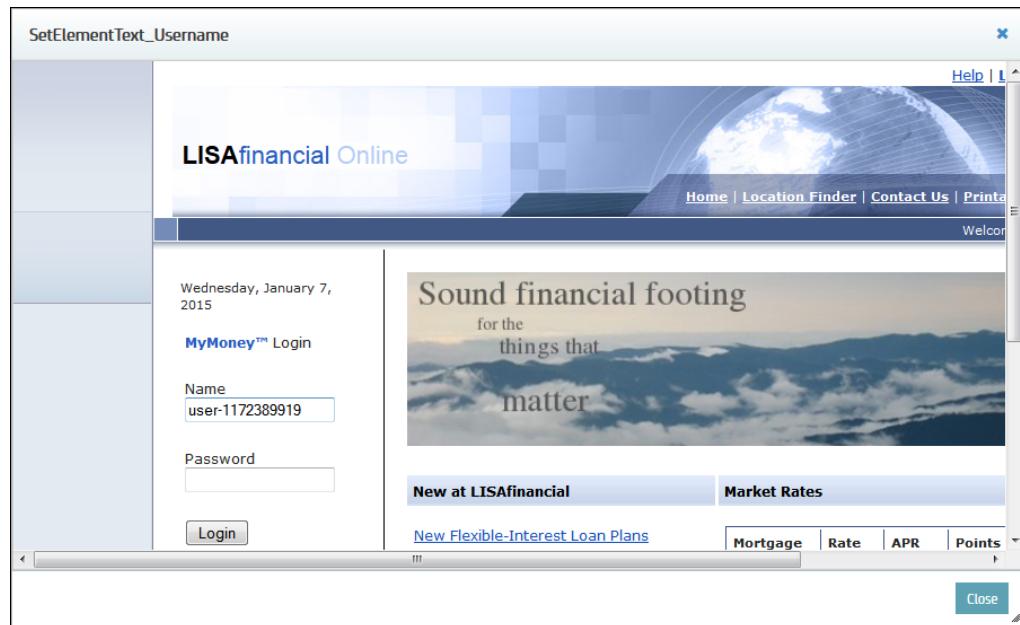
To view a screenshot of a test step, locate a step that displays the **Screenshot** button.

> Deposit Money - Assumes that ATM deposit puts a message on the queue. - The money is deposited into the account via a messa...	Request	Response	1/7/2015, 14:58:18	140ms	0 bytes
> Get Go to Lisa Bank home page on local machine	Request	Response	1/7/2015, 14:58:29	15032ms	0 bytes
> SetElementText_Username Sets value for user name	Request	Response	1/7/2015, 14:58:44	5407ms	0 bytes
> SetElementText_Password Sets value for password	Request	Response	1/7/2015, 14:58:50	3789ms	0 bytes
> ClickElement_Login Perform login action	Request	Response	1/7/2015, 14:58:55	3885ms	0 bytes
> Verify_User Name On Page Verify if user login successfully	Request	Response	1/7/2015, 14:58:59	525ms	0 bytes
> ClickElement_CheckingAccountId Go to checking account activity	Request	Response	1/7/2015, 14:59:05	537ms	0 bytes

Tip: In the **examples** project, the **multi-tier-combo-selenium.tst** has several steps that can display screenshots.

Follow these steps:

1. Click **Screenshot**.



2. The screenshot window appears.

3. Size the window by using the gray arrow at the bottom right corner of the screen to drag the corner.

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.

DevTest Solutions - 8.4

The screenshot shows a DevTest assertion result for the step 'list users'. The message indicates a failed assertion: 'Assert [Assert user is now in [list]] fired false of type Result as String. Contains Given String Result did not contain string. Looked for: 1-csv_usertest-2123150799 in result.' The stack trace is long and detailed, showing various Java classes and methods from org.apache.http.impl.client.DefaultHttpClient and org.apache.http.impl.conn.ManagedClientConnectionImpl. The 'Result false' section shows the actual value as a single-line string: '1 HTTP'.

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.

The screenshot shows a DevTest assertion result for the step 'new user'. The message states 'The assertion of type "Any Non-Empty Result" passed evaluation.' The 'Result true' section shows the actual value as a single-line string: '1 HTTP'.

Result of an evaluated assertion in the DevTest Portal, Monitor Tests

Details of Filters

Details for filters display the properties that the filter set.

The screenshot shows the properties of an 'XML XPath Filter'. It has one LISA property named 'fl_login_get'. The 'Value Before Filter' is 'n/a' and the 'Value After Filter' is 'webapp-623189486'.

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.

The screenshot shows the 'Companions Panel' of the DevTest Solutions interface. It lists several test steps and their status:

- Set Final Step to Execute**: Status: Success (green icon)
- ds_login**: Status: Failed (red icon)
- DataSet1**: Status: Failed (red icon)
- Add User Object XML**: Status: Failed (red icon) with 1 error. Description: "adds a new customer to the bank from a XML dataset - the customers's login is verified in the return...". A 'Request' button is shown next to the description.
- abort**: Status: Failed (red icon)
- Fail Test Case Companion**: Status: Success (green icon)

A note at the bottom of the panel states: "Test case "CopyOfmulti-tier-combo" passed if all of contained steps passed; otherwise, it failed if one or more contained steps failed."

Companions Panel

Monitor with CVS (Continuous Validation Service)

This release includes a [preview \(see page 204\)](#) of the CVS feature in the DevTest Portal.

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 \(see page 50\)](#).

You can open the **CVS Dashboard** from the DevTest Portal, DevTest Workstation, or from a web browser.

Open the CVS Dashboard

To open the CVS Dashboard from the DevTest Portal:

Select **Monitor, CVS** from the left navigation panel.

To open the CVS Dashboard from DevTest Workstation:

Select **View, CVS Dashboard** from the main menu.

To open the CVS Dashboard from a web browser:

1. Ensure that the [registry \(see page 319\)](#) is running.
2. Enter <http://localhost:1505/> in a web browser.
If the registry is on a remote computer, replace **localhost** with the name or IP address of the computer.
The DevTest Console appears.
3. Click **Continuous Validation Service**.



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.

More Information:

- [CVS Dashboard Overview \(see page 338\)](#)
- [Deploy a Monitor to CVS \(see page 342\)](#)
- [Run a Monitor Immediately \(see page 344\)](#)
- [View Test Details \(see page 344\)](#)
- [Email Notification Settings \(see page 343\)](#)
- [CVS Manager \(see page 344\)](#)

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 the 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 \(see page 342\)](#).
- **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 the 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 the 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:

- **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 the 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

Contents

- [Deploy a Monitor through the DevTest Portal \(see page 342\)](#)
- [Deploy a Monitor through DevTest Workstation \(see page 342\)](#)
- [Deploy a Monitor through the CVS Dashboard \(see page 342\)](#)
- [Deploy a Monitor through the cvsMonitors Directory \(see page 343\)](#)
- [Deploy a Monitor through CVS Manager \(see page 343\)](#)

Deploy a Monitor through the DevTest Portal

Before you can perform this procedure, a Model Archive for the monitor must be [created \(see page 531\)](#).

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 \(see page 528\)](#).
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 \(see page 531\)](#).

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 \(see page 531\)](#).

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 \(see page 344\)](#).

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 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 those
alerts
# lisa.alert.email.emailAddr=lisa@itko.com

# And this is the email server we will attempt to route emails with (SMTP server)
# lisa.alert.email.defHosts=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 \(see page 598\)](#). 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

This example deploys a monitor to CVS.

```
CVSManager -d monitor.mar -u user -p password
```

Example: Delete Monitor

This example deletes that same monitor (assuming the service and monitor names).

```
CVSManager -x -s OrderManager -n CheckOrders -u user -p password
```

This example deletes all monitors in the OrderManager service.

```
CVSManager -X -s OrderManager -u user -p password
```

This example deletes all monitors.

```
CVSManager -X -u user -p password
```

Manage Test Artifacts

To manage test artifacts, on the left navigation panel, select **Manage**, then select from the following:

- **API Tests:** API tests that were created with the 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

The **Manage API Tests** option lets you edit tests that were created in the 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 \(see page 329\)](#).

Manage Tests

You can manage tests from the **Manage Tests** window.

To view information and documentation about a test, click **Edit**.

To delete a test, click **Delete**.

To run a test, click **Options** and select **Run**.

To run a test with the options to change the configuration file, staging document, or coordinator server, click **Options** and select **Run with Options**. The **Run Test Case** dialog appears.

1. (Optional) To change the configuration file for the test, enter a new configuration file name in the **Configuration** field. To display the values in the configuration file, click  **Info**.
2. (Optional) To change the staging document for the test, select a staging document from the drop-down list.
3. (Optional) To change the coordinator server for the test, select a coordinator server from the drop-down list.
4. Click **Run**.

To download the [MAR \(see page 524\)](#) file with the test, click **Options** and select **Download MAR**.

Mobile Testing Runtime Properties

You can add or edit runtime properties on a mobile test using the DevTest Portal. These properties are loaded when a mobile test case starts. You can tag test jobs with these properties, so when you execute report filters, certain test cases can be grouped and located much easier.

Follow these steps:

1. In the DevTest Portal, click the test case, then select **Run with options**.
2. Click the **Additional Properties** tab (at the bottom).
3. Click **New**.
4. Enter the Property **name**.
5. Enter the Property **value**.
6. Click **Run**.

Manage Test Suites

You can manage test suites from the **Manage Test Suites** window.

To view information and documentation about a test suite, click **Edit**.

To delete a test suite, click **Delete**.

To run a test suite, click **Options** and select **Run**.

To run a test suite with the options to change the configuration file, click **Options** and select **Run with Options**. The **Run Test Suite** dialog appears.

1. (Optional) To change the configuration file for the test suite, enter a new configuration file name in the **Configuration** field. To display the values in the configuration file, click  **Info**.
2. Click **Run**.

To download the [MAR \(see page 524\)](#) file for the test suite, click **Options** and select **Download MAR**.



More Information:

- Video about the [All Resources window](https://www.youtube.com/watch?v=utlwLF7Kug&index=5&list=PLUo5-4tntpYJmxYRsnMkYSWPTdQ-vrKd8) (<https://www.youtube.com/watch?v=utlwLF7Kug&index=5&list=PLUo5-4tntpYJmxYRsnMkYSWPTdQ-vrKd8>)

Using the Workstation and Console with CA Application Test

Contents

- [DevTest Workstation \(see page 348\)](#)
- [DevTest Console \(see page 349\)](#)
 - [Reporting Dashboard \(see page 349\)](#)
 - [Continuous Validation Service \(see page 349\)](#)
 - [Server Console \(see page 349\)](#)

This section provides detailed information about using the DevTest Workstation and DevTest Console to build and run test cases and suites for features that have not yet migrated to the Portal.

The DevTest Portal is a web-based application that is intended to become the primary user interface for DevTest Solutions. The Portal provides capabilities for some of the most-used workflows for DevTest products. Over time, CA will enhance the functionality of the DevTest Portal and eventually sunset the other interfaces in the LISA product line, including DevTest Workstation.

- For a quick summary of the functionality available in the Portal, see [DevTest Portal Functionality \(see page 48\)](#).
- [Using the DevTest Portal with CA Application Test \(see page 324\)](#) provides detailed information about using the Portal to build and run test cases and suites for supported features.

DevTest Workstation

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, all in a code-less manner: 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 DevTest test cases and virtual services (test and virtual service model authoring IDE) manages, schedules, and orchestrates DevTest test cases continually for unit, functional, load, and performance tests.

DevTest Console

The web-based DevTest Console provides access to the following consoles and dashboards:

- **Reporting Dashboard**
- **Continuous Validation Service**
- **Server Console**

Reporting Dashboard

A report viewer displays event and metric information, and information that is derived from data that was captured during the running of tests. You can use a staging document to set the events and metrics to capture for reporting purposes.

For more information, see [Reports \(see page 598\)](#).

Continuous Validation Service

The Continuous Validation Service (CVS) lets you schedule tests and test suites to run regularly over an extended time period.

For more information, see [Continuous Validation Service \(CVS\) \(see page 598\)](#).

Server Console

The **Server Console** enables you to manage labs and to configure role-based access control. The **Server Console** is also where you access the VSE Dashboard.

For more information, see [Cloud DevTest Labs \(see page 585\)](#), [Access Control \(ACL\) \(see page 1417\)](#), and [VSE Dashboard \(see page 973\)](#).

Open DevTest Workstation

When you open DevTest Workstation, you are prompted to specify a [registry \(see page 319\)](#).

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 \(see page 1409\)](#).

To open DevTest Workstation:

1. Do one:

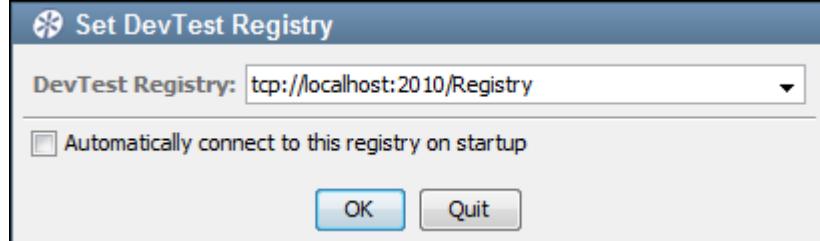
- Open a command prompt, go to the **LISA_HOME\bin** directory, and run the DevTest Workstation executable.



DevTest
Workstation

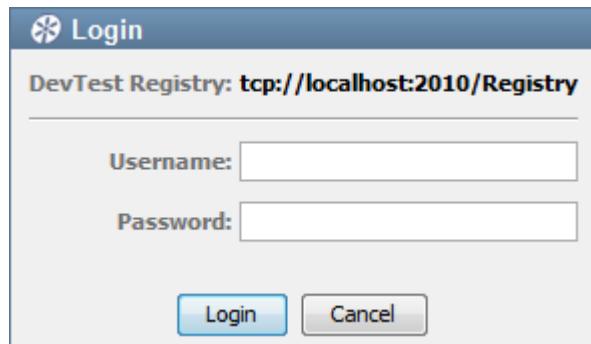
- If you have a DevTest application icon  on your desktop, double-click the application icon.
- Click **Start Menu, All Programs, DevTest, DevTest Workstation.**

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**.
The **Login** dialog opens.



Login dialog box

4. 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 \(see page 361\)](#) 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 \(see page 482\)](#).
- **Create New VS Model**
For detailed information, see [Work with Virtual Service Models \(see page 862\)](#). This feature requires an extra license.
- **Create New VS Image**
For detailed information, see [Create a Service Image \(see page 760\)](#). This feature requires an extra license.

- **Create New Staging Document**

For detailed information, see [Building Staging Documents \(see page 495\)](#).

- **Create New Suite**

For detailed information, see [Building Test Suites \(see page 516\)](#).

- **Create New Test Audit**

For detailed information, see [Building Audit Documents \(see page 510\)](#).

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.

- **rawSoap**

Test MAR that includes the rawSoap test case, the 1User0Think_RunContinuously 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 data sets, keystores, and WSDLs 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 CAI disabled and no maximum run time.

- **monitorRunMultiple.stg**

Staging document with one user, run continuously, 136 percent think time, with CAI enabled and a maximum run time of 15 seconds.

- **monitorSLARun.stg**

Staging document with one user, one cycle, and 100 percent think time, with CAI 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 in the Examples 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 CAI 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 CAI because the agent records the fact that an exception was logged. Thus, DevTest determines that something is wrong.

An alternative to using CAI 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".

- **rawSoap**

The rawSoap step is a one-step test case demonstrating a simple raw SOAP request in the "listUsers" step.

- **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 protocol handlers, 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 \(see page 712\)](#).

- **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

- **statefullATM.tst**

- **statelessATM.tst**

- **web-app-proxy.tst**

- **DatabaseModel.vsm**

- **kioskV4model.vsm**

- **kioskV5.vsm**

- **kioskV6.vsm**

- **WebServicesModel.vsm**

- **webservices-vs.vsm**

Images

- **DatabaseModel.vsi**
- **kioskV4ServiceImage.vsi**
- **kioskV6.vsi**
- **si-kioskV5-dynamic.vsi**
- **si-kioskV5.vsi**
- **WebServicesModel.vsm**

Create a Project

You can create a project from scratch or 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

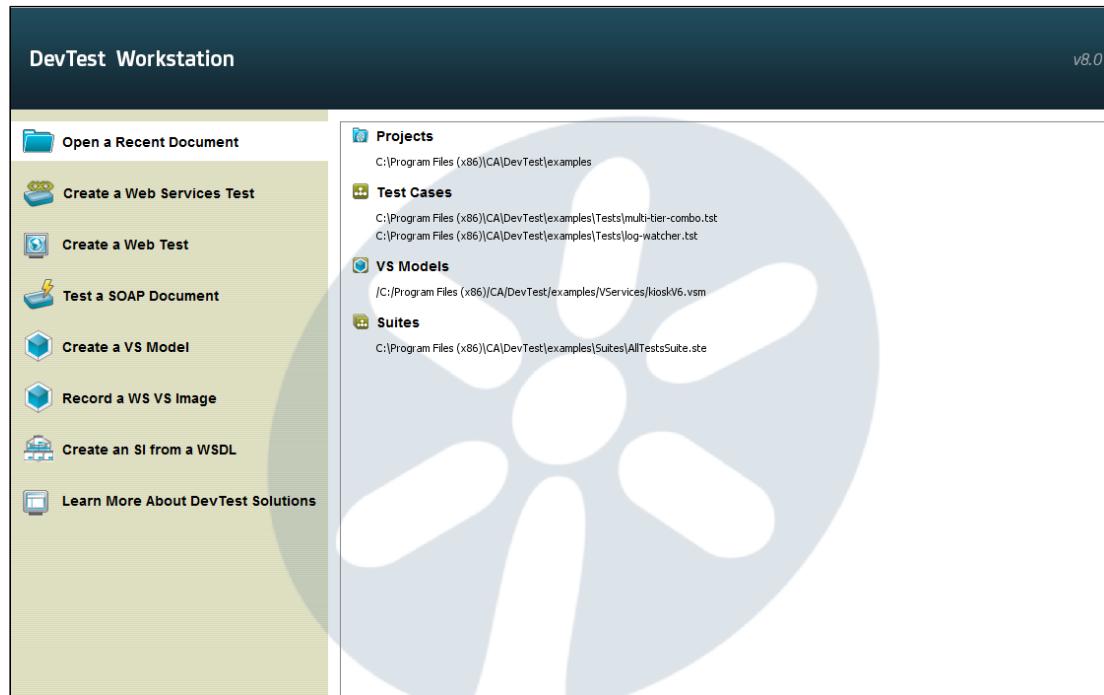
Contents

- [Open Recent \(see page 363\)](#)
- [New WS Test \(see page 364\)](#)
- [New Web Test \(see page 365\)](#)
- [Send SOAP Doc \(see page 365\)](#)
- [Create VSM \(see page 366\)](#)
- [Record WS VSI \(see page 367\)](#)
- [VSI from WSDL \(see page 368\)](#)
- [Learn More \(see page 368\)](#)
- [DevTest Workstation Memory Meter \(see page 369\)](#)

The **Quick Start** window is the first one you see when you open DevTest Workstation.



Note: The **Quick Start** window is always available as a tab after more tabs are opened.



Quick Start window

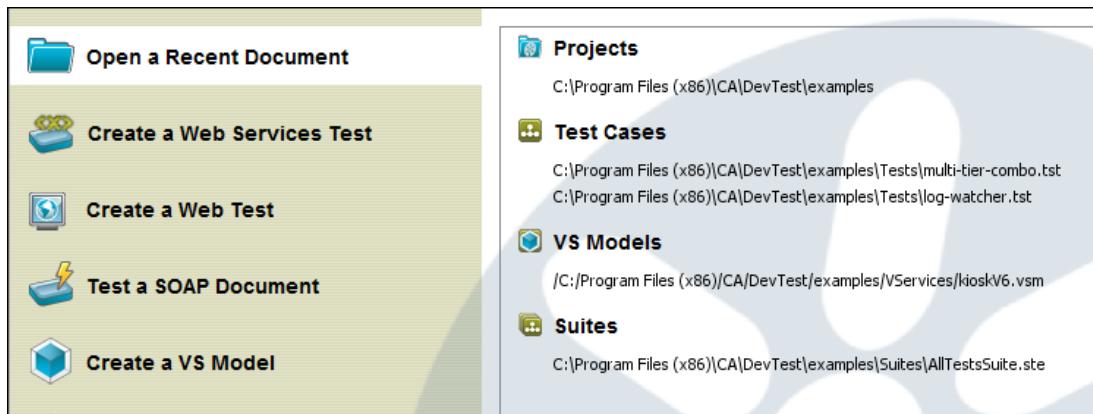
The **Quick Start** window has some of the most useful options in DevTest Workstation. When you click an option, its parameters are listed on the right side of the window.

The following choices are available on this window. Depending on your screen resolution, you could see the compact or the expanded wording for each menu choice. To view all menu options, click the down arrow at the bottom of the window.

- **Open Recent or Open a Recent Document**
- **New WS Test or Create a Web Services Test**
- **New Web Test or Create a Web Test**
- **Send SOAP Doc or Test a SOAP Document**
- **Create VSM or Create a VS Model**
- **Record WS VSI or Record a WS VS Image**
- **VSI from WSDL or Create an SI from a WSDL**
- **Learn More or Learn More About DevTest**

Open Recent

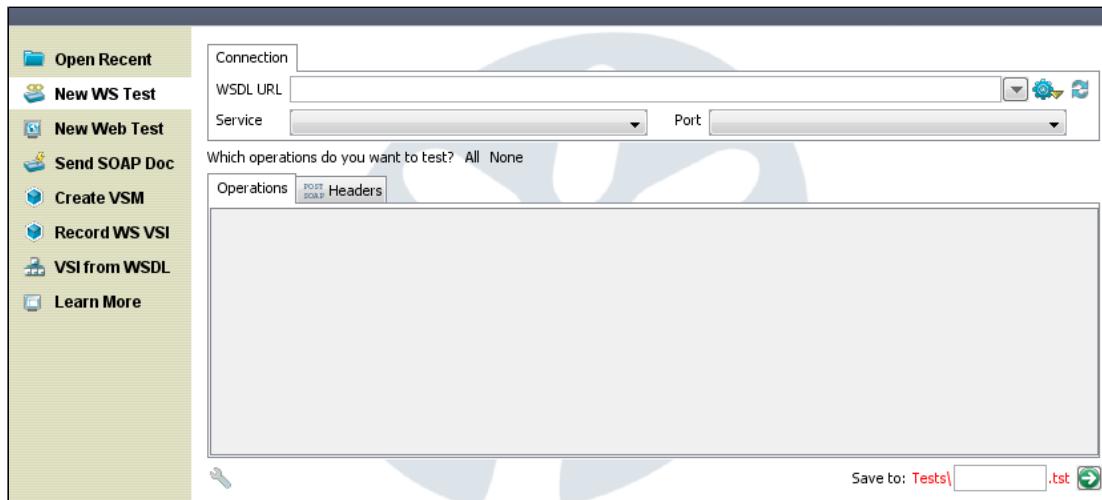
When DevTest Workstation opens, by default this option is selected. The right pane displays recently opened projects, test cases and suites, staging documents, VS models or VS images (if applicable).



Quick Start window, Open Recent option

New WS Test

The **New WS Test** option in the **Quick Start** window lets you create a Web Service test case. The parameters that are needed are displayed in the right pane.



Quick Start window, New WS test option

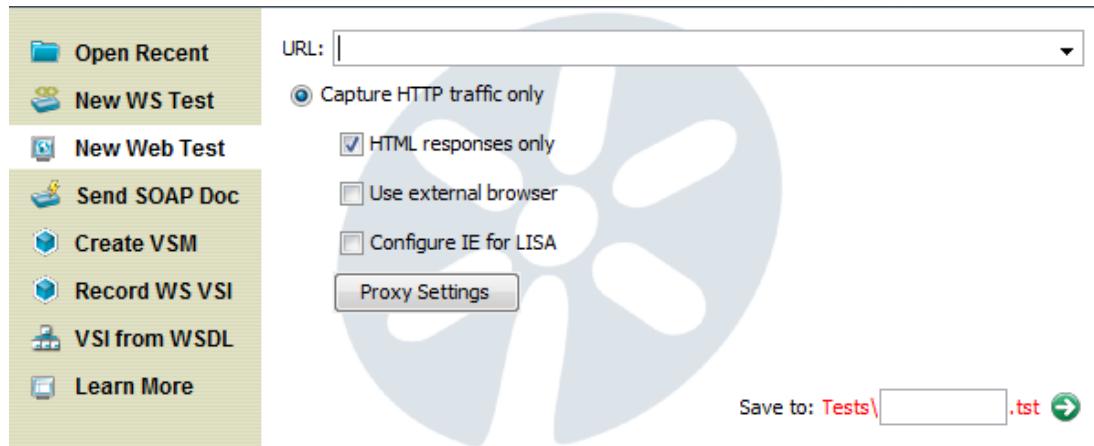
Follow these steps:

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 \(see page 632\)](#).

New Web Test

The **New Web Test** option in the **Quick Start** window lets you create a web test.



Quick Start window, New Web Test option

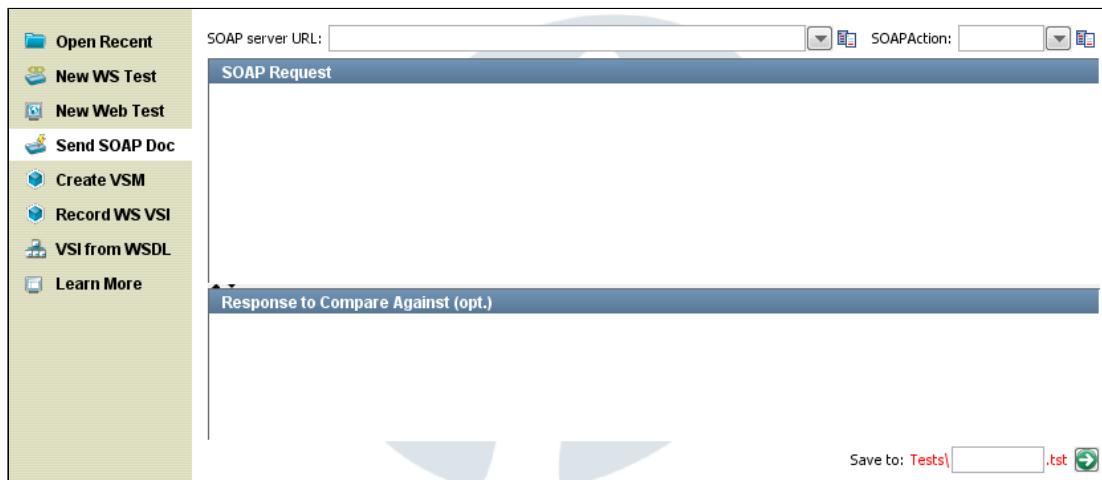
Follow these steps:

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 \(see page 633\)](#).

Send SOAP Doc

The Send SOAP Doc option in the **Quick Start** window lets you create a SOAP Request test case.



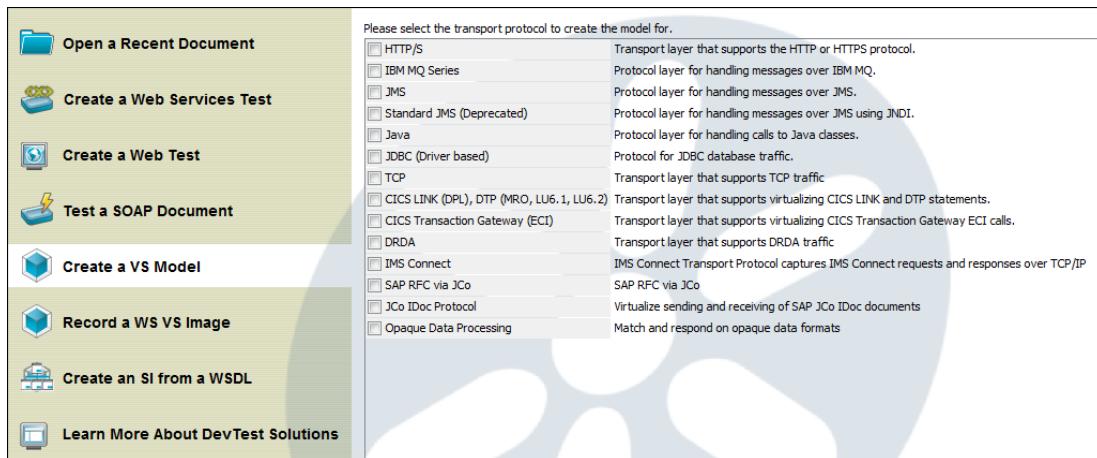
Quick Start window, Send SOAP Doc option

Follow these steps:

1. Enter the SOAP server URL and SOAP Action.
2. Enter the URL of the web service endpoint in the **SOAP Server URL** field.
3. In the **SOAP Action** field, enter the SOAP action as indicated in the <soap: operation> tag in the WSDL for the method being called. This value is required for SOAP 1.1 and often required to be left blank for SOAP 1.2.
4. Type or paste the SOAP Request into the editor.
5. Enter the name of the test in the **Save to** field.
6. To create the test case, click the green arrow .

Create VSM

The **Create VSM** option in the **Quick Start** window lets you create a Virtual Service Model and a corresponding Virtual Service Image.



Quick Start window, Create VSM option

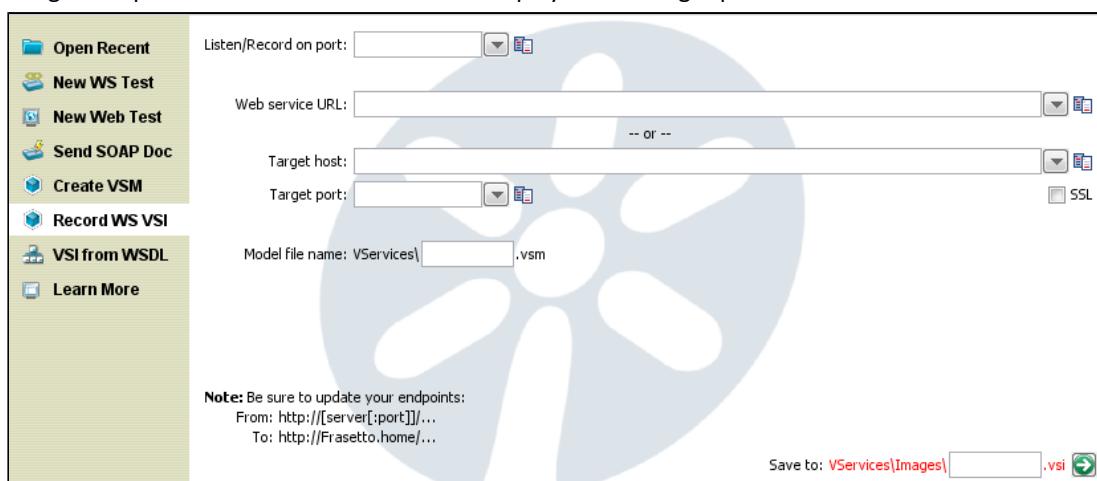
Follow these steps:

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 \(see page 760\)](#).

Record WS VSI

The **Record WS VSI** option in the **Quick Start** window lets you create a web services Virtual Service Image. The parameters that are needed are displayed in the right pane.



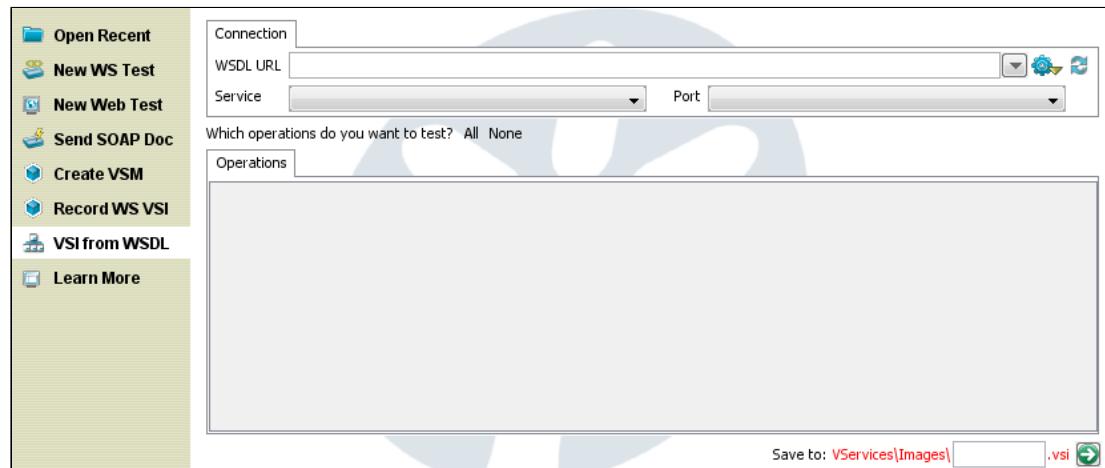
Quick Start window, Record WS VSI option

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 .

VSI from WSDL

The **VSI from WSDL** option in the **Quick Start** window lets you create a Virtual Service Image from a WSDL. The properties that are required are displayed in the right pane.



Quick Start window, 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 in the **Quick Start** 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 \(see page 1775\)](#) 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
```

Open the DevTest Console

You can open the DevTest Console from the main menu of DevTest Workstation or from a web browser.

The DevTest Console home page lets you access the **Reporting Dashboard**, the **CVS Dashboard**, and the **Server Console**. The home page also includes a link to the CAI portion of the DevTest Portal. The lower right area of the home page displays the version number.

To open the DevTest Console from the DevTest Workstation main menu:

1. Select **View, DevTest Portal** from the main menu.
A login dialog appears.
2. Enter your user name and password, and click **Login**.

To open the DevTest Console from a web browser:

1. Ensure that the [registry \(see page 319\)](#) is running.

2. Enter **http://localhost:1505/** in a web browser. If the registry is on a remote computer, replace **localhost** with the name or IP address of the computer.
A login dialog appears.
3. Enter your user name and password, and click **Login**.

Web Server Timeouts

By default, the web server that the DevTest Console uses waits 90 seconds for a process to run on the server. If a process takes longer than 90 seconds, the connection is aborted and the client application or browser handles this abort appropriately.

You can change the default timeout value by adding the **lisa.webserver.socket.timeout** property to the **local.properties** file. The value is in milliseconds. For example:

```
lisa.webserver.socket.timeout=120000
```

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 \(see page 371\)](#)
- [Properties \(see page 381\)](#)
- [Configurations \(see page 393\)](#)
- [Assets \(see page 398\)](#)
- [Filters \(see page 405\)](#)
- [Assertions \(see page 419\)](#)
- [Data Sets \(see page 433\)](#)
- [Companions \(see page 441\)](#)

- [Complex Object Editor \(COE\) \(see page 443\)](#)
- [Building Test Steps \(see page 473\)](#)
- [Creating Test Cases \(see page 481\)](#)
- [Building Subprocesses \(see page 489\)](#)

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 \(see page 371\)](#)
- [Multi-tier-combo Test Case \(see page 375\)](#)
- [Elements of a Test Case \(see page 376\)](#)
- [Elements of a Test Step \(see page 378\)](#)

Test Case Quick Start

To start working with test cases:

1. Start the registry.
See [DevTest Registry \(see page 319\)](#).
2. Create or open a project in DevTest Workstation.
See [Project Panel \(see page 351\)](#).
3. Create a test case in the project.
See [Create a Test Case \(see page 482\)](#).
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 \(see page 375\)](#).

DevTest Workstation is divided into the following main areas (from left to right):

- Project Panel
- Model editor
- Element Panel

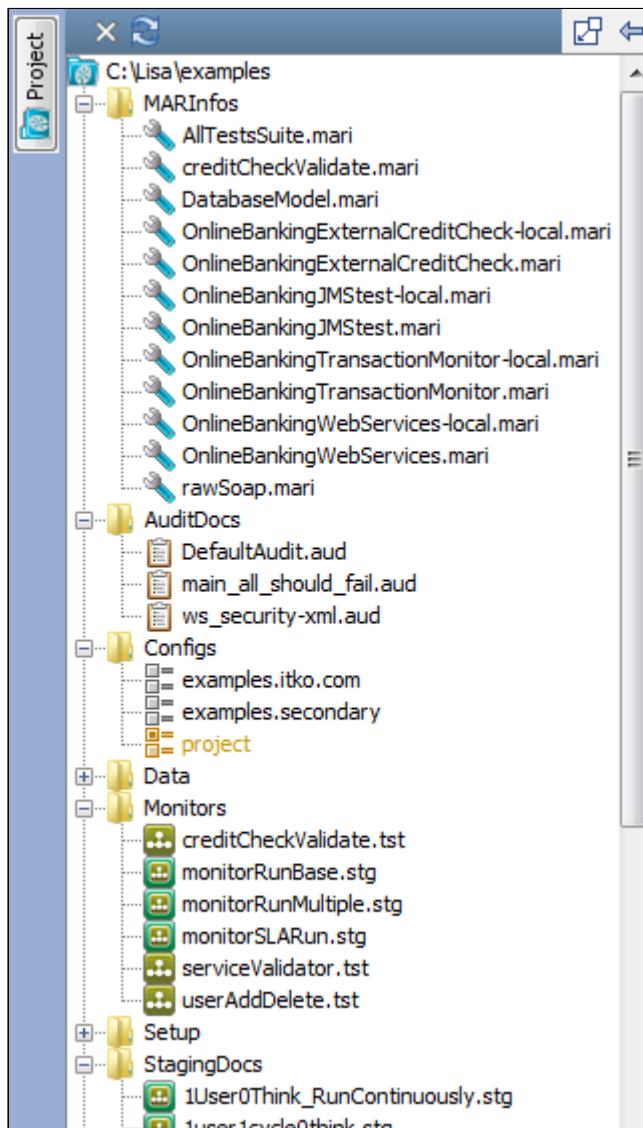
Project Panel

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.



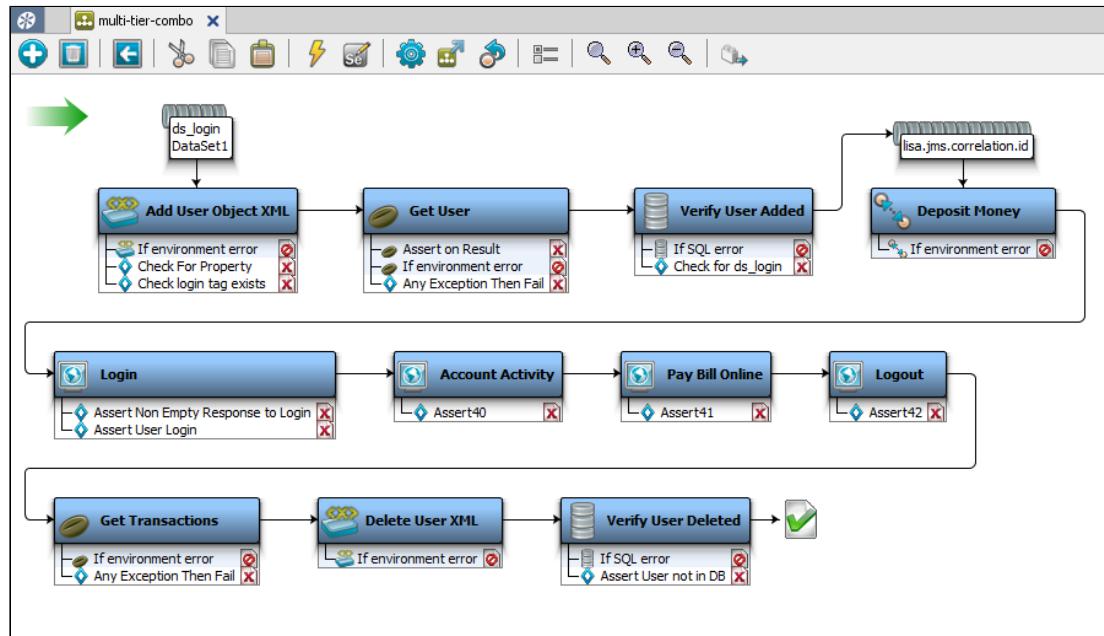
Project panel

For more information, see [Project Panel. \(see page 351\)](#)

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

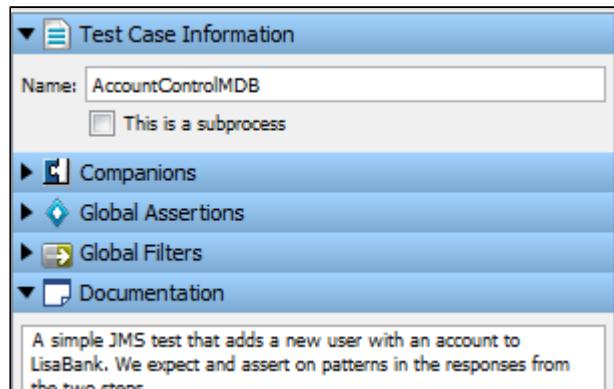
The green arrow marks the start of a test case.

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.

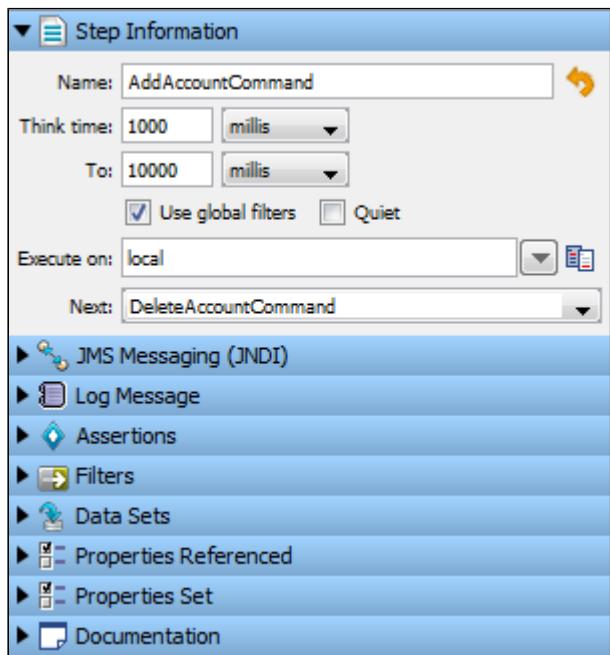
In the model editor, an icon in the workflow represents each step in the test case. The icons change according to the type of test step. For example, if you have a database-related step, a database icon and the associated filters and assertions are attached to the step.

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 \(see page 353\)](#) 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 \(ITR\) \(see page 534\)](#) window, the test creates a single user from the first row of the spreadsheet and finishes.

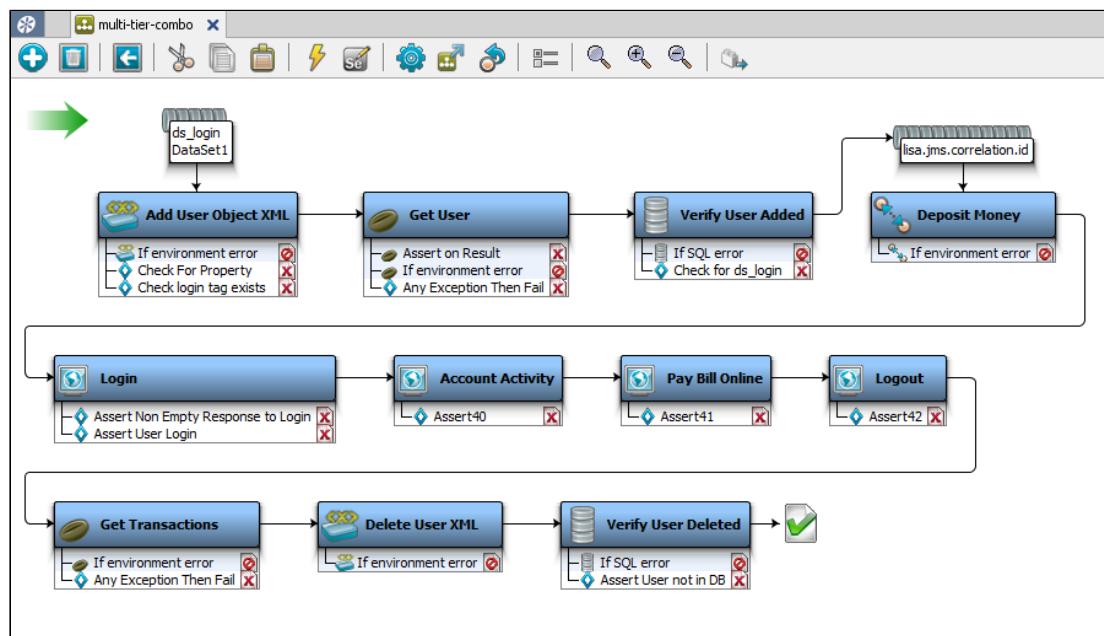
If you [stage the test \(see page 550\)](#) with the example **1User0Think_RunContinuously** staging document, the test restarts until it reaches the end of the data set. This method is the preferred way to iterate repeatedly over a large data set. You can introduce a loop in the test case, but that 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.

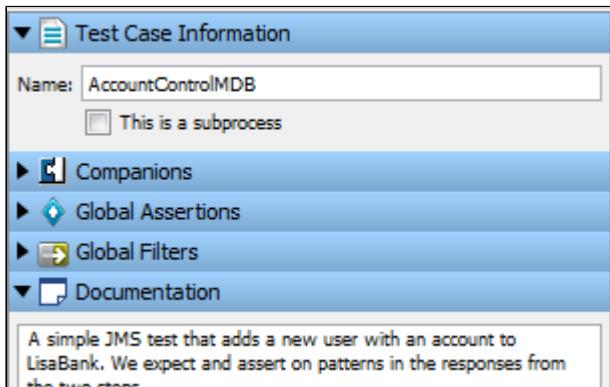
Here you can see in the model editor section the test case information.



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.

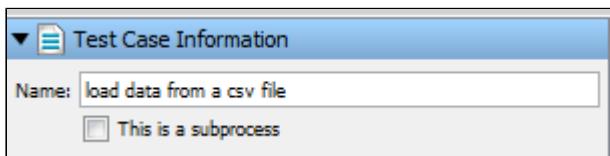


Screenshot of Element panel for a test case

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.



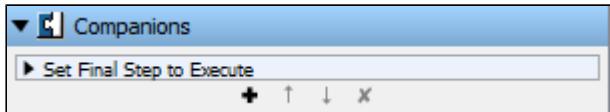
Screenshot of the Test Case Information tab

For more information about subprocesses, see [Building Subprocesses \(see page 489\)](#).

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.



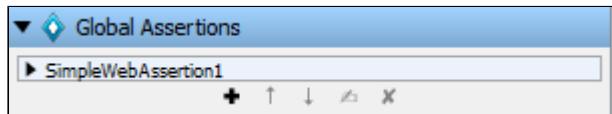
Screenshot of Companions editor

For more information, see [Companions \(see page 441\)](#).

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.



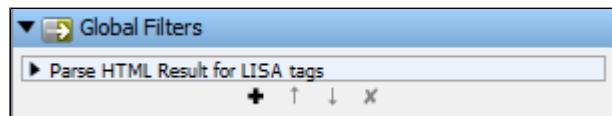
Screenshot of Global Assertions tab

For more information, see [Assertions \(see page 419\)](#).

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 \(see page 405\)](#).

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.

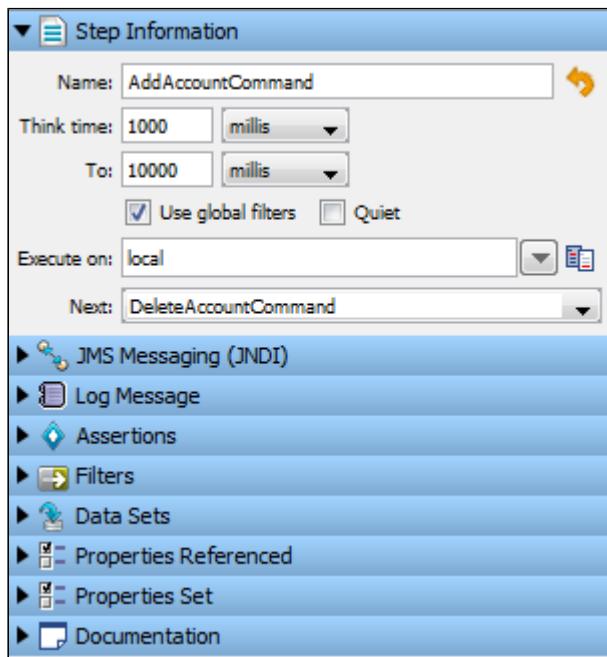


Screenshot of Documentation tab

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 \(see page 473\)](#).

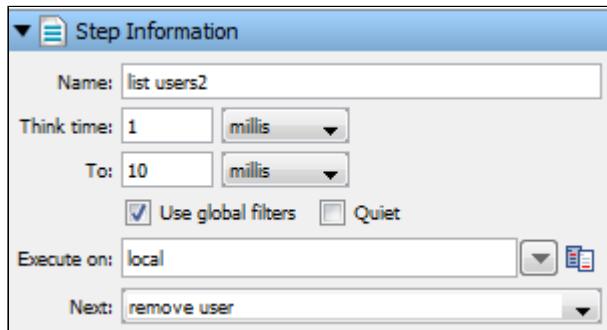


Screenshot of test step collapsed

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 \(see page 473\)](#).

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.



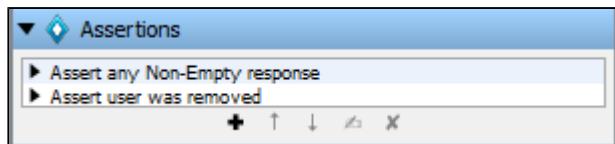
Screenshot of log message section

For more information, see [Test Step Logger \(see page 1451\)](#).

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. \(see page 419\)](#)

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 \(see page 405\)](#).

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.



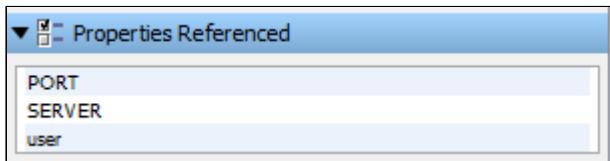
Screenshot of data sets section

For more information, see [Data Sets \(see page 433\)](#).

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 \(see page 381\)](#).

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.

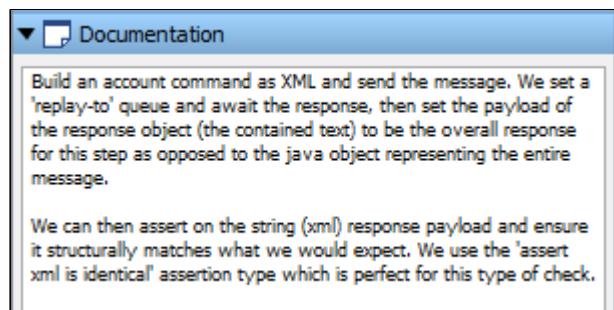


Screenshot of properties set section

For more information, see [Properties \(see page 381\)](#).

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 \(see page 1581\)](#) 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.



More Information:

- [Specify a Property \(see page 382\)](#)
- [Property Expressions \(see page 383\)](#)
- [String Patterns \(see page 383\)](#)
- [Property Sources \(see page 389\)](#)
- [Common Properties and Environment Variables \(see page 390\)](#)
- [Property Files \(see page 392\)](#)
- [Use the Properties Pane \(see page 392\)](#)

Specify a Property

The syntax for a property is {{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 like WordPad.



Note: All property names that start with **lisa.** are reserved for internal use. DevTest may hide or delete properties that start with **lisa.**

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. 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 \(see page 657\)](#) or www.beanshell.org (<http://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.

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.

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.

Patterns

These patterns generate Strings based on the following definitions:

Pattern-based

- D - digit (0-9)
- H - hex digit (0-9, A-F)
- h - hex digit (0-9, a-f)
- X - hex digit (0-9, A-F, a-f)
- L - letters (A-Z)
- I - letters (a-z)
- A - alphanumeric
- P - punctuation
- . - (dot) anything printable
- [1,2,3] - randomly choose among the options shown
- {0,9,8} - do not allow these on the previous spec
- \ - take the following char as a literal
- *(N[-M]) - repeat the pattern N times, or randomly N-M times if M is present

Anything that is not a pattern char IS a literal, for example JohnDDD would be "John123" or the like.

Built-in String Generator Patterns		
Name	Pattern	Category
CC Expiry		TestData
CC Verification Code		TestData
City		TestData
Country		TestData
Credit Card		TestData
Email		TestData
First Name		TestData
Last Name		TestData
Middle Initial		TestData
SSN		TestData
State Code		TestData
Street Address		TestData
Telephone		TestData
UPS Tracking Code		TestData
Zip Code		TestData

Find:

Custom String Generator Patterns		
Name	Pattern	Category

+ ↑ ↓ Delete Find: Search
 Sample: Exp:

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

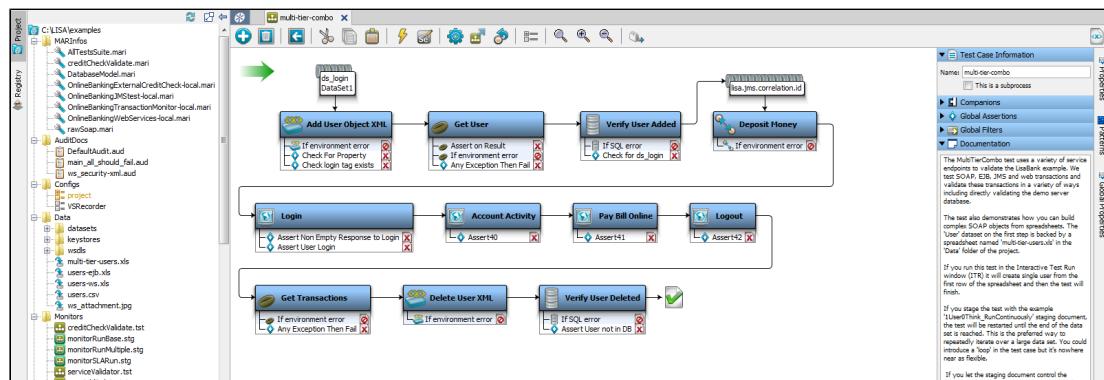
When you add your own data, the only thing to be careful about is to assign the ID correctly. Start with 1 and increase with no gaps until you get to your number of rows.

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.

Example

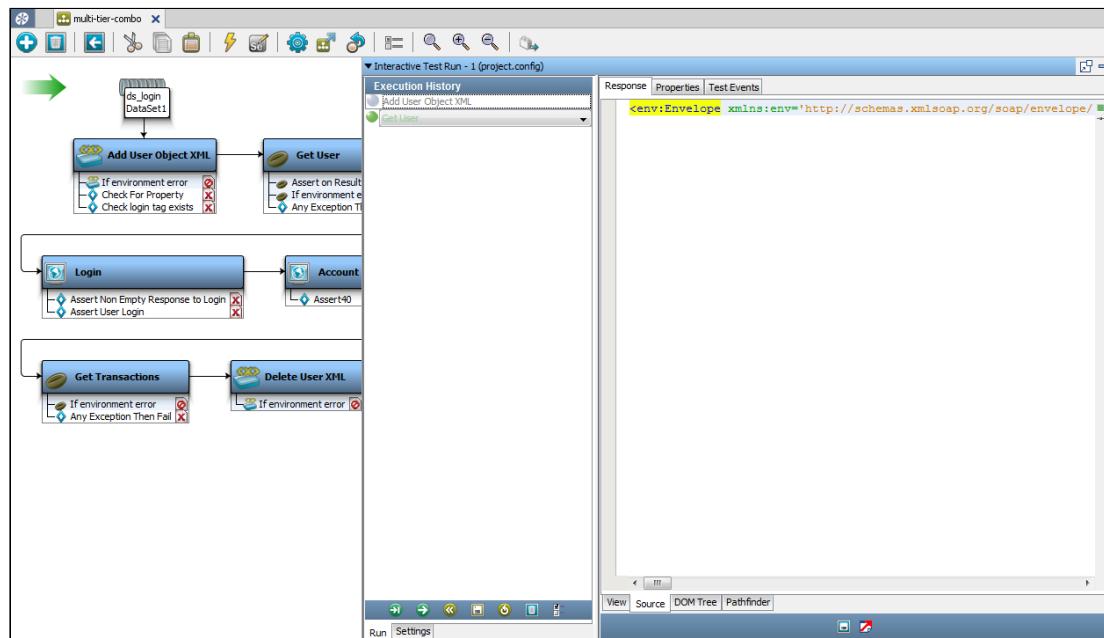
A good way to get some practice with property expressions is to build a simple test case that has a single step: an Output Log Message. This step only writes to a log and displays the response in the **Interactive Test Run (ITR) Response panel**. Therefore, you can experiment with using property expressions.

The following example uses the multi-tier-combo test case in the examples directory (multi-tier-combo.tst):



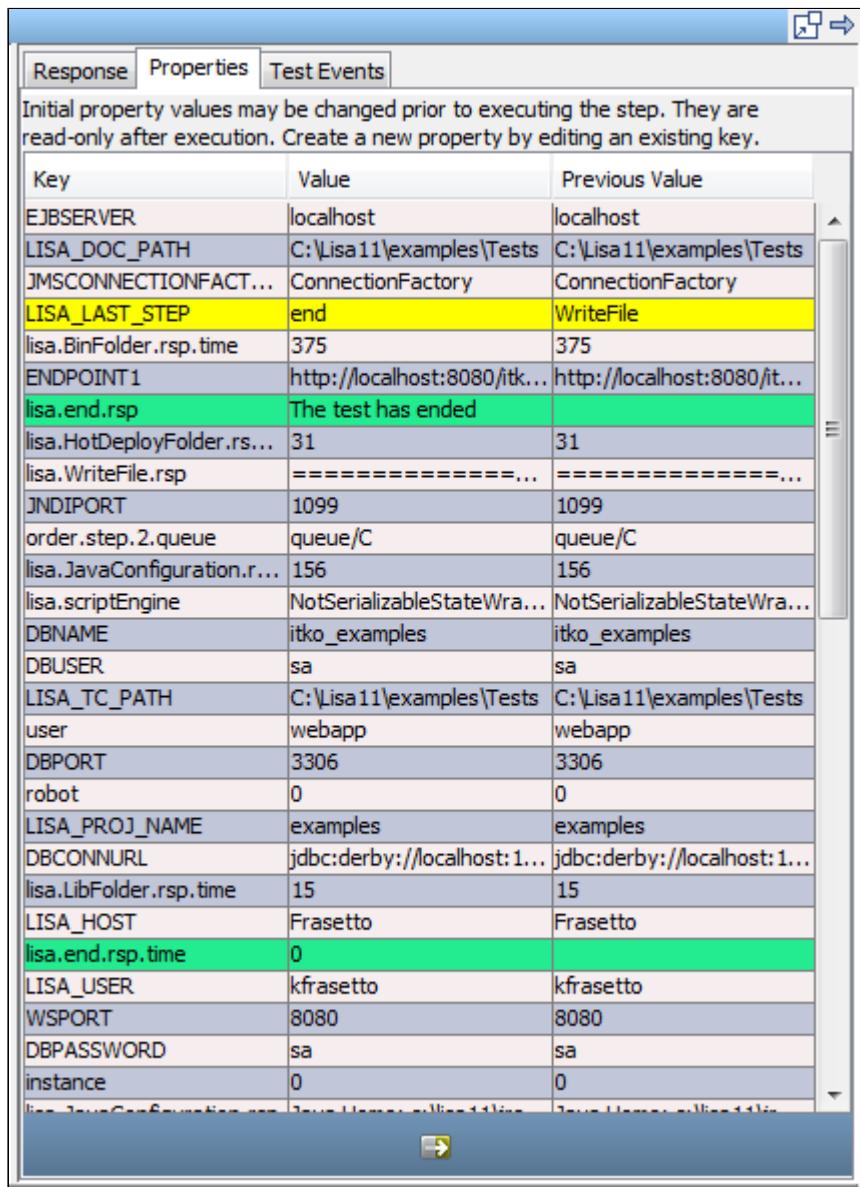
Screenshot of multi-tier-combo test case

This graphic shows our example in the ITR utility.



Screenshot of ITR running multi-tier-combo test case

These graphics show the **Properties** tab in the ITR. The tab that is shown is for the step Get User.



Screenshot 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...

Screenshot 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.

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

▪ **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\LISA\examples
```

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/LISA/examples
```

- **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/LISA/examples
```

- **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 \(lisa.properties\) \(see page 1638\)](#)
- [Custom Property Files \(see page 1671\)](#)

Use the Properties Pane

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 to open the Properties pane:
 - 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 WSOPORT. 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 \(see page 381\)](#).

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 \(see page 441\)](#).



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 \(see page 398\)](#).

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 \(see page 394\)](#)
- [Add a Configuration \(see page 395\)](#)
- [Mark a Configuration as Active \(see page 396\)](#)
- [Edit a Configuration \(see page 396\)](#)
- [Apply a Configuration While Running a Test Case \(see page 398\)](#)

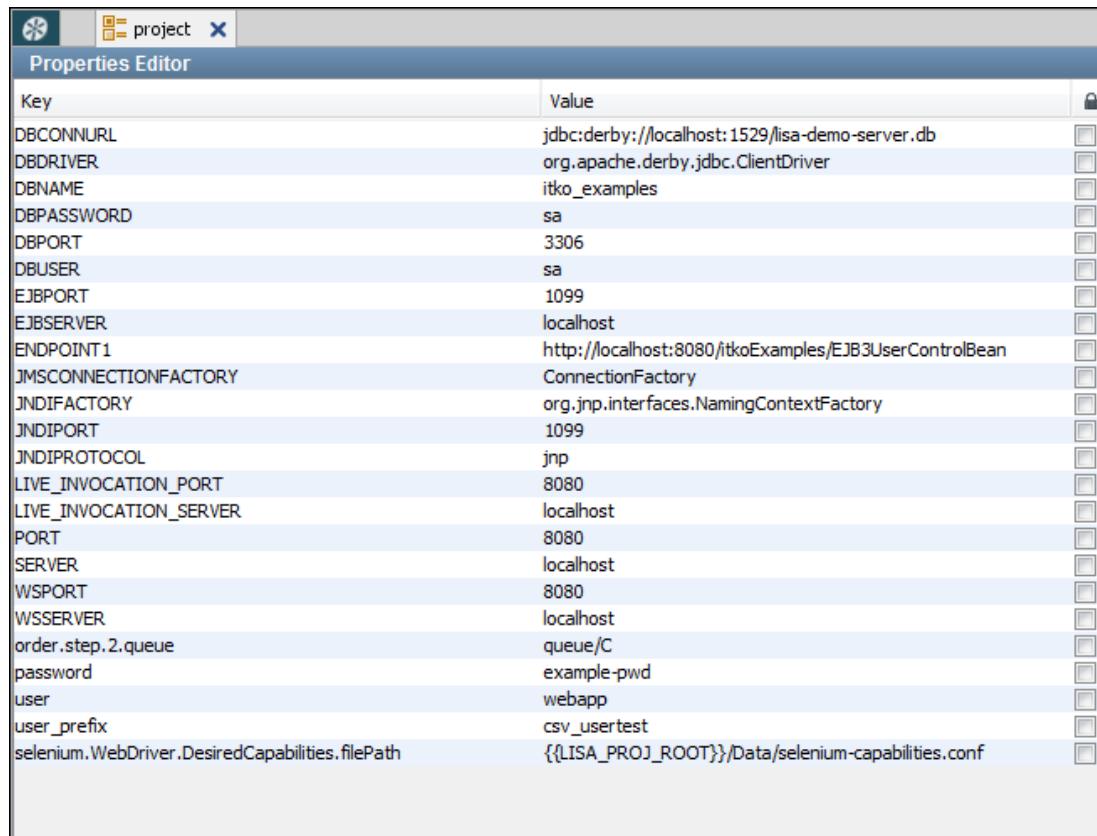
Project Configuration

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.



The screenshot shows the 'Properties Editor' window with the title bar 'Properties Editor'. The window contains a table with two columns: 'Key' and 'Value'. The 'Key' column lists various configuration parameters, and the 'Value' column shows their corresponding values. Some values are explicitly set, while others are derived from environment variables or default values. The last row shows a key 'selenium.WebDriver.DesiredCapabilities.filePath' with a value containing a placeholder for the LISA project root.

Key	Value
DBCONNURL	jdbc:derby://localhost:1529/lisa-demo-server.db
DBDRIVER	org.apache.derby.jdbc.ClientDriver
DBNAME	itko_examples
DBPASSWORD	sa
DBPORT	3306
DBUSER	sa
EJBPORT	1099
EJB SERVER	localhost
ENDPOINT1	http://localhost:8080/itkoExamples/EJB3UserControlBean
JMSCONNECTIONFACTORY	ConnectionFactory
JNDIFACTORY	org.jnp.interfaces.NamingContextFactory
JNDIPORT	1099
JNDIPROTOCOL	jnp
LIVE_INVOCATION_PORT	8080
LIVE_INVOCATION_SERVER	localhost
PORT	8080
SERVER	localhost
WSPORT	8080
WSERVER	localhost
order.step.2.queue	queue/C
password	example-pwd
user	webapp
user_prefix	csv_usertest
selenium.WebDriver.DesiredCapabilities.filePath	{LISA_PROJ_ROOT}/Data/selenium-capabilities.conf

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 \(see page 396\)](#) 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.

When the project configuration is active, it is marked orange and other configurations are black. When a secondary configuration is marked active, it is marked purple and the project configuration remains orange.



Project configs folder

Each test case in a single 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:

1. 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 \(see page 1445\)](#).

Properties Editor	
Key	Value
DBCONNURL	jdbc:derby://loc
DBDRIVER	org.apache.der
DBNAME	itko_examples
DBPASSWORD	sa
DBPORT	3306
DBUSER	sa
EJBPORT	1099
EJBSERVER	localhost
ENDPOINT1	http://localhost
JMSCONNECTIONFACTORY	ConnectionFact
JNDIFACTORY	org.jnp.interfac
JNDIPORT	1099
JNDIPROTOCOL	jnp
LIVE_INVOCATION_PORT	8080
LIVE_INVOCATION_SERVER	localhost
PORT	8080
SERVER	localhost
WSPORT	8080
WSSERVER	localhost
order.step.2.queue	queue/C
password	example-pwd
user	webapp
user_prefix	csv_usertest
HOT_DEPLOY	
MORE_JARS	
NOTIFY_ON_FAIL	
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 \(see page 550\)](#).

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 \(see page 393\)](#). 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 \(see page 399\)](#)
- [Asset Editor \(see page 400\)](#)
- [Asset Inheritance \(see page 402\)](#)
- [Runtime Scope \(see page 402\)](#)
- [Create Assets \(see page 403\)](#)
- [Create Assets from Test Steps \(see page 403\)](#)
- [Modify Assets \(see page 404\)](#)
- [Verify Assets \(see page 404\)](#)

Asset Browser

When you open a configuration, the asset browser is located to the right of the properties editor.



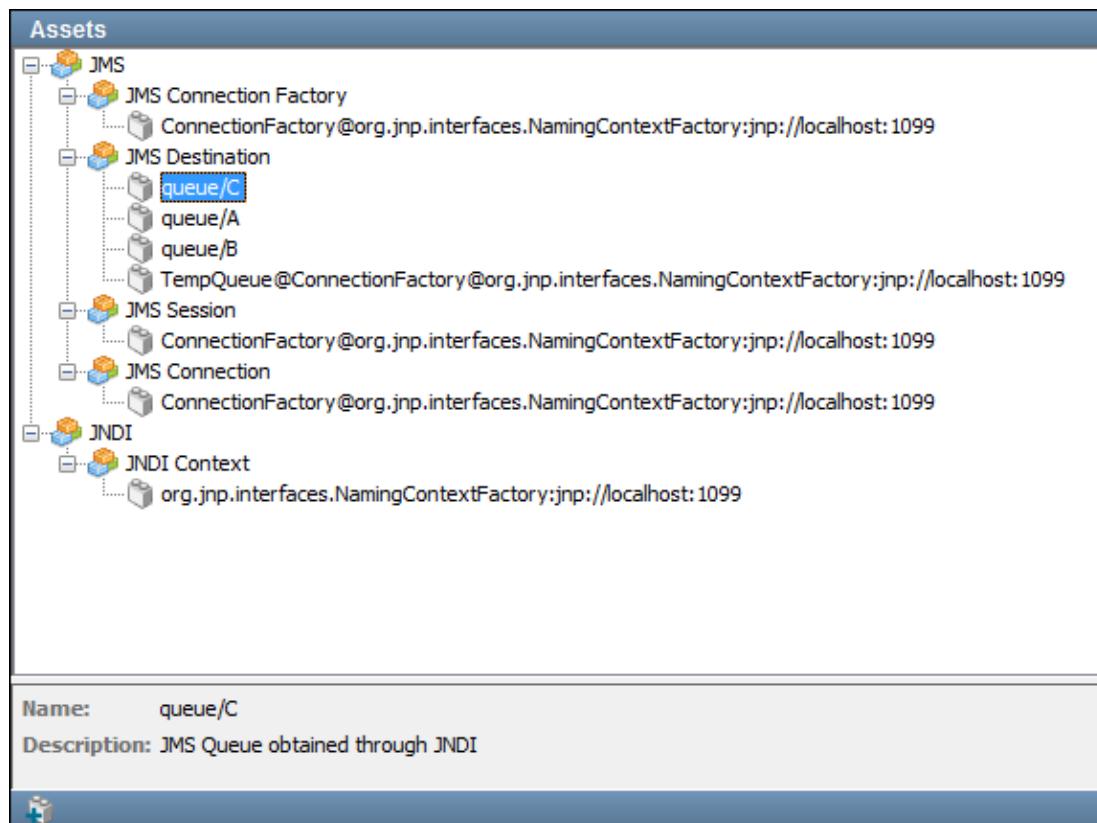
Note: If the asset browser is empty, then no assets have been created.

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.

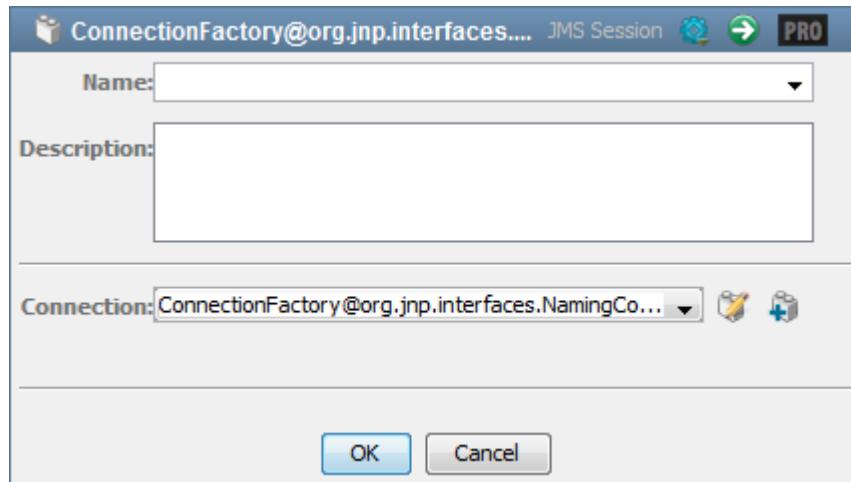
Asset Editor

You use the asset editor to perform the following tasks:

- Create assets from scratch
- Modify assets

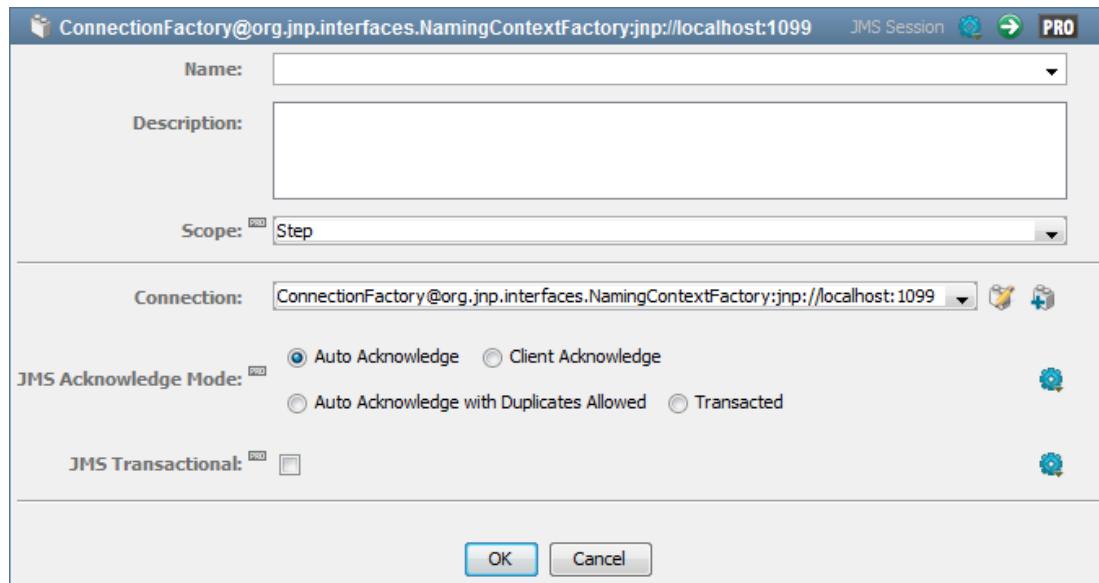
The asset editor has two modes: basic and advanced.

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 \(see page 404\)](#) the asset.

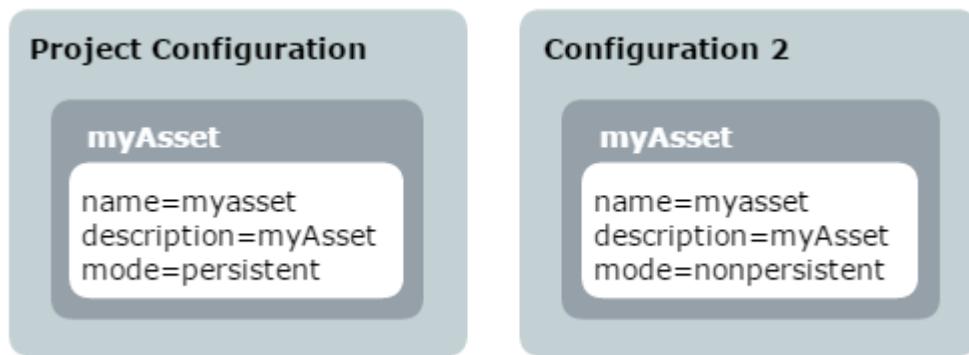
Asset Inheritance

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 \(see page 399\)](#) 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 \(see page 399\)](#) or the [asset editor \(see page 400\)](#).

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 \(see page 400\)](#) 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 \(see page 405\)](#)
- [Drag and Drop a Filter \(see page 418\)](#)

Add a Filter

You can add a filter in the following ways:

- [Add a Filter Manually \(see page 405\)](#)
- [Add a Filter from an HTTP Response \(see page 407\)](#)
- [Add a Filter from a JDBC Result Set \(see page 410\)](#)
- [Add a Filter from a Returned Java Object \(see page 415\)](#)

Add a Filter Manually

To add a filter manually, select the filter type from a list and enter the parameters for the filter.

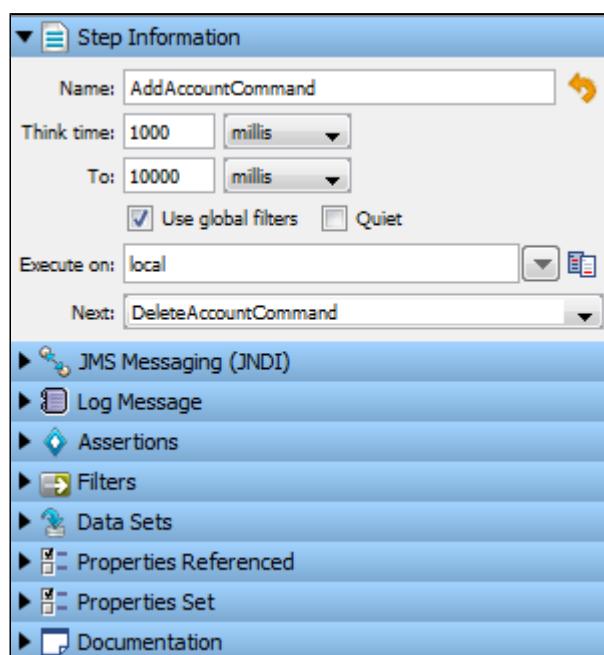
You can add two types of filters manually: global filters and step filters.

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 \(see page 1625\)](#) or [Integration Support for webMethods Integration Server \(see page 1625\)](#)
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 \(see page 1587\)](#).

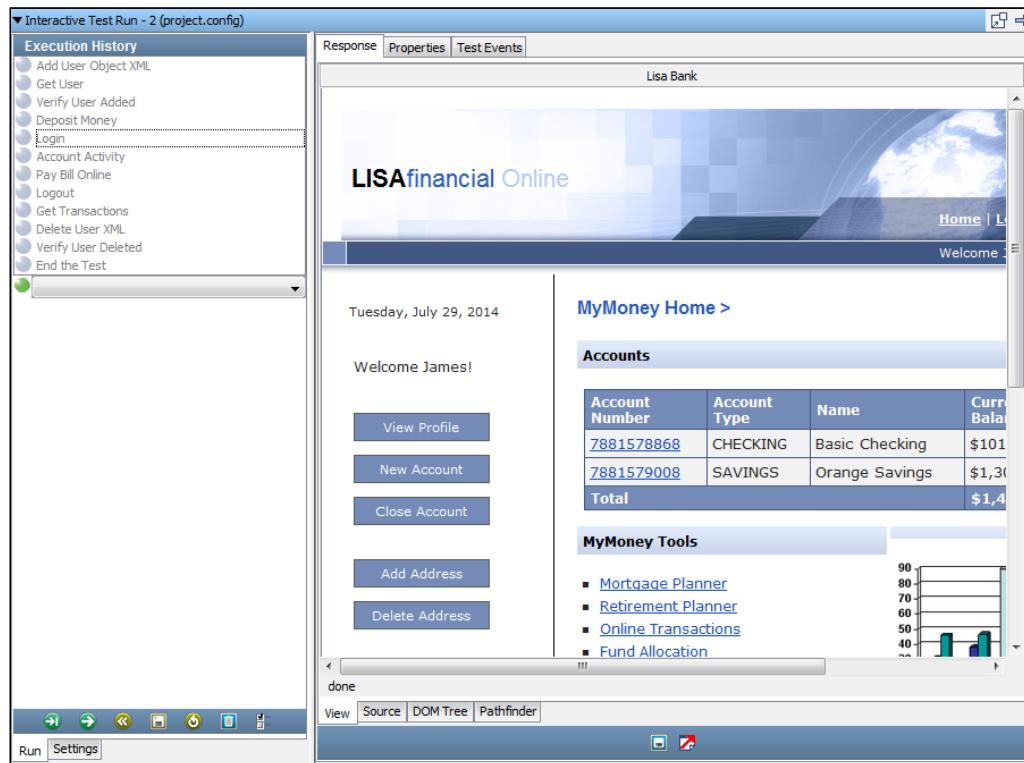
Add a Filter from an HTTP Response

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).

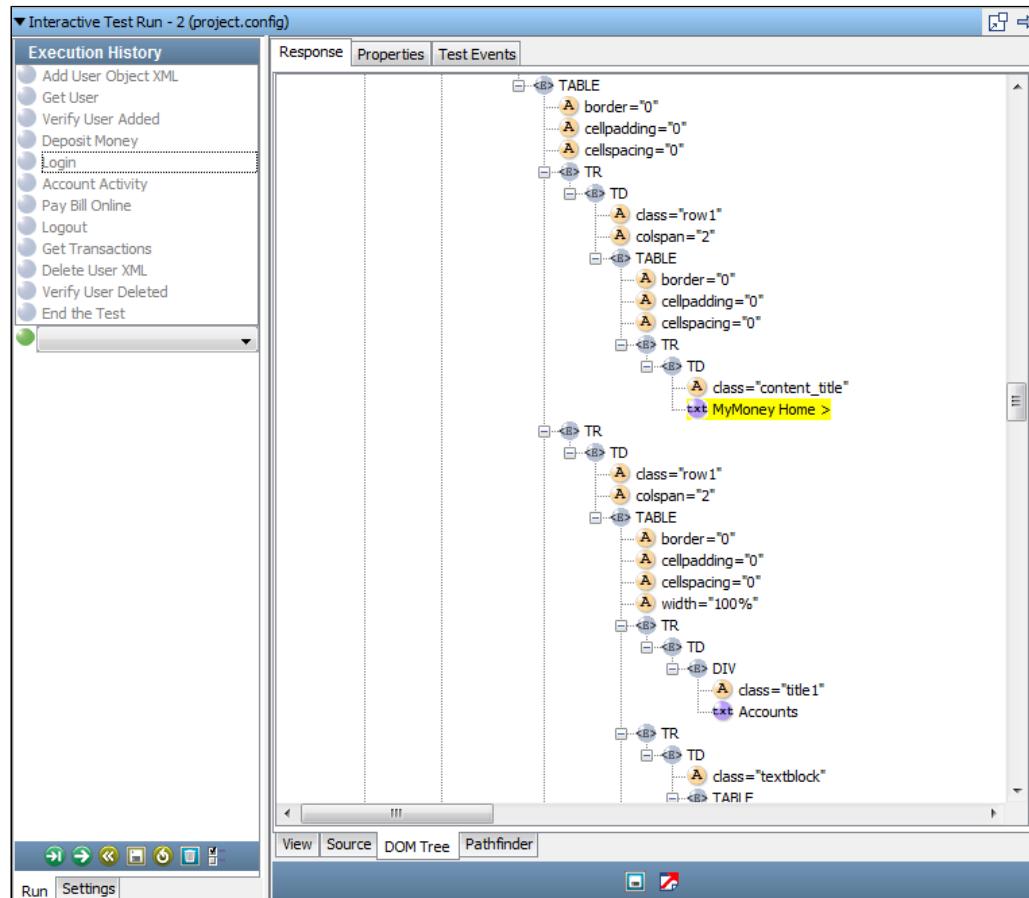
Follow these steps:

1. Run the multi-tier-combo test case in the ITR, and then select the Login test step.



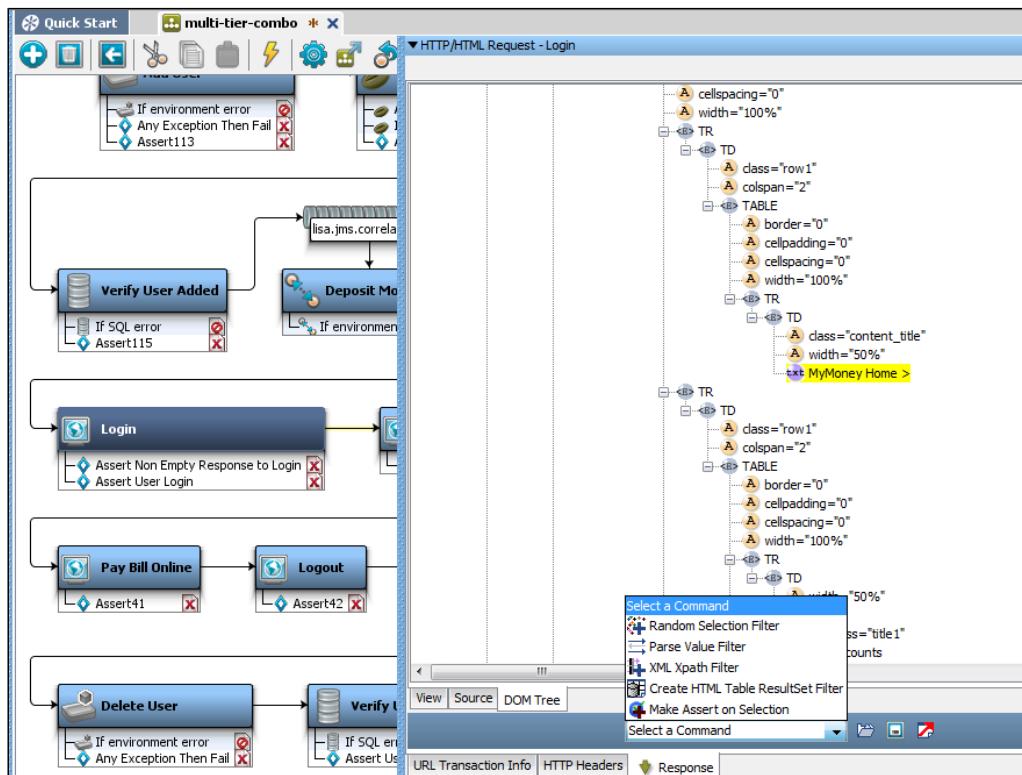
Screenshot of multi-tier-combo in ITR with Login step selected

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.



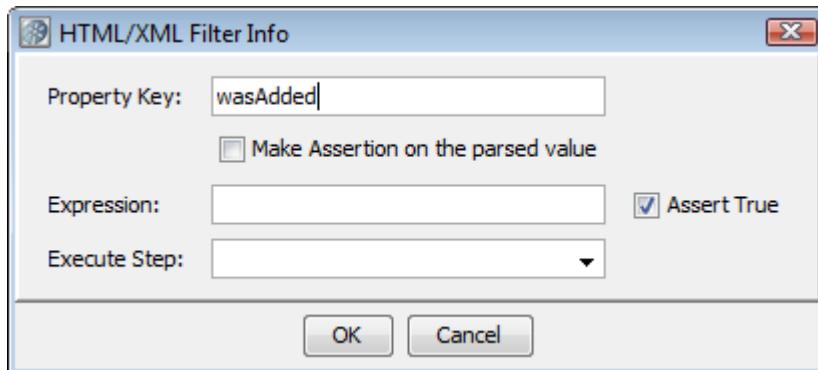
Screenshot of multi-tier-combo in ITR, DOM Tree view of Login step

4. To apply an inline filter, double-click the login step in the model editor to open the HTTP /HTML Step Editor.



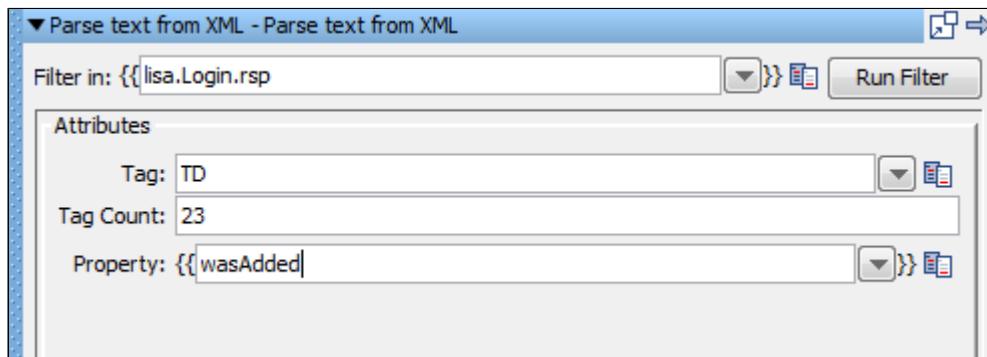
Screenshot of multi-tier-combo in ITR

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.

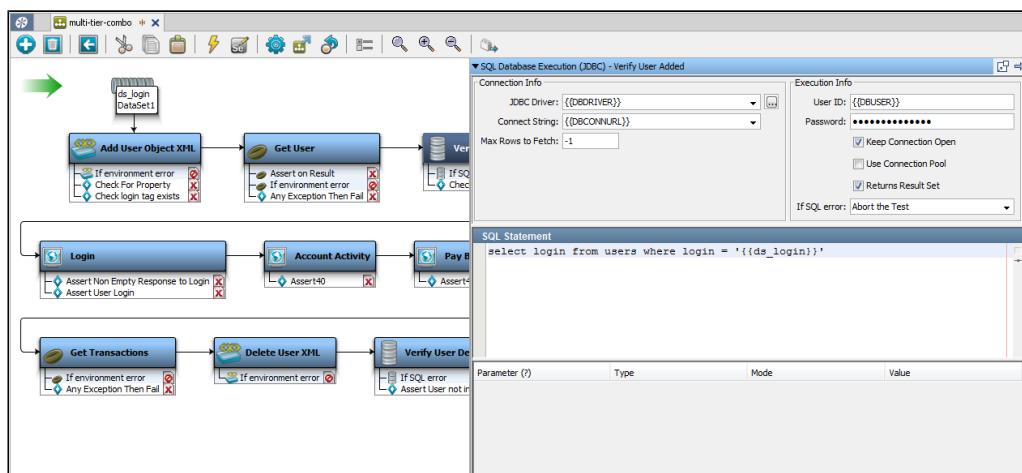
Add a Filter from a JDBC Result Set

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.

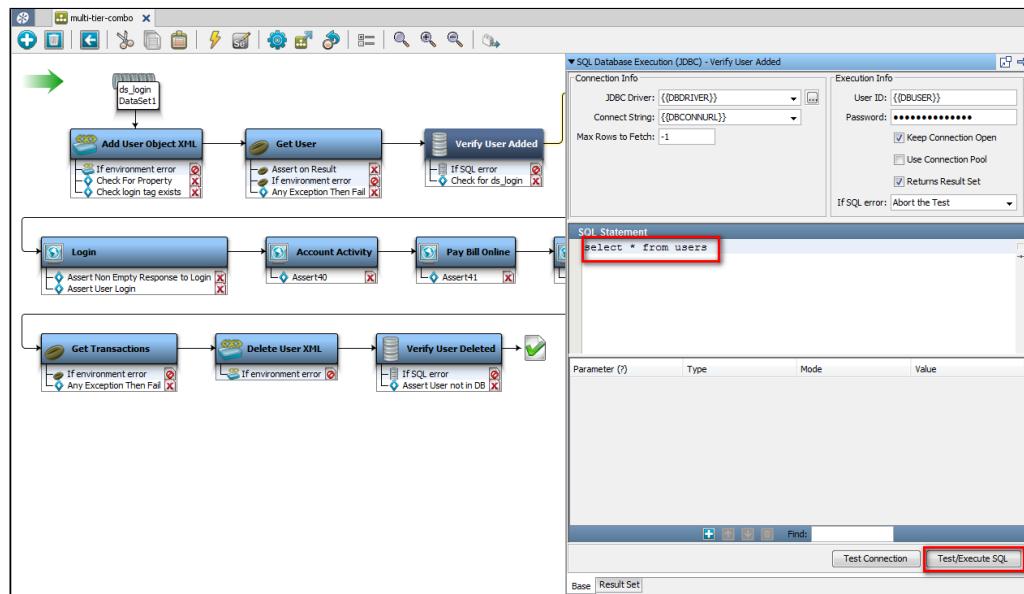
Follow these steps:

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	tIDAdNa...	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 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*.

▼ 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	ODPiKuNIr...	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	RQIl0Ldp...	1	Boaty	Rabbit	boaty@ra...	333-4521	1	616-51-0...	
itko	qUqP5cyx...						650-234-...	140-72-2...	
lisa_simpson	60fAFoq+						123-456-...	1	295-20-0...
virtuser	nU4eI71b...						123-456-...	1	297-55-9...
demo	89yJPVNm						817-433-...	1	677234567

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 - theLogin

Filter in: {{ lisa.Verify User Added.rsp }} Run Filter

Attributes

Column (1-based or Name): 1
Row (0-based): 19
Property: {{ theLogin }}

Filter Run Results

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:

- From the result set, select the two values in two different columns from the same row, using the Ctrl key.

▼ SQL Database Execution (JDBC) - Verify User Added

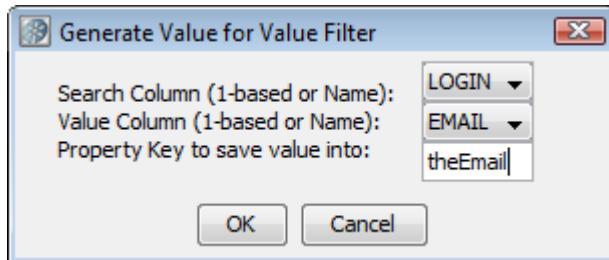
Result Set

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@itk...	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	ODPIKuNIr...	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	RQIIoLDp...	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 a Filter from a JDBC Result Set - Result set with two cells selected

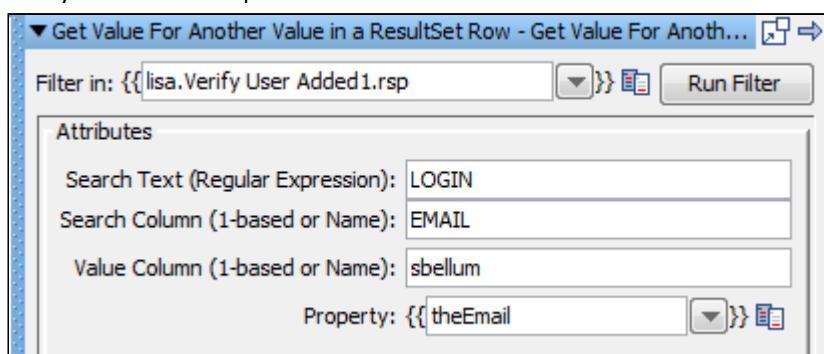
- Select **Filter for a value** and then get another column value filter using the icon. To create this filter, select two cells in the same row. One is the search column and the other is the column whose value you want to extract.
- In the dialog that opens, select or reassign the columns for the search and the value.
- 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.

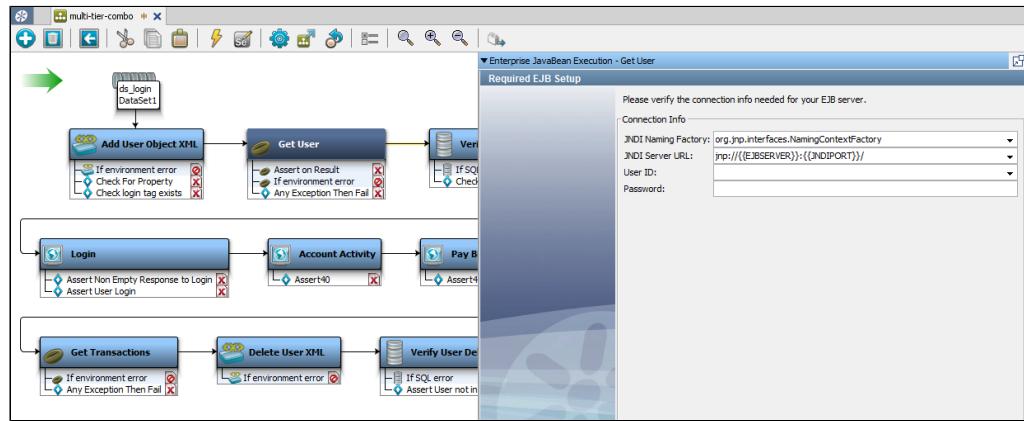
Add a Filter from a Returned Java Object

When the test step result is a Java object, you can use the **Inline Filter** panel in the **Complex Object Editor** (see page 443) to filter the returned value directly from the method call.

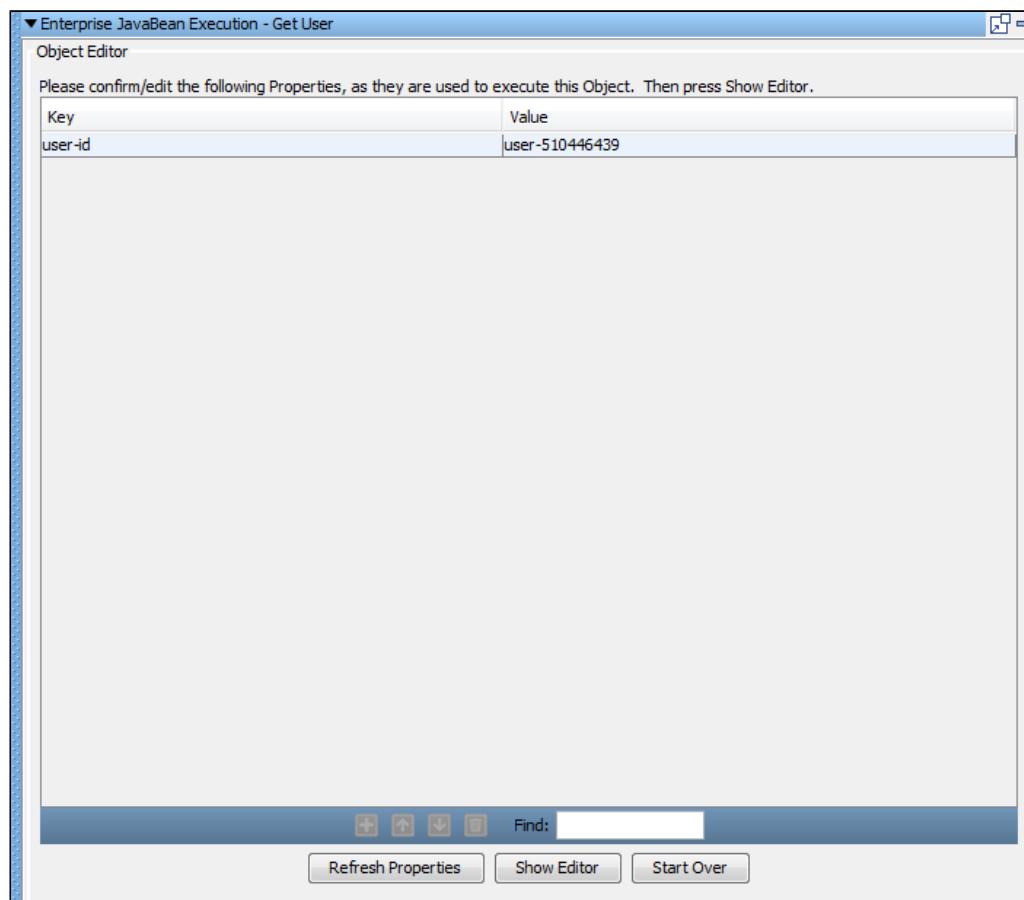
This example uses the Get User step (EJB step) in the multi-tier-combo test case in the examples directory.

Follow these steps:

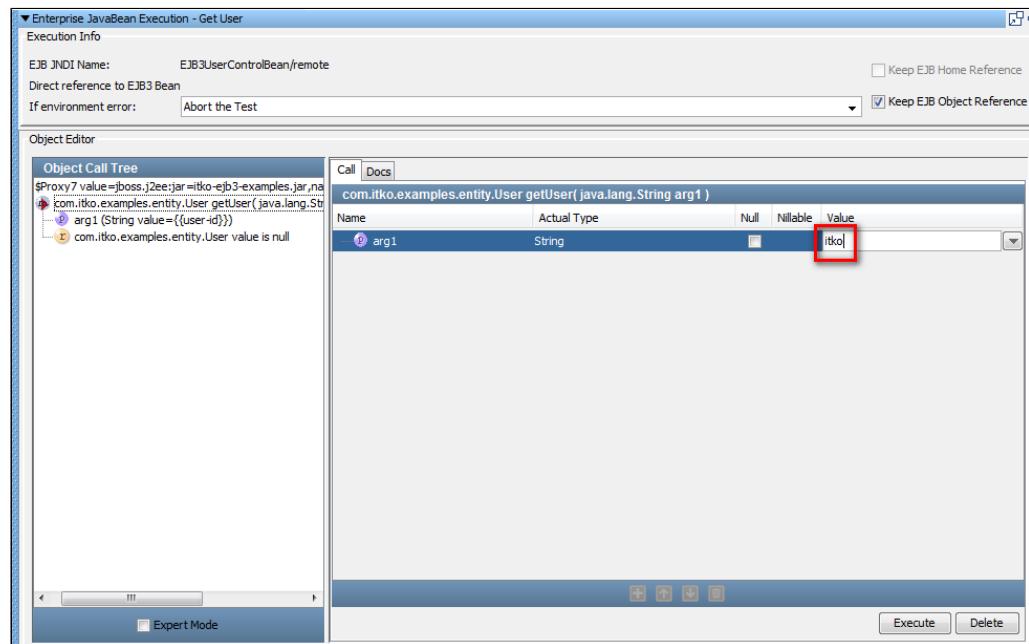
1. To open the step editor for the Get User step, double-click the step.



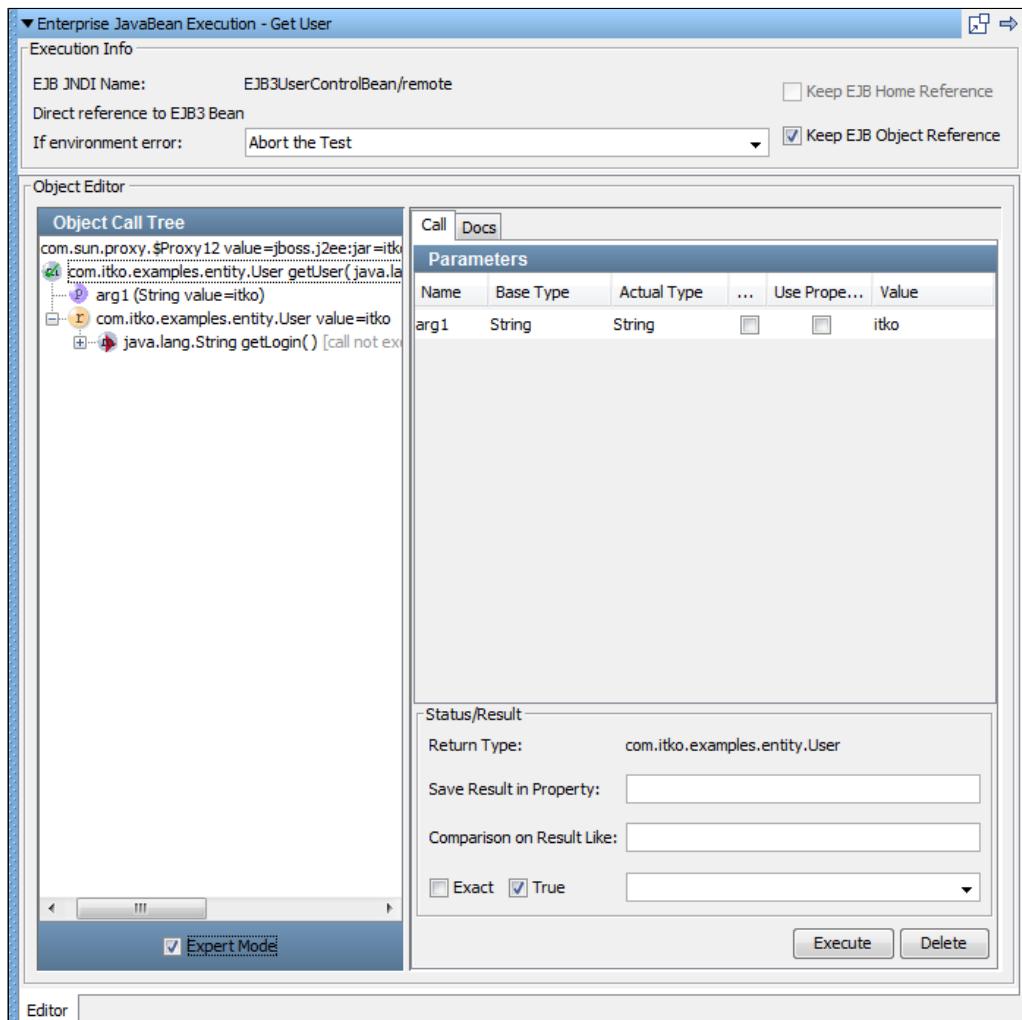
2. Click **Next**. On the next window, click **Finish**.



3. To open the Object Call Tree, click **Show Editor**.



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.



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 \(see page 443\)](#).

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 \(see page 420\)](#)
- [Assertions Toolbar \(see page 432\)](#)
- [Reorder an Assertion \(see page 432\)](#)
- [Rename an Assertion \(see page 432\)](#)
- [Drag and Drop an Assertion \(see page 432\)](#)
- [Configure the Next Step of an Assertion \(see page 433\)](#)

Add an Assertion

You can add an assertion in the following ways:

- [Add an Assertion Manually \(see page 420\)](#)
- [Add an Assertion from an HTTP Response \(see page 423\)](#)
- [Add an Assertion from a JDBC Result Set \(see page 427\)](#)
- [Add an Assertion for Returned Java Object \(see page 429\)](#)
- [Add an Assertion to a Mobile Test Step \(see page 430\)](#)

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

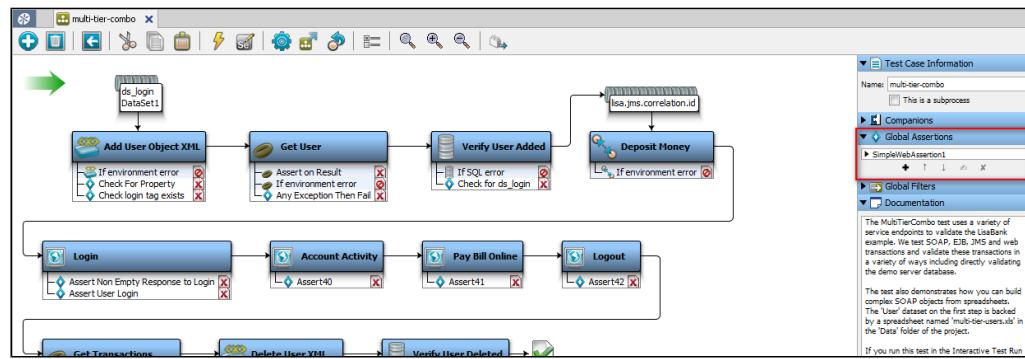
Follow these steps:

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.

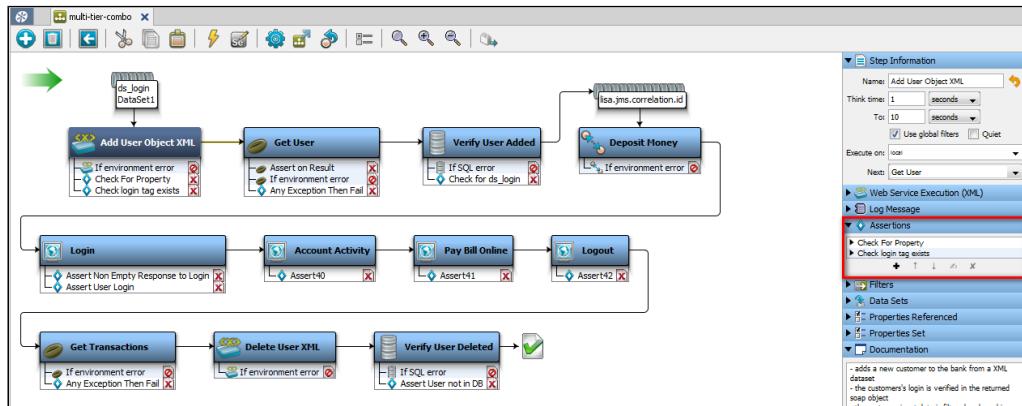


Add a Step Assertion

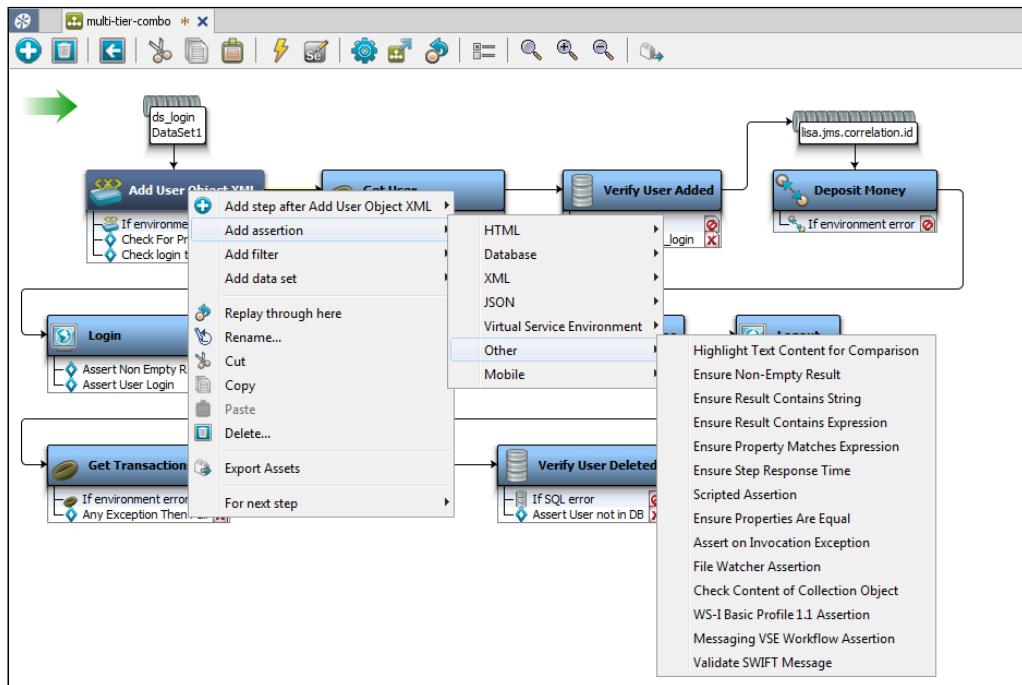
Follow these steps:

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.

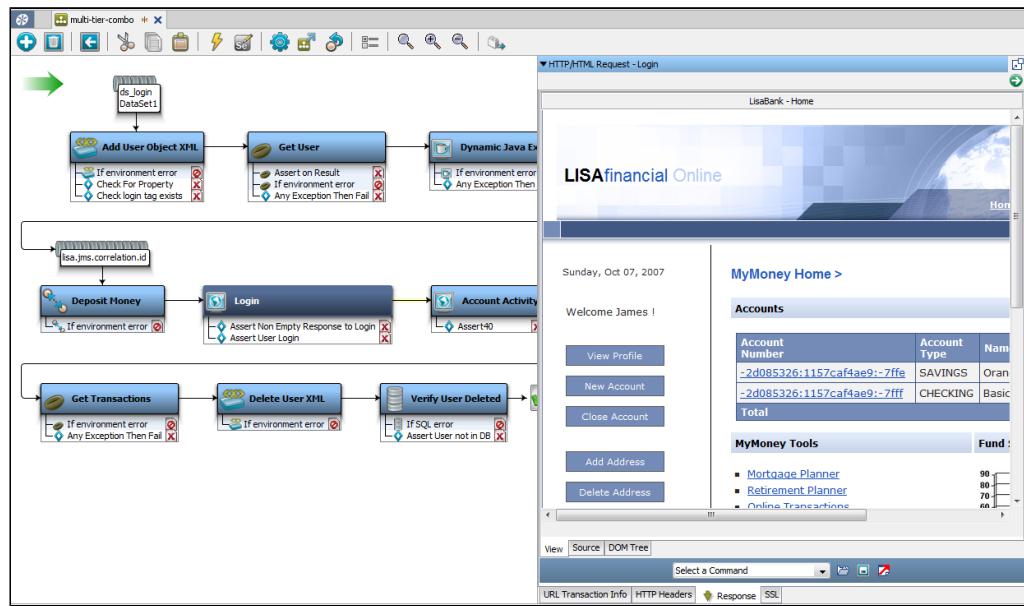
Add an Assertion from an HTTP Response

When you have access to the response from an HTTP-based step, you can use the response to add an assertion directly.

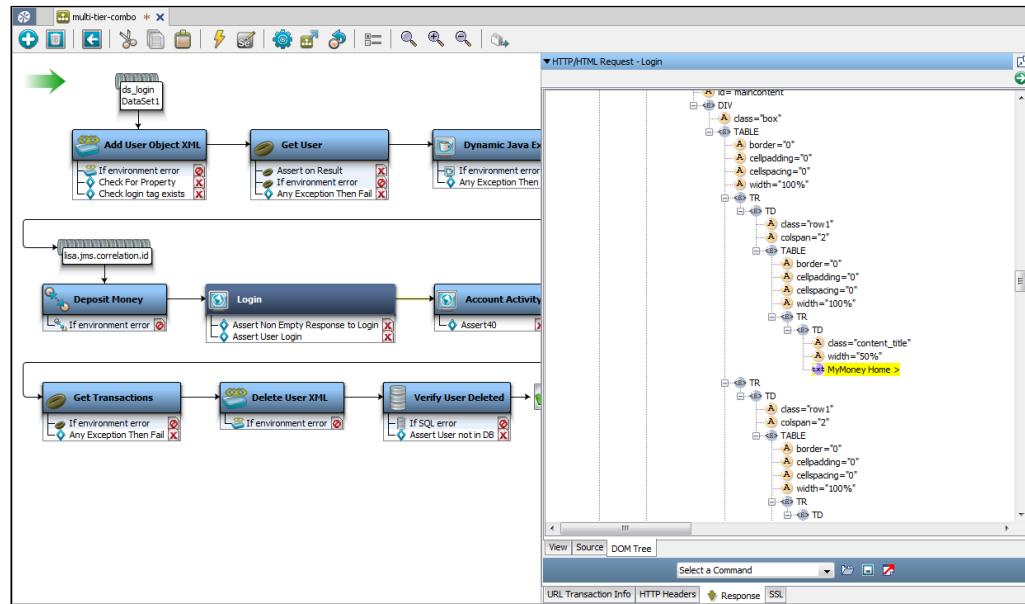
This example of the HTTP/HTML response uses the login step in the multi-tier-combo test case. The point of this example is to test whether the text "MyMoney Home" appears in the response.

Follow these steps:

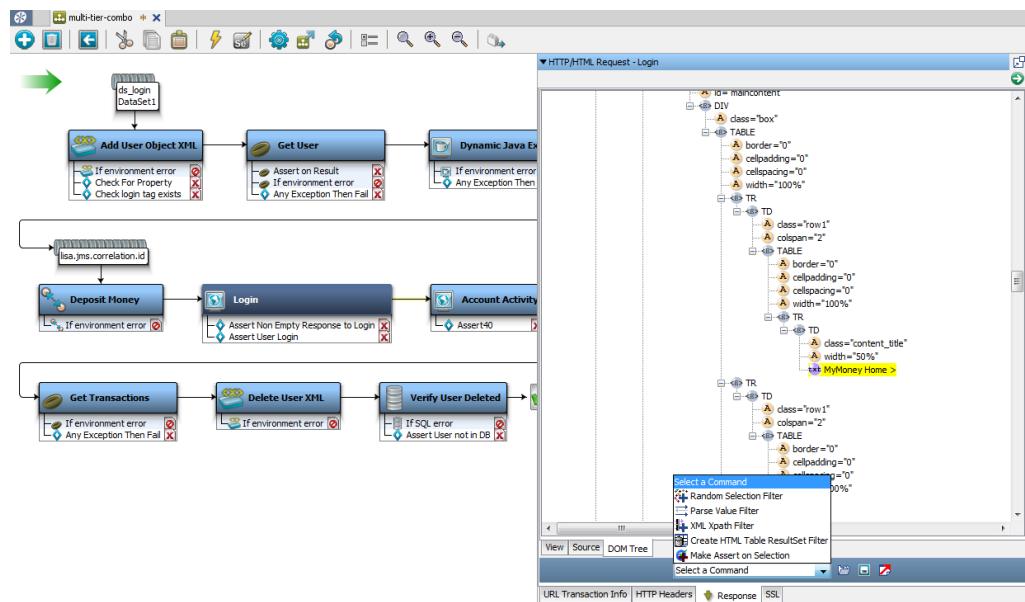
1. Run the multi-tier-combo test case in the ITR.
2. Double-click the Login step in the model editor.



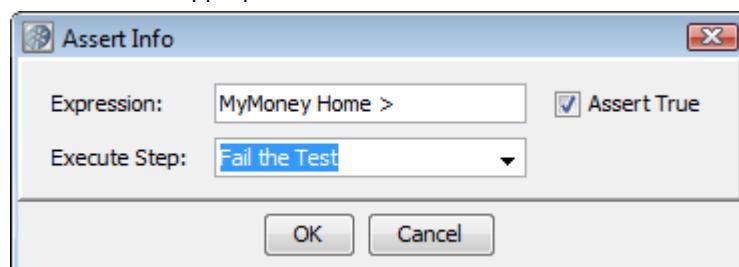
3. Select the text "MyMoney Home" in the **View** tab.
4. To view and verify that this text is selected in the tree, click the **DOM Tree** tab.



5. From the **Select a Command** pull-down menu at the bottom of the panel, select **Make Assert on Selection**.



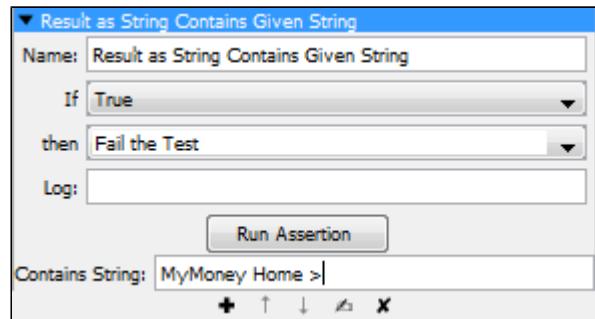
6. In the displayed window, enter the expression that you expect the selected text to match, then select the appropriate assertion behavior.



In this example, the assertion fires if the text "MyMoney Home" is not present, and then redirects to the fail step.

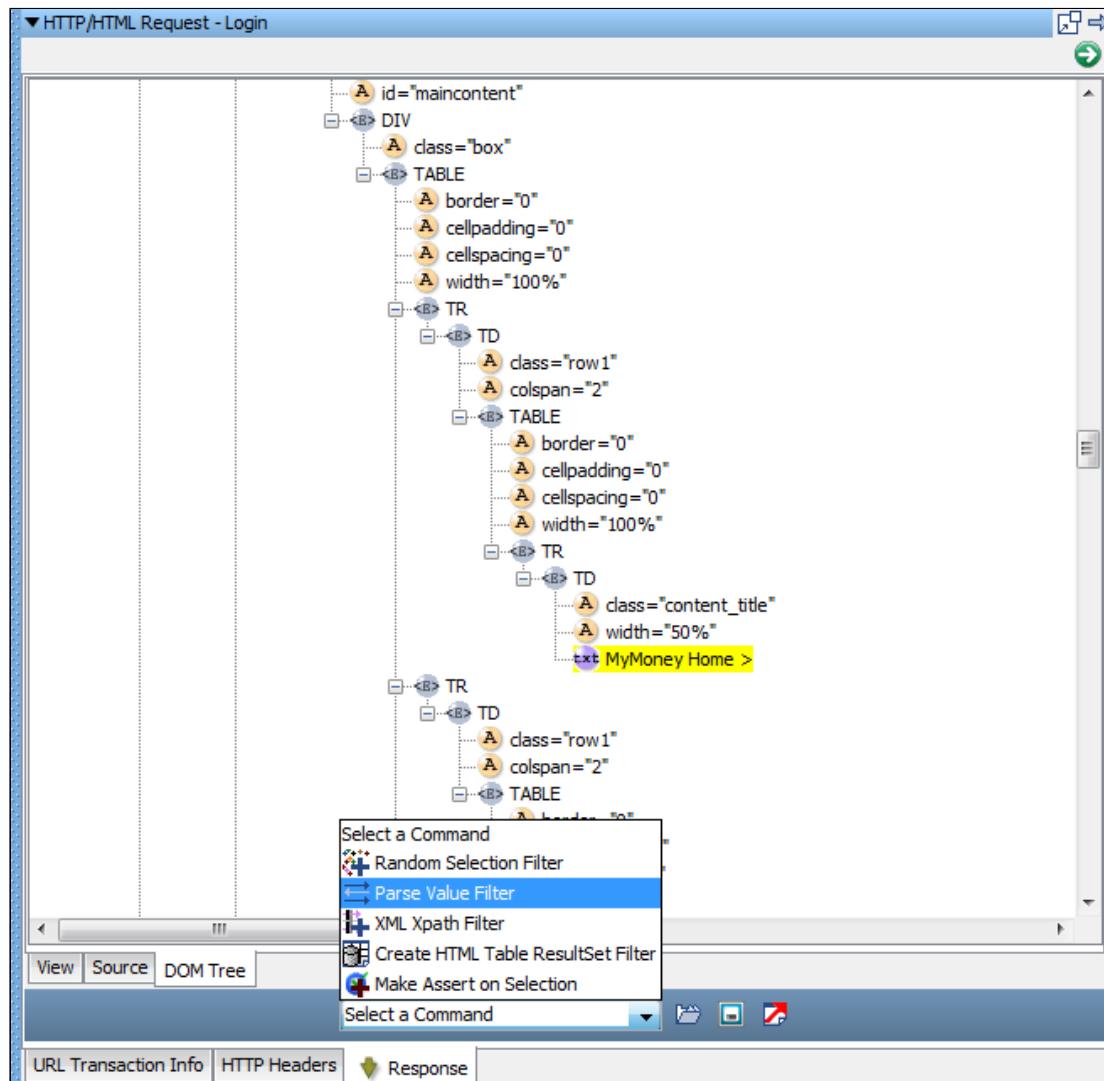
7. Click **OK** to save the assertion.

The assertion that was generated appears in the login step.

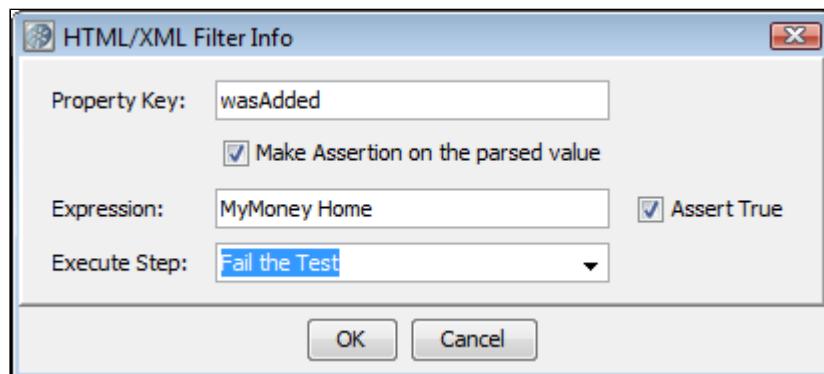


Running one Filter and one Assertion

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.

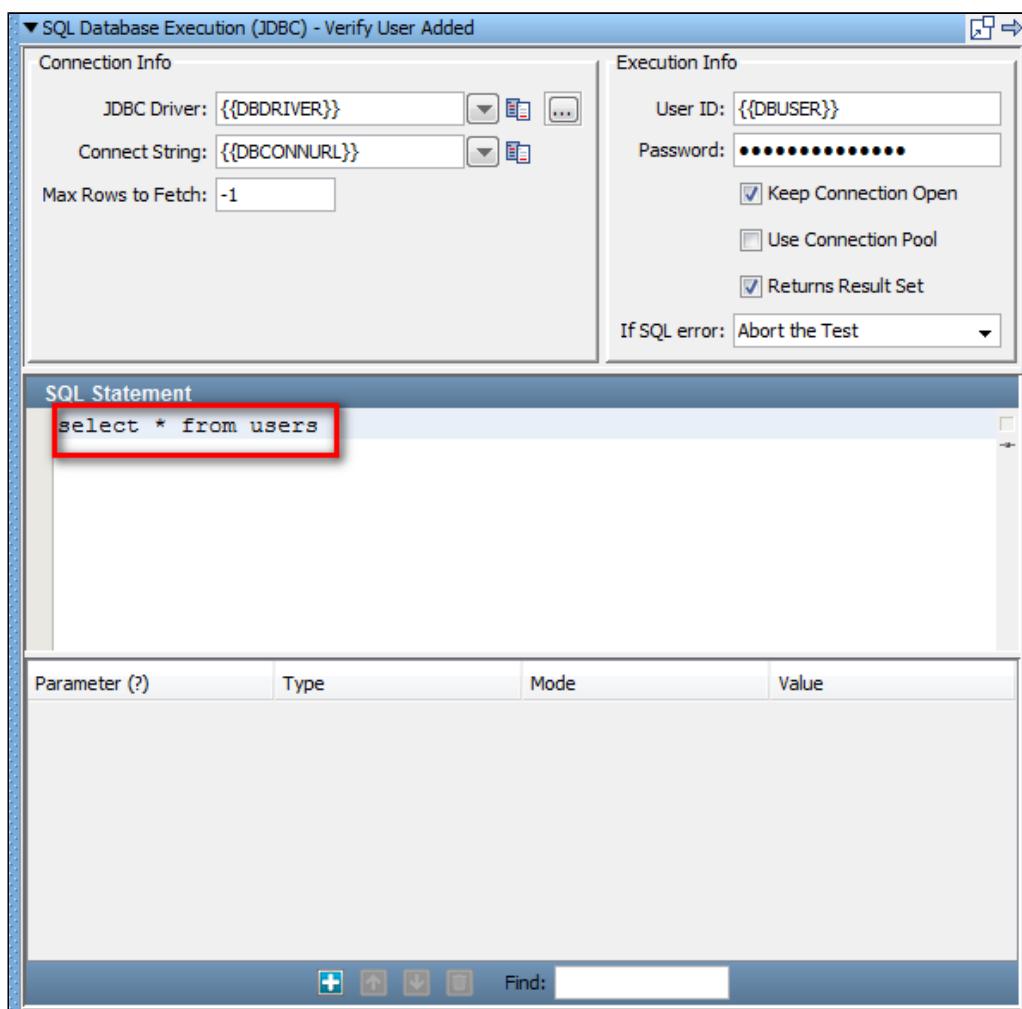
Add an Assertion from a JDBC Result Set

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**).

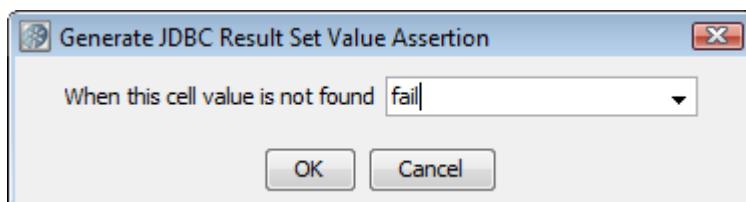
▼ SQL Database Execution (JDBC) - Verify User Added

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@itk...	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@it...	232-4345	1	614-40-1...
wpiece	/UuJ0Me...	1	Warren	Piece	wpiece@it...	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@it...	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@it...	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](#) (see page 1478) in the Verify User Added step.

▼ Ensure JDBC Result Set Contains Expression - Ensure JDBC Result Set Contains Expression

Name:	It Set Contains Expression	If	False	then	Fail the Test
Log:	<input type="text"/>				
Column (1-based or Name):	<input type="text" value="1"/>				
Regular Expression:	<input type="text" value="sbellum"/>				



Note: The same assertion capabilities are available when a JDBC result set is displayed in the step editor.

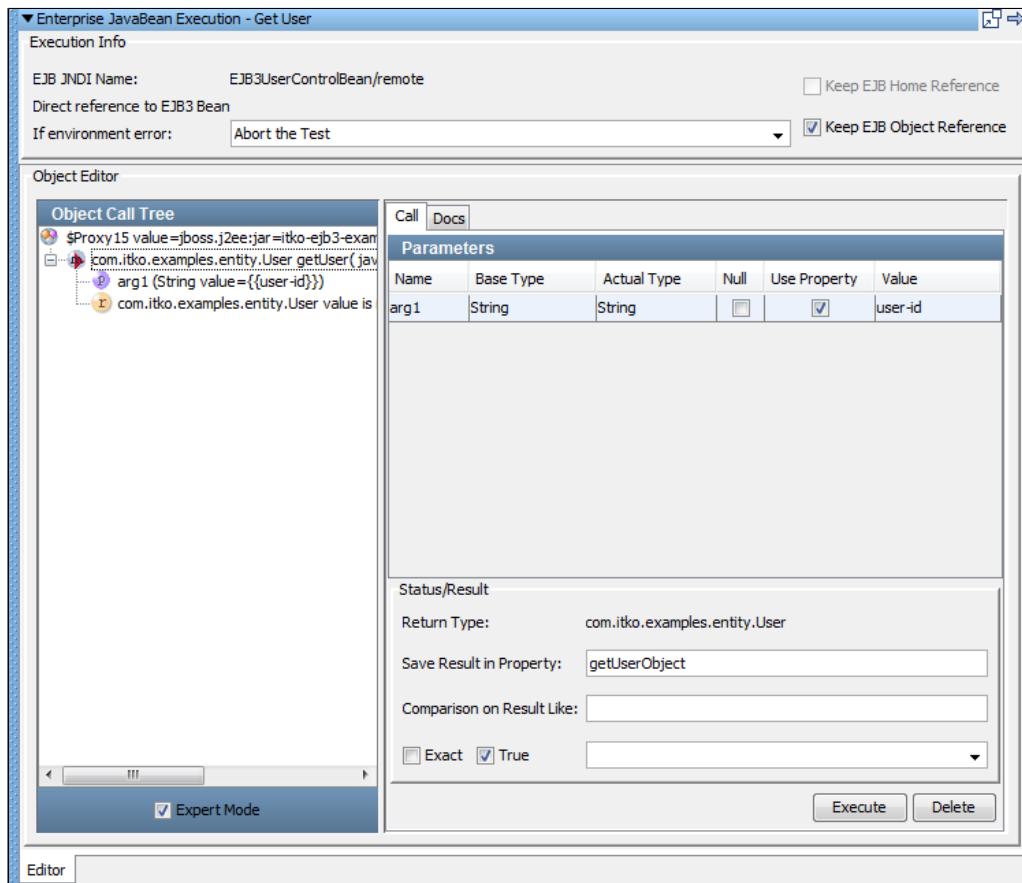
Add an Assertion for Returned Java Object

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.

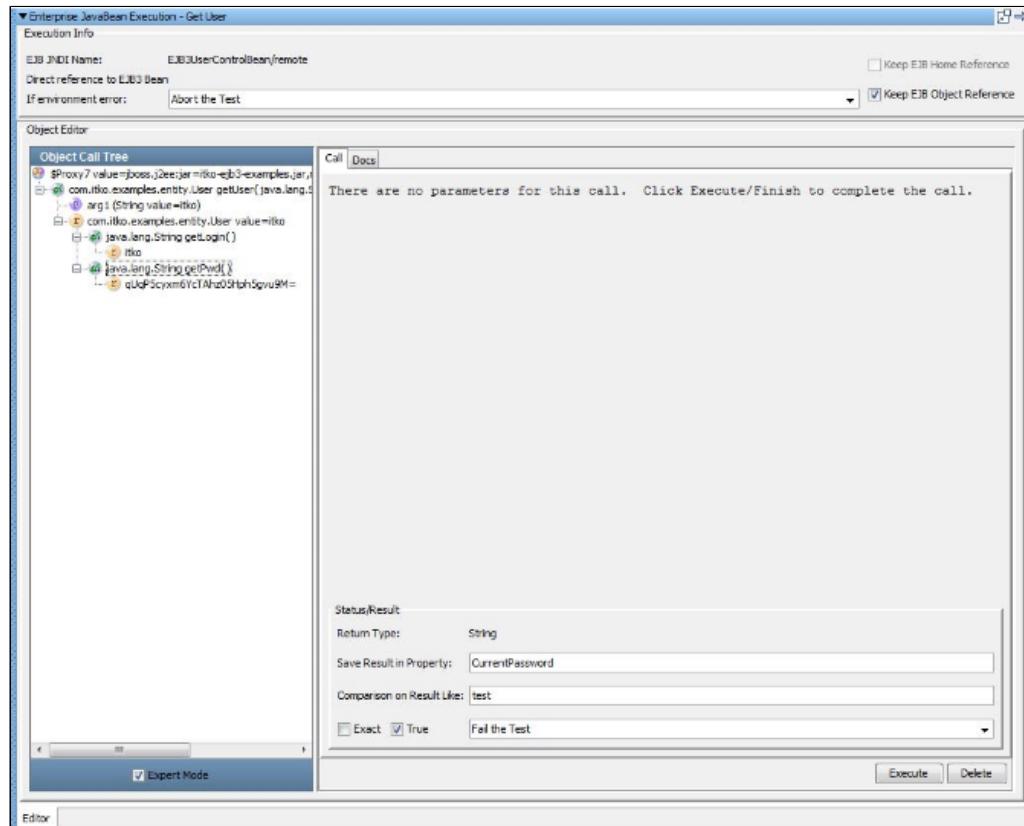
Follow these steps:

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 \(COE\) \(see page 443\)](#).

Add an Assertion to a Mobile Test Step

Contents

- [Add an Assertion \(see page 431\)](#)
- [Select and Edit an Assertion \(see page 431\)](#)

You can apply assertions to each action that is contained in a mobile test step.



Note: One test step contains as many gestures (tapping, swiping) as needed to complete the step. These sub-steps, or actions, appear in the Actions table when you double-click a test step.

Add an Assertion

Follow these steps:

1. Open the mobile test case that you want to update.
2. To view the details of each step:
 - a. Click the test step that you want to review.
 - b. Click **Mobile testing step** in the element tree on the right to expand the details for the step. You can also double-click the test step.
The Mobile Testing Step tab opens and shows a screenshot of the test application. The Actions section at the top of the tab shows the individual actions that are performed in the test step.
 - c. To view the screenshot that is associated with a specific action, click the action in the Actions section.
3. Right-click the screenshot or a specific screen element in the bottom portion of the tab.
4. Click **Add Assertion** and select the assertion that you want to add.
The selected assertion is added to the test step.
5. Click **Save**.

Select and Edit an Assertion

Follow these steps:

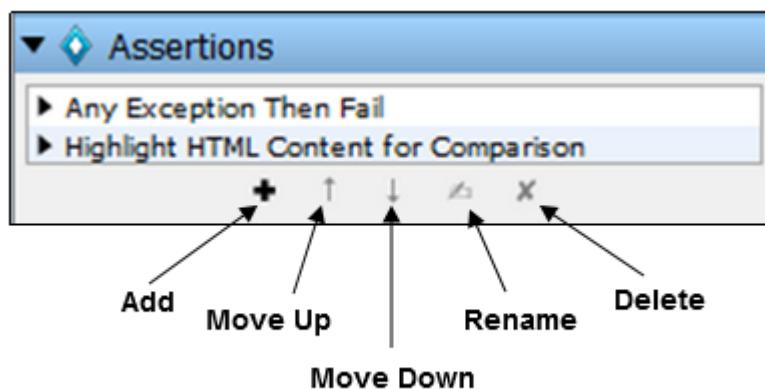
1. Close the Mobile test step editor.
2. Click **Assertions** in the element tree on the right to view the assertions for the selected test step.
The Assertions tab expands.
3. Double-click the assertion that you want to modify.
The Assertion editor opens.
4. Complete the fields for the selected assertion.
The editor is unique for each type of assertion. For more information about a specific type of assertion, see the following pages:
 - [Ensure Same Mobile Screen \(see page 1508\)](#)
 - [Ensure Screen Element Matches Expression \(see page 1509\)](#)

- Ensure Screen Element is Removed (see page 1509)

5. Click **Save**.

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

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

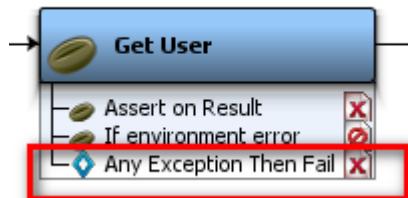
To drag and drop assertions in the model editor from one test step to another test step:

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 \(see page\)](#) 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 \(see page 434\)](#)
- [Random Data Sets \(see page 437\)](#)
- [Example Scenarios \(see page 437\)](#)
- [Add a Data Set \(see page 438\)](#)
- [Rename a Data Set \(see page 440\)](#)
- [Data Set Next Step Selection \(see page 440\)](#)
- [Data Sets and Properties \(see page 441\)](#)

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

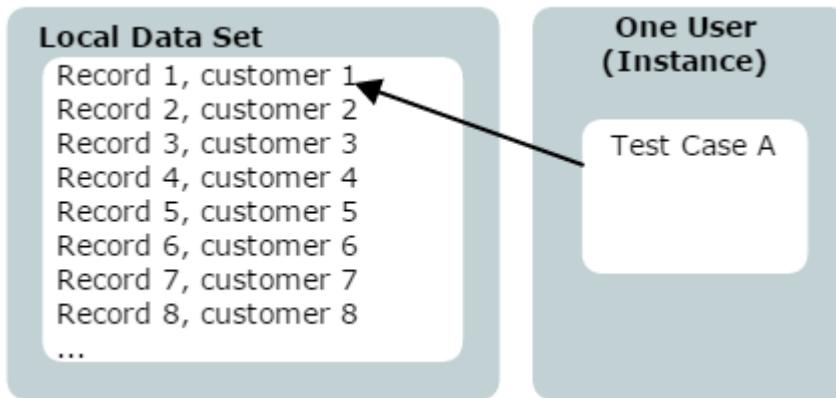
This example has the following components:

- 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

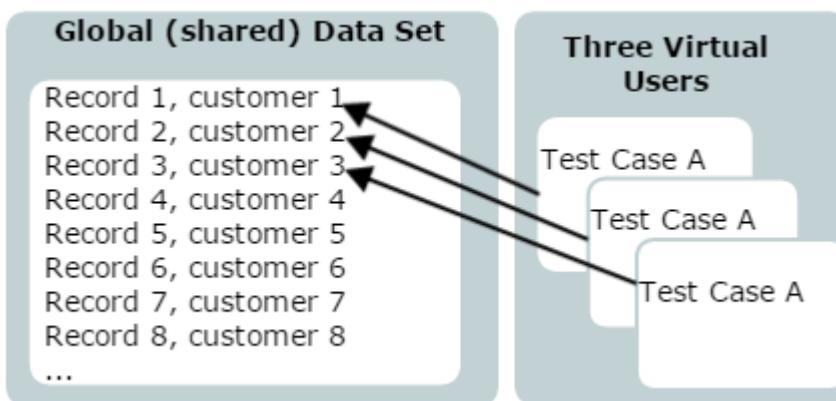
A single run (one vuser): Test Case A, gets the first row of data: Record 1, customer 1



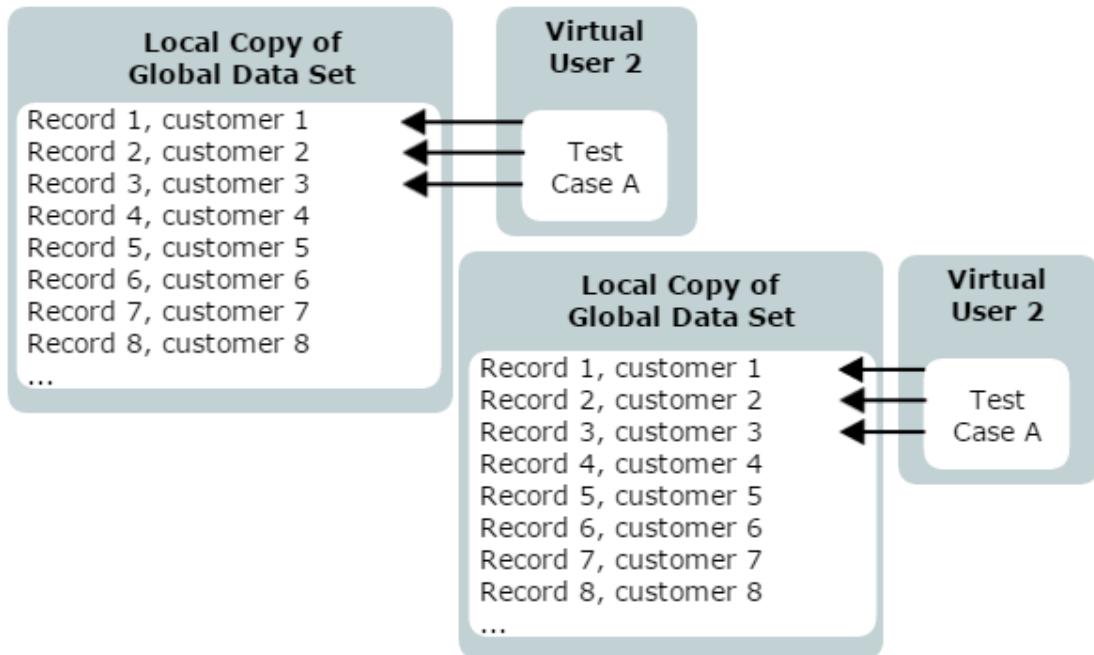
A data set that is shared across virtual users is *global*.

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.

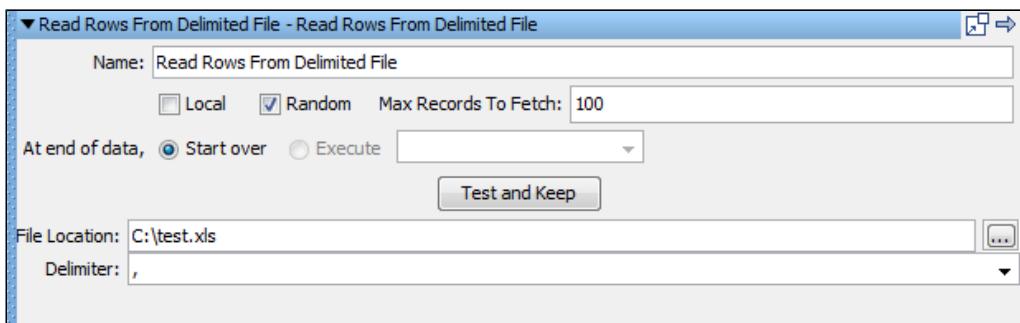
When a random wraps a data set, n copies of the data set are added to the random list, where $n = \text{Max Number of rows}$. So when $n=10$, DevTest makes 10 copies of the rows from the data set.

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

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.

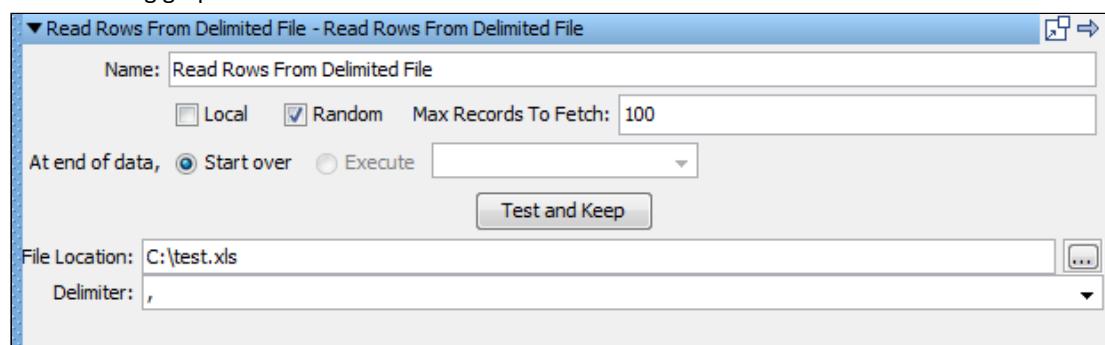
Add a Data Set

Follow these steps:

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:

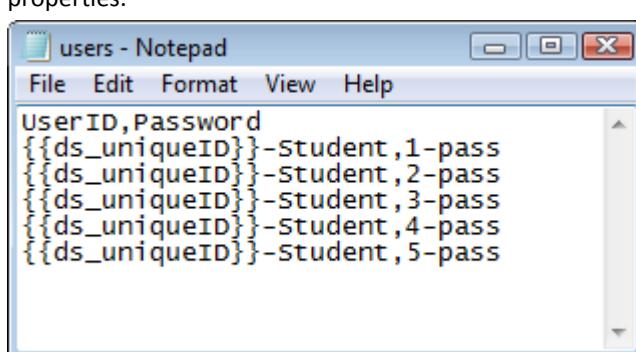
- **File Location**

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.



```

users - Notepad
File Edit Format View Help
UserID,Password
{{ds_uniqueID}}-Student,1-pass
{{ds_uniqueID}}-Student,2-pass
{{ds_uniqueID}}-Student,3-pass
{{ds_uniqueID}}-Student,4-pass
{{ds_uniqueID}}-Student,5-pass

```

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.

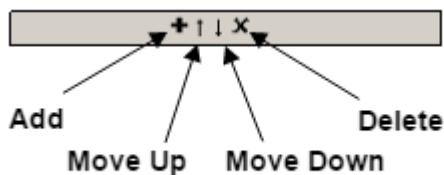
- Data set values can contain properties. These values are evaluated when the value is read. For example, we could set up a login value in the data set as student _1. Then, if the current value of student is **Bart**, the resulting data value becomes **Bart_1**.
- Local data sets can use properties from 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 \(see page 442\)](#)
- [DevTest Hooks \(see page 442\)](#)

Add a Companion

Follow these steps:

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 \(see page 1533\)](#).

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)

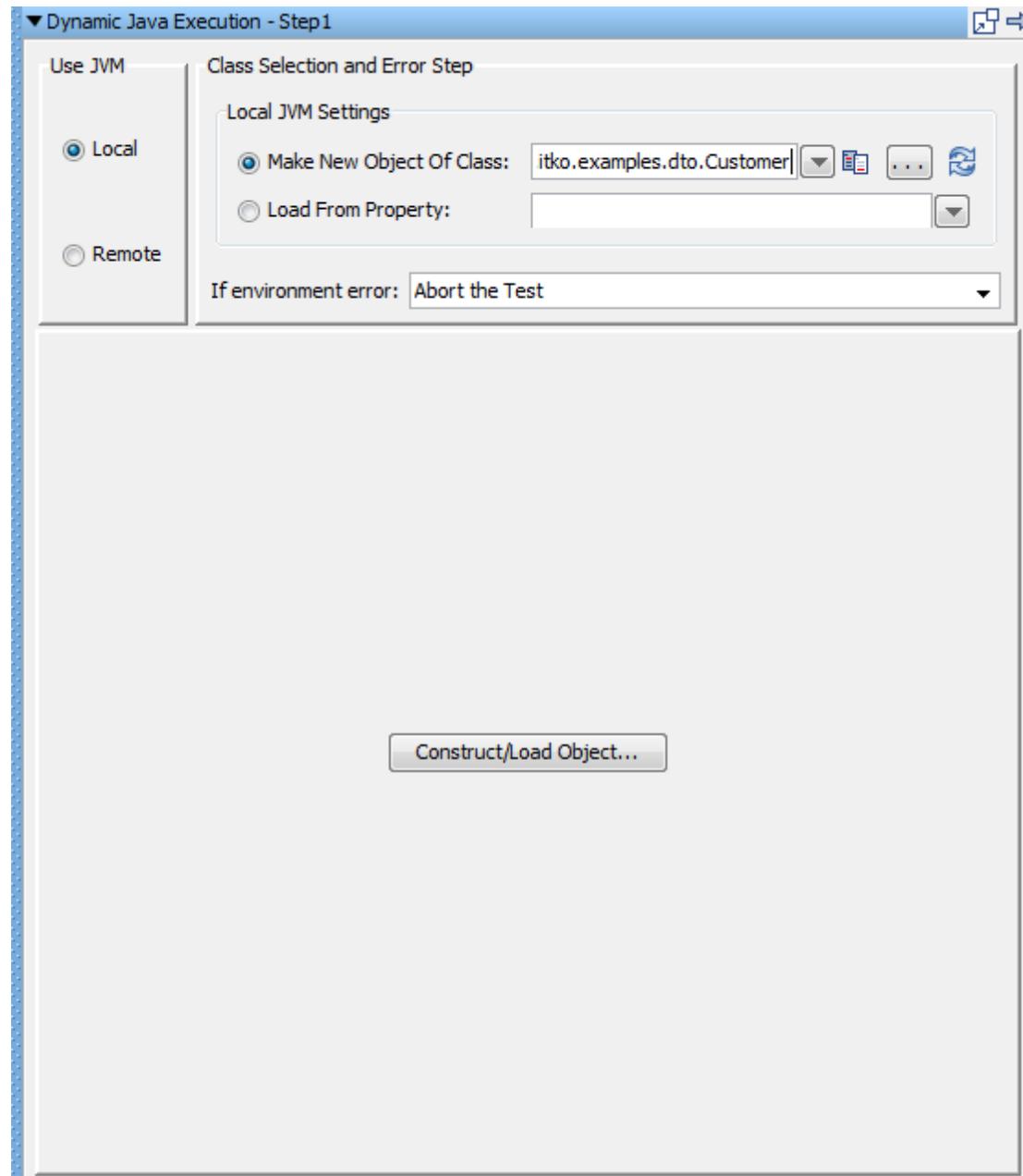
The Complex Object Editor (COE) lets you interact with Java objects without having to write Java code.

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 an example, when you invoke the Dynamic Java Execution step, using the Customer class, you see the following window:

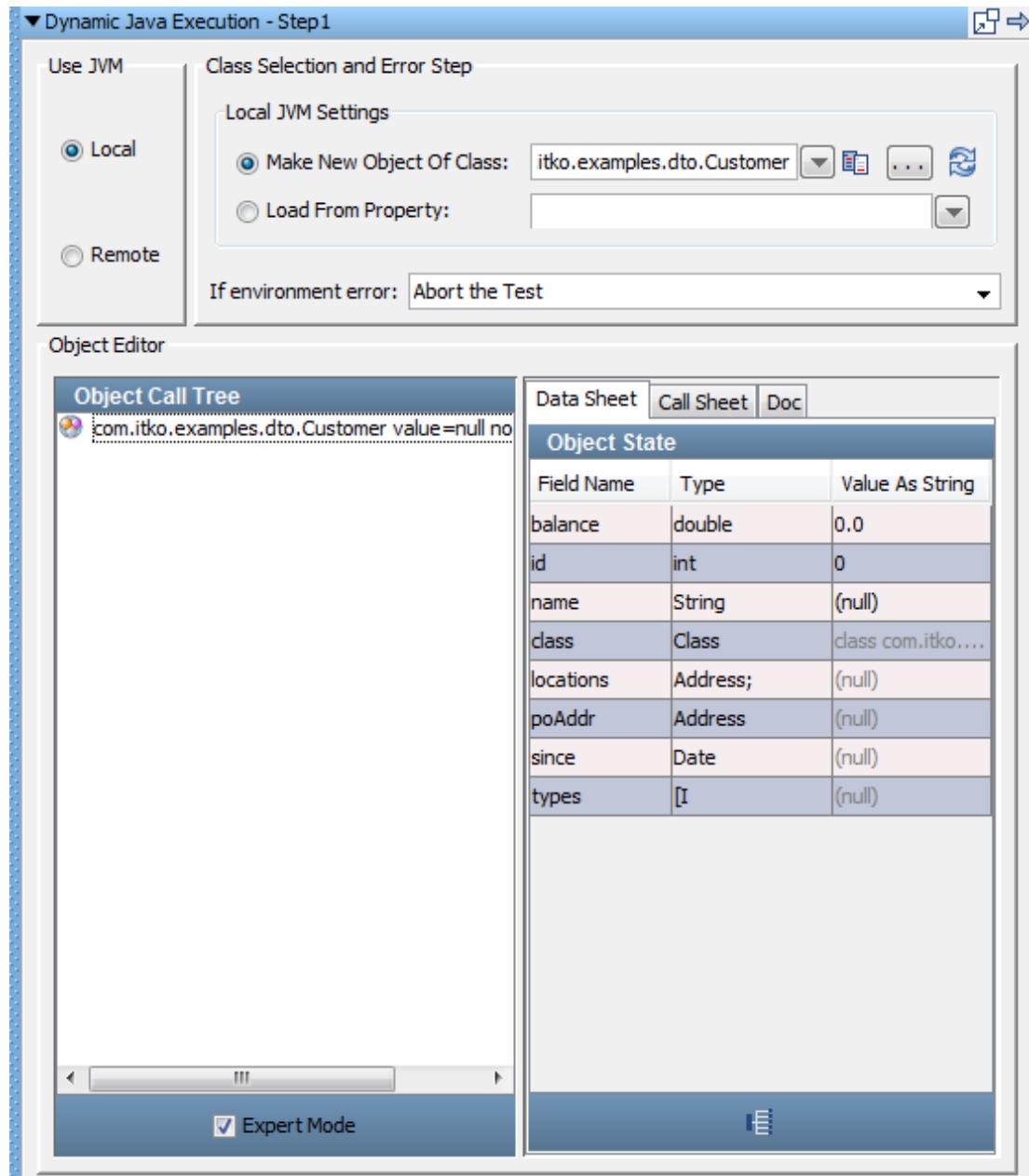


COE - Dynamic Java Execution step

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 [Object Call Tree Panel \(see page 446\)](#) is described in detail in the next page.

The right panel is the **Object State**, with a set of dynamic tabs ([Data Sheet Panel](#), [Call Sheet Panel](#), [Doc Panel \(see page 447\)](#)) 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. This 5-MB default limit can be adjusted with 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.

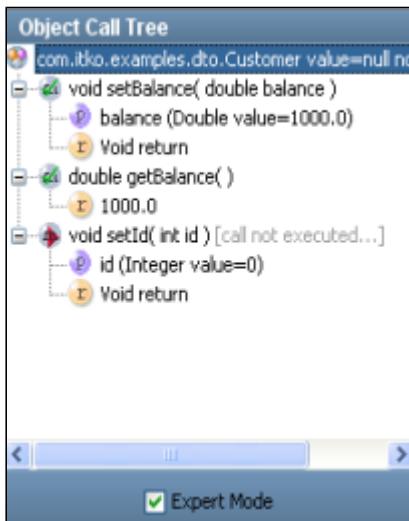
 **More Information:**

- [Object Call Tree Panel \(see page 446\)](#)
- [Data Sheet and Call Sheet Panels \(see page 447\)](#)
- [Using Data Sets in the COE \(see page 456\)](#)
- [Usage Scenarios for Simple Objects \(see page 458\)](#)
- [Usage Scenarios for Complex Objects \(see page 463\)](#)

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
	The type (class) of the object currently loaded, followed by response from calling <code>toString</code> method of object. Icon for the Complex Object Editor type
	The constructor that was called. This is shown if multiple constructors exist. Icon for the Complex Object Editor constructor
	A method call that has not been executed. Icon for the Complex Object Editor non-executed method call
	A method call that has been executed. Icon for the executed method call
	The input parameters (type and current value) for the enclosing method. Icon for the Complex Object Editor input parameter
	The return value (current value if call has been executed) for the enclosing method. Icon for the Complex Object Editor return value

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

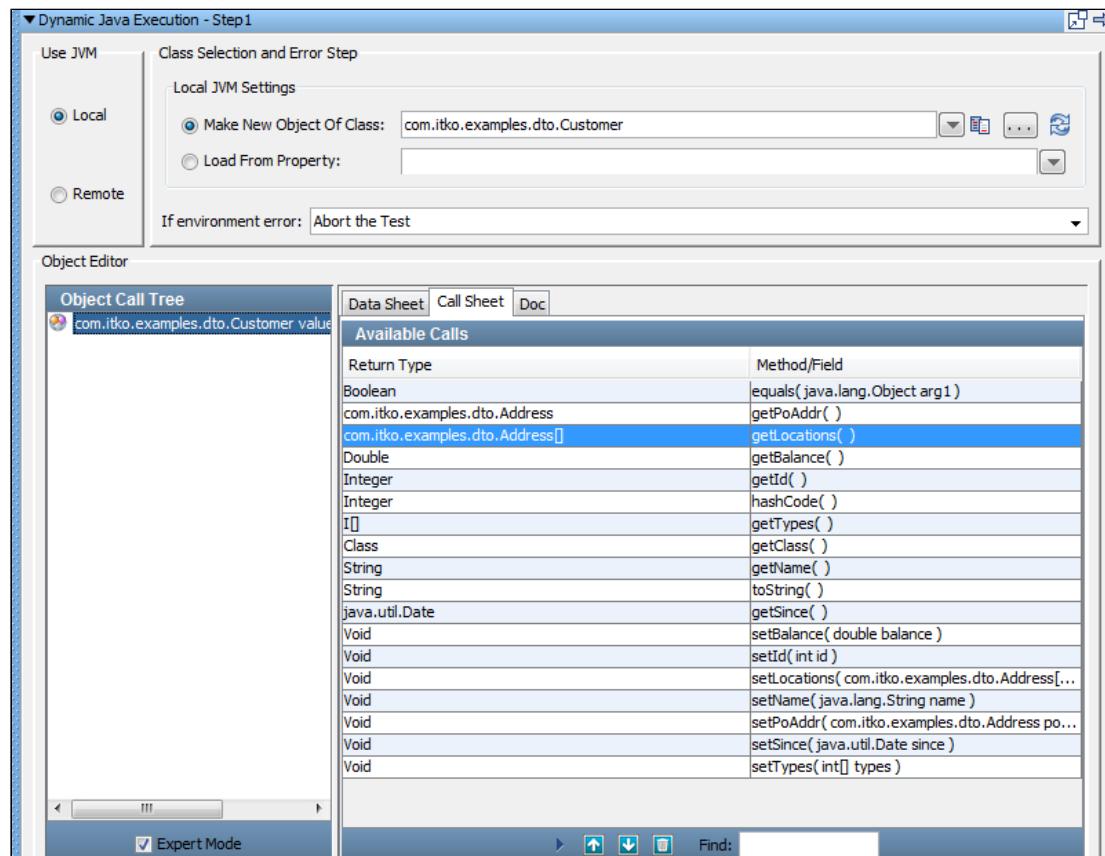
Data Sheet Panel

The right panel shows three tabs, with the **Data Sheet** tab active by default.

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 the **Data Sheet** panel, but can be edited in other windows. The address field, for example, is an object of type Address that cannot be edited in this tab.

Call Sheet Panel

In the **Call Sheet** tab, the COE shows the methods/fields calls that are available, and their return types.



COE Call Sheet panel

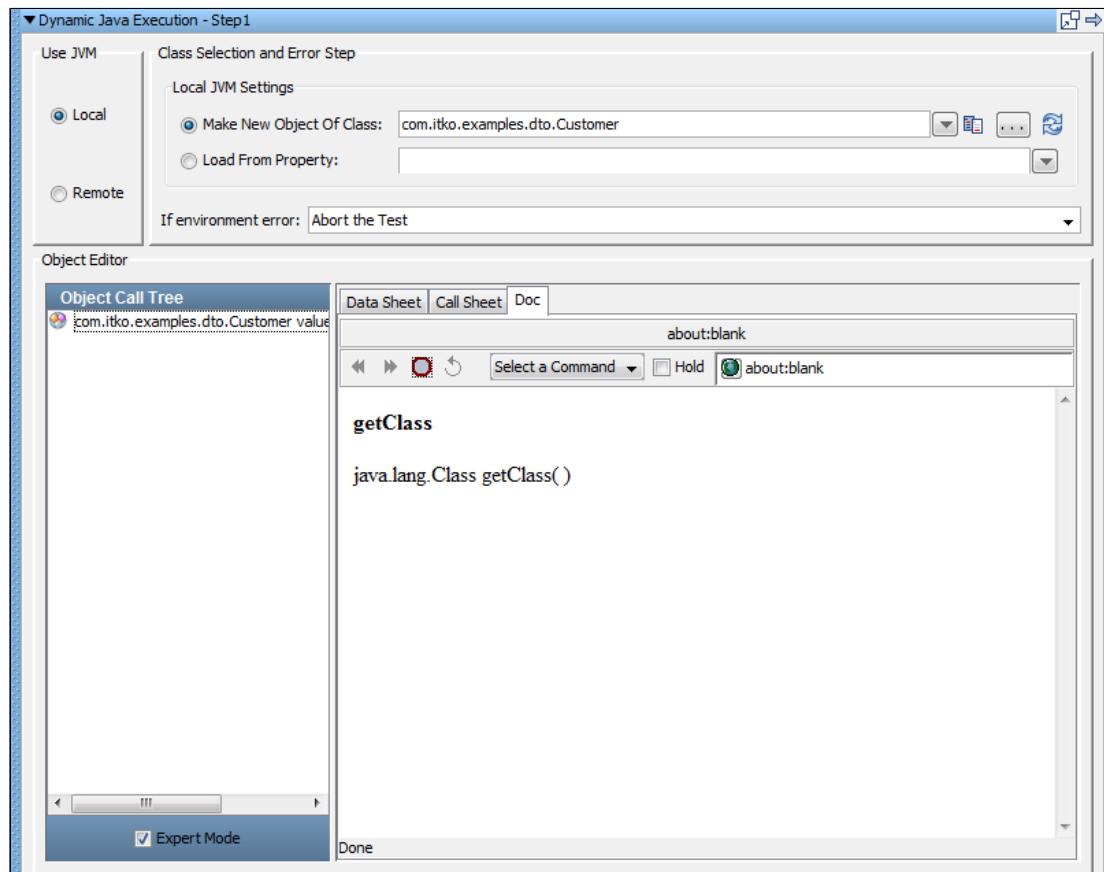


To add a method, select it in the **Methods/Fields** list and click **Add Method** at the bottom of the tab. The selected method now appears in the **Object Call Tree** (in the left panel).

When you are in the object call tree, you can provide input parameters and can invoke the method.

Doc Panel

The **Doc** tab displays any Java API documentation that has been made available for this class.

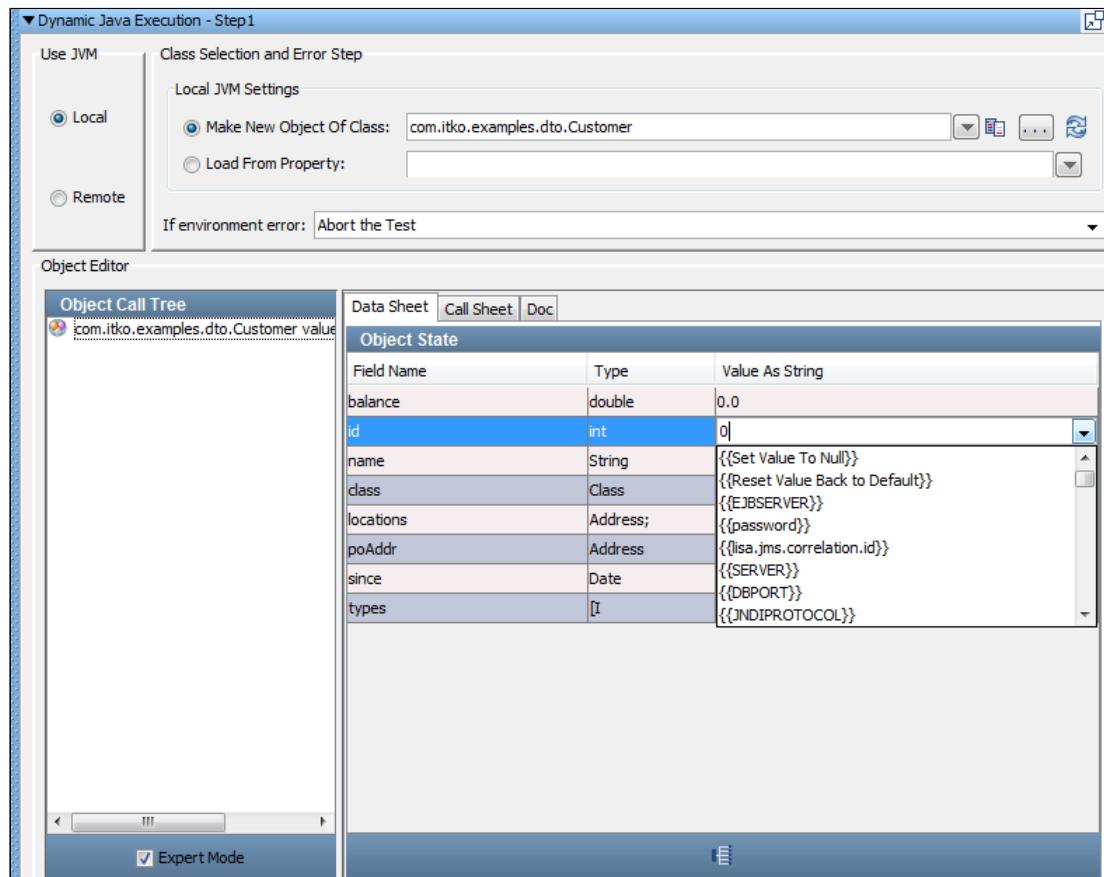


COE Doc panel

Object Interaction Panels

Each action that is displayed in the **Data Sheet** and **Call Sheet** has a different interface.

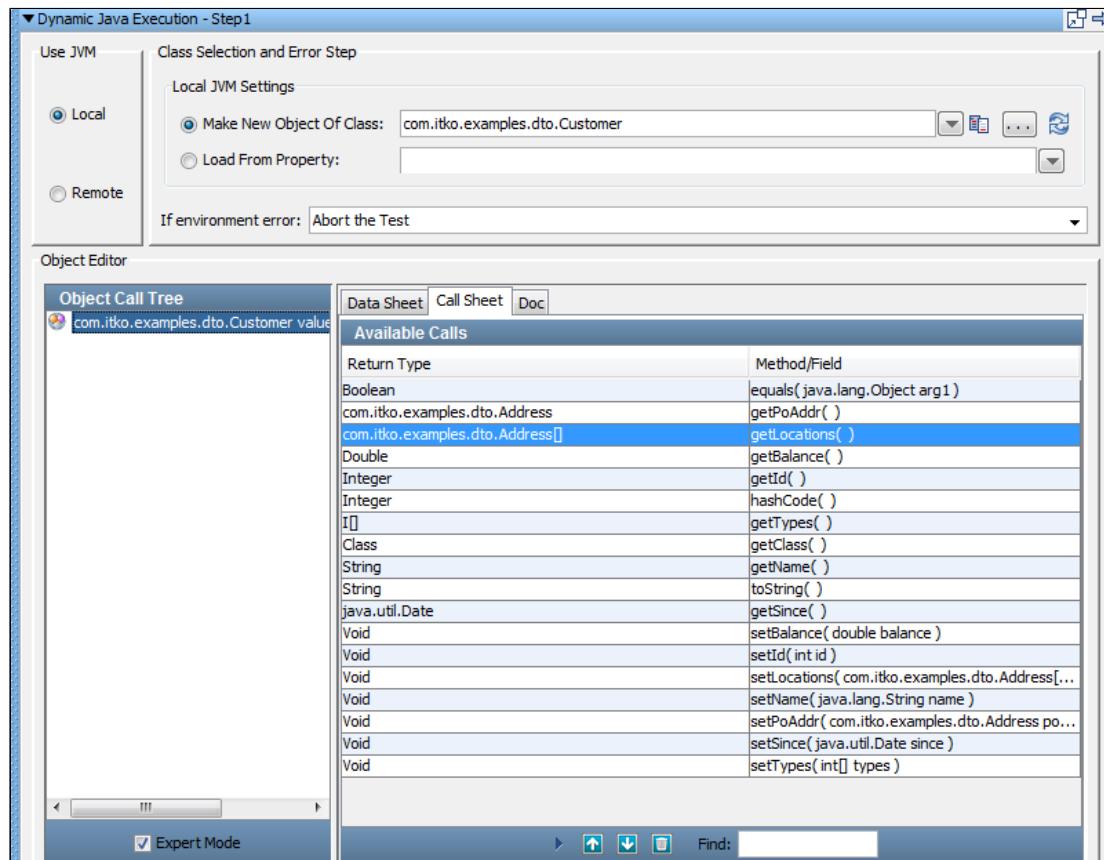
The tab and its interface change depending on what you have selected in the **Object Call Tree** in the left panel.



COE Data Sheet panel

Object Panels

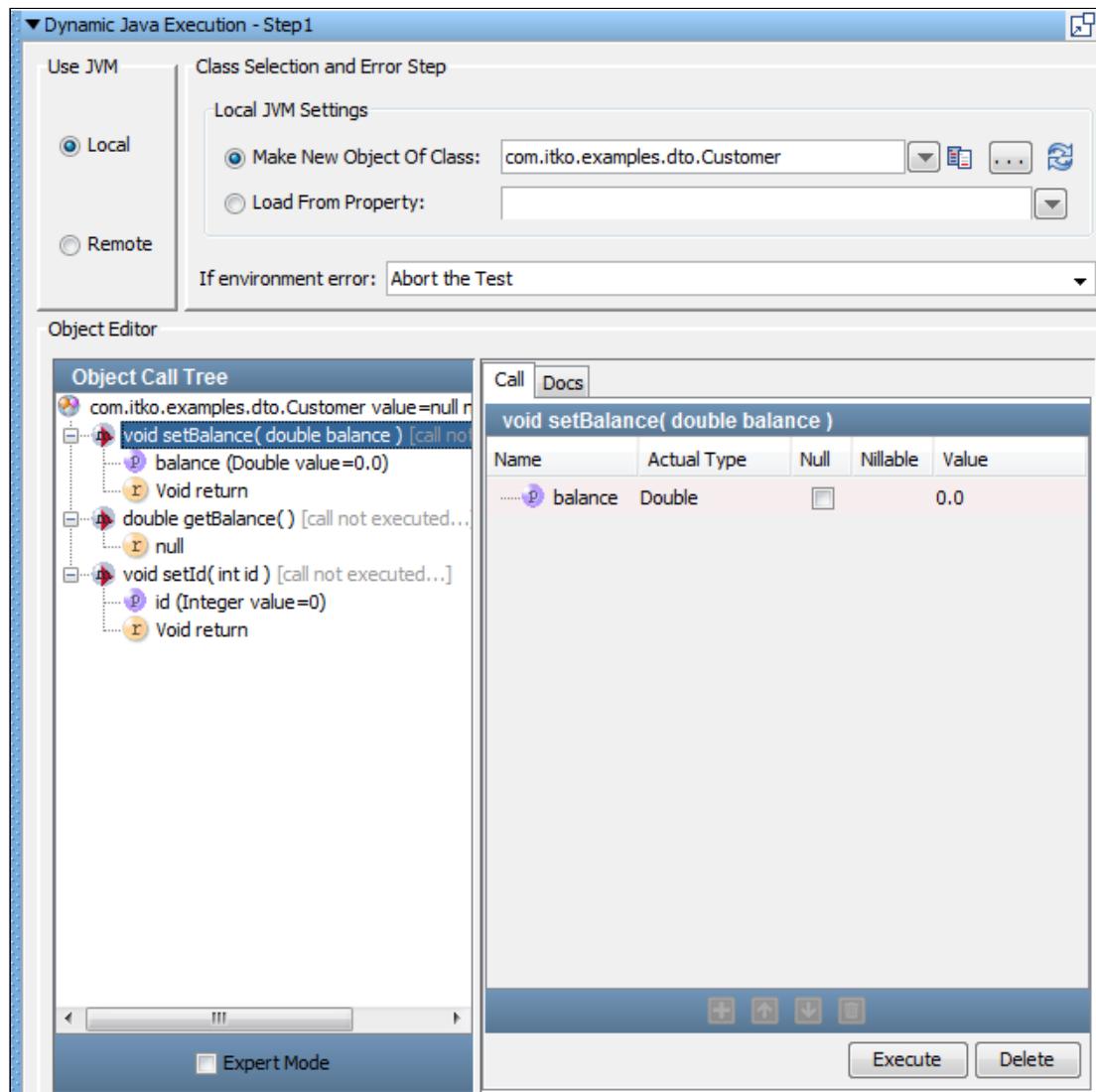
The **Data Sheet**, **Call Sheet**, and **Doc** tabs are available when an object is selected in the **Object Call Tree**.



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

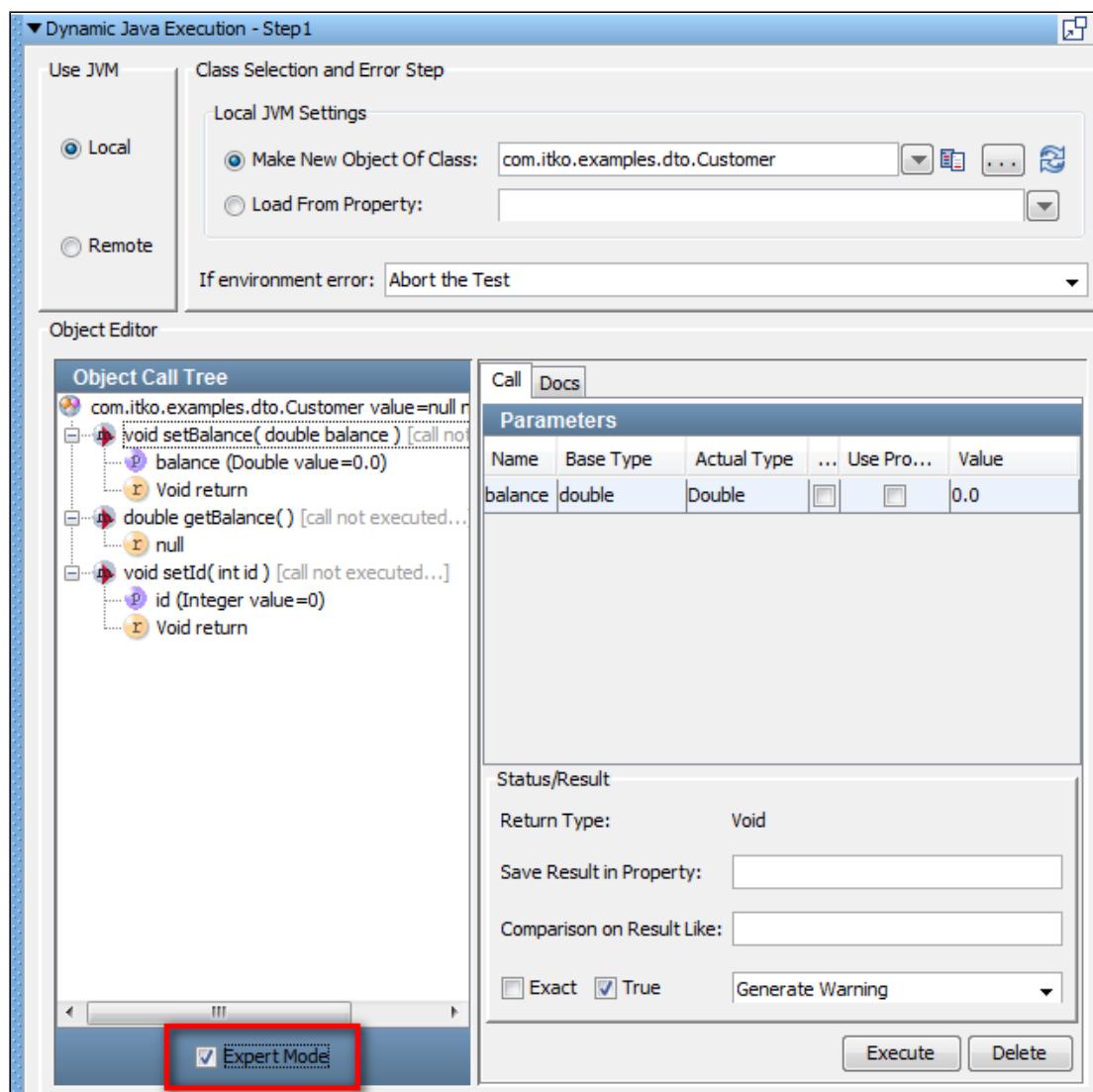
Two editing modes are available: simple and expert.

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.

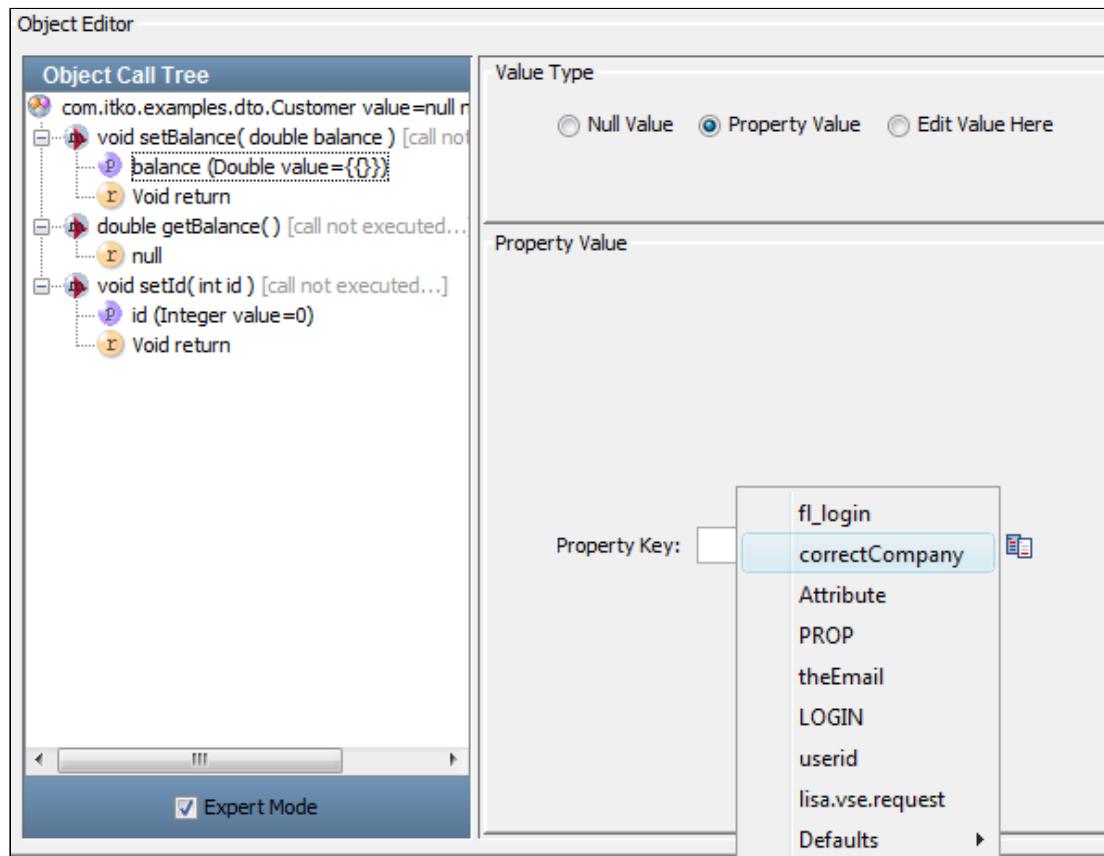
For input parameters that are primitives and strings, the following panel is displayed.

The screenshot shows the 'Object Editor' window with the 'Object Call Tree' tab selected. The tree view displays a customer object with methods like setBalance, getBalance, and setId. The 'Value Type' tab is active, showing three radio button options: 'Null Value', 'Property Value', and 'Edit Value Here'. The 'Edit Value Here' option is selected, and below it is a 'Simple Value' field with a placeholder 'Simple Value: []'.

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.

The screenshot shows the DevTest Solutions Object Editor interface with the 'Data Sheet' tab selected. The 'Object Call Tree' panel on the left shows the same method calls as before. The 'Data Sheet' panel on the right contains a table titled 'Object State' with the following data:

Field Name	Type	Value As String
city	String	(null)
line1	String	(null)
line2	String	(null)
state	String	(null)
zip	String	(null)
class	Class	class com.itko.examples...

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.

The screenshot shows the 'Object Editor' interface with two panels. The left panel, titled 'Object Call Tree', displays a call tree for a 'Customer' object. One of the calls is 'double getBalance()' with a return value of '850.0'. The right panel, titled 'String Value of Object', contains a single entry: ':850.0'.

COE String Value of Object panel

For input parameters that are objects, the following panel is displayed.

The screenshot shows the 'Object Editor' interface with two panels. The left panel, titled 'Object Call Tree', displays a call tree for a 'Customer' object. One of the calls is 'void setLocations(com.itko.examples.dto.Address[] locations)' with a return value of 'Void return'. The right panel, titled 'String Value of Object', contains a single entry: 'Void return'.

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.

Field Name	Type	Value As String
city	String	{{AddressDataS...}}
line1	String	{{LISA_TC_PATH}}
line2	String	{{robot}}
state	String	{{LISA_HOST}}
zip	String	{{Instance}}
class	Class	{{LISA_TC_URL}}

COE - Using Excel Data Set as input

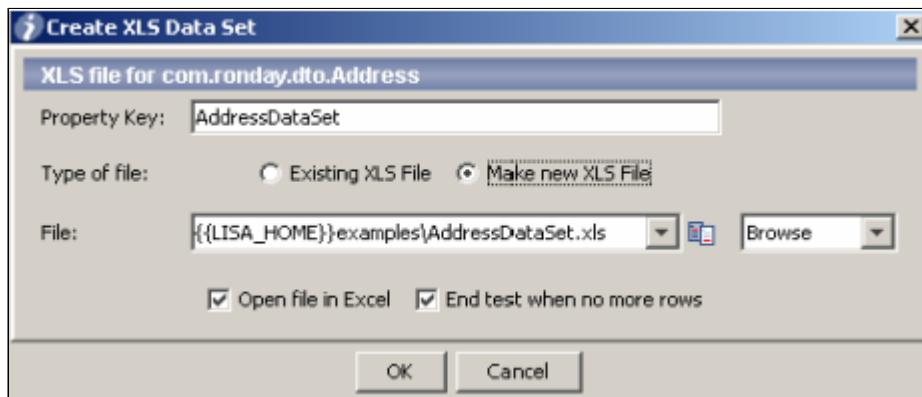
To initiate the creation of a new DTO data set from the object editor:

- When a DTO object appears in the call list, right-click the **Name** or **Actual Type** to open a menu.

Name	Actual Type	Null	Nillable	Value
poAddr	Address			

COE - Right-click menu on DTO object

- 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**

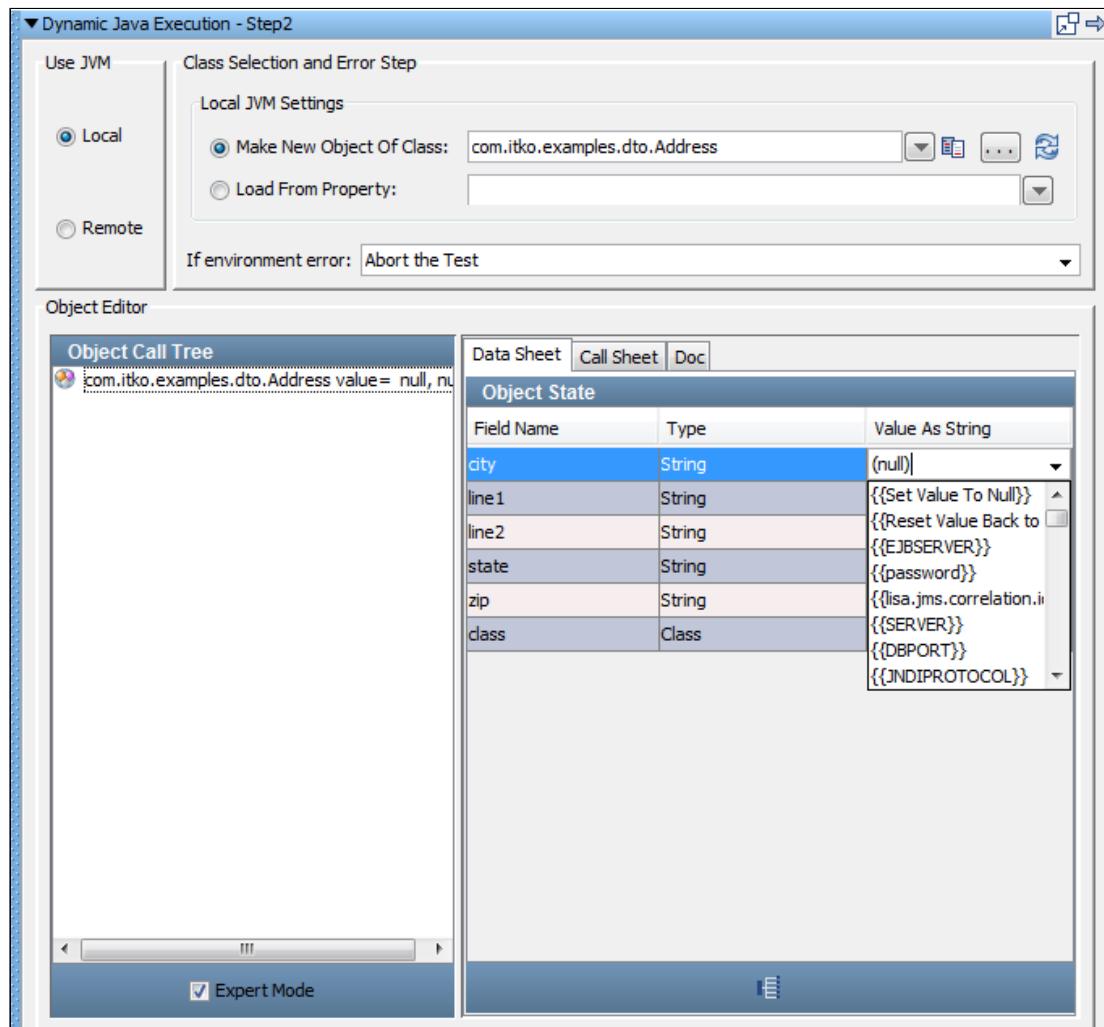
Ends the test after all rows are read.

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

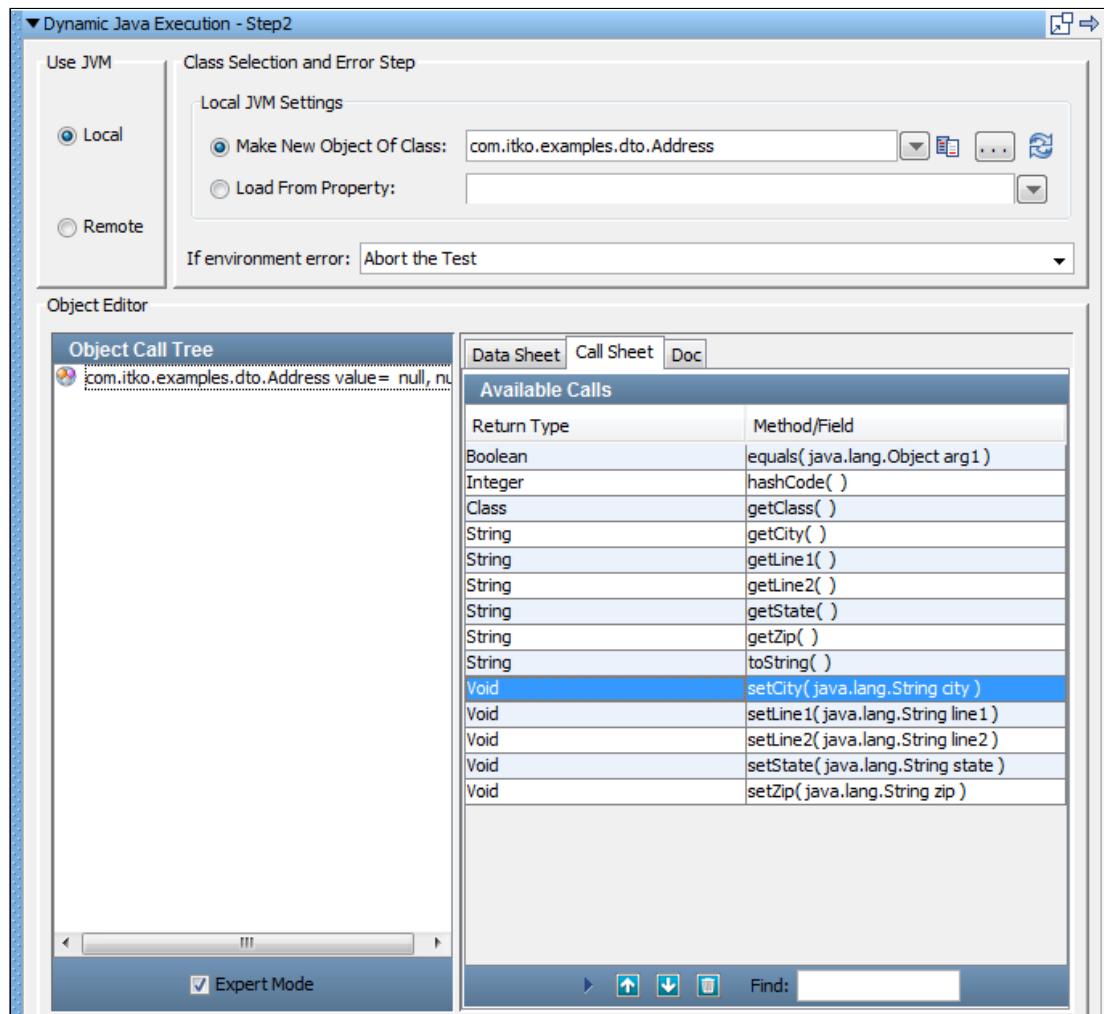
A simple DTO com.itko.examples.dto.Address has been loaded in the **COE** using a Dynamic Java Execution step. The Address class has simple properties only.



A simple DTO com.itko.examples.dto.Address has been loaded in the COE using a Dynamic Java Execution step

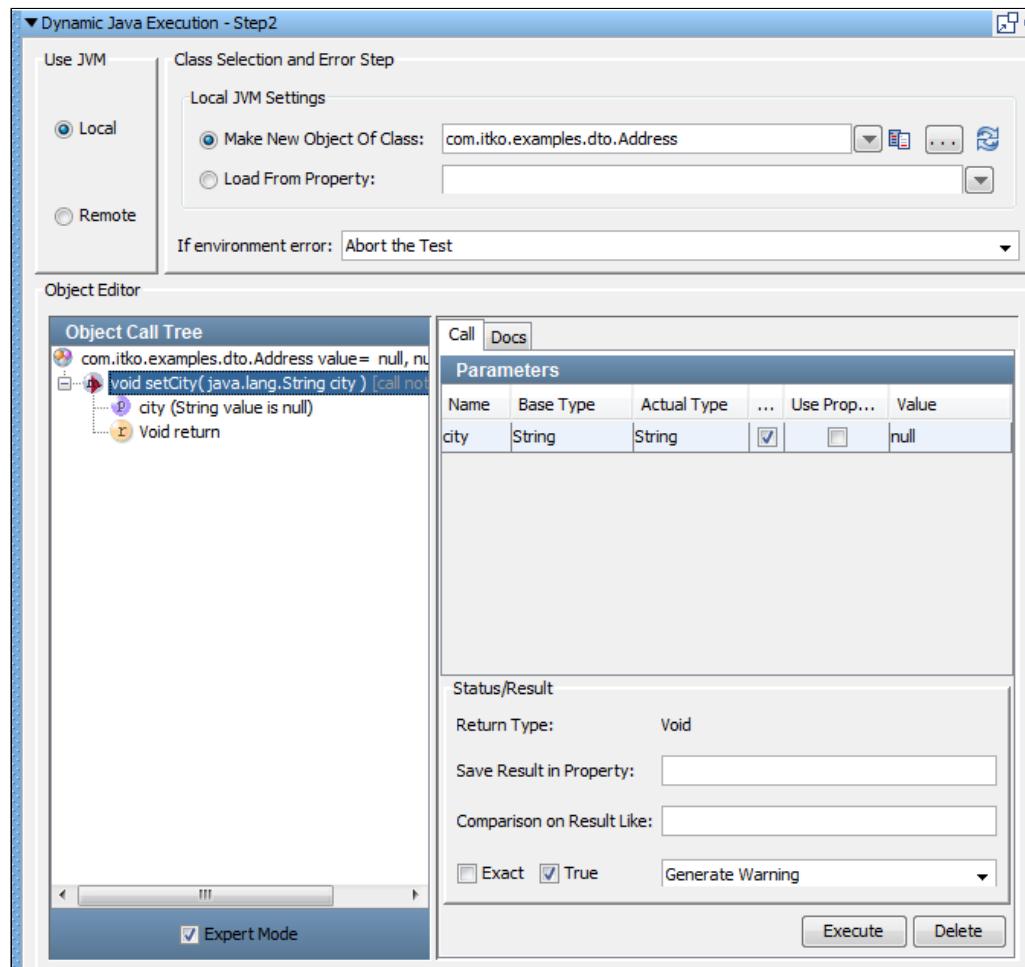
For a simple DTO, parameter values can be entered into the **Data Sheet** panel as fixed values or properties. In the previous example, city could be set equal to the property currentCity.

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.

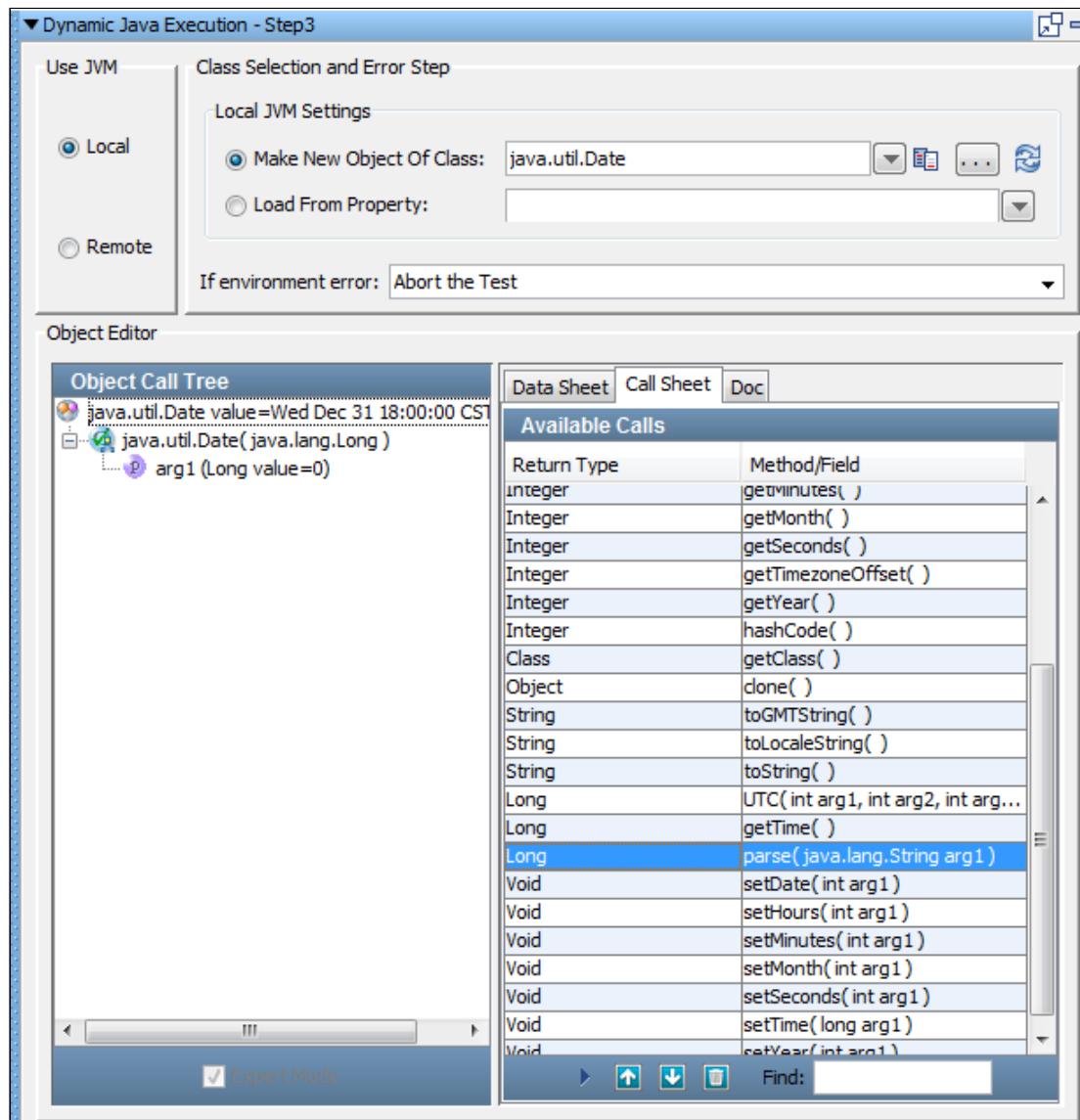


COE Call tab opened

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.

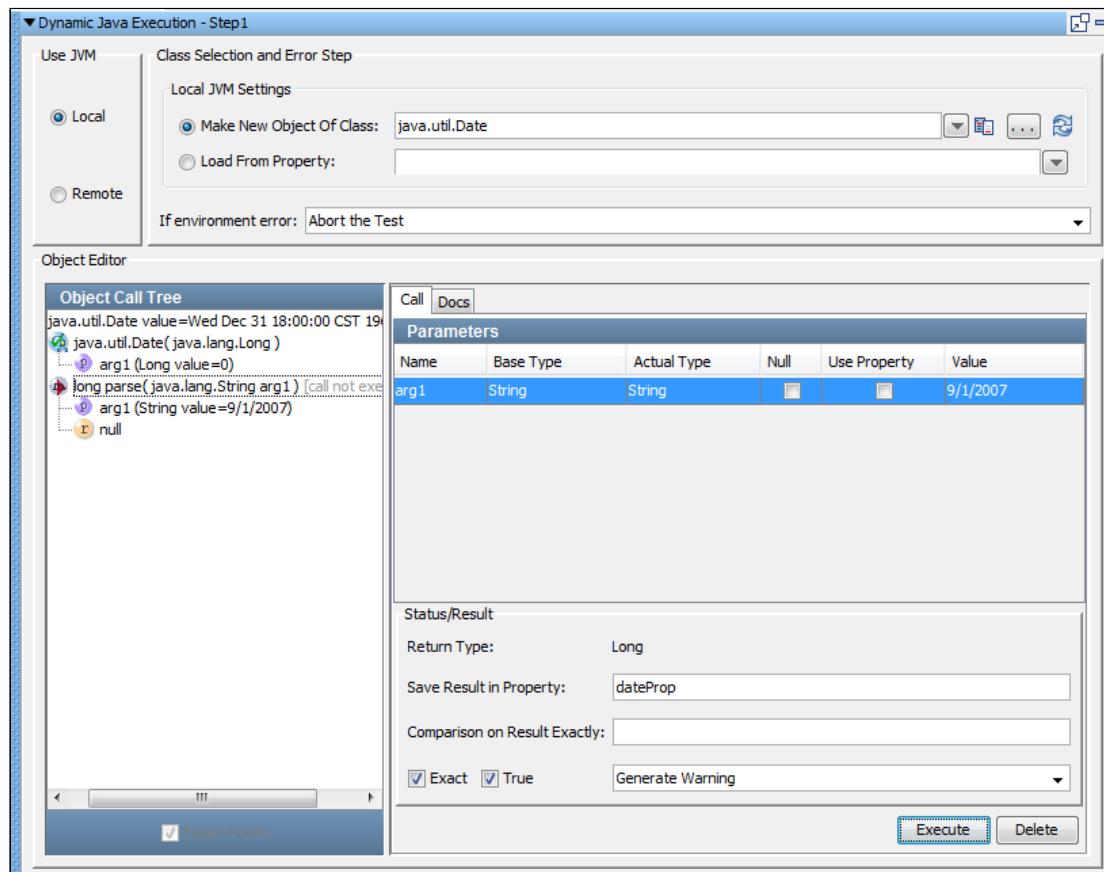
Simple Java Object Scenario 2

A simple Java object, **java.util.Date**, has been loaded in the **COE** using a Dynamic Java Execution step.



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.

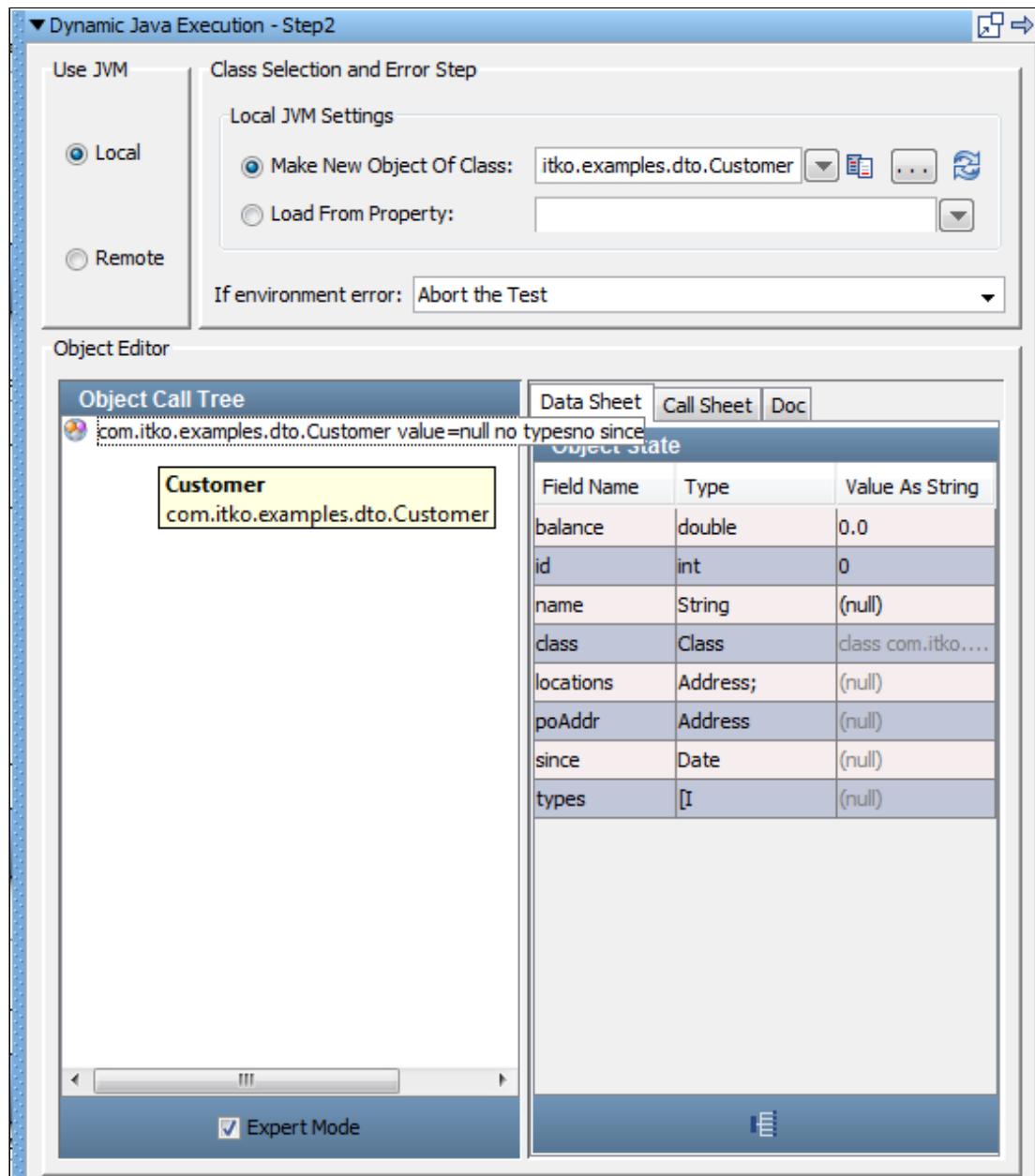
Because we are in Expert Mode, we can add inline filters and assertions.

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

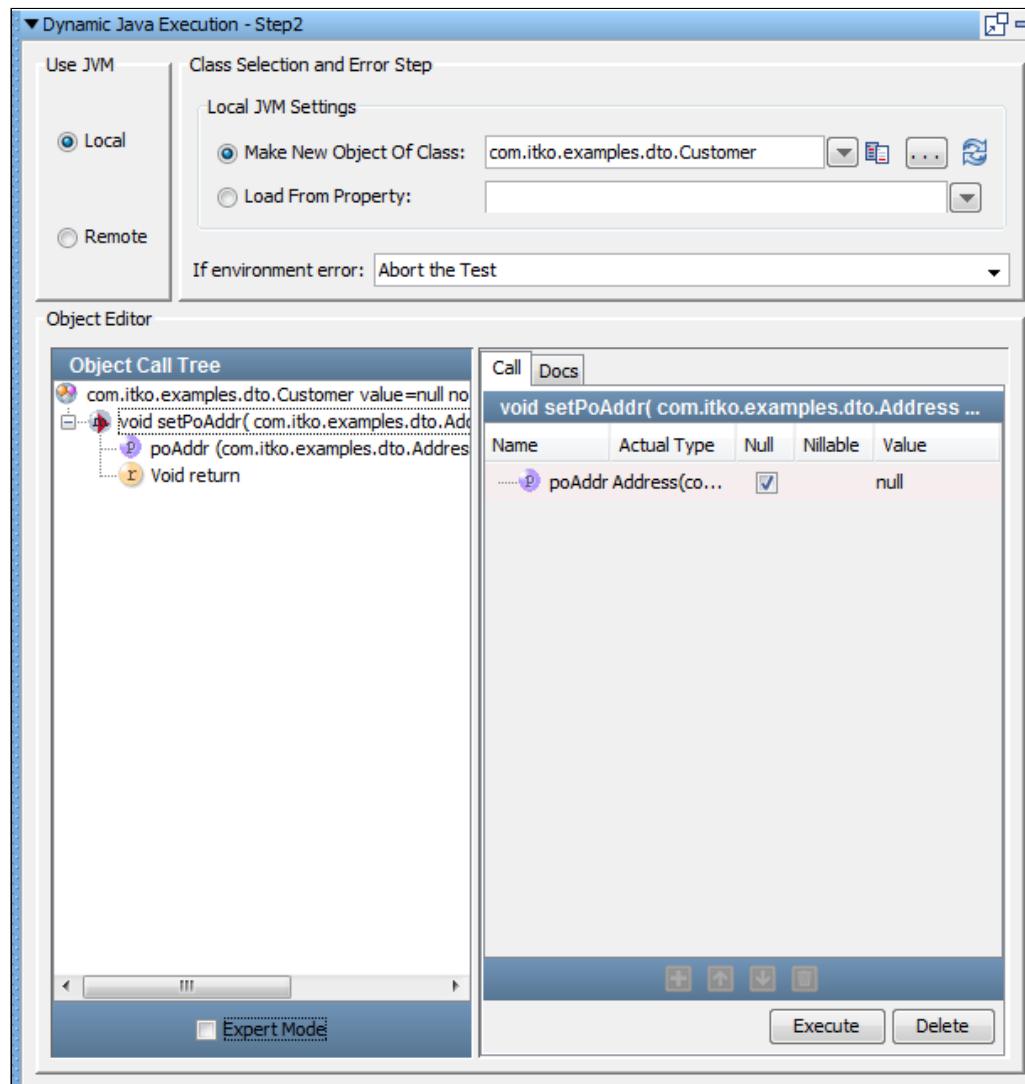
- **types**

An array of integers

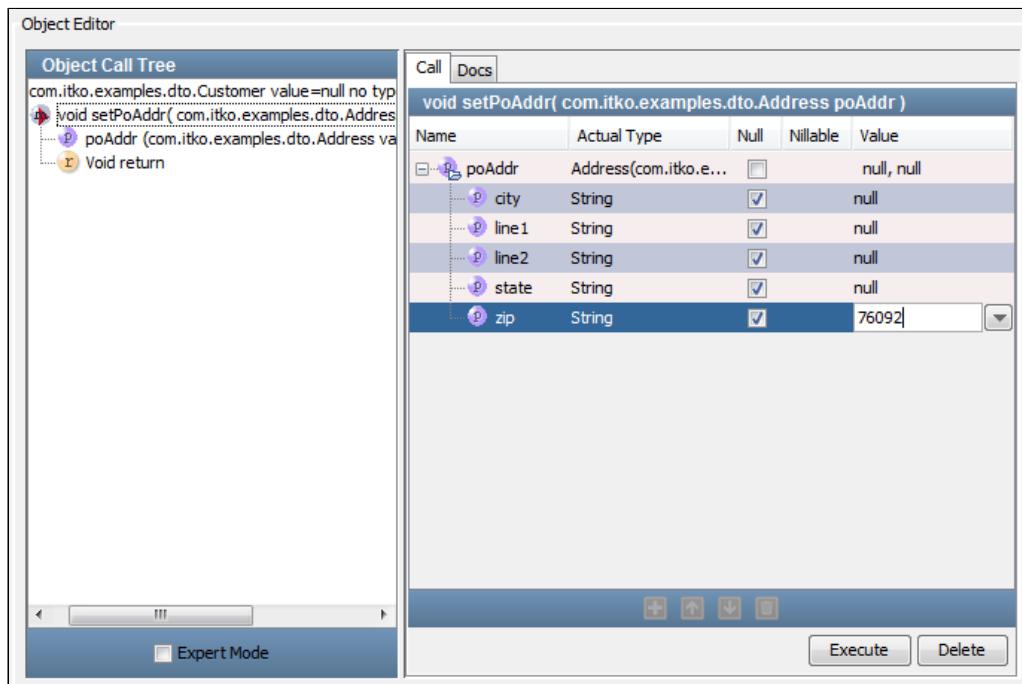
1. Starting with the poAddr object, identify the setPoAddr method in the **Call Sheet**.

Return Type	Method/Field
Boolean	equals(java.lang.Object)
com.itko.examples.dto.Address	getPoAddr()
com.itko.examples.dto.Address[]	getLocations()
Double	getBalance()
Integer	getId()
Integer	hashCode()
I[]	getTypes()
Class	getClass()
String	getName()
String	toString()
java.util.Date	getSince()
Void	setBalance(double balance)
Void	setId(int id)
Void	setLocations(com.itko.examples.dto.Location[] locations)
Void	setName(java.lang.String name)
Void	setPoAddr(com.itko.examples.dto.Address address)
Void	setSince(java.util.Date since)
Void	setTypes(int[] types)

2. Select the setPoAddr method and double-click or click  **Add Method** to run this method.

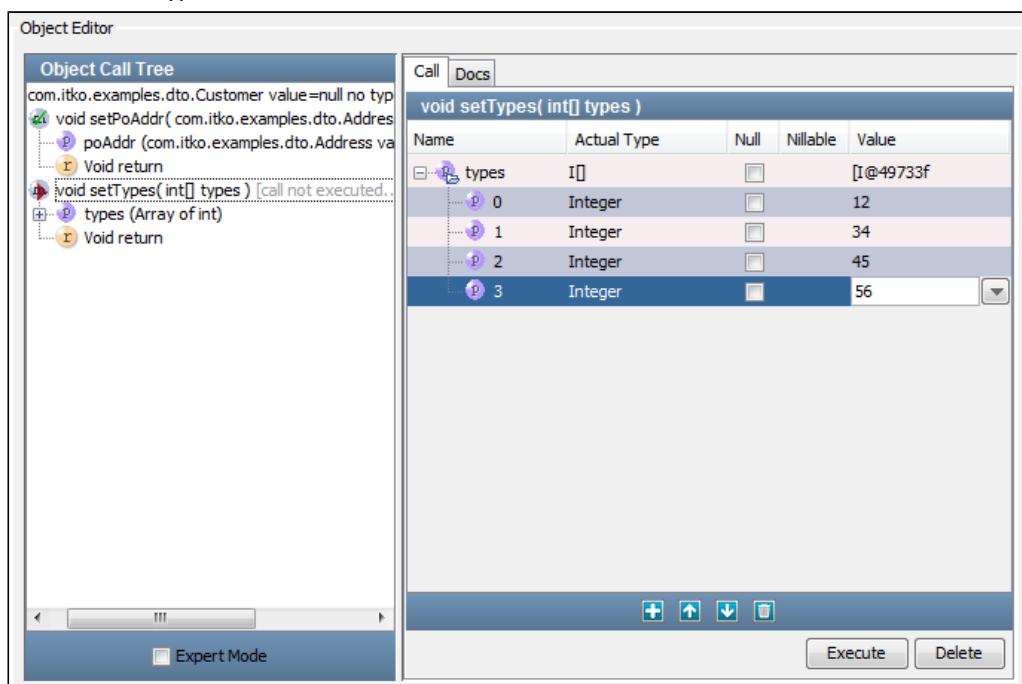


3. This DTO enables the use of Simple mode. Do not select the **Expert Mode** check box. The **poAddress** property is identified as type **Address**.
4. In a Simple mode, when you clear the **Null** parameter, the **Address** object is expanded to expose its properties. You know, from the previously 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.

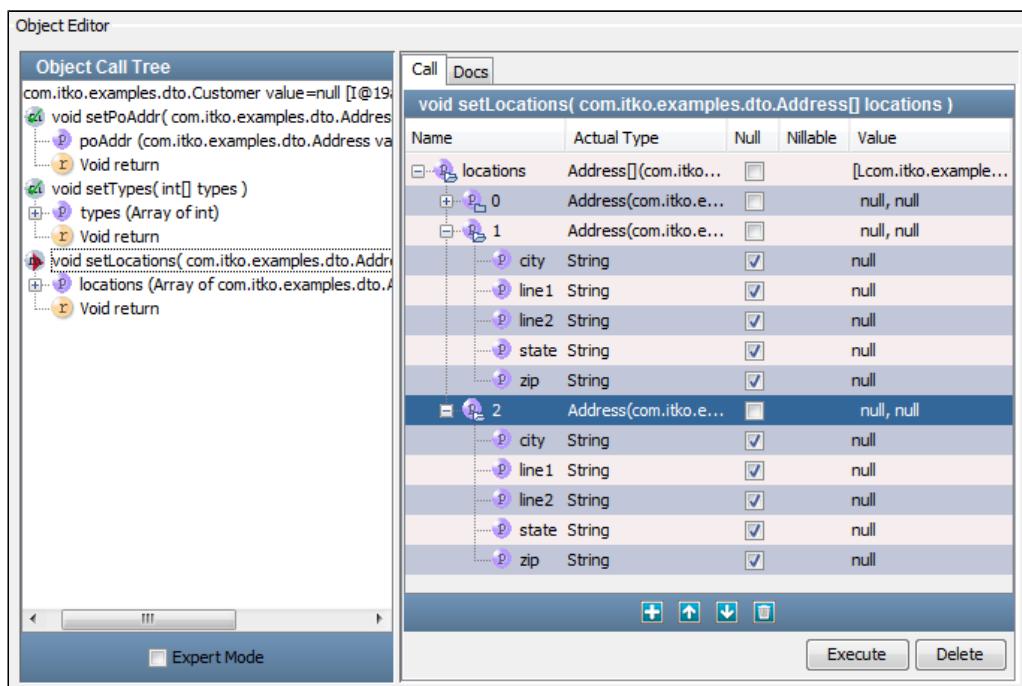
6. Select the setTypes method on the **Call Sheet** and click **Invoke Method**.



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**.



The screenshot shows the DevTest Object Editor interface. On the left, the **Object Call Tree** pane displays a call tree for a Customer object, with the `void setLocations(com.itko.examples.dto.Address[] locations)` method selected. On the right, the **Call Sheet** pane shows the method signature and a data sheet for the `locations` parameter. The data sheet has columns for Name, Actual Type, Null, Nullable, and Value. It lists three Address objects (index 0, 1, 2) with their properties: city, line1, line2, state, and zip. Each property has a checked checkbox under the Nullable column, and the Value column shows 'null'. At the bottom of the Call Sheet are buttons for **Execute** and **Delete**.

void setLocations(com.itko.examples.dto.Address[] locations)				
Name	Actual Type	Null	Nullable	Value
<code>locations</code>	Address[] (com.itko.e...)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	[Lcom.itko.example...
<code>0</code>	Address(com.itko.e...)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	null, null
<code>1</code>	Address(com.itko.e...)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	null, null
<code>city</code>	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	null
<code>line1</code>	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	null
<code>line2</code>	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	null
<code>state</code>	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	null
<code>zip</code>	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	null
<code>2</code>	Address(com.itko.e...)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	null, null
<code>city</code>	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	null
<code>line1</code>	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	null
<code>line2</code>	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	null
<code>state</code>	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	null
<code>zip</code>	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	null

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.

The screenshot shows the Object Editor interface. On the left, the **Object Call Tree** pane displays a hierarchical list of method calls and their parameters for a Customer object. On the right, the **Object State** pane shows a data sheet with fields like city, line1, line2, state, zip, and class, each with its type and value.

Field Name	Type	Value As String
city	String	Alto
line1	String	2109 Busy Bee Street
line2	String	Suite 45
state	String	TX
zip	String	75925
class	Class	class com.itko.examples....

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 screenshot shows the Object Editor with the **Call Sheet** tab selected. It displays the `void setSince(java.util.Date since)` method and its input parameters: since, date, hours, minutes, month, seconds, time, and year. The `since` parameter is currently selected.

Name	Actual Type	Null	Nillable	Value
since	Date(java.util)	<input type="checkbox"/>		Tue May 10 15:25:...
date	Integer	<input type="checkbox"/>		10
hours	Integer	<input type="checkbox"/>		15
minutes	Integer	<input type="checkbox"/>		25
month	Integer	<input type="checkbox"/>		4
seconds	Integer	<input type="checkbox"/>		48
time	Long	<input type="checkbox"/>		1305059148775
year	Integer	<input type="checkbox"/>		111

The input parameters for the Data object are displayed and can be given values.

13. Click **Execute**.

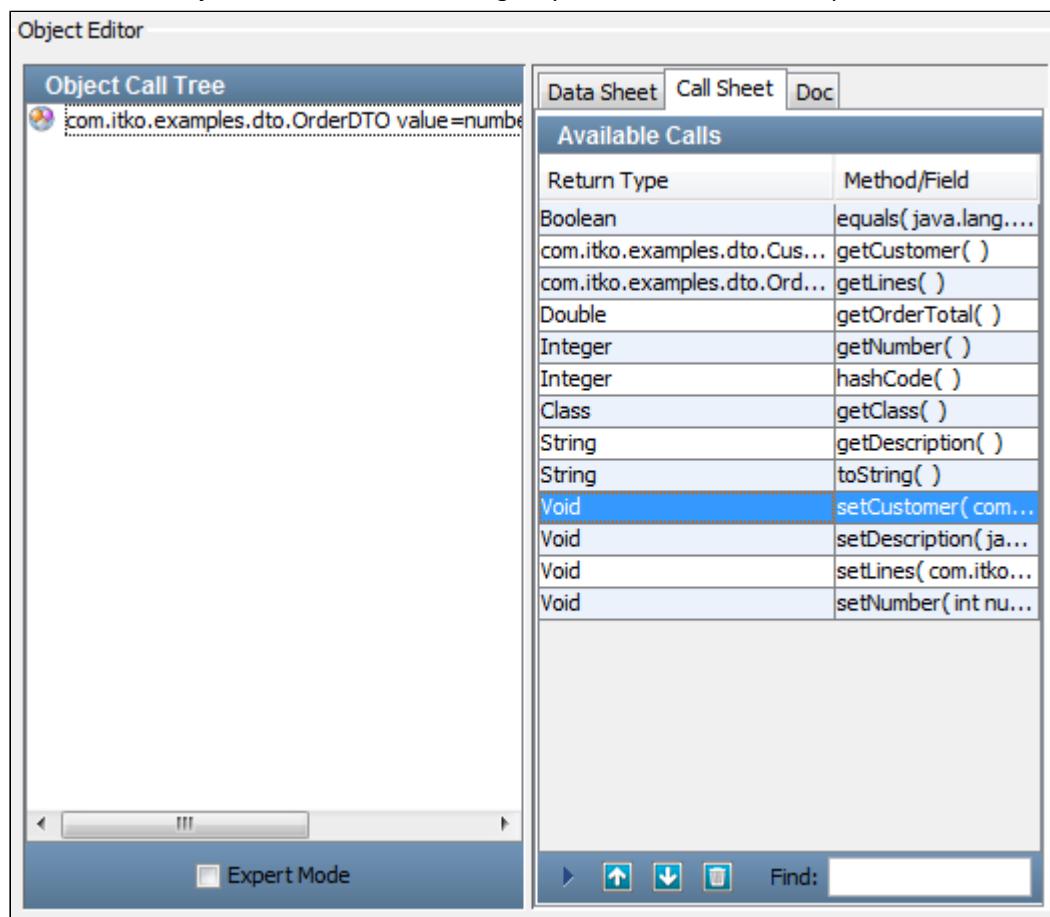
The Customer object is now fully specified and can be used in your test case.

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.

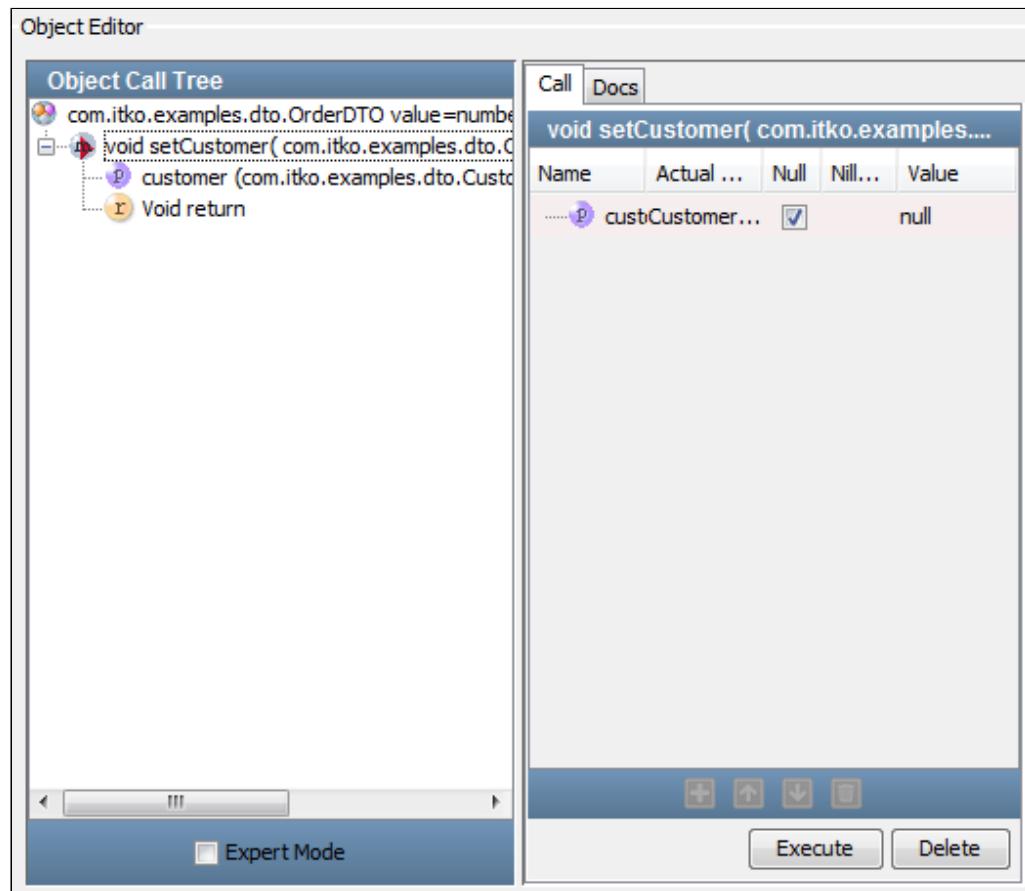


Again we can use Simple Mode for the OrderDTO object.

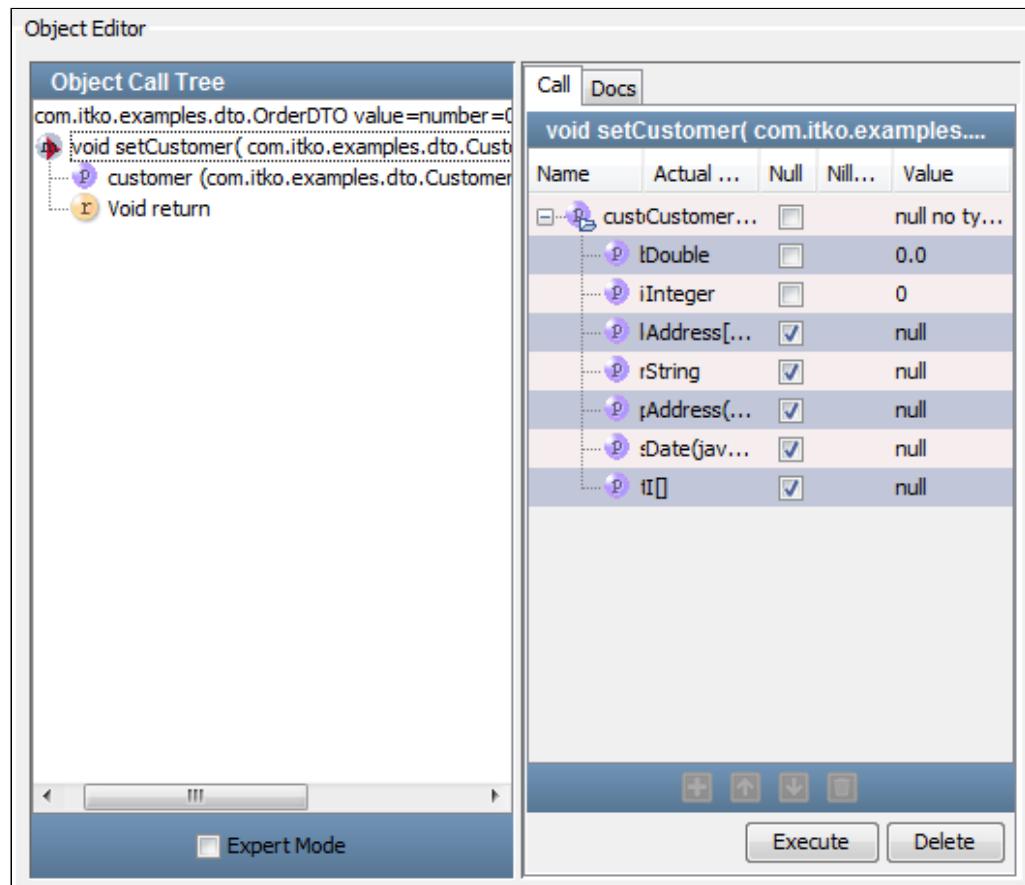
Follow these steps:



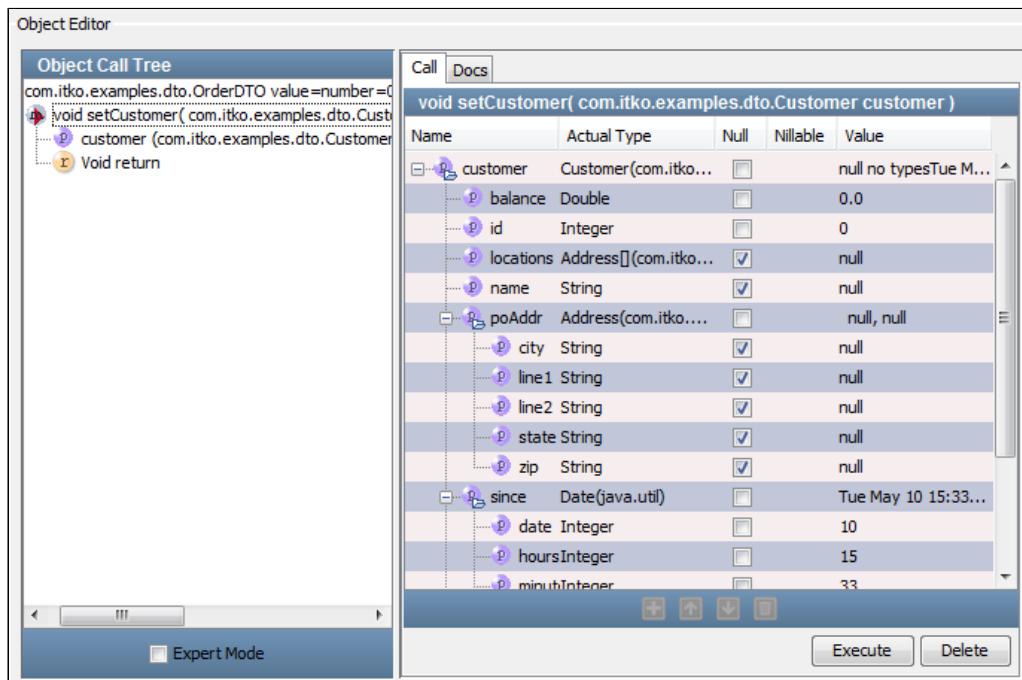
1. Select the setCustomer method in the **Call Sheet** and click **Invoke Method**. As expected, the input parameter is a Customer object.



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 \(see page 474\)](#)
- [Configure Test Steps \(see page 477\)](#)

- [Add Filters - Assertions - Data Sets to a Step \(see page 478\)](#)
- [Configure Next Step \(see page 479\)](#)
- [Replay to a Step \(see page 480\)](#)
- [Set a Starter Step \(see page 480\)](#)
- [Generate Warnings and Errors \(see page 480\)](#)

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 \(see page 474\)](#)

Add a Test Step - Example

Contents

- [Add Step Information \(see page 475\)](#)

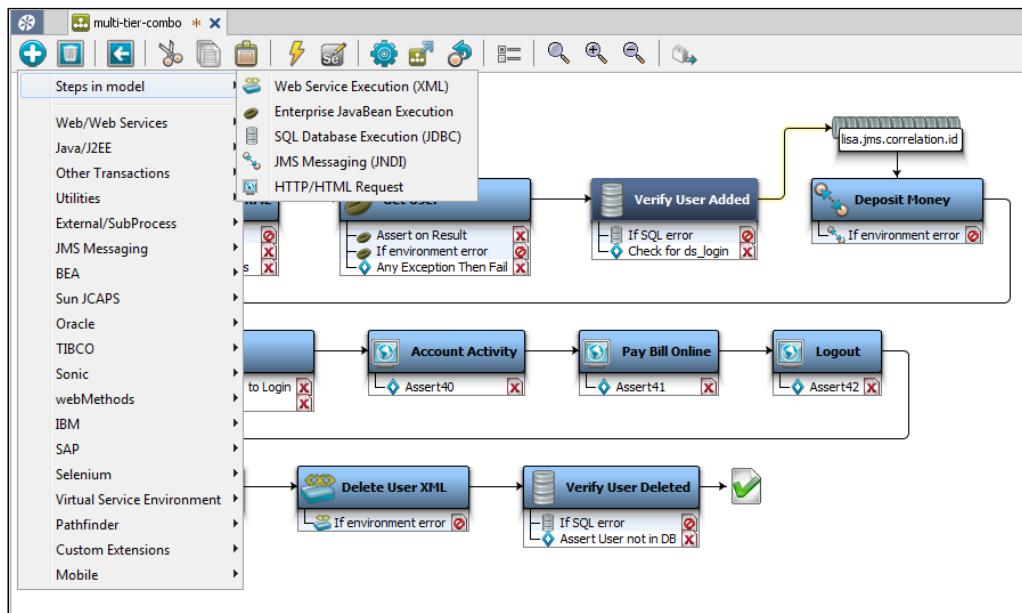
This example adds a step to the multi-tier-combo test case in the examples directory. A new Dynamic Java Execution step will be added to the multi-tier-combo test case, after the Get User step.

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).

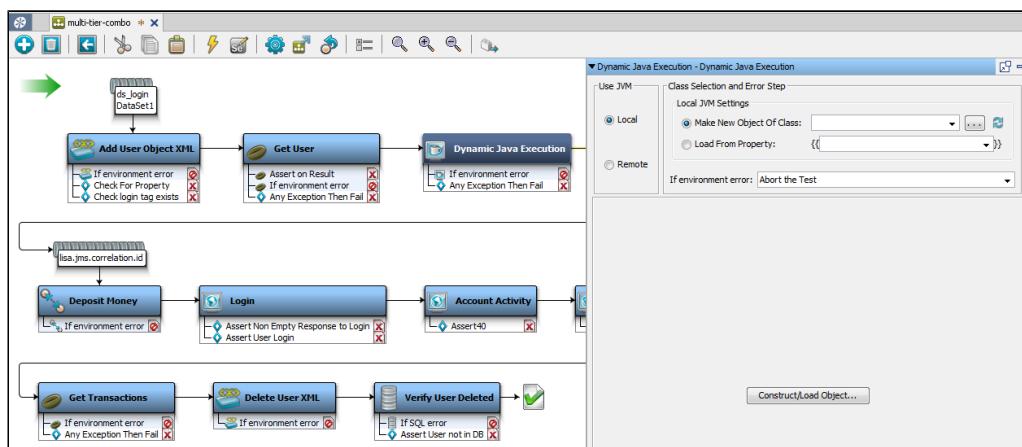
The sub category opens.

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.



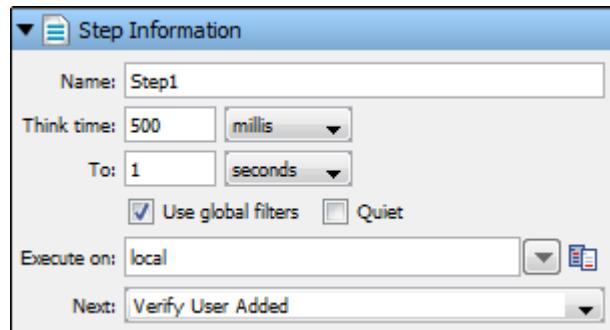
Step editor for the Dynamic Java Execution step

Add Step Information

Follow these steps:

- 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

- 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 \(see page 405\)](#).

- **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

Contents

- [Step Element Toolbar \(see page 478\)](#)

Follow these steps:

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 \(see page 474\)](#).

▪ 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**

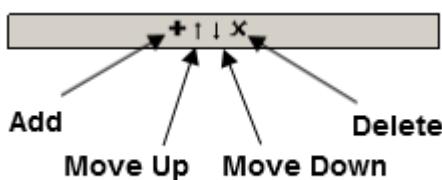
A read-only list of properties the step sets (assigns a value).

- **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 \(see page 405\)](#).
- For detailed information about adding assertions, see [Add an Assertion \(see page 420\)](#).

Add Filters - Assertions - Data Sets to a Step

Filters, assertions, and data sets are added 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 \(see page 405\)](#)

- [Assertions \(see page 419\)](#)
- [Data Sets \(see page 433\)](#)

Configure Next Step

Contents

- [Assign the Next Step \(see page 479\)](#)
- [End Step \(see page 479\)](#)
- [Fail Step \(see page 479\)](#)
- [Abort Step \(see page 479\)](#)

Assign the Next Step

In a test case workflow, you can assign the "next step" to a selected test 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

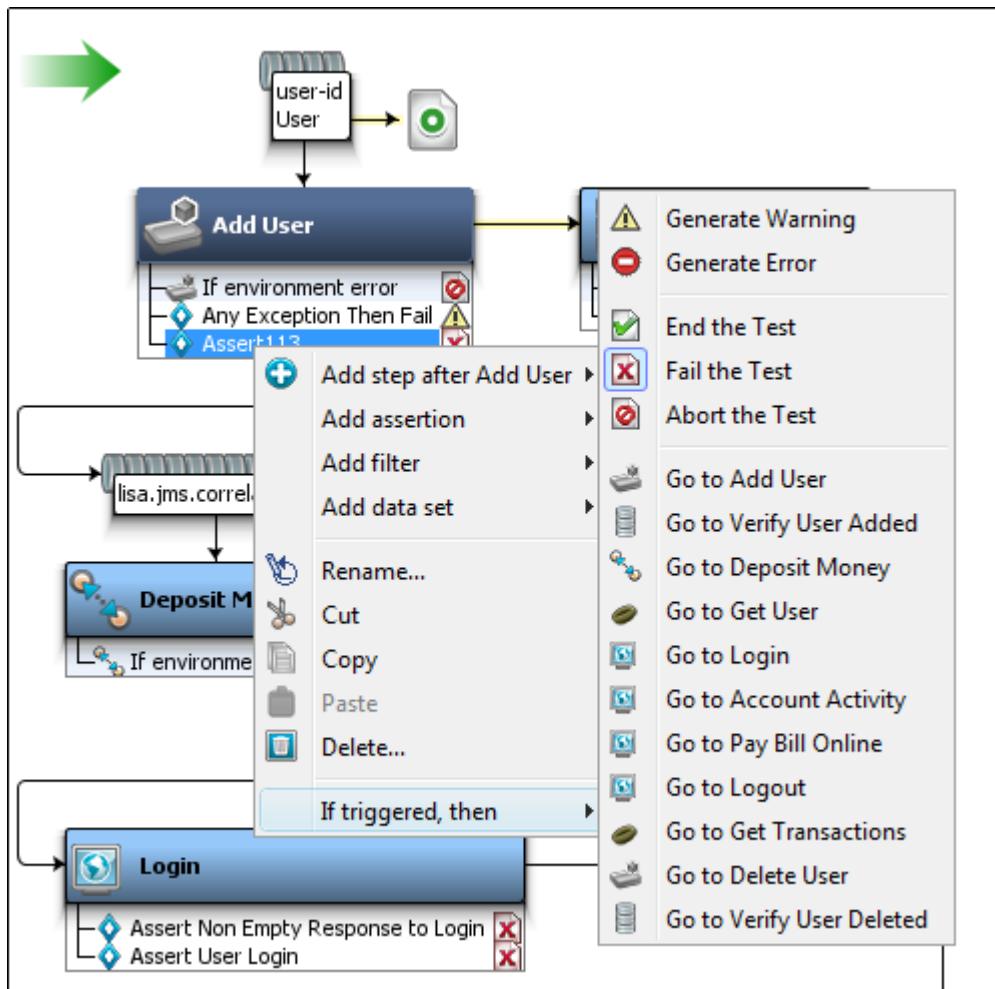
You can configure two other types of tests as next steps.

- 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 \(see page 351\)](#). In this project, you can create single or multiple test cases.



More Information:

- [Create a Test Case \(see page 482\)](#)
- [Open a Test Case \(see page 482\)](#)
- [Save a Test Case \(see page 483\)](#)
- [Test Cases in Model Editor \(see page 483\)](#)
- [Add Test Steps \(see page 484\)](#)
- [Configure the Next Step \(see page 484\)](#)
- [Branching and Looping in a Test Case \(see page 485\)](#)
- [Import Test Cases \(see page 486\)](#)
- [Response \(.rsp\) Documents \(see page 486\)](#)
- [Accessing Files in Another Project \(see page 487\)](#)
- [Test Case Toolbar \(see page 487\)](#)

Create a Test Case

You can create a test case by opening a project or [creating a project \(see page 351\)](#).

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 \(see page 483\)](#) 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 \(see page 486\)](#).

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:

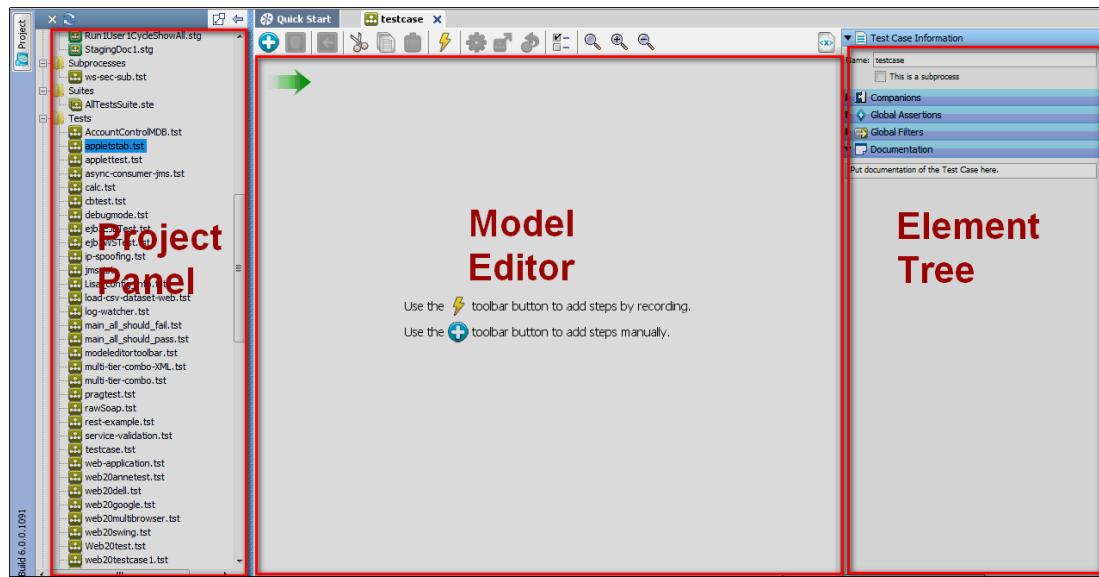
Do one:

- 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.

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 \(see page 420\)](#).

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 \(see page 433\)](#).

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 \(see page 405\)](#).

For more information about using the HTTP response to create assertions, see [Assertions \(see page 419\)](#).

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



Test case toolbar

Icon	Description
	Create a new step.



Create a New Step Icon



Delete a test step.

Delete a Test Step Icon



Set the currently selected step as the starting step in the workflow.

Set the Starting Step Icon



Cut the selected text.

Cut icon



Copy the selected text.

Copy Icon



Paste the selected text.

Clipboard icon



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)
- Mobile Recorder



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

Selenium Test Step Icon



Start a new Interactive Test Run.

Blue Gear Icon (Start the ITR)



Stage a quick test.

Stage Quick Test icon



Replay a test to a specific point.

Replay Test to Point Icon

Show model properties.



Model Properties icon



Reset zoom scale to 1:1.

Zoom icon



Zoom in.

Zoom in icon



Zoom out.

Zoom out icon



View XML source.

Icon - page with XML symbols

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 \(see page 490\)](#)).

- 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 \(see page 1788\)](#) test step simplifies calling a subprocess test case.



More Information:

- [Create a Subprocess Test Case \(see page 490\)](#)
- [Convert an Existing Test Case into a Subprocess \(see page 491\)](#)
- [Subprocess Example \(see page 492\)](#)

Create a Subprocess Test Case

Contents

- [Define Subprocess Input Properties \(see page 491\)](#)
- [Define Subprocess Output Parameters \(see page 491\)](#)

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

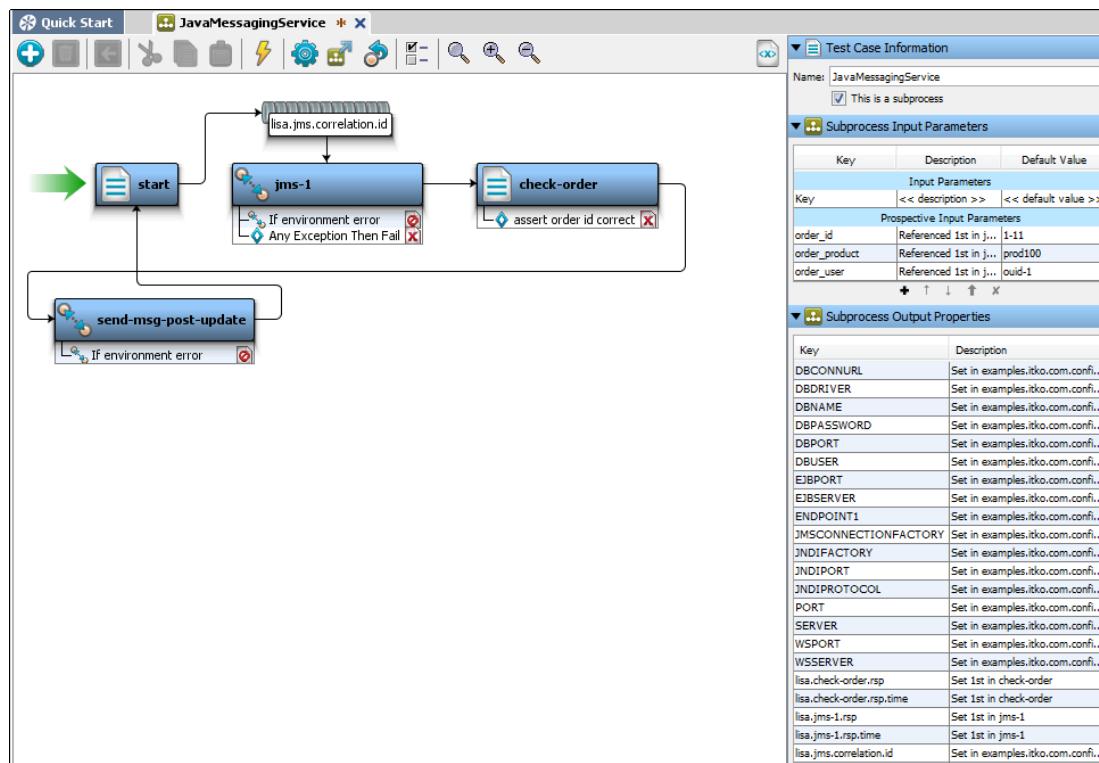
Follow these steps:

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.

Test Case Information

Name: JavaMessagingService
 This is a subprocess

Subprocess Input Parameters

Key	Description	Default Value
Input Parameters		
order_id	Referenced 1st in j...	1-11
order_num	order_num	4534433
order_product	Referenced 1st in j...	prod100
order_user	Referenced 1st in j...	oid-1
Prospective Input Parameters		
<input type="button" value="+"/> <input type="button" value="↑"/> <input type="button" value="↓"/> <input type="button" value="↑"/> <input type="button" value="X"/>		

Subprocess Example 2

The following graphic shows a test case with one test step: Execute Subprocess.

Parameters to Sub Process		
Key	Description	Value
order_id	Referenced 1st in jms-1	1-11
order_num	order_num	4534433
order_product	Referenced 1st in jms-1	prod100
order_user	Referenced 1st in jms-1	ouid-1

Result Properties

- DBCONNURL
- DBDRIVER
- DBNAME
- DBPASSWORD
- DBPORT
- DBUSER
- EJBPORT
- EJB SERVER
- ENDPOINT1
- JMSCONNECTIONFACTORY
- JNDIFACTORY
- JNDIIMPORT
- JNDIPROTOCOL
- PORT
- SERVER
- WSSPORT
- WSSERVER
- lisa.check-order.rsp
- lisa.check-order.rsp.time
- lisa.jms-1.rsp
- lisa.jms-1.rsp.time
- lisa.jms.correlation.id
- lisa.send-msg-post-update.rsp
- lisa.send-msg-post-update.rsp.time
- lisa.start.rsp
- lisa.start.rsp.time
- message
- order
- order.step.2.queue
- order_data_RowNum
- order_id

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 \(see page 495\)](#)
- [Building Audit Documents \(see page 510\)](#)
- [Understanding Events \(see page 513\)](#)
- [Generating Metrics \(see page 516\)](#)
- [Building Test Suites \(see page 516\)](#)

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 \(see page 496\)](#), 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 \(see page 503\)](#), specify the type of report that you want to create at run time.
6. In the [Metrics tab \(see page 504\)](#), specify the metrics that you want to record at run time.
7. In the [Documentation tab \(see page 507\)](#), enter descriptive information about the staging document.
8. (Optional) In the [IP Spoofing tab \(see page 507\)](#), enable, and configure IP spoofing.
9. (Optional) In the [Source View tab \(see page 510\)](#), 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.

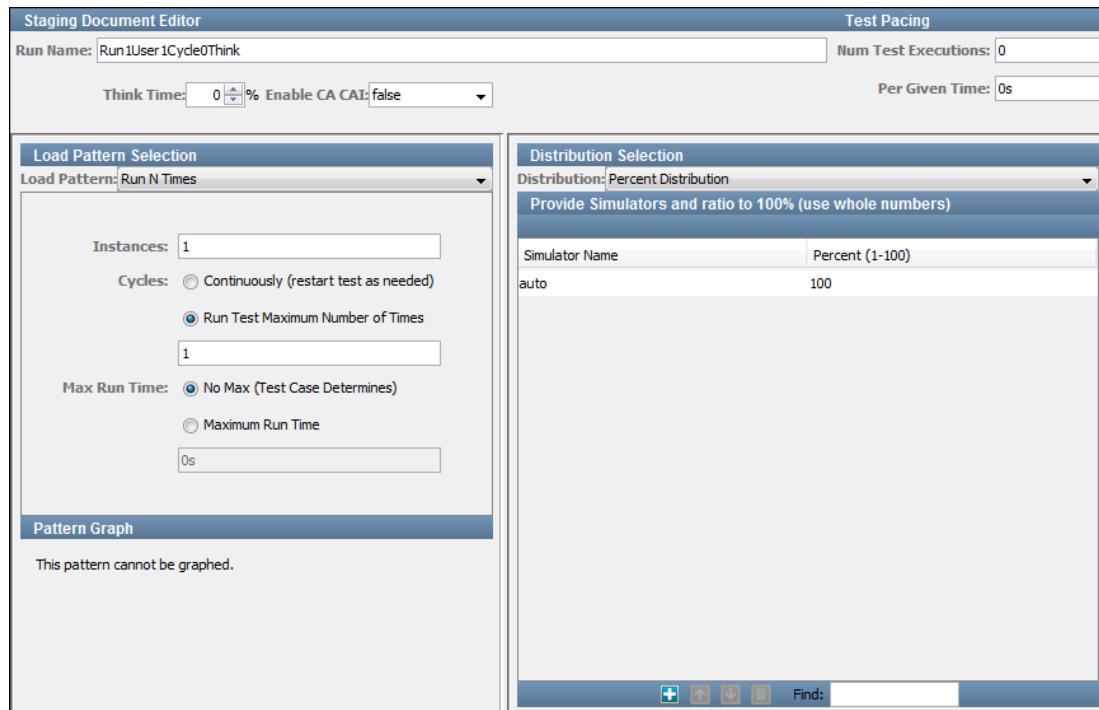
The Staging Document Editor contains the following tabs:

- [Base \(see page 496\)](#): To specify the basic parameters.
- [Reports \(see page 503\)](#): To select and add reports.
- [Metrics \(see page 504\)](#): To select metrics and specify your sampling intervals.
- [Documentation \(see page 507\)](#): To enter descriptive information about the staging document.
- [IP Spoofing \(see page 507\)](#): 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 \(see page 510\)](#): 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 CA Continuous 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 \(see page 562\)](#) 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 \(see page 498\)](#).

Distribution Selection Panel

The **Distribution Selection** panel lets you distribute virtual users (instances) over your running simulators. For more information, see [Distribution Selection \(see page 501\)](#).

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

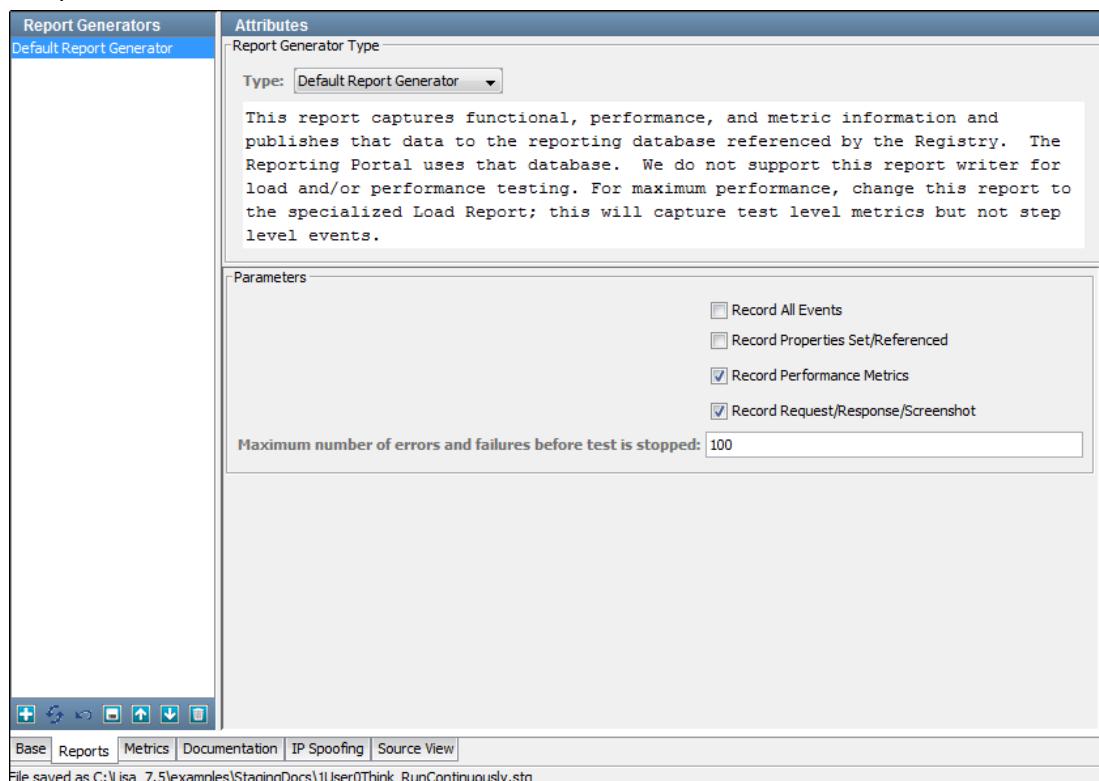
The **Round Robin Distribution** requests that DevTest control the allocation of virtual users (instances), based on a simple round-robin distribution pattern.

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 \(see page 598\)](#).

The metrics to capture for the reports are discussed more fully in [Generating Metrics \(see page 516\)](#).

Reports can be selected in the following modules and can be seen in the **Report Viewer**:

- [Stage a Quick Test \(see page 543\)](#)
- [Building Staging Documents \(see page 495\)](#)
- [Run a Test Suite \(see page 557\)](#)

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](#)

[Contents](#)

- [Add a Metric \(see page 506\)](#)
- [Set an Email Alert \(see page 506\)](#)
- [Sampling Parameters \(see page 506\)](#)

The **Metrics** tab of the Staging Document Editor lets you select the test metrics to record.

You can also set the sampling specifications for the collection of the metrics and set an email alert on any metric that you have selected.

Color	Short Name	Long Name	Scale	Edit Email Alert
Red	Event: Cycle failed/*	A LISA Event Metric: Event: Cycle failed/*	Auto	Set Alert
Red	Event: Step error/*	A LISA Event Metric: Event: Step error/*	Auto	Set Alert
Blue	Event: Step started/*	A LISA Event Metric: Event: Step started/*	Auto	Set Alert
Red	Event: Abort/*	A LISA Event Metric: Event: Abort/*	Auto	Set Alert
Magenta	LISA: Instances	A LISA built-in metric: number of running instances	Auto	Set Alert
Magenta	LISA: Avg Resp Time (ms)	A LISA built-in metric: average response time in milliseconds	Auto	Set Alert
Cyan	LISA: Steps per second	A LISA built-in metric: Steps per second (non-quiet) since the test started	Auto	Set Alert

Sample rate: Find: [+](#) [-](#) [↑](#) [↓](#) [X](#)

Samples per interval: 10 25 50 60 75 100 150 200 300 400 500

Base Reports Metrics Documentation IP Spoofing Source View

File Saved As: C:\Lisa\examples\StagingDocs\1User0Think_RunContinuously.stg

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 \(see page 1627\)](#).

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 \(see page 516\)](#).

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

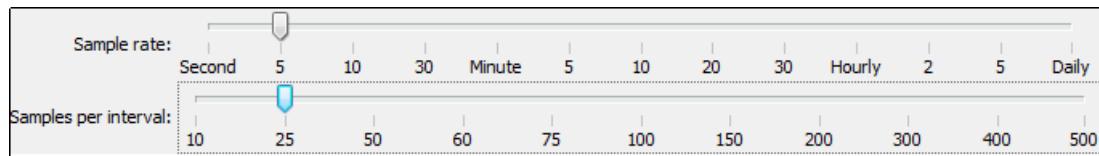
The bottom panel has two slider bars that let you set sampling parameters:

- **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 \(see page 503\)](#)).

Staging Document Editor - Documentation Tab

The **Documentation** tab of the Staging Document Editor lets you enter descriptive information about the staging document.

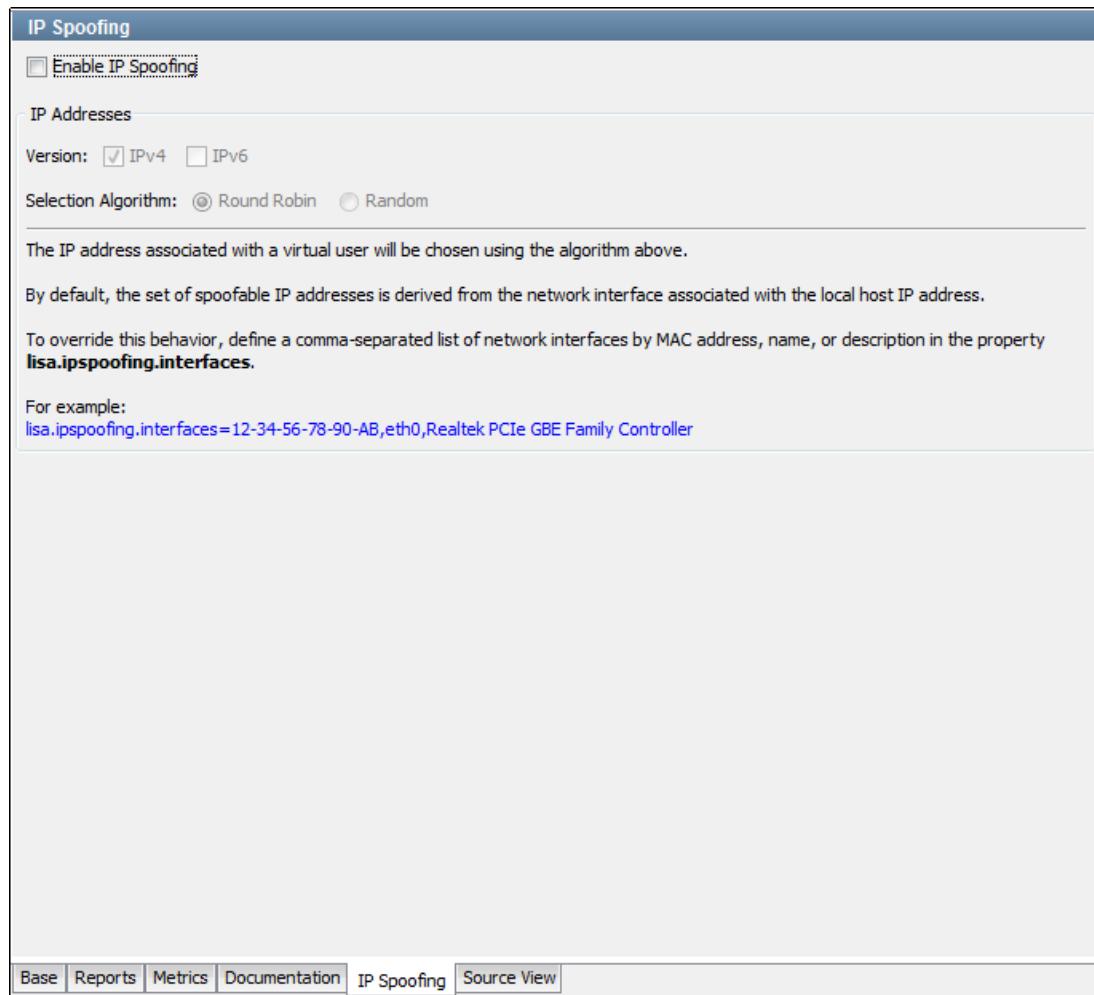
Staging Document Editor - IP Spoofing Tab

Contents

- [IP Spoofing Support \(see page 508\)](#)
- [IP Spoofing Scenarios \(see page 509\)](#)
- [Configuring IP Addresses in Windows \(see page 509\)](#)

The **IP Spoofing** tab of the Staging Document Editor lets multiple IP addresses on a network interface to be used when making network requests.

In a performance testing scenario, enabling IP spoofing against a system under test gives the appearance that requests are originating from many different virtual users. For 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 (see page 353). 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. CAI is enabled.
- **Run1User1CycleShowAll.stg:** This staging document runs a single virtual user one time with 100 percent think time. CAI is enabled.

Building Audit Documents

Contents

- [Event Audits Panel \(see page 512\)](#)
- [Run for Audit Info Panel \(see page 512\)](#)

An audit document lets you set success criteria for a test case in a suite.

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 \(see page 517\)](#) of the suite document editor. Be sure to select the **Record All Events** check box in the [Reports tab \(see page 520\)](#).

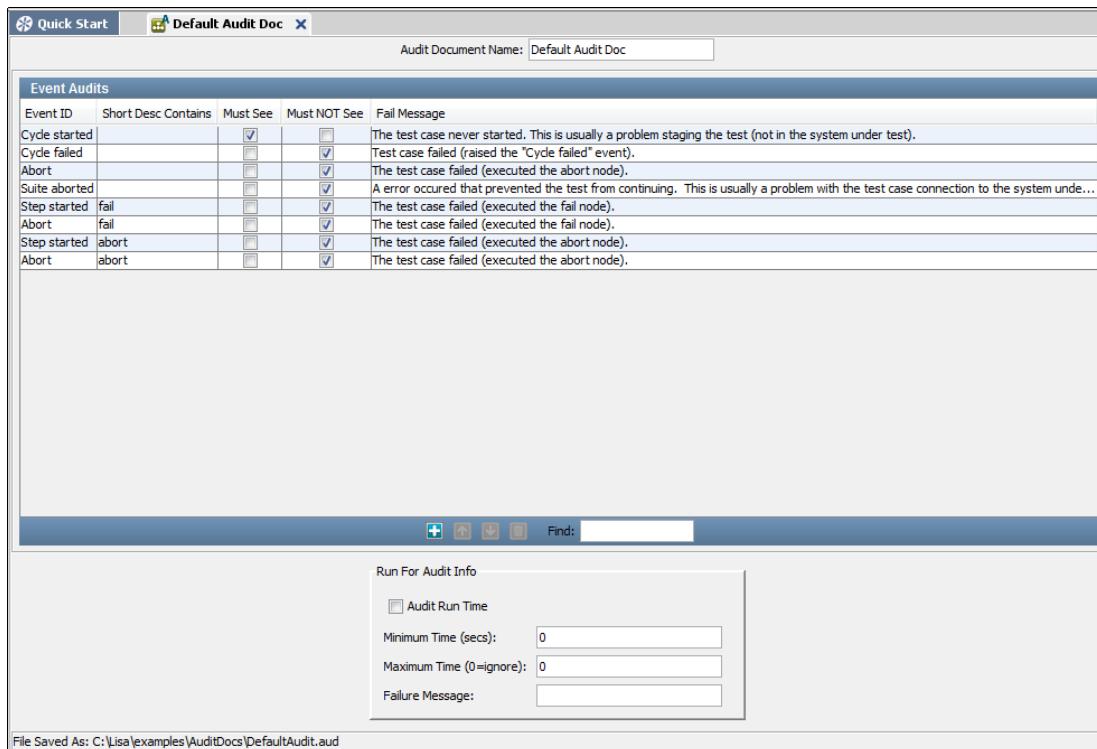
Audit documents are located in the **AuditDocs** folder of a project. The file extension is **.aud**.

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](#) (see page 512) panel.
4. To audit the execution time, configure the parameters in the [Run for Audit Info](#) (see page 512) 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.

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.

Events to Filter Out		Test Events					
	No Filter	Timestamp	Event	Simulator	Instance	Short Info	Long Info
<input type="checkbox"/>	Coordinator server started	2012-08-17 08:22:15,016	Log message	local	0/48	Will execute the default next step	null
<input type="checkbox"/>	Coordinator server ended	2012-08-17 08:22:14,762	Step response	local	0/48	create-consumer	0
<input type="checkbox"/>	Coordinator started	2012-08-17 08:22:14,762	Step response time	local	0/48	create-consumer	EXAMPLE-ASYNC-WRAPPER
<input type="checkbox"/>	Coordinator ended	2012-08-17 08:22:14,762	Property set	local	0/48	EXAMPLE-ASYNC-WRAPPER	NotSerializableStateWrapper >> AsyncQ...
<input type="checkbox"/>	Test started	2012-08-17 08:22:14,762	Property set	local	0/48	async.queuewrapper.control	AsyncQueueWrapperControl [Total Wrap...
<input type="checkbox"/>	Test ended	2012-08-17 08:22:14,762	Property set	local	0/48	lisa.hidden.jms.jnp://localhost:10...	NotSerializableStateWrapper >> SpySe...
<input type="checkbox"/>	Instance started	2012-08-17 08:22:14,762	Property set	local	0/48	lisa.hidden.jms.jnp://localhost:10...	NotSerializableStateWrapper >> Connec...
<input type="checkbox"/>	Instance ended	2012-08-17 08:22:14,762	Property set	local	0/48	lisa.hidden.jms.jnp://localhost:10...	NotSerializableStateWrapper >> javax.n...
<input type="checkbox"/>	Simulator started	2012-08-17 08:22:14,762	Property set	local	0/48	lisa.msg.sharingEngines.holder	SharingEngines [Size: 0]
<input type="checkbox"/>	Simulator ended	2012-08-17 08:22:14,762	Cycle started	local	0/48	create-consumer	
<input type="checkbox"/>	Cycle initialized	2012-08-17 08:22:14,762	Cycle started	local	0/48	986BD-4BED9102898BF64CCC3E8...	
<input type="checkbox"/>	Cycle started	2012-08-17 08:22:14,762	Property set	local	0/47	lisa.hidden.asyncconsumer.stop	true
<input type="checkbox"/>	Cycle ended normally	2012-08-17 08:22:14,762	Cycle history	local	0/47	986BD-4BED9102898BF397702E86...	
<input type="checkbox"/>	Cycle ended normally	2012-08-17 08:22:14,762	Cycle ending	local	0/47	986BD-4BED9102898BF397702E86...	Signaled to stop test
<input type="checkbox"/>	Cycle ended normally	2012-08-17 08:22:14,762	Log message	local	0/47	Clean Up	Executed clean up logic on Managed Stat...
<input type="checkbox"/>	Cycle ended normally	2012-08-17 08:22:14,762	Log message	local	0/47	Clean Up	Executed clean up logic on Managed Stat...
<input type="checkbox"/>	Cycle ended normally	2012-08-17 08:22:14,762	Log message	local	0/47	Clean Up	Executed clean up logic on Managed Stat...
<input type="checkbox"/>	Cycle ended normally	2012-08-17 08:22:14,762	Log message	local	0/47	Clean Up	Executed clean up logic on Managed Stat...
<input type="checkbox"/>	Cycle ended normally	2012-08-17 08:22:14,762	Log message	local	0/47	Clean Up	Executed clean up logic on Managed Stat...
<input type="checkbox"/>	Cycle ended normally	2012-08-17 08:22:14,762	Log message	local	0/47	Clean Up	Executed clean up logic on Managed Stat...
<input type="checkbox"/>	Cycle ended normally	2012-08-17 08:22:14,762	Property set	local	0/47	lisa.hidden.jms.jnp://localhost:10... <removed>	
<input type="checkbox"/>	Cycle ended normally	2012-08-17 08:22:14,742	Cycle ended normally	local	0/47	986BD-4BED9102898BF397702E86...	N/A
<input type="checkbox"/>	Cycle ended normally	2012-08-17 08:22:14,742	Step history	local	0/47	986BD-4BED9102898BF397702E86...	
<input type="checkbox"/>	Cycle ended normally	2012-08-17 08:22:14,742	Log message	local	0/47	Executing 'end of dataset' step end	N/A
<input type="checkbox"/>	Cycle ended normally	2012-08-17 08:22:14,742	Log message	local	0/47	Executing 'end of dataset' step co...	N/A
<input type="checkbox"/>	Cycle ended normally	2012-08-17 08:22:14,742	Log message	local	0/47	Will execute the default next step	
<input type="checkbox"/>	Cycle ended normally	2012-08-17 08:22:14,488	Step response	local	0/47	create-consumer	null
<input type="checkbox"/>	Cycle ended normally	2012-08-17 08:22:14,488	Step response time	local	0/47	create-consumer	0

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 \(see page 598\)](#). For more information about metrics, see [Generating Metrics \(see page 516\)](#).

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 \(see page 1581\)](#)

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 \(see page 598\)](#). For more information about metrics, see [Generating Metrics \(see page 516\)](#).

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 \(see page 1581\)](#)

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 the reporting console. 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 \(see page 543\)](#): Use to monitor the test.
- [Staging Documents \(see page 504\)](#): Use to include in reports.
- [Test Suite Documents \(see page 522\)](#): Use to include in reports.
- [Test Monitors \(see page 544\)](#): Use to monitor tests.

Building Test Suites

You can run/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 \(see page 517\)](#), specify basic information about the suite.
This information includes the suite name and the suite entries.

3. In the [Reports tab \(see page 520\)](#), specify the type of report that you want to create at run time.
4. In the [Metrics tab \(see page 522\)](#), specify the metrics that you want to record at run time.
5. In the [Documentation tab \(see page 524\)](#), enter descriptive information about the suite.
6. Select **File, Save** from the main menu.

Test Suite Editor

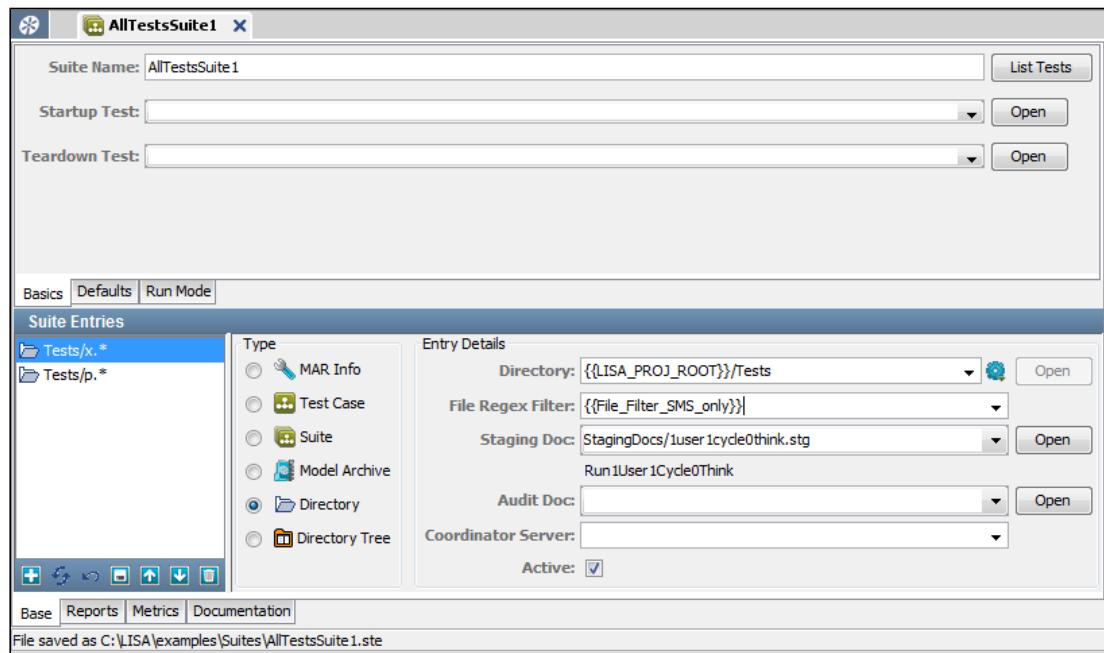
When you create a test suite, and when you open an existing test suite, the Test Suite Editor opens.

For more information about the specific tabs on this page, see the following:

- [Test Suite Editor - Base Tab \(see page 517\)](#)
- [Test Suite Editor - Reports Tab \(see page 520\)](#)
- [Test Suite Editor - Metrics Tab \(see page 522\)](#)
- [Test Suite Editor - Documentation Tab \(see page 524\)](#)

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/Tear down 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.

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.

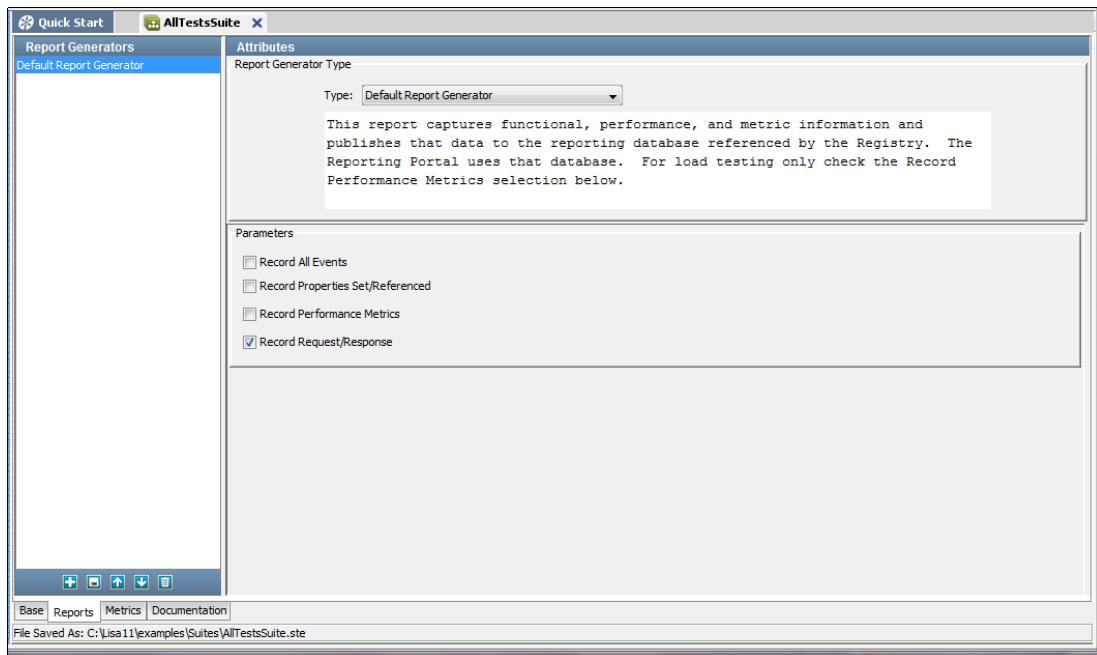
Click **Add**  on the toolbar to add this entry to the list shown in the left panel. You can delete entries by clicking **Delete** . You can rearrange entries by using the **Move Up**  and **Move Down**  icons.

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 \(see page 598\)](#). The metrics to capture for the reports are discussed more fully in [Metrics \(see page 516\)](#).



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 \(see page 600\)](#)
- [XML Report Generator \(see page 600\)](#)

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**

If selected, records the request and response.

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				
Color	Short Name	Long Name	Scale	Edit Email Alert
Red	Event: Cycle failed/*	A Event Metric: Event: Cycle failed/*	Auto	<input type="button" value="Set Alert"/>
Red	Event: Step error/*	A Event Metric: Event: Step error/*	Auto	<input type="button" value="Set Alert"/>
Blue	Event: Step started/*	A Event Metric: Event: Step started/*	Auto	<input type="button" value="Set Alert"/>
Red	Event: Abort/*	A Event Metric: Event: Abort/*	Auto	<input type="button" value="Set Alert"/>
Pink	DevTest: Instances	A built-in metric: number of running instances	Auto	<input type="button" value="Set Alert"/>
Pink	DevTest: Avg Resp Time (ms)	A built-in metric: average response time in milliseconds	Auto	<input type="button" value="Set Alert"/>
Cyan	DevTest: Avg steps per second since start	A built-in metric: Steps per second (non-quiet) since the test started	Auto	<input type="button" value="Set Alert"/>

Sampling Settings														
Sample rate:		Second	5	10	30	Minute	5	10	20	30	Hourly	2	5	Daily
Samples per interval:		10	25	50	60	75	100	150	200	300	400	500		

File saved as C:\Program Files (x86)\CA\DevTest\examples\Suites\AllTestsSuite.ste

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 \(see page 1627\)](#)
- [DevTest Test Event Metrics \(see page 1628\)](#)
- [JMX Metrics \(see page 1630\)](#)
- [SNMP Metrics \(see page 1629\)](#)
- [TIBCO Hawk Metrics \(see page 1633\)](#)
- [Windows Perfmon Metrics \(see page 1634\)](#)
- [UNIX Metrics Via SSH \(see page 1636\)](#)

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 \(see page 516\)](#).

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).

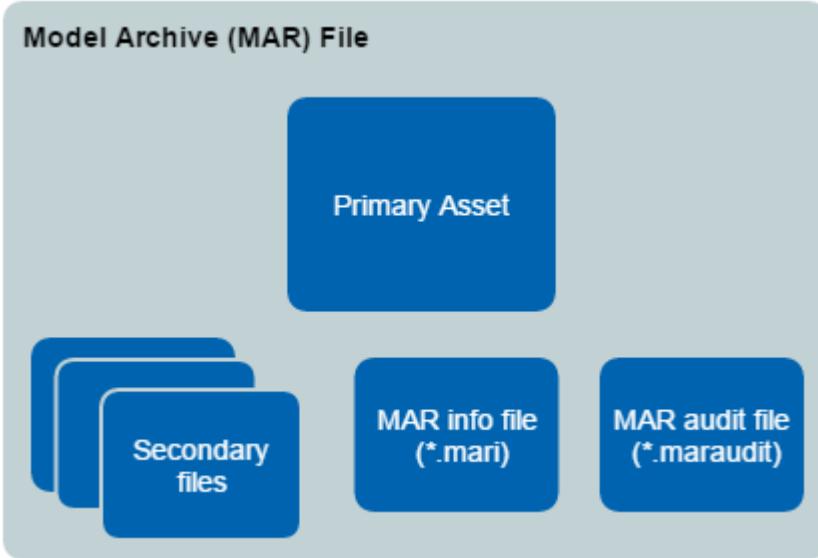
You can configure MARs from DevTest Workstation or by using the Make Mar command-line utility.

The main deployment artifact is a type of file referred to as a Model Archive (MAR). 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

Model Archive (MAR) 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.

In addition, MAR files contain the following files:

- [MAR info file \(see page 525\)](#)
- [MAR audit file \(see page 526\)](#)

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
---------------	--

MAR info files are located in the **Tests**, **Suites**, or **VirtualServices** folder of a project. When you use the stage/deploy related options, a MAR info file is created but not saved.

The **examples** project contains MAR info files that you can review.

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 \(see page 526\)](#)
- [Implicit \(see page 527\)](#)

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 \(see page 568\)](#) 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 that 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 \(see page 568\)](#):
 - 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 \(see page\)](#), 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 [Continuous Validation Service \(CVS\) \(see page 598\)](#).

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.

- **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)** (see page 524).

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** (see page 528) 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 \(ITR\) \(see page 534\)](#) 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 \(see page 480\)](#) in the case.
- In DevTest Workstation, you can [run a test case quickly \(see page 543\)](#) with minimal setup.
- In DevTest Workstation, you can [stage a test case \(see page 550\)](#). 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 \(see page 568\)](#) is a command-line utility that lets you run tests as a batch process.
- [LISA Invoke \(see page 572\)](#) is a REST-like web application that enables you to run test cases and suites with a URL.
- In the **Server Console**, you can [deploy the Model Archive \(MAR\) for a test case to a lab \(see page 597\)](#).
- You can schedule tests to run at regular time intervals by using the [Continuous Validation Service \(CVS\) \(see page 598\)](#).
- You can run tests as part of an automatic build process by using [Ant and JUnit \(see page 1371\)](#).



More Information:

- [Using the Interactive Test Run \(ITR\) Utility \(see page 534\)](#)
- [Stage a Quick Test \(see page 543\)](#)
- [Stage a Test Case \(see page 550\)](#)
- [Run a Test Suite \(see page 557\)](#)
- [Run a Selenium Integration Test Case \(see page 563\)](#)
- [Assumption of Load Testing \(see page 567\)](#)
- [Test Runner \(see page 568\)](#)
- [LISA Invoke \(see page 572\)](#)
- [REST Invoke API \(see page 1698\)](#)

- [Using the HP ALM - Quality Center Plug-in \(see page 576\)](#)
- [HTTP and SSL Debug Viewer \(see page 582\)](#)

Using the 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 \(see page 534\)](#)
- [Examine the Results of an ITR Run \(see page 537\)](#)
- [Graphical Text Diff Utility \(see page 541\)](#)

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.

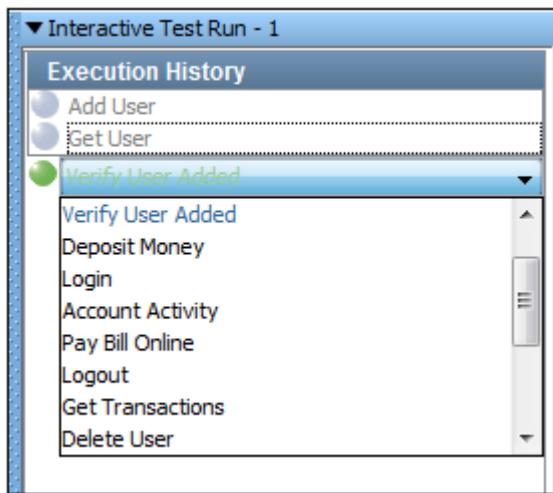
2. (Optional) To change any settings, use the **Settings** tab.

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.

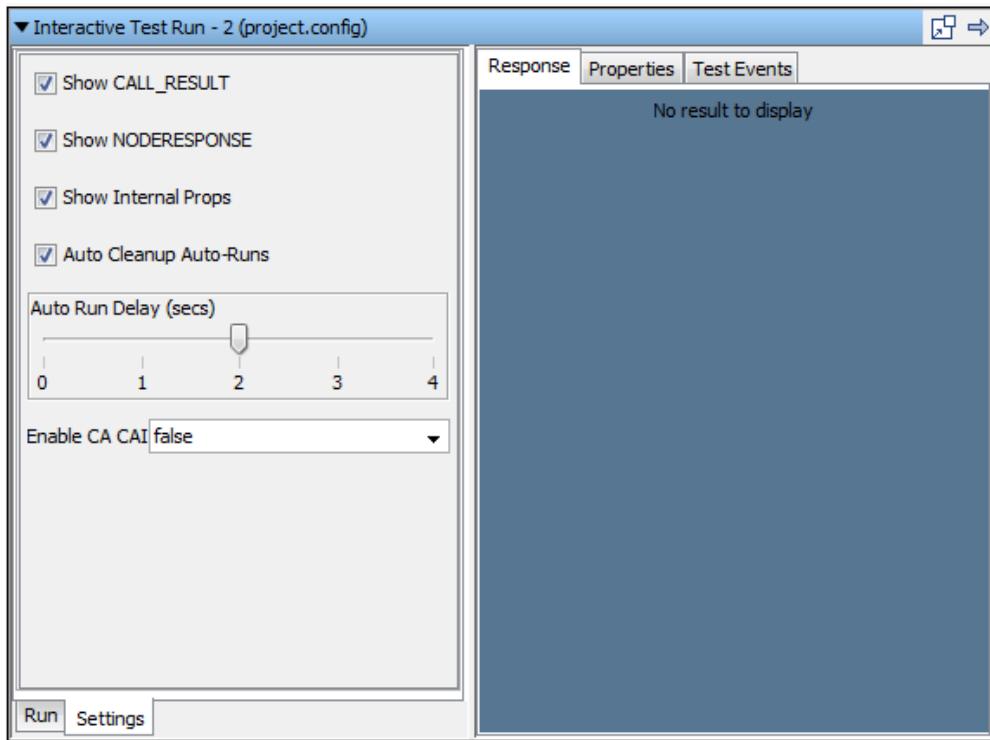
You manage the ITR by using the toolbar at the bottom of the **Run** tab.



-
- Run the next step in the test case. After the step has run, you can [examine the results \(see page 537\)](#) 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 CAI**
Controls whether CAI is enabled.
Default: false

Examine the Results of an ITR Run

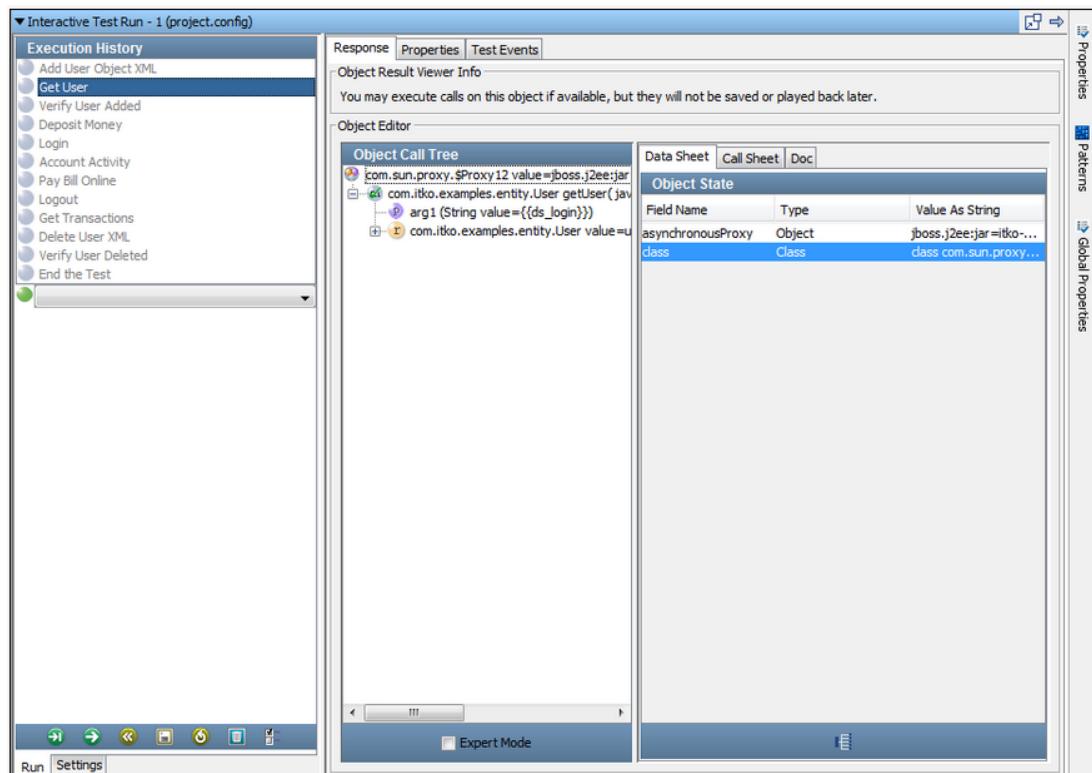
You can examine the results of the execution of a test step, during the run, between two steps, or at the end of the run.

Select the step in the **Execution History** list. Examine the information in the right panel, which contains the following tabs.

Response Tab

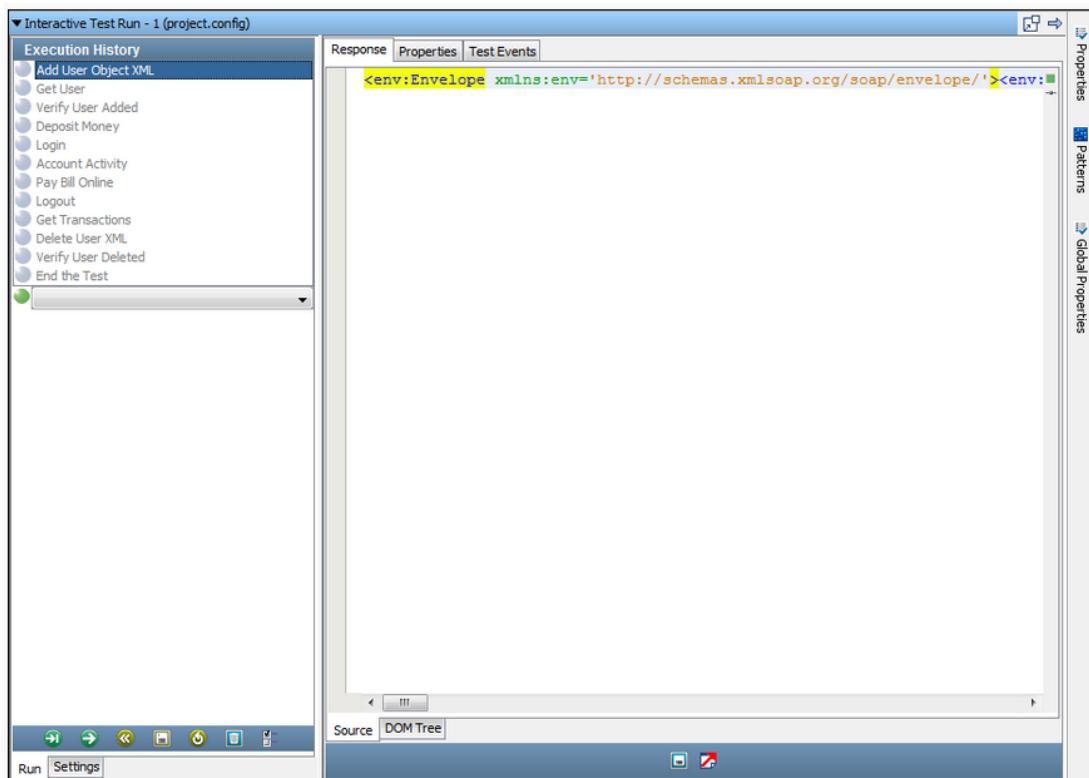
The **Response** tab displays the step response after executing a step.

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](https://wiki.ca.com/pages/viewpage.action?pageId=159580205) (<https://wiki.ca.com/pages/viewpage.action?pageId=159580205>).



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.

Key	Value	Previous Value
EISERVER	localhost	localhost
LISA_PROJ_URL	file:/C:/Program%20Files%20(x86)/CA.../CA...	file:/C:/Program%20Files%20(x86)/CA...
lisa.hidden.ejb.FF4CA9EFAD5F11E40...	SerialNum=139, of class unknown simp...	SerialNum=139, of class unknown simp...
LISA_LAST_STEP	end	Verify User Deleted
lisa.hidden.ejb.j2ee:jar=itko-ejb3-examples-1.0....jboss,j2ee:jar=itko-ejb3-examples-1.0....	deleteUser	deleteUser
LISA_DOC_PATH	C:\Program Files (x86)\CA\DevTest\ex...	C:\Program Files (x86)\CA\DevTest\ex...
JMSCONNECTIONFACTORY	ConnectionFactory	ConnectionFactory
ds_login	user-1868318651	user-1868318651
lisa.hidden.jms.jnp://localhost:1099 C...	NotSerializableStateWrapper >> org.jb...	NotSerializableStateWrapper >> org.jb...
lisa.Login.http.responseCode	200	200
ENDPOINT1	http://localhost:8080/itkoExamples/EJB... http://localhost:8080/itkoExamples/EJB...	
lisa.msg.sharingEngines.holder	SharingEngines [Size: 0]	SharingEngines [Size: 0]
lisa.end.rsp	The test has ended	
fl_checking_account_id	17312746583	17312746583
lisa.Verify User Added.rsp.time	37	37
lisa.hidden.jms.jnp://localhost:1099Qu...	NotSerializableStateWrapper >> Conn...	NotSerializableStateWrapper >> Conn...
LISA_MODEL_ID	456306a0	456306a0
lisa.Deposit Money.rsp	SpyTextMessage {Header { jmsDesti... SpyTextMessage {Header { jmsDesti...	
lisa.Add User Object XML.http.respons...	200	200
lisa.Delete User XML.httpheaders.rsp	HTTP/1.1 200 OK=&Date=Thu, 05 Feb ... HTTP/1.1 200 OK=&Date=Thu, 05 Feb ...	
lisa.ws.conprops	org.apache.axis.components.net.Defa... org.apache.axis.components.net.Defa...	
lisa.Pay Bill Online.http.responseCode	200	200
order.step.2.queue	queue/C	queue/C
JNDIPORT	1099	1099
LISA_COOKIE_localhost/lisabank/JSESS...	[version: 0][name: JSESSIONID][value:... [version: 0][name: JSESSIONID][value:...	
lisa.Login.rsp	<!- jspstart: rootLayout.jsp --><html... <!- jspstart: rootLayout.jsp --><html...	
lisa.Logout.http.headers	Server=Apache-Coyote/1.1&X-Powere... Server=Apache-Coyote/1.1&X-Powere...	
lisa.Login.rsp.time	5620	5620
lisa.hidden.ITR	true	true
lisa.Delete User XML.rsp	<env:Envelope xmlns:env='http://sche... <env:Envelope xmlns:env='http://sche...	
lisa.Verify User Added.rsp	<?xml version='1.0' encoding='UTF-8'?<?xml version='1.0' encoding='UTF-8'?<	
DBNAME	itko_examples	itko_examples
DBUSER	sa	sa
lisa.Add User Object XML.cookies2.rsp		

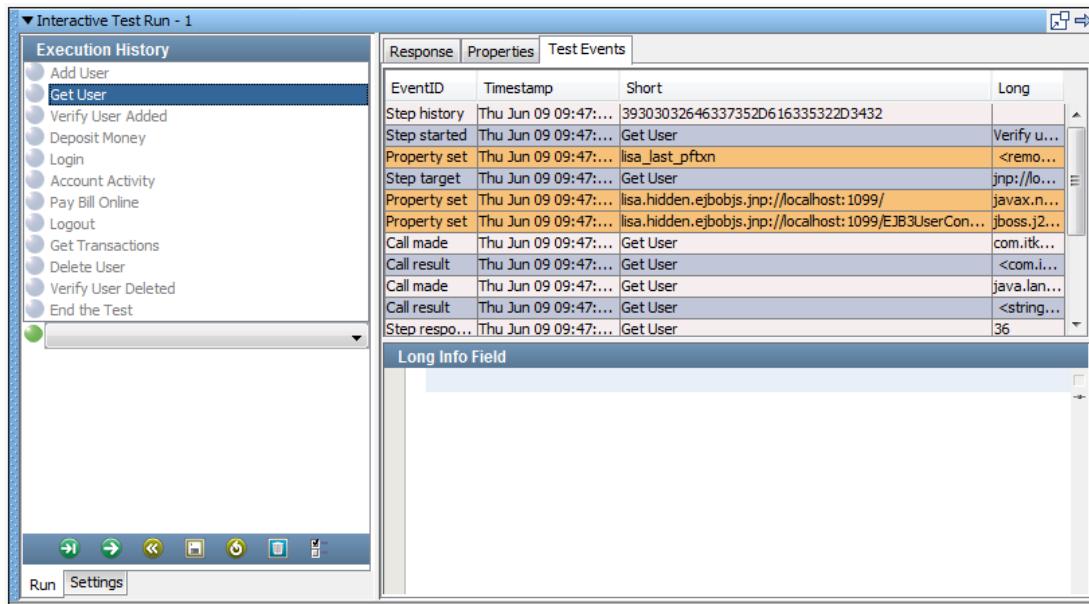
ITR Results Properties tab

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 \(see page 1581\)](#) 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

Contents

- [Start the Graphical Text Diff Utility \(see page 541\)](#)
- [Graphical Text Diff Utility Properties \(see page 543\)](#)

A graphical XML diff engine and visualizer are available to compare XML files and graphically display their differences.

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.

```

<?xml version="1.0" encoding="UTF-8"?>
<env>http://schemas.xmlsoap.org/soap/envelope/</env>
<?xml version="1.0" encoding="UTF-8"?>
<env>http://schemas.xmlsoap.org/soap/envelope/</env>
<rResponse xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/">
<ns2:to>itko.test@itko.com</ns2:to>
<ns2:from>itko</ns2:from>
<ns2:subject>test</ns2:subject>
<ns2:login>itko</ns2:login>
<ns2:phone>650-234-1212</ns2:phone>
<ns2:roleKey>REGULAR</ns2:roleKey>
<ns2:>
<ns2:erResponse>

```

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 \(see page 1482\)](#).

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 \(see page 495\)](#) as in the case of a proper [test case \(see page 550\)](#). 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 \(see page 544\)](#) window opens.

Test Monitor Window - Quick Test

When you [stage a quick test \(see page 543\)](#), 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 \(see page 544\)](#) Tab displays collected metrics as a function of time. After the test starts, this display cannot be changed.
- The [Events Tab \(see page 546\)](#) 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 \(see page 548\)](#) is displayed.

If you run a baseline test case, then the **Consolidated Baseline** tab is also displayed. For more information, see [Using CA Continuous Application Insight \(see page 1007\)](#).

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.

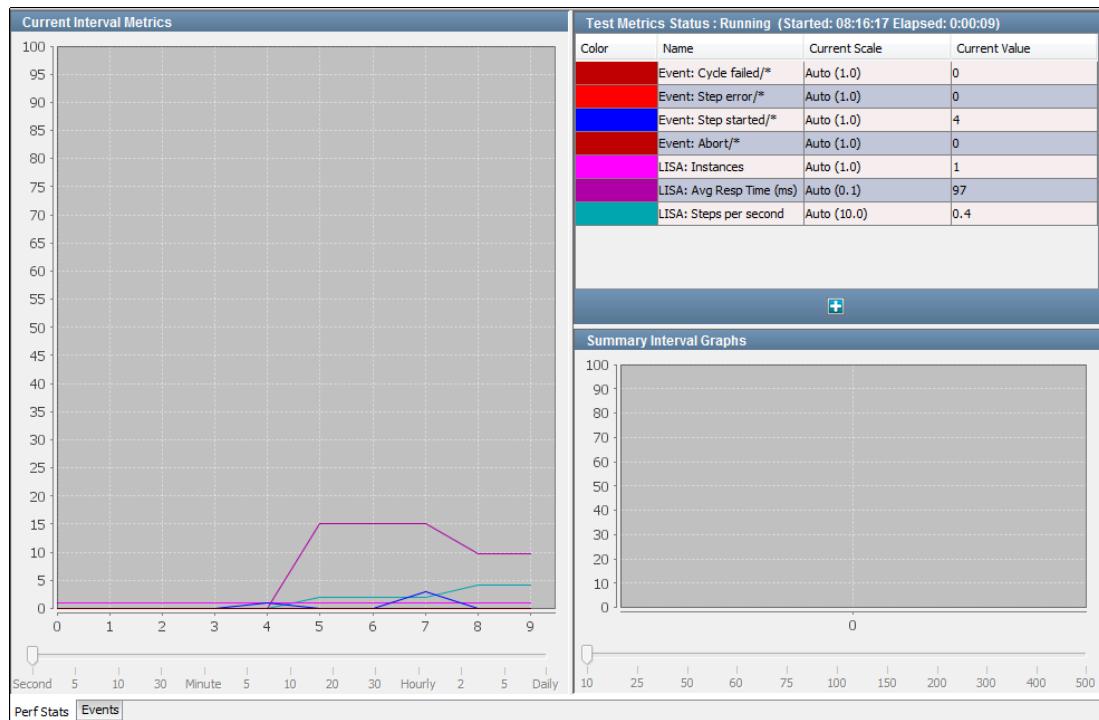


Image of the Perf Stats tab of the Test Monitor window

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 \(see page 516\)](#).

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**

The screenshot shows the DevTest Solutions interface with the following panels:

- Events to Filter Out**: A list of events with checkboxes. The "No Filter" option is selected.
- Test Events**: A table with columns: Timestamp, Event, Simulator, Instance, Short Info, and Long Info. The table lists various log messages and step responses from a local simulator instance.
- Simulators**: A table showing a single local simulator instance.
- Add Simulator**: A form for adding a new simulator.
- Long Info Field**: A large text area for viewing detailed event information.
- Perf Stats** and **Events**: Buttons at the bottom of the interface.

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.

The screenshot shows the DevTest Solutions Test Monitor window. At the top, there are three tabs: 'Quick Start', 'multi-tier-combo', and 'multi-tier-combo [Run1User1Cycle]'. The third tab is selected. Below the tabs is a table with columns: Timestamp, Event, Simulator, Instance, Short Info, and Long Info. A single row is present: '2012-05-03 07:56:01,104' under Timestamp, 'Cycle failed' under Event, 'local' under Simulator, '0/0' under Instance, '3263639356332662D656366652D3433' under Short Info, and '<?xml version="1.0"?><CycleHistory><Ende...' under Long Info. Below the table is a status bar with icons for back, forward, stop, and refresh, followed by the path 'C:\Lisa\reports\error_report9847224898932858.html'. The main content area displays the 'Cycle Execution History for Test: multi-tier-combo' report. This report includes a table with the following data:

Run Status	Failed
Unique ID	3263639356332662D656366652D3433
Staging ID	3234393333238322D393436392D3432
Test Case Run	multi-tier-combo [Run1User1Cycle]
Simulator	local
Instance	0
Cycle	0
Warning Count	0
Error Count	0
Configuration	C:\Users\arhoades\lisatmp_6.0.7\lads\3234393333238322D393436392

At the bottom of the report area, there are tabs for 'Done', 'Perf Stats', 'Events', and 'Failures(1)'. The 'Failures(1)' tab is selected.

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 on the main toolbar.

When the quick test starts, the **Play** icon changes to a **Stop** icon.



To stop a quick test, click 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 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.

The main toolbar in DevTest Workstation includes a **Lab Status** icon. If you stage a test case to a cloud-based lab, the icon indicates when the lab is being provisioned and then ready.

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 \(see page 393\)](#) from the **Configuration** drop-down list.
If you leave this field blank, the default configuration is used.
4. Select the [staging document \(see page 495\)](#) from the **Staging doc** drop-down list.
5. Select the [coordinator server \(see page 321\)](#) or (if available) a [cloud-based lab \(see page 585\)](#) from the **Coordinator server** drop-down list.
Coordinator server names include the associated lab. For example, Coordinator@Default indicates that the Default lab is used. For cloud-based labs, the lab is started and the test case is staged to the coordinator in the lab.



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**.

- If you selected a coordinator server, the [Test Monitor \(see page 551\)](#) window opens. You can now select the metrics and events to monitor, and then start the test case.
- If you selected a cloud-based lab, a message indicates that it takes some time to provision the lab. You are notified when the process has completed. Click **OK**. When the **Provisioning Complete** message appears, click **OK**. The [Test Monitor \(see page 551\)](#) 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 \(see page 550\)](#), 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 \(see page 551\)](#) Tab lets you select metrics, and display them as a function of time.
- The [Events \(see page 553\)](#) 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 \(see page 555\)](#) is also displayed.

If you run a baseline test case, then the **Consolidated Baseline** tab is also displayed. For more information, see [Using CA Continuous Application Insight \(see page 1007\)](#).

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.

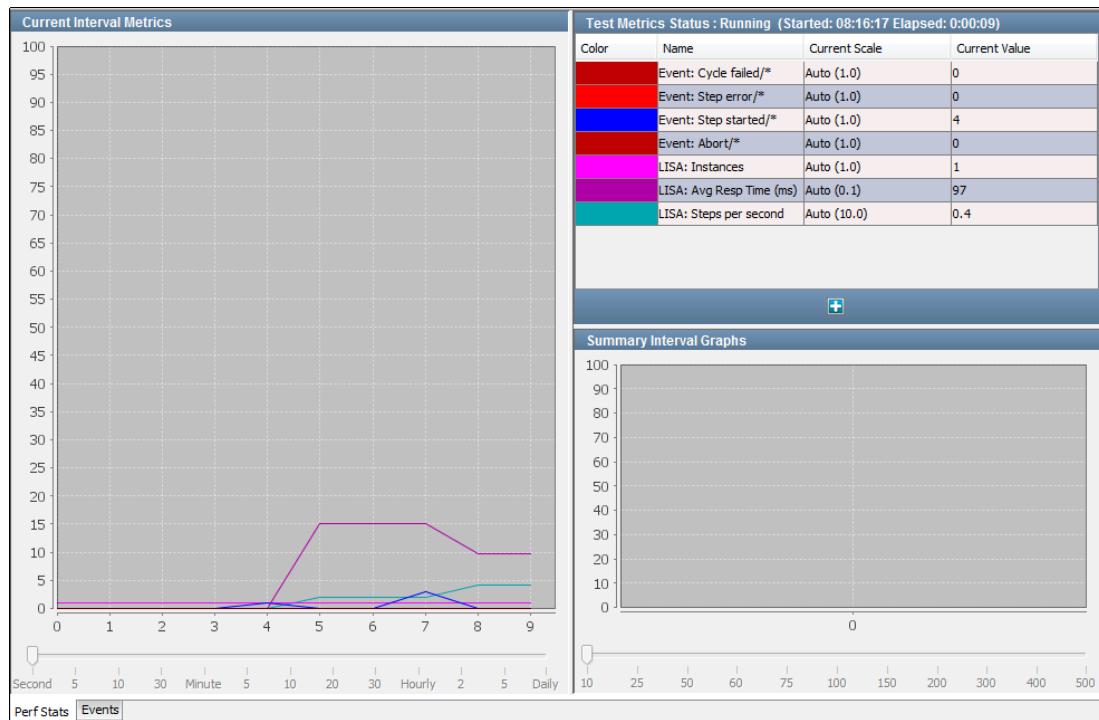


Image of the Perf Stats tab of the Test Monitor window

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 \(see page 516\)](#).

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.

Timestamp	Event	Simulator	Instance	Short Info	Long Info
2012-08-17 08:22:15,016	Log message	local	0/48	Will execute the default next step	
2012-08-17 08:22:14,762	Step response	local	0/48	create-consumer	null
2012-08-17 08:22:14,762	Step response time	local	0/48	create-consumer	0
2012-08-17 08:22:14,762	Property set	local	0/48	EXAMPLE-ASYNC-WRAPPER	NotSerializableStateWrapper >> AsyncQ...
2012-08-17 08:22:14,762	Property set	local	0/48	async.queuewrapper.control	AsyncQueueWrapperControl [Total Wrap...
2012-08-17 08:22:14,762	Property set	local	0/48	lisa.hidden.jms.jnp://localhost:10...	NotSerializableStateWrapper >> SpySe...
2012-08-17 08:22:14,762	Property set	local	0/48	lisa.hidden.jms.jnp://localhost:10...	NotSerializableStateWrapper >> Connec...
2012-08-17 08:22:14,762	Property set	local	0/48	lisa.hidden.jms.jnp://localhost:10...	NotSerializableStateWrapper >> javax.n...
2012-08-17 08:22:14,762	Step started	local	0/48	create-consumer	
2012-08-17 08:22:14,762	Cycle started	local	0/48	986BD-4BED91028988F64CCC3E8...	
2012-08-17 08:22:14,762	Property set	local	0/47	lisa.hidden.asyncconsumer.stop	true
2012-08-17 08:22:14,762	Cycle history	local	0/47	986BD-4BED91028988F397702E86...	
2012-08-17 08:22:14,762	Cycle ending	local	0/47	986BD-4BED91028988F397702E86...	Signaled to stop test
2012-08-17 08:22:14,762	Log message	local	0/47	Clean Up	Executed clean up logic on Managed Stat...
2012-08-17 08:22:14,762	Log message	local	0/47	Clean Up	Executed clean up logic on Managed Stat...
2012-08-17 08:22:14,762	Log message	local	0/47	Clean Up	Executed clean up logic on Managed Stat...
2012-08-17 08:22:14,762	Log message	local	0/47	Clean Up	Executed clean up logic on Managed Stat...
2012-08-17 08:22:14,762	Log message	local	0/47	Clean Up	Executed clean up logic on Managed Stat...
2012-08-17 08:22:14,762	Property set	local	0/47	lisa.hidden.jms.jnp://localhost:10... <removed>	
2012-08-17 08:22:14,742	Cycle ended normally	local	0/47	986BD-4BED91028988F397702E86...	N/A
2012-08-17 08:22:14,742	Cycle history	local	0/47	986BD-4BED91028988F397702E86...	
2012-08-17 08:22:14,742	Log message	local	0/47	Executing 'end of dataset' step end	N/A
2012-08-17 08:22:14,742	Log message	local	0/47	Executing 'end of dataset' step co...	N/A
2012-08-17 08:22:14,742	Log message	local	0/47	Will execute the default next step	
2012-08-17 08:22:14,488	Step response	local	0/47	create-consumer	null
2012-08-17 08:22:14,488	Step response time	local	0/47	create-consumer	0

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.

The screenshot shows the DevTest Solutions Test Monitor window. At the top, there are three tabs: 'Quick Start', 'multi-tier-combo', and 'multi-tier-combo [Run1User1Cycle]'. The third tab is selected. Below the tabs is a table with columns: Timestamp, Event, Simulator, Instance, Short Info, and Long Info. A single row is present: '2012-05-03 07:56:01,104' in 'Timestamp', 'Cycle failed' in 'Event', 'local' in 'Simulator', '0/0' in 'Instance', '3263639356332662D656366652D3433' in 'Short Info', and '<?xml version="1.0"?><CycleHistory><Ende...' in 'Long Info'. Below the table is a status bar with icons for back, forward, stop, and refresh, followed by the path 'C:\Lisa\reports\error_report9847224898932858.html'. The main content area displays the 'Cycle Execution History for Test: multi-tier-combo' report. This report includes a summary table with the following data:

Run Status	Failed
Unique ID	3263639356332662D656366652D3433
Staging ID	3234393333238322D393436392D3432
Test Case Run	multi-tier-combo [Run1User1Cycle]
Simulator	local
Instance	0
Cycle	0
Warning Count	0
Error Count	0
Configuration	C:\Users\arhoades\lisatmp_6.0.7\lads\3234393333238322D393436392

Below the summary table is a 'Done' button and a navigation bar with 'Perf Stats', 'Events', and 'Failures(1)'. The 'Failures(1)' tab is highlighted.

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 on the main toolbar.

When the test case starts, the **Play** icon changes to a **Stop** icon.



To stop a test case, click 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 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.

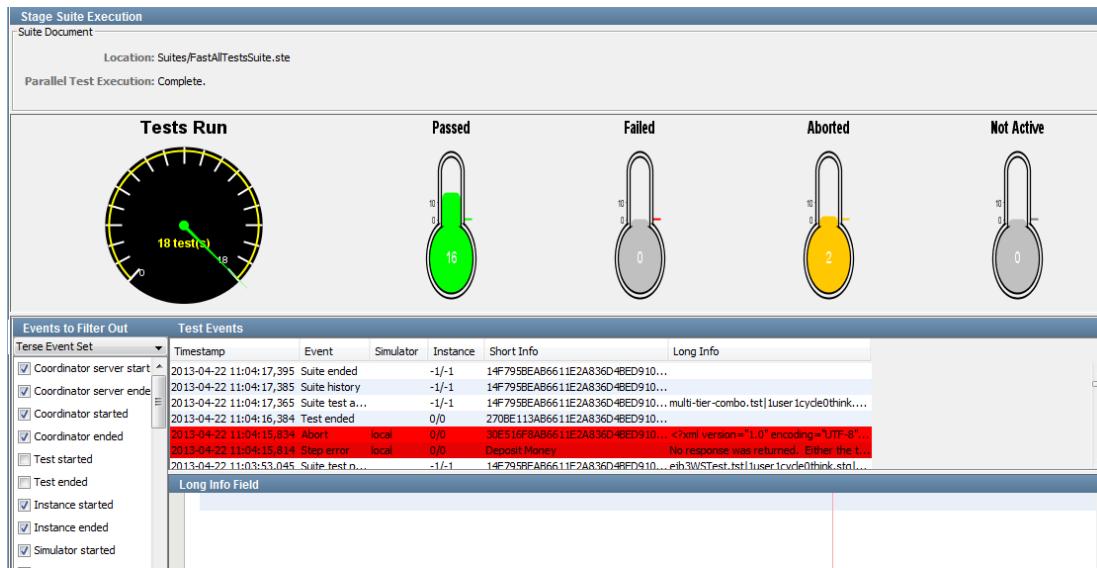
Follow these steps:

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 \(see page 557\)](#) window opens and the tests start.

Stage Suite Execution

When you click **Stage** during the process of [running a test suite \(see page 557\)](#) 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 \(see page 558\)](#): Lists requested events as they occur
- [Results Tab \(see page 559\)](#): 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

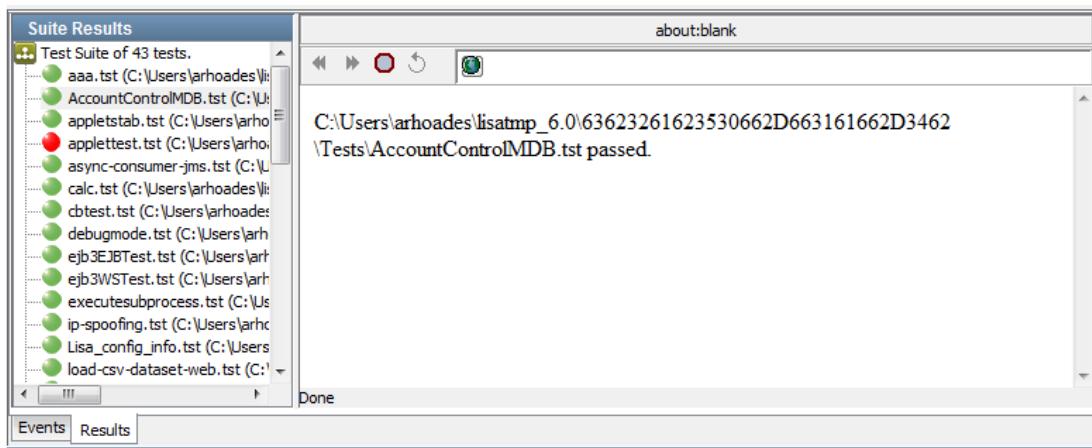
The **Test Events** area lists the events as they occur, with the most recent on the top of the list. You can list events in real time by selecting the **Auto Refresh** check box at the bottom of the panel. You can refresh the list manually by clicking the **Refresh** icon, with the **Auto Refresh** 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.



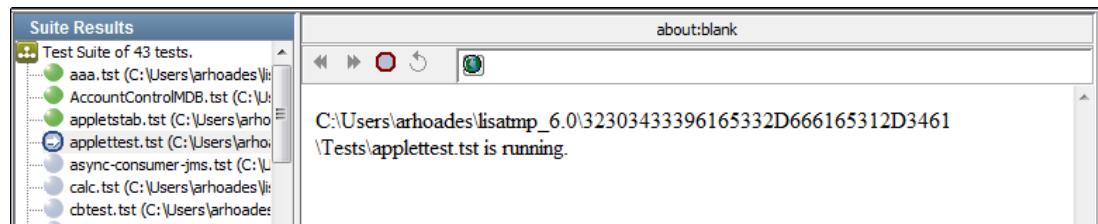
To display the test monitor for tests that are still running, click **View Test** at the bottom of the panel.

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

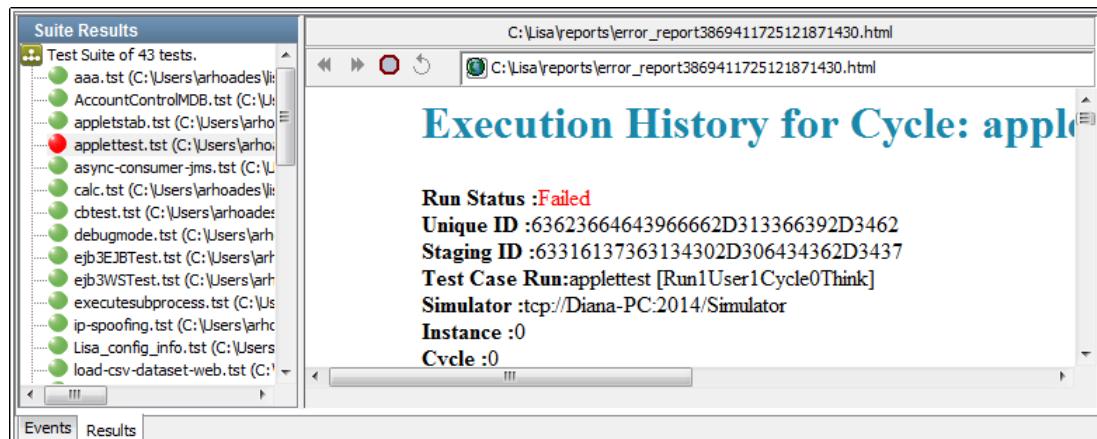
The **Status** window displays the status of the test that is selected on the left.

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 Stops the test after it reaches the next logical end/fail. It does not start a new cycle.

Stop Test icon



Kill Test Stops the test immediately after the current step completes.

Kill Test icon



View Test This tool will work only for currently running tests. This starts the test monitor for the selected test.

View Test icon

The second set of icons manages the test suite itself, or when a suite is running:



Run Suite Starts/restarts the suite test.

Run Suite icon



Close Suite Close the **Stage Suite Execution** window.

Close Suite icon

	View Test	Starts the test monitor for the selected test.
View Test icon		

**More Information:**

- [Use the Registry Monitor \(see page 1458\)](#)

Use the Load Test Optimizer

The Load Test Optimizer appears in the [Registry Monitor \(see page 1458\)](#) 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

Contents

- [Run a Selenium Integration Test on Google Chrome \(Local\) \(see page 563\)](#)
- [Run a Selenium Integration Test on Microsoft Internet Explorer \(Local\) \(see page 564\)](#)
- [Run a Selenium Integration Test on Apple Safari \(Local\) \(see page 565\)](#)
- [Run a Selenium Integration Test on a Remote Browser \(see page 566\)](#)

[Selenium Integration test steps \(see page 1860\)](#) let you import test scripts for web-based user interfaces from Selenium Builder to DevTest Solutions. The recording of these test scripts requires Selenium Builder, which is only supported in Firefox. After you import the test to DevTest, you can run the test in Mozilla Firefox, Google Chrome, Apple Safari, or Internet Explorer. You can also run the test in either a local or remote browser.

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 \(see page 533\)](#).

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. 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.



Safari is only supported 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.safariflattened** 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`

- 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
```

- 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.browser.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 \(see page 393\)](#).



Note: Add these properties to project.config before you add them to other project configuration files.

- 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>.

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 CAI 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.

Other results of the load testing assumption are:

- The **Server Console** displays an asterisk next to a test name.
- The **Reporting Console** does not show the normal amount of detail.
- 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] [[-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 Solutions with Ant and JUnit \(see page 1371\)](#).



More Information:

- [Run a MAR with Test Runner \(see page 570\)](#)
- [Run a Test Case with Test Runner \(see page 570\)](#)
- [Run a Suite with Test Runner \(see page 570\)](#)
- [Other Test Runner Options \(see page 571\)](#)
- [Multiple Test Runner Instances \(see page 572\)](#)
- [Test Runner Log File \(see page 572\)](#)

Run a MAR with Test Runner

To run a [Model Archive \(MAR\) \(see page 524\)](#) 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 \(see page 571\)](#).

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 \(see page 571\)](#).

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.

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 \(see page 1448\)](#).

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}}
```

LISA Invoke

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

The syntax for running test cases with LISA Invoke is:

```
/lisa-invoke/runTest?testCasePath=testCasePath&stagingDocPath=stagingDocPath&
[configPath=configPath]&[async=true]&[coordName=csName]
```

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.

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]
```

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.

Run Model Archives with LISA Invoke

The syntax for running model archives (MARs) with LISA Invoke is:

```
/lisa-invoke/runMar?marOrMariPath=[marOrMariPath]&[async=true]
```

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.

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.

Using the HP ALM - Quality Center Plug-in

The HP ALM - Quality Center plug-in lets you load and run a DevTest test case as a Quality Center test from the HP ALM - Quality Center suite. You can import into and can run DevTest tests from Quality Center. 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 and DevTest callbacks returning from any system under test. DevTest tests are executable inside the workflow of Quality Center, and they report back results to maintain the context and status of the testing process.



Note: For information about installing the HP ALM - Quality Center plug-in, see *Installing*.



More Information:

- [Set up DevTest Tests in HP ALM - Quality Center \(see page 576\)](#)
- [Run DevTest Tests in HP ALM - Quality Center \(see page 579\)](#)
- [QCRunner Command-Line Interface \(see page 580\)](#)
- [HP ALM - Quality Center Plug-in Troubleshooting \(see page 581\)](#)

Set up DevTest Tests in HP ALM - Quality Center

The plug-in uses VAPI-XP to integrate with HP ALM - Quality Center.



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 clicking **Start**, **All Programs**, **DevTest**, **Quality Center Plugin**. The templates are functionally equivalent.

The DevTest test can be a test case file (as well as the configuration file), a MAR file, or a MAR info file.

For test case steps to appear in ALM, ensure they are not marked as "Quiet". For test cases, you can also attach a staging document. If you do not attach a staging document, a staging document is automatically created with these characteristics:

- One user
- One cycle
- Zero think time

To link an ALM test to DevTest, using the "Test Plan" feature of ALM, follow these steps:

1. Create a new test.
2. Select **VAPI-XP-TEST** from the Type dropdown and give the test a name.
3. Click **OK**.
4. Choose the scripting language you would like to use and then click **Finish**.
You will add the script later.
5. Once the test is created, click on 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 has been added, the final step is to add references to the test, staging, and config files in the Attachments tab. We recommend that the test, staging, and config files be added as links (URLs) rather than attaching the actual file itself. This will allow for any changes to the DevTest test to not have to be uploaded back to ALM. To do this, choose the link icon and then file a properly constructed URL to the file. If the file is on the same machine as the ALM instance, it should be in a form similar to this:

```
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
```

For MAR files and MAR info files, you cannot attach any additional files.

In this procedure, you create one or more URL attachments. Here is an example URL for a test case file:

```
file:///C:/Lisa/examples/Tests/rest-example.tst
```

Here is an example URL for a MAR info file:

```
file:///C:/Lisa/examples/MARInfos/rest-example.mari
```

The following graphic shows the **Attachments** tab. The URL for a test case file has been added.

The screenshot shows the 'Attachments' tab in the DevTest Solutions interface. A single attachment, 'URL Attachment.url', is listed. The table has columns for Name, Size, and Modified. The attachment is 1 KB and was modified on 6/12/2012 at 3:24:08 PM. Below the table, there is a 'Description:' section containing the file path 'file:///C:/Lisa/examples/Tests/rest-example.tst'.

Name	Size	Modified
URL Attachment.url	1 KB	6/12/2012 3:24:08 PM

Description:
file:///C:/Lisa/examples/Tests/rest-example.tst

HP ALM Attachments tab

Follow these steps:

1. Install HP ALM - Quality Center Connectivity Add-in and register the HP ALM - Quality Center Client.
 - a. From a browser navigate to your ALM or QC 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 Quality Center or 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, or MAR info file.
5. Attach a DevTest test file, staging document, and configuration file to the test plan in Quality Center.
6. Attach a MAR file to the test plan in Quality Center or ALM.

7. Save the VAPI-XP test.

Run DevTest Tests in HP ALM - Quality Center

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 - Quality Center, 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:

- If a coordinator is running, it is used to run the test case.
- If a coordinator is not running, the Test Runner utility is used to run the test case locally.

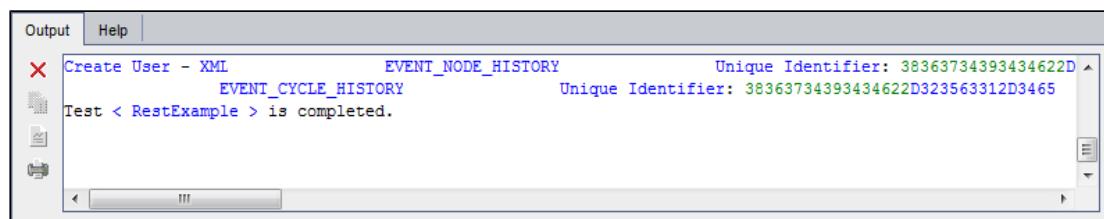
For MAR files and MAR info files:

- If the file specifies a coordinator, the coordinator must be running.
- If the file does not specify a coordinator, the plug-in creates and starts a local coordinator.

By default, the Reload command is set to run with each test run. This command populates the design steps for that test that are necessary for a successful test run. Any changes to the test are updated before the test is run. For some tests, the reload process can take some time. If you know that the underlying DevTest test file has not changed, then you can comment out that line from the script file and the step is skipped.

To run the test from the Test Plan (debug mode), locate the test and click on the **Test Script** tab. From there, you can click the green arrow (Execute Script). The starts executing, and its output shows up in the output window. You are notified when the test completes.

The following graphic shows the results from a run of the **rest-example** test case in the Test Plan.



HP ALM results from a run of the rest-example test case in the Test Plan

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. From there, either 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.

Last Run Report					Steps Details
	Step Name	Status	Exec Date	Exec Time	Description: load the user list as XML
	List Users - XML	✓ PASSED	6/12/2012	4:16:11 PM	
	List Users - JSON	✓ PASSED	6/12/2012	4:16:11 PM	
	Get User - XML	✓ PASSED	6/12/2012	4:16:12 PM	
	Create User - XML	✓ PASSED	6/12/2012	4:16:12 PM	

HP ALM results from a run of the rest-example test case in the Test Lab

QCRunner Command-Line Interface

The **QCRunner** executable in the **LISA_HOME\bin** directory lets you run Quality Center DevTest tests from the command line. You can persist results in the Quality Center database.

This executable has the following format:

```
QCRunner [-h your_HPQC_host] [-P your_HPQC_port] [-u your_HPQC_user] [-p your_HPQC_password] [-D your_HPQC_domain] [-l your_HPQC_project] run|debug|reload name_of_your_test_plan_in_HPQC|all
```

The default host is *localhost*. The default port is *8080*. The default user is *admin*. The default password is *admin*. The default domain is *DEFAULT*. The default project is *Test*.

▪ run

Runs the test name that you specify as an argument. The output appears in the command window. The results are persisted in Quality Center.

▪ debug

Runs the test name that you specify as an argument. The output appears in the command window. The results are not persisted in Quality Center.

▪ reload

Reloads the test name that you specify as an argument, or all DevTest tests (if the argument is all).



Tip: When you use the run parameter, the test must be included in a test lab in Quality Center. If you see the error "unable to load the test" with the QCRunner command, it is because you created a test plan, but did not include the test plan in any test sets,

The following example shows a run of the **rest-example** test case.

```
QCRunner -h machine.example.com -P 8080 -u admin -p mypassword -D DEFAULT -l myproject run RunWithTestandStage

Connecting...
Connected
Running RunWithTestAndStage with the following parameters:
Test Doc: file:///c:/lisa/examples/tests/rest-example.tst
Staging Doc: file:///c:/lisa/examples/stagingdocs/luser1cycle0think.stg
Config:
Mar:
List Users - XML      EVENT_NODE_HISTORY      Unique Identifier: 63366561643365642D64383
4302D3461 ...
List Users - JSON     EVENT_NODE_HISTORY      Unique Identifier: 63366561643365642D6438
34302D3461 ...
Get User - XML        EVENT_NODE_HISTORY      Unique Identifier: 63366561643365642D6438343
02D3461 ...
Create User - XML    EVENT_NODE_HISTORY      Unique Identifier: 63366561643365642D6438
34302D3461 ...
Disconnecting...
Disconnected
```

HP ALM - Quality Center Plug-in Troubleshooting

To improve its performance, the DevTest bridge always keeps a reference to the DevTest COM server. Thus, the server is not instantiated for each API call. When the process hosting the bridge terminates, this reference is released unless the host is a native app like a web browser so the **QCRunner.exe** process stays alive. This behavior is only a problem if something gets in a bad state (for example, because of an abrupt termination). In that case, consider manually terminating the lingering **QCRunner.exe** process before proceeding.

Test Director Test Case Execution Fails

1. Check the URLs in the **Attachment** tab and ensure that the PATH and member name is correct. The Test Director Log file may contain an exception indicating that the member was not found.
2. For more complex problems, start DevTest Workstation, navigate to the test, and execute the test in ITR mode. Watch for exceptions that cause the test case to fail.
3. Ensure that the XML responses and the WSDL have not changed. DevTest test cases rely on XPATH commands to navigate through XML in search of specific values. If the XML has changed, the XPATH commands (filters and assertions) may not be finding their intended targets, resulting in failed test cases.

Test Director Test Case - Permission Denied

If you see an error message indicating, "You do not have the required permissions to execute this action," while trying to execute tests, ensure:

1. That an HPQC administrator has granted your ID permission to execute tests.
2. DevTest Workstation is installed on the computer from which the test is being executed. If DevTest Workstation is installed, verify that the Quality Center Plug-in is also installed.
3. The browser version is compatible with HPQC certified versions.
4. HPQC .DLLs are properly installed in the C:\Program Files\Common\Mercury Interactive\Quality Center folder.
5. Try to execute the test case from both the QC Test Plan and QC Test Lab to ensure that the error results are identical.

[Unhandled Exception executing Test Director tests](#)

If you see an error message indicating, "Unhandled Exception: System.Runtime.InteropServices.COMException," while trying to execute tests, ensure that you are running a 32-bit version of DevTest.

[Common Issues with ALM and DevTest](#)

1. A user running the test may 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 **local.properties** file on the Workstation computer: **lisa.registry.url=tcp://<ipaddress> : 2010/Registry**.
3. Verify that the file attachment format in Test Plan is using the correct format. No spaces are allowed for file path or project or file name.
4. If the test does not run in ALM, verify that the same test runs correctly using the TestRunner Command, then using the QCRunner command-line tool.
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.

[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 \(see page 583\)](#) 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 \(see page 582\)](#) 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
```

Cloud DevTest Labs

You can use a cloud-based infrastructure to provision development and test environments.

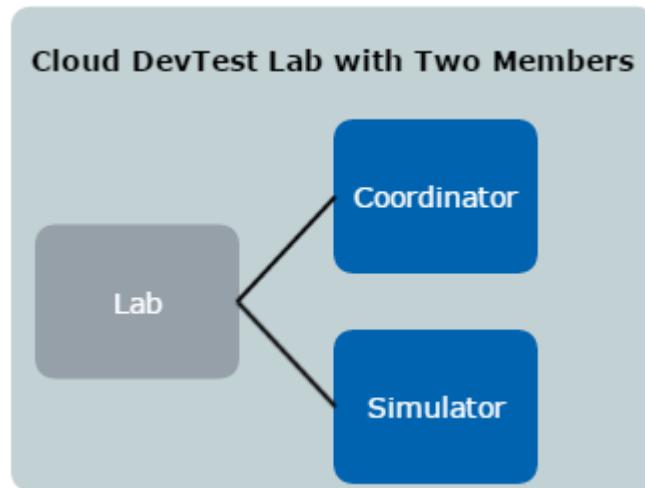
Labs and Lab Members

A *lab* is a logical container for one or more lab members.

A *lab member* can be a DevTest server or a non-DevTest server.

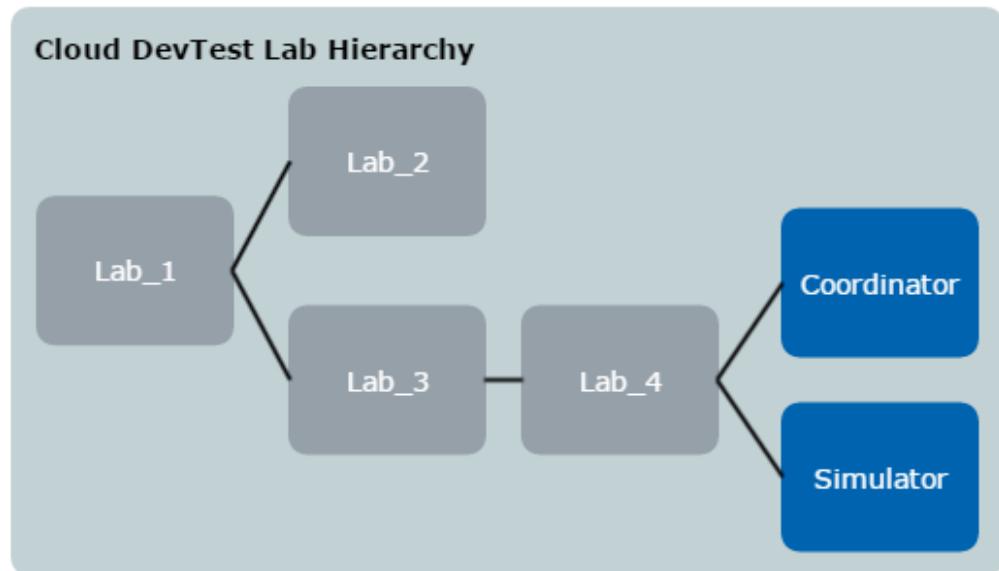
- The valid types of DevTest servers are coordinator, simulator, and Virtual Service Environment.
- Examples of non-DevTest servers include a database and a web server.

The following graphic shows a lab with two members: a coordinator and a simulator.



Lab and lab members diagram

A lab can have one or more child labs. The following graphic shows the hierarchical nature of labs. Lab_1 is the parent of Lab_2 and Lab_3. Lab_3 is the parent of Lab_4.



Lab hierarchy

The fully qualified name of a child lab uses a forward slash as the separator. For example, the fully qualified name of Lab_4 in the previous graphic is Lab_1/Lab_3/Lab_4.

A lab named **Default** is included with DevTest. If you start a coordinator, simulator, or Virtual Service Environment without specifying a lab, the Default lab is used.

A lab member has one of the following statuses:

- Uninitialized
- Starting
- Running
- Unknown

If a lab member is a DevTest server, the status proceeds from Uninitialized to Starting to Running.

If a lab member is a non-DevTest server, the status goes directly from Uninitialized to Running.

Virtual Lab Manager (VLM)

The cloud-based environment where the labs run is known as a Virtual Lab Manager (VLM).

The following VLM provider is supported:

- VMware vCloud Director 1.0 and 1.5

Each VLM provider has a unique prefix. The following table lists the prefixes.

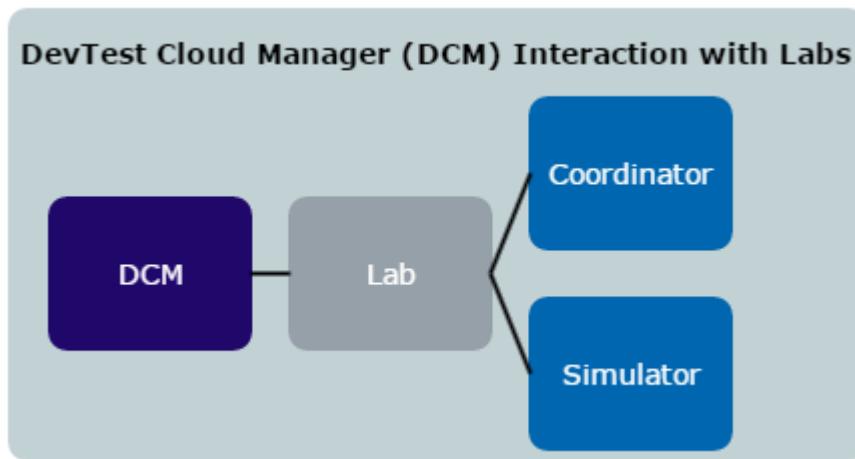
VLM Provider	Prefix
VMware vCloud Director	VCD

In DevTest Workstation, the prefix and a colon appear at the beginning of a fully qualified lab name to indicate which VLM provider is being used. For example:

VCD:MyOrganization/MyCatalog/MyLab

DevTest Cloud Manager

The DevTest Cloud Manager (DCM) is the DevTest Solutions component that interacts with the labs.



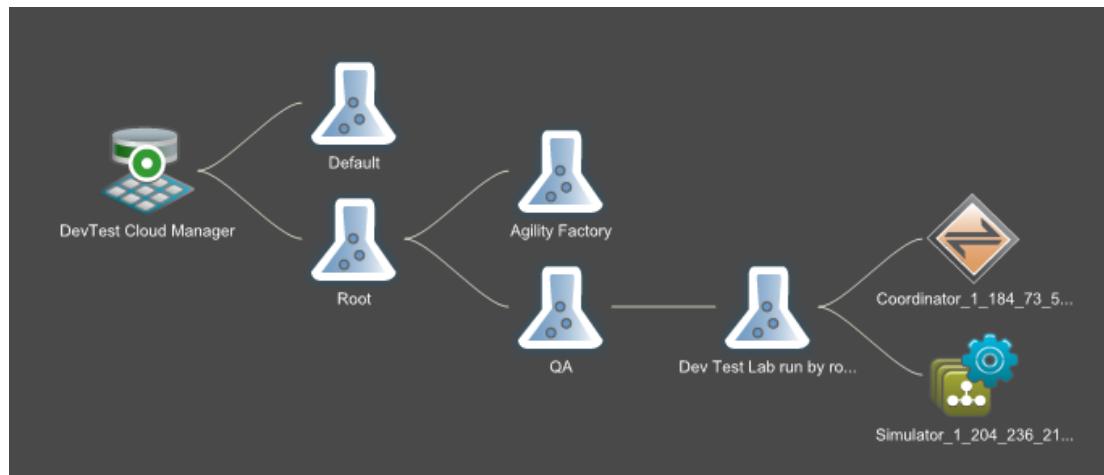
DCM interaction with labs diagram

The responsibilities of the DCM include:

- Sending cloud-related properties to a lab
- Notifying the Virtual Lab Manager (VLM) provider that a lab should be started
- Sending the [Model Archive \(MAR\)](#) (see page 524) to a lab

The DCM lets you view, start, monitor, and shut down labs from the **Server Console**.

The **Server Console** displays the DCM as part of the network graph. In the following graphic, the DCM is interacting with the Default lab and a lab named Root. The Root lab has two child labs: Agility Factory and QA. The QA lab has a child lab named Dev Test Lab. The fully qualified name of Dev Test Lab is Root/QA/Dev Test Lab.



Server Console displaying the DCM and related labs

Configure DCM Properties

To enable the provisioning of development and test labs, configure properties on the computer where the registry is located.

Some properties apply to all Virtual Lab Manager (VLM) providers. Other properties are specific to a VLM provider.

DCM General Properties

This section describes the properties that apply to all VLM providers.

You must configure the following property in the **local.properties** file:

- **lisa.net.bindToAddress**

You must configure the following properties in the **site.properties** file:

- **lisa.dcm.labstartup.min**
- **lisa.dcm.lisastartup.min**
- **lisa.dcm.lisashutdown.min**
- **lisa.dcm.lab.cache.sec**
- **lisa.dcm.lab.factories**
- **lisa.net.timeout.ms**

- **lisa.net.bindToAddress**

The fully qualified domain name or IP address of the computer where the registry is located. The domain name or IP address must be addressable by any computer that is connected to a network outside of the network in which the registry resides.

Syntax:

```
lisa.net.bindToAddress=<fully-qualified-domain-name-or-IP-address>
```

- **lisa.dcm.labstartup.min**

The number of minutes that DevTest waits for the VLM provider to start a lab.

Syntax:

```
lisa.dcm.labstartup.min=<integer>
```

- **lisa.dcm.lisastartup.min**

The number of minutes that DevTest waits after the VLM provider has started a lab for DevTest to be properly initialized.

Syntax:

```
lisa.dcm.lisastartup.min=<integer>
```

- **lisa.dcm.lisashutdown.min**

The number of minutes that DevTest will wait after the test execution has completed to shut down a lab.

Syntax:

```
lisa.dcm.lisashutdown.min=<integer>
```

- **lisa.dcm.lab.cache.sec**

DevTest maintains a cache of the VLM project configuration. This property specifies how often DevTest accesses the VLM provider to see whether the cache must be updated (for example, an environment may have been added). The value is in seconds.

Syntax:

```
lisa.dcm.lab.cache.sec=<integer>
```

Default: 180

- **lisa.dcm.lab.factories**

The name of the object that contains cloud support logic for a specific VLM provider. The only valid value is **com.itko.lisa.cloud.vCloud.vCloudDirectorCloudSupport**.

Syntax:

```
lisa.dcm.lab.factories=<object-name>
```

- **lisa.net.timeout.ms**

The timeout value (in milliseconds) used by the underlying messaging system. This property is not cloud specific. Modify the value for cloud integration because some operations can take longer than the default.

Syntax:

```
lisa.net.timeout.ms=<integer>
```

Default: 30

vCloud Director Properties

This section describes the properties that are specific to VMware vCloud Director.

Configure the following property in the **site.properties** file:

- **lisa.dcm.vCLOUD.baseUri**

Also configure the following properties.

- lisa.dcm.vCLOUD.userId
- lisa.dcm.vCLOUD.password

DevTest creates the **lisa.dcm.vCLOUD.userId** and **lisa.dcm.vCLOUD.password** custom permissions and assigns them to each role. You can specify the initial values for these custom permissions by configuring the **lisa.dcm.vCLOUD.userId** and **lisa.dcm.vCLOUD.password** properties in the **site.properties** file. If you do not specify the initial values, set the values from the Server Console. See [Adding, Updating, and Deleting Roles \(see page 1436\)](#).

- **lisa.dcm.vCLOUD.baseUri**

The login URL of the vCloud API.

Syntax:

```
lisa.dcm.vCLOUD.baseUri=<url>
```

- **lisa.dcm.vCLOUD.userId**

A user name that can log in to vCloud Director. Typically, the format consists of the user name, followed by an ampersand (@), followed by the organization name.

Syntax:

```
lisa.dcm.vCLOUD.userId=<user-name>@<organization-name>
```

- **lisa.dcm.vCLOUD.password**

The password for the user name that is defined in the **lisa.dcm.vCLOUD.userId** property.

Syntax:

```
lisa.dcm.vCLOUD.password=<password>
```

DCM Properties Example

The following example shows a vCloud Director-based configuration.

This property is located in the **local.properties** file:

```
lisa.net.bindToAddress=myserver.example.com
```

These properties are located in the **site.properties** file:

```
lisa.dcm.labstartup.min=6
lisa.dcm.lisastartup.min=4
lisa.dcm.lisashutdown.min=2
lisa.dcm.lab.cache.sec=240
lisa.dcm.lab.factories=com.itko.lisa.cloud.vCloud.vCloudDirectorCloudSupport

lisa.dcm.vCLOUD.baseUri=https://www.example.com/api/versions
lisa.dcm.vCLOUD.userId=administrator@System
lisa.dcm.vCLOUD.password=mypassword

lisa.net.timeout.ms=60000
```

Configure vCloud Director

To enable the provisioning of development and test labs with VMware vCloud Director as the Virtual Lab Manager (VLM) provider, perform configuration steps in vCloud Director.

This page uses the following vCloud Director concepts:

- catalog
- vApp
- vApp template

vApps and vApp templates in vCloud Director correspond to labs in DevTest.

During the configuration procedure, you create virtual machine images for the following DevTest Server components:

- Coordinator
- Simulator
- Virtual Service Environment

If you want the labs to include only a coordinator and simulator, then you do not need to create an image for the VSE.

If you want the labs to include only a Virtual Service Environment, then you do not need to create images for the coordinator and simulator.

Each server component must be configured to start in **remotelInit** mode. This mode causes the server component to start and then wait until the DevTest Cloud Manager (DCM) sends the required initialization settings.

To configure vCloud Director:

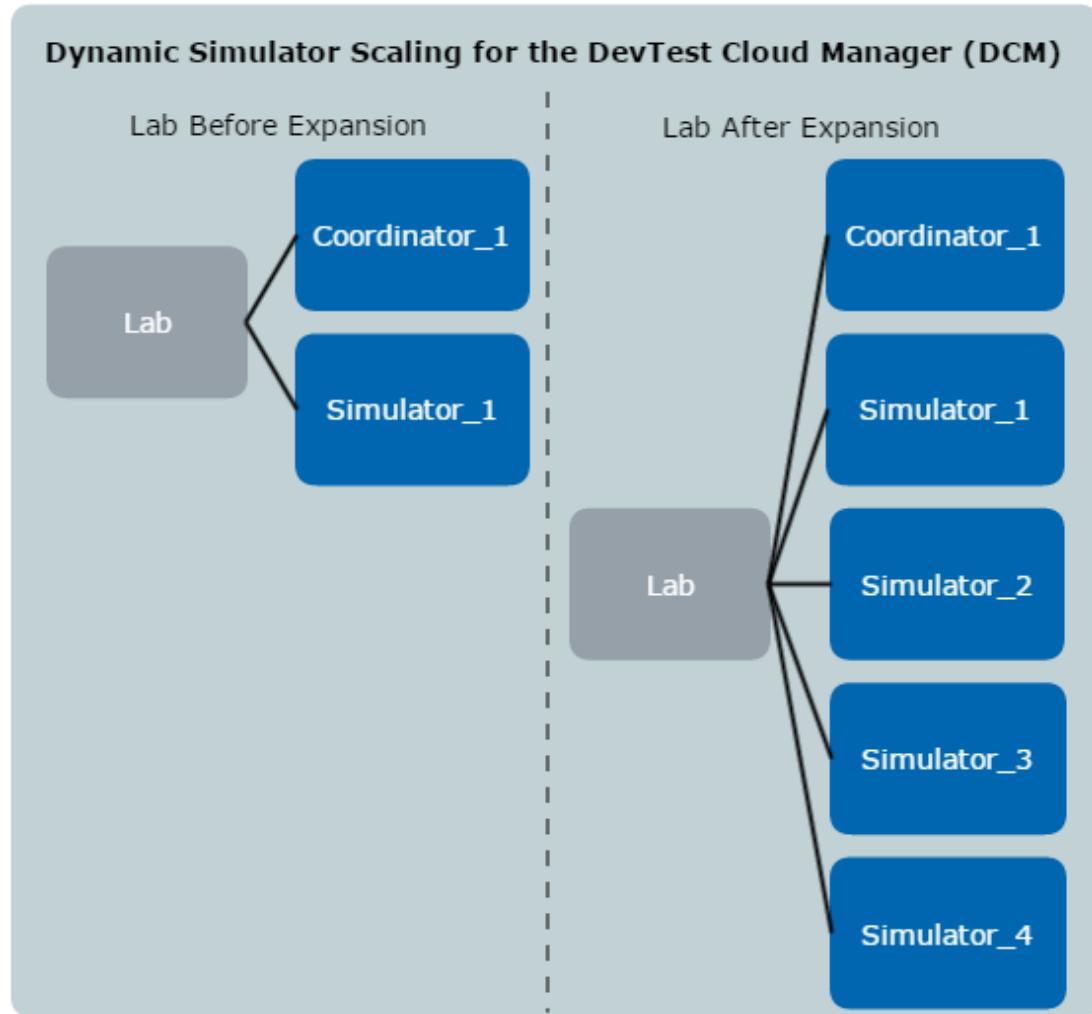
1. Using an application such as VMware Workstation, create a virtual machine image for each server component that you want to include in a lab.
 - To start the server component in **remotelInit** mode, configure each image.
 - The name of a coordinator must include the term **Coordinator**.
 - The name of a simulator must include the term **Simulator**.
 - The name of a Virtual Service Environment must include the term **VSE**.
2. Log in to the vCloud Director web console as an administrator.
3. Import each virtual machine image that you created as a vApp template.
4. Create a vApp from the vApp templates.
5. Add the vApp to a catalog.

Dynamic Expansion of Test Labs

DevTest can determine that more capacity is required to meet the needs of a running test and then automatically expand the lab.

Use a staging document that has the Dynamic Simulator Scaling with the DCM distribution pattern. For more information, see [Distribution Selection \(see page 501\)](#).

The following graphic shows an example. In the left portion, the lab initially has one coordinator and one simulator. In the right portion, the lab has one coordinator and four simulators after the expansion happens.



List the Available Labs

You can use the **Server Console** or LISA Invoke to list the development and test labs that are available in the cloud environment.

Listing the Available Labs from the Server Console

For the instructions about accessing the **Server Console**, see [Open the Server Console. \(see page 1455\)](#)

The following graphic shows a list of available labs in the **Server Console**. The tree structure is expanded to show two child labs.

The screenshot shows the 'Available Labs' section of the DevTest Solutions Server Console. At the top, there are two buttons: 'Start Lab' (with a play icon) and 'Refresh List' (with a circular arrow icon). Below these buttons is a table with a single column labeled 'Name'. The tree structure is as follows:

- Root** (indicated by a minus sign and a folder icon)
 - QA** (indicated by a plus sign and a folder icon)
 - INT-TESTS** (indicated by a minus sign and a folder icon)
 - Lab-2** (indicated by a plus sign and a folder icon)
 - Simulator
 - Coordinator
 - Lab-1** (indicated by a plus sign and a folder icon)
 - Simulator
 - Coordinator
- DEV** (indicated by a plus sign and a folder icon)

Server Console Available Labs

To list the available labs:

1. Display the **Network** panel in the **Server Console**.
2. Expand the **DevTest Cloud Manager** node.
3. Click the **Available Labs** node.
The available labs appear in the right panel. The tree structure is initially collapsed.

Listing the Available Labs by Using LISA Invoke

You can use [LISA Invoke \(see page 572\)](#) to list the available labs. The syntax is as follows:

```
/lisa-invoke/listRunnableLabs
```

The response is an XML document. For each lab, the following information is provided:

- The name of the lab
- The lab key
- The name of each lab member

The lab key is a required parameter in the **startLab**, **getLab**, and **killLab** operations.

Example

The following URL makes a request to list the available labs.

```
http://localhost:1505/lisa-invoke/listRunnableLabs
```

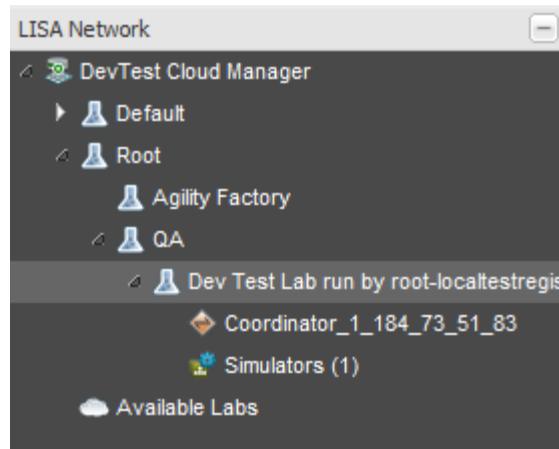
The following response contains a list of three available labs.

```
<?xml version="1.0" encoding="UTF-8" ?>
<invokeResult>
  <method name="ListRunnableLabs">
    <params />
  </method>
  <status>OK</status>
  <result>
    <lab name="MM-Test" key="VCD:7">
      <labMember name="Simulator" />
      <labMember name="Coordinator" />
    </lab>
    <lab name="MM-Test run by rich-47" key="VCD:47">
      <labMember name="Simulator_1_184_72_204_206" />
      <labMember name="Simulator_2_107_21_199_227" />
      <labMember name="Coordinator_1_184_73_83_115" />
    </lab>
    <lab name="VSE-Cluster run by rich-46" key="VCD:46">
      <labMember name="V_S_E_Member_1_23_21_11_148" />
      <labMember name="V_S_E_Member_2_23_21_3_45" />
      <labMember name="V_S_E_Member_4_184_73_65_217" />
      <labMember name="V_S_E_Member_3_174_129_88_179" />
    </lab>
  </result>
</invokeResult>
```

Start a Lab

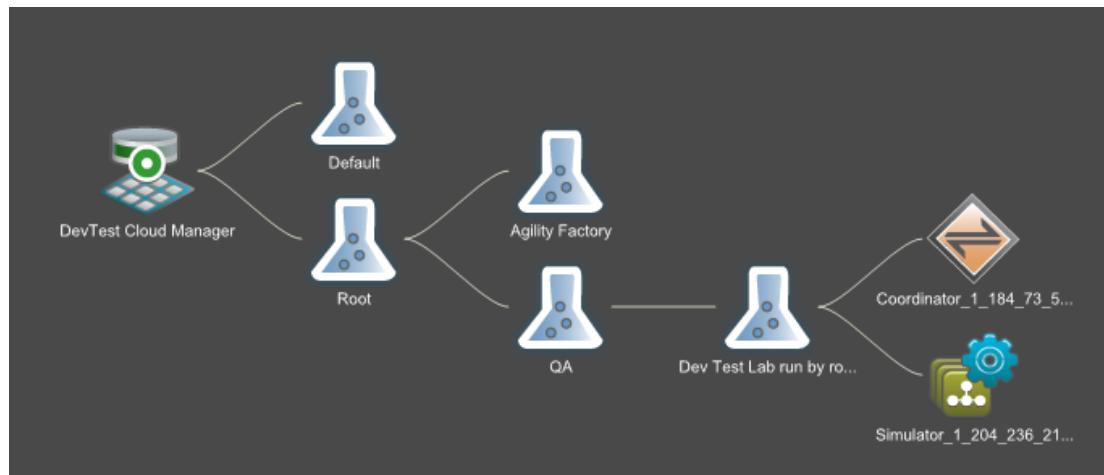
When you start a cloud-based lab, you are starting an instance of the lab. A lab can have multiple instances, all of which are independent of each other.

In the **Server Console**, the **Network** panel displays the started lab in a tree structure. The following graphic shows the **Network** panel.



DevTest Network panel

The right panel displays the started lab in the network graph. The following graphic shows the network graph.



Server Console displaying the DCM and related labs

The **lisa.dcm.labstartup.min** property controls the number of minutes that DevTest waits for the Virtual Lab Manager (VLM) provider to start the lab. For more information, see [Configure DCM Properties \(see page 588\)](#).



Note: You do not need to start the Default lab explicitly.

You can start a lab in the following ways:

- [From the command line \(see page 595\)](#)
- [From the Server Console \(see page 595\)](#)
- [By using LISA Invoke \(see page 596\)](#)

Start a Lab from the Command Line

You can start a lab by invoking one of the DevTest Server executables, and specifying a server name and a lab name.

For example, the following command starts a lab named MyLab, which is a child of MyParentLab. The MyLab lab has one lab member: a coordinator named Dev-Coord.

```
CoordinatorServer -n Dev-Coord -l MyParentLab/MyLab
```

Start a Lab from the Server Console

This procedure assumes that you have [listed the available labs \(see page 592\)](#) from the **Server Console**.

Follow these steps:

1. Select the lab from the list of available labs.

2. Click **Start Lab**.
3. Wait for the lab to start.
When the startup sequence is finished, a message indicates that the lab was started. The **Network** panel displays the started lab in a tree structure. The right panel displays the started lab in the network graph.
4. Wait for the lab members to start.
When the startup sequence for a lab member is finished, the status of the member changes to **Running**. In addition, statistics about the lab member start appearing in the **Component Health Summary tab** (see page 1455).

Start a Lab by Using LISA Invoke

You can use [LISA Invoke \(see page 572\)](#) to start a lab. The syntax is as follows:

```
/lisa-invoke/startLab?labKey=LAB:key
```

The available lab keys are included in the response to the **listRunnableLabs** operation. For more information, see [List the Available Labs \(see page 592\)](#).

The response is an XML document that includes a status message.

Example

The following URL makes a request to start a lab whose key is **VCD:49**.

```
http://localhost:1505/lisa-invoke/startLab?labKey=VCD:49
```

The following response indicates that the lab was started.

```
<?xml version="1.0" encoding="UTF-8" ?>
<invokeResult>
  <method name="StartLab">
    <params />
  </method>
  <status>OK</status>
  <result>
    <lab name="MM-Test run by rich-49" key="VCD:49" />
  </result>
</invokeResult>
```

Obtain Information About a Lab

You can use [LISA Invoke \(see page 572\)](#) to obtain basic information about a running lab. The syntax is as follows:

```
/lisa-invoke/getLab?labKey=LAB:key
```

The available lab keys are included in the response to the **listRunnableLabs** operation. For more information, see [List the Available Labs \(see page 592\)](#).

The response is an XML document. The following information is provided:

- The name of the lab
- The lab key

- The name, service name, and IP address of each lab member.

If the lab is in the middle of starting, then some of the information is not available.

If the lab cannot be found, the response contains an error message.

Example

The following URL makes a request to obtain information about a lab whose key is **VCD:50**.

`http://localhost:1505/lisa-invoke/getLab?labKey=VCD:50`

The following response contains the lab information.

```
<?xml version="1.0" encoding="UTF-8" ?>
<invokeResult>
  <method name="GetLab">
    <params />
  </method>
  <status>OK</status>
  <result>
    <lab name="MM-Test run by rich-50" key="VCD:50">
      <labMember name="Coordinator_1_50_16_15_3" serviceName="tcp://50.16.15.3:2011
/Coordinator_1_50_16_15_3" ip="50.16.15.3" />
      <labMember name="Simulator_2_23_20_80_144" serviceName="tcp://23.20.80.144:2014
/Simulator_2_23_20_80_144" ip="23.20.80.144" />
      <labMember name="Simulator_1_23_21_5_69" serviceName="tcp://23.21.5.69:2014
/Simulator_1_23_21_5_69" ip="23.21.5.69" />
    </lab>
  </result>
</invokeResult>
```

Deploy a MAR to a Lab

You can deploy the Model Archive (MAR) for a test case or suite to a lab that has been [started \(see page 594\)](#). The lab must include a coordinator and (in most cases) a simulator.

Follow these steps:

1. In the Network panel of the **Server Console**, click the coordinator.
The details window for the coordinator appears in the right panel.
2. In the right panel, click **Deploy MAR**.
The **Deploy MAR** dialog opens.
3. Click **Browse** and select the .mar file.
4. Click **Deploy**.
A message indicates that the model archive has been successfully deployed. The test case or suite is now running.
5. Click **OK**.

Stop a Lab

You can use the **Server Console** or LISA Invoke to stop a currently running lab. This action is permanent and cannot be undone.

If the lab is a child lab, the parent labs are not stopped.



Note: You cannot stop the Default lab.

Stop a Lab from the Server Console

In the network graph, right-click the lab and select **Stop**.

When the action is finished, a message indicates that the lab was stopped.

Stop a Lab by Using LISA Invoke:

You can use [LISA Invoke \(see page 572\)](#) to stop a lab. The syntax is:

```
/lisa-invoke/killLab?labKey=LAB:key
```

The available lab keys are included in the response to the **listRunnableLabs** operation. For more information, see [List the Available Labs \(see page 592\)](#).

The response is an XML document that includes a status message.

Example

The following URL makes a request to stop a lab whose key is **VCD:49**.

```
http://localhost:1505/lisa-invoke/killLab?labKey=VCD:49
```

The following response indicates that the lab was stopped.

```
<?xml version="1.0" encoding="UTF-8" ?>
<invokeResult>
  <method name="KillLab">
    <params />
  </method>
  <status>OK</status>
  <result>
    <message>Lab with key VCD:49 has been deleted.</message>
  </result>
</invokeResult>
```

Continuous Validation Service (CVS)

Continuous Validation Service (CVS) lets you schedule tests and test suites to run regularly over an extended time period.

The CVS Dashboard is available in both the DevTest Portal and DevTest Workstation. Documentation for the CVS Dashboard is available [here \(see page 337\)](#).

Reports

You determine what data is collected for reports by specifying reporting parameters in one of three areas:

- [Quick Tests \(see page 543\)](#)

- [Staging Documents \(see page 495\)](#)
- [Test Suites \(see page 516\)](#)

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.

Open the Reporting Portal

You can open the **Reporting Portal** from DevTest Workstation or from a web browser.

To open the Reporting Portal from DevTest Workstation:

Select **View, Reporting Console** from the main menu.

To open the Reporting Portal from a web browser:

1. Ensure that the [registry \(see page 319\)](#) is running.
2. Enter **http://localhost:1505/** in a web browser.
If the registry is on a remote computer, replace **localhost** with the name or IP address of the computer.
The DevTest Console opens.
3. Click **Reporting Dashboard**.



More Information:

- [Report Generator Types \(see page 600\)](#)
- [Reporting Portal Layout \(see page 600\)](#)
- [Filtering Reports \(see page 602\)](#)
- [Viewing Reports \(see page 604\)](#)
- [Exporting Reports \(see page 630\)](#)
- [Changing Reporting Databases \(see page 631\)](#)
- [Troubleshooting Reporting Performance \(see page 631\)](#)

Report Generator Types

You can select the following types of report generators in the **Reports** tab of the [Staging Document Editor](#) (see page 496) or the [Test Suite Editor](#) (see page 517).

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 **Reporting Console** contains 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.

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. To view this report, import the file into the **Reporting Portal**.

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 reporting console. To export the XML data to a file, see [Exporting Reports](#) (see page 630).

Reporting Portal Layout

Contents

- [Reporting Portal - Criteria](#) (see page 601)
- [Reporting Portal - Right Panel](#) (see page 601)

The **Reporting Portal** lets you view all the reports that have run earlier in DevTest Workstation. You can configure the reports either through the staging documents, quick tests, running test cases, or test suites.

This section looks at the reports that are generated by running the multi-tier-combo test case, which is located in the examples directory.



DevTest Console Reporting portal

Reporting Portal - Criteria

In the left panel, you can select the date and time criteria and can select the filters for use.

▪ Start Date/End Date

Select the start/end date by clicking **Calendar** . By default, the start and end dates are in the range of the last hour. Select the start and end time by clicking the **1-12** and **AM/PM** drop-downs. After you enter start and end dates, click **Apply** to apply your changes.

▪ Recent

To reset the date and time criteria automatically to find the last test/suite that was run and the hour before it, click this button.

▪ Filter

You can create filters of your choice here. Enter the name of the filter and click **Save**. You can also delete a filter by clicking **Delete**. To show all filters, not merely ones you have created, select **Show All Filters**.

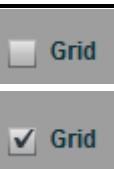
Select from the AND/OR operators for the criteria and click **Add** or **Delete** .

Reporting Portal - Right Panel

The right panel consists of the graphs charted depending on the selected criteria. For more information, see [Reports - Graphical View \(see page 604\)](#).

The **Reporting Toolbar** appears at the top of the right panel and allows you to:

Icon	Function
	Changes the report view from the default chart view to the grid view, and back. For more information, see Reports - Grid View (see page 619) .



 Copies the URL of the report you are viewing to the operating system clipboard so you can share the report with other users.

URL Copy icon



Exports the report data to a PDF file or Excel file to view report details in those formats. For more information, see [Exporting Reports \(see page 630\)](#).



Displays information about the reporting database.

Database

Details icon

You can filter the reports on the results of test cases that have passed or failed. You can also select to show or hide errors and warnings, and show or hide test suites or test cases. For more information, see [Filtering Reports \(see page 602\)](#).

The **Zoom** slider lets you customize the size of the report display.

The **Refresh** button  refreshes the display. To set an auto-refresh, select the **Autorefresh** check box and set an interval for auto-refresh.

Filtering Reports

The right panel of the [Reporting Portal \(see page 600\)](#) displays the reports.

You can filter reports by using specific criteria. After you select the criteria, the report viewer will show graphs only for selected criteria.

▪ **Pass**

Show the test cases/suites that have passed.

▪ **Fail**

Show the test cases/suites that have failed.

▪ **Aborted**

Show the test cases/suites that have aborted.

▪ **Errors**

Show the test cases/suites that have generated errors.

- **Warnings**

Show the test cases/suites that have generated warnings.

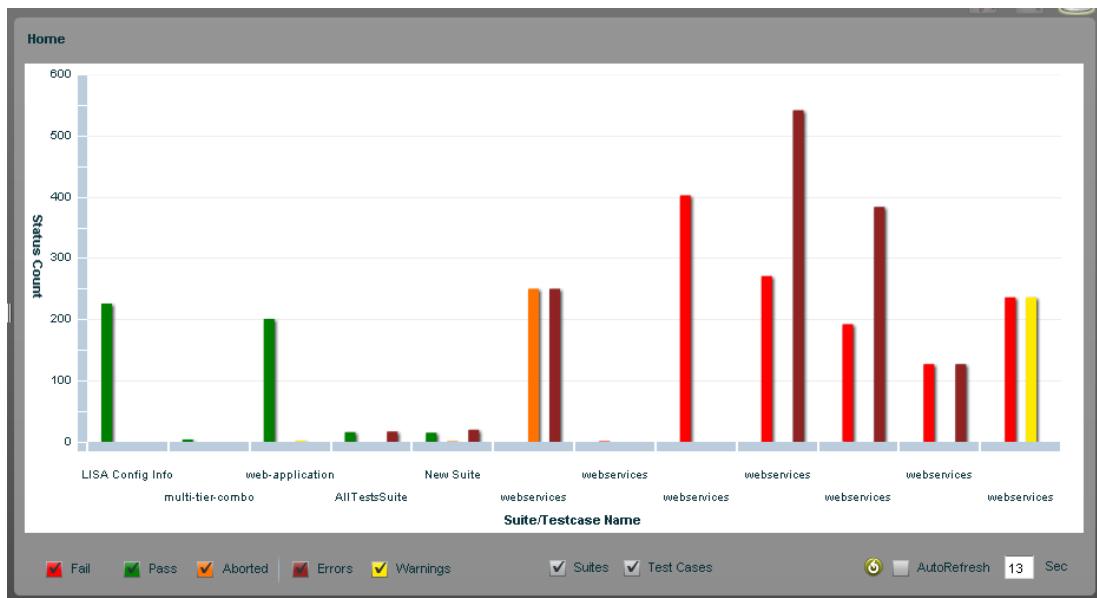
- **Suites**

Show the test suite results.

- **Test Cases:**

Show the test case results.

The **Reporting Portal** shows reports for all the test cases and test suites that have run and that are currently in its database.



Reporting Console

In the previous report, no search criteria are specified, as all filters are selected. Therefore, the report shows all test cases and test suites that have passed, failed, aborted, had errors, and had warnings.



Note: You can combine the **result** criteria with the **test run** criteria for more specificity. For example, you could select **Fail** and **Error** and **Test Case** to see only failed test cases or those that completed with some errors.

To refresh the reports

Click **Refresh** .

To automatically refresh the reports

1. Select the **Auto Refresh** check box.

2. Enter the number of seconds after which you want the reports to be refreshed.
For example, entering **15** refreshes the report every 15 seconds.

Viewing Reports

In the **Reports viewer**, you can view the reports in two formats:

- [Graphical View \(see page 604\)](#): The default view of the **Reporting Portal**. You can see all the reports in the graphical format.
- [Grid View \(see page 619\)](#): You can select the grid view to arrange the data in a grid format.



More Information:

- [Standard Reports \(see page 621\)](#)
- [Interpreting Reports \(see page 629\)](#)

Reports - Graphical View

Contents

- [Reports - Reporting Menu \(see page 607\)](#)

By default, the reports appear as graphs.

The following sample report contains three test cases:

- multi-tier-combo
- main_all_should_fail
- ejb3EJBTest

Because not all of the filter criteria boxes are selected, the report only shows test cases that have passed, failed, or aborted.

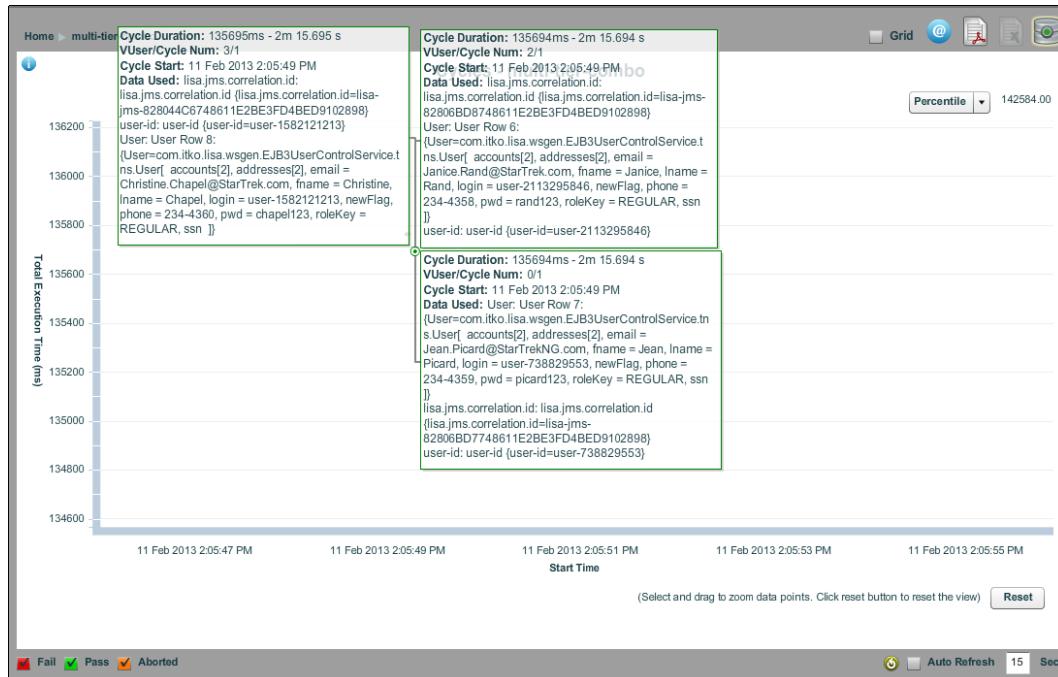
The following features are available in the graphical view:

- Mousing over a test case opens an information box that shows the test case name, the date of execution, and test execution details.



Reporting Console Graph View

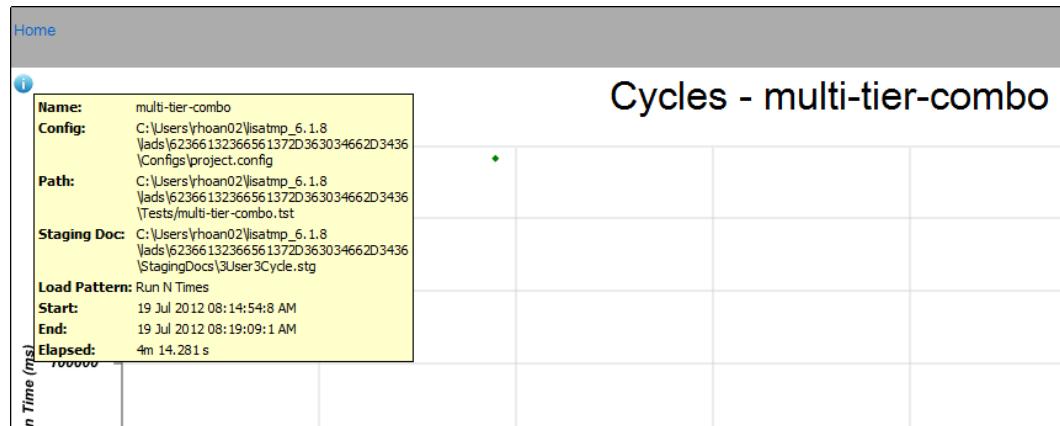
- Double-clicking a test case in the graph that passed produces a Cycles diagram for the test case. When you display the Cycles diagram in graphical view, you can select options from a Percentile drop-down to indicate the percentile value to display in the graphical form (a line across the diagram). The percentile value indicates the maximum number of milliseconds for which a specific percentage of step runs were completed. For example, if the 75th longest running cycle of Step A finished in 7.9 seconds, the 75th percentile would be 7900 or greater.



Reporting Console graph view

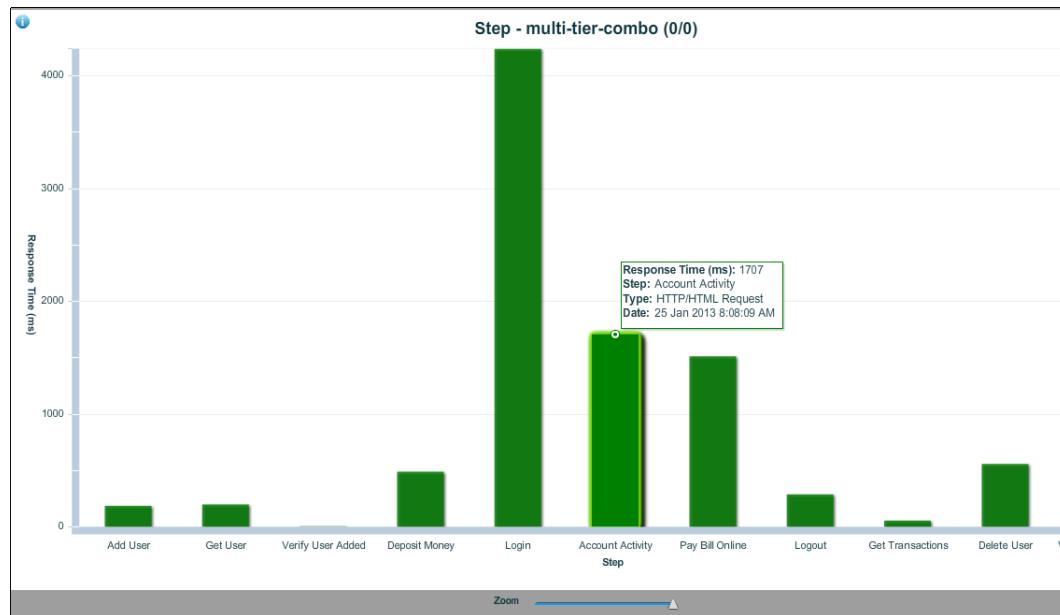
When you move your mouse over these dots, you see that each dot represents a cycle of the test case, and the details for each cycle are shown: response time in milliseconds, the cycle/instance number, the date of the cycle, and the data used. A cycle represents a complete test case from beginning to end.

- Clicking **Information** at the upper left of the window gives you more information about the test case environment.



Reporting Console graph view cycles

- To inspect a subset of cycles more closely, select a group of dots and zoom to see only that subset of cycles.
- To return to the complete cycle view, click **Reset**.
- Double-clicking a test case shows a view of each step of the case, with information about response times.



Reporting Console step report with response times

- Reports that are shown as line graphs can auto scale. 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 that is greater than 100, the Y-axis limit also must grow to accommodate the larger values.

Reports - Reporting Menu

When you right-click a test case or suite, the following options are available. To view examples of the reports, see [Reports - Graphical View Examples \(see page 607\)](#).

Analyze	<ul style="list-style-type: none"> Top Ten Longest Transactions: The ten longest-running transactions in a pie chart Average Transaction Response: Transaction response time in a line chart Metrics: DevTest event metrics in a line chart by time Requests/Second: Number of requests for each second in a line chart by time Performance Summary: Average response time and standard deviation by step in a bar chart Cycle Performance Summary: Cycle time and cycle execution time in milliseconds in a bar chart HTTP Details: HTTP traffic details in a grid format by name HTTP Summary: Cumulative HTTP traffic summary in a line chart
View Error Reports	<ul style="list-style-type: none"> Detailed Failures Detailed Errors Detailed Warnings Detailed Aborts
View Launch Properties	View all launch properties with a time stamp, property name, and property value in grid form.
View History	<p>Shrinks the current graph on to the top half of the window and displays two additional reports:</p> <ul style="list-style-type: none"> A grid listing of test cases run and their results A line chart showing historic execution times. <p>The history report shows information on previous runs of the same test case.</p>
Delete	Deletes the selected test case or suite from the report.
Pin	Keeps the test case or suite from being auto-deleted after 30 days. Any unpinned test older than 30 days are auto-deleted. For more information, see Database Maintenance (see page 1394) .
Import	Imports the test case or suite information from an XML file.
Export	Exports the test case or suite information to an XML file.
Save Image	Saves the graphical image as a .png file. Right-click the step bar for a step menu.

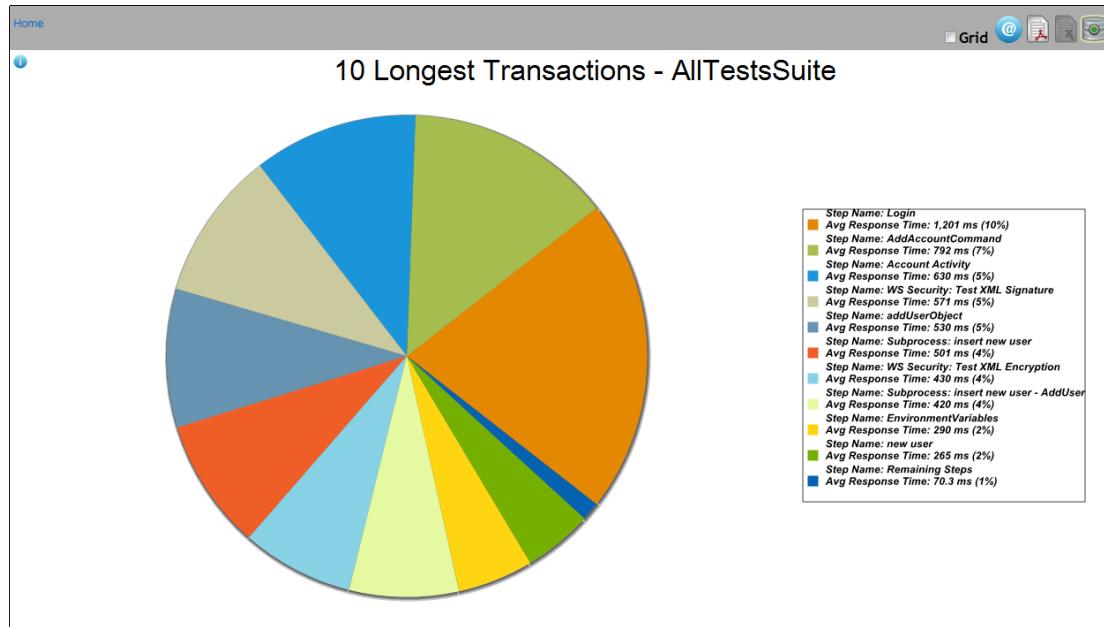
Reports - Graphical View Examples

This section provides examples of the following reporting menu options:

- [Analyze \(see page 608\)](#)
- [View Error Reports \(see page 614\)](#)
- [View History \(see page 618\)](#)

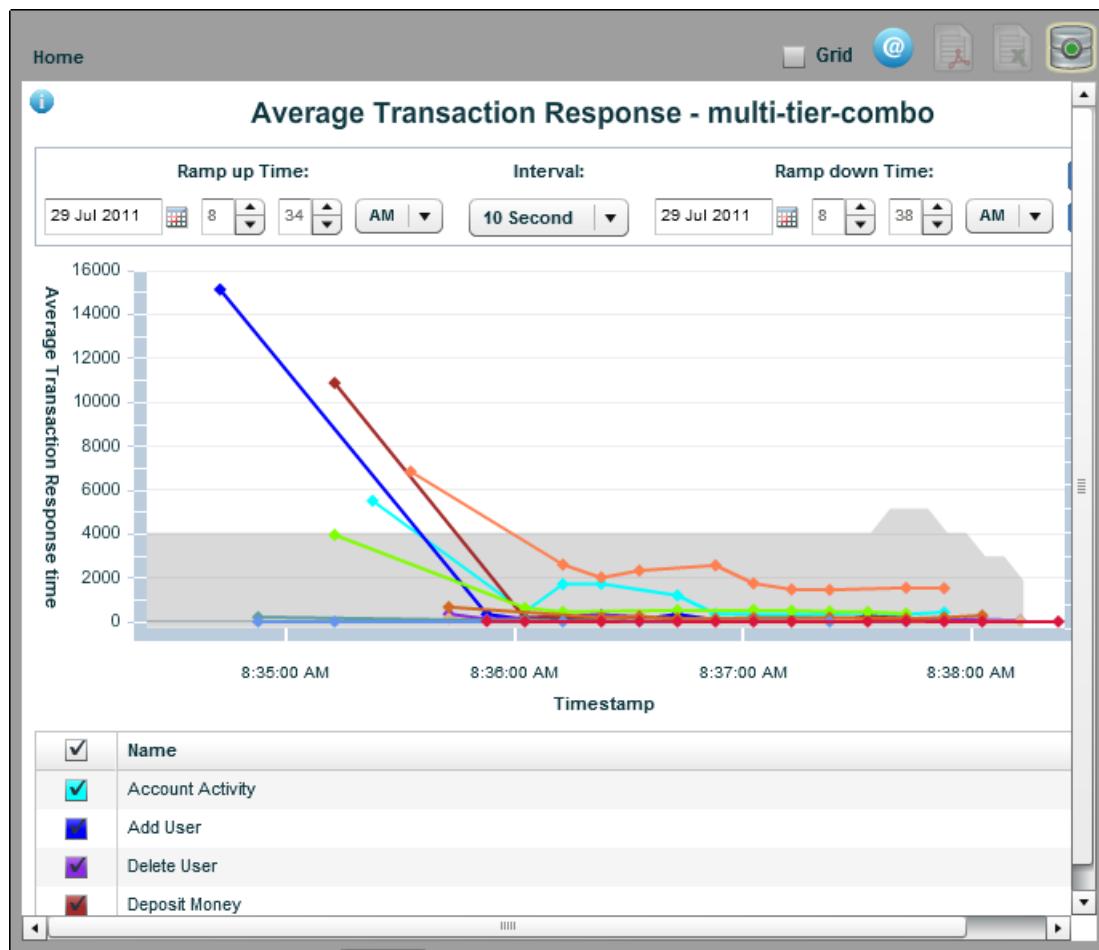
Analyze Report Examples

Top Ten Longest Transactions



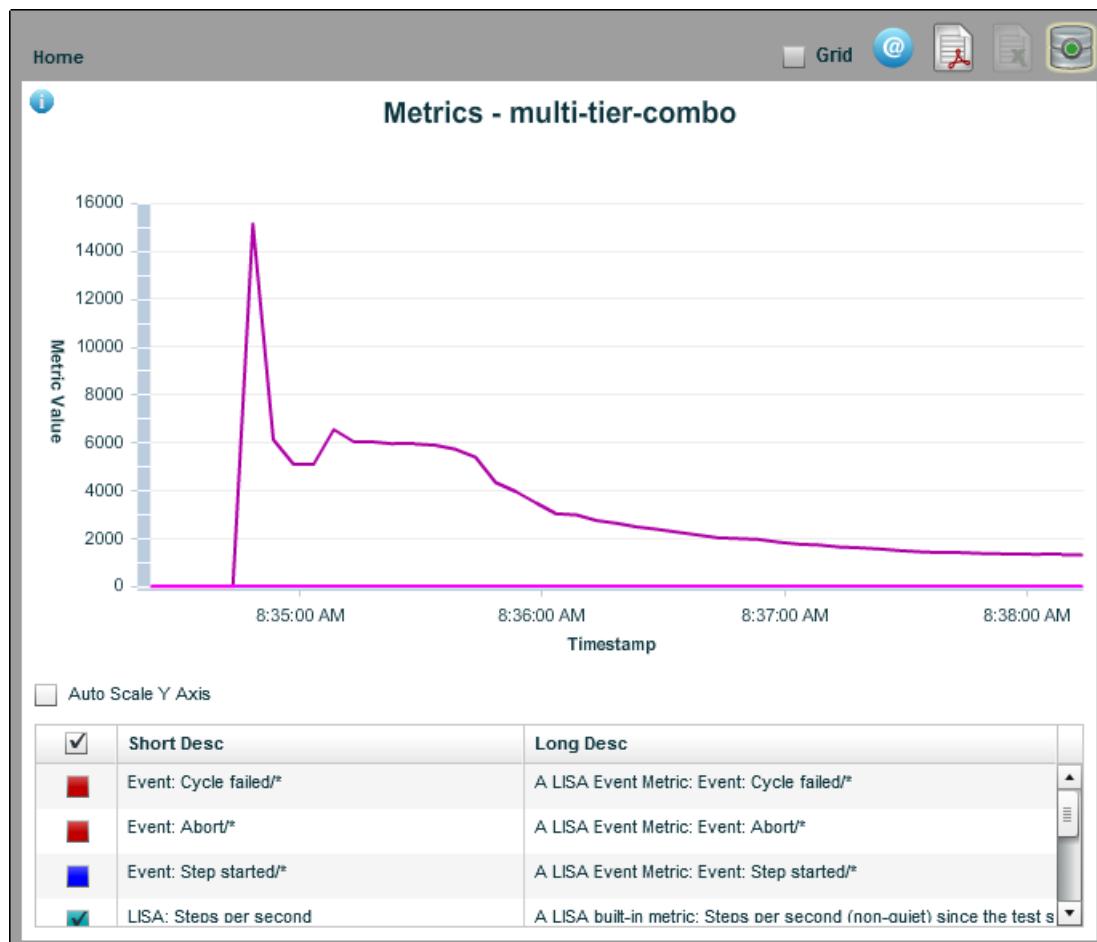
Report: 10 Longest Transactions

Average Transaction Response



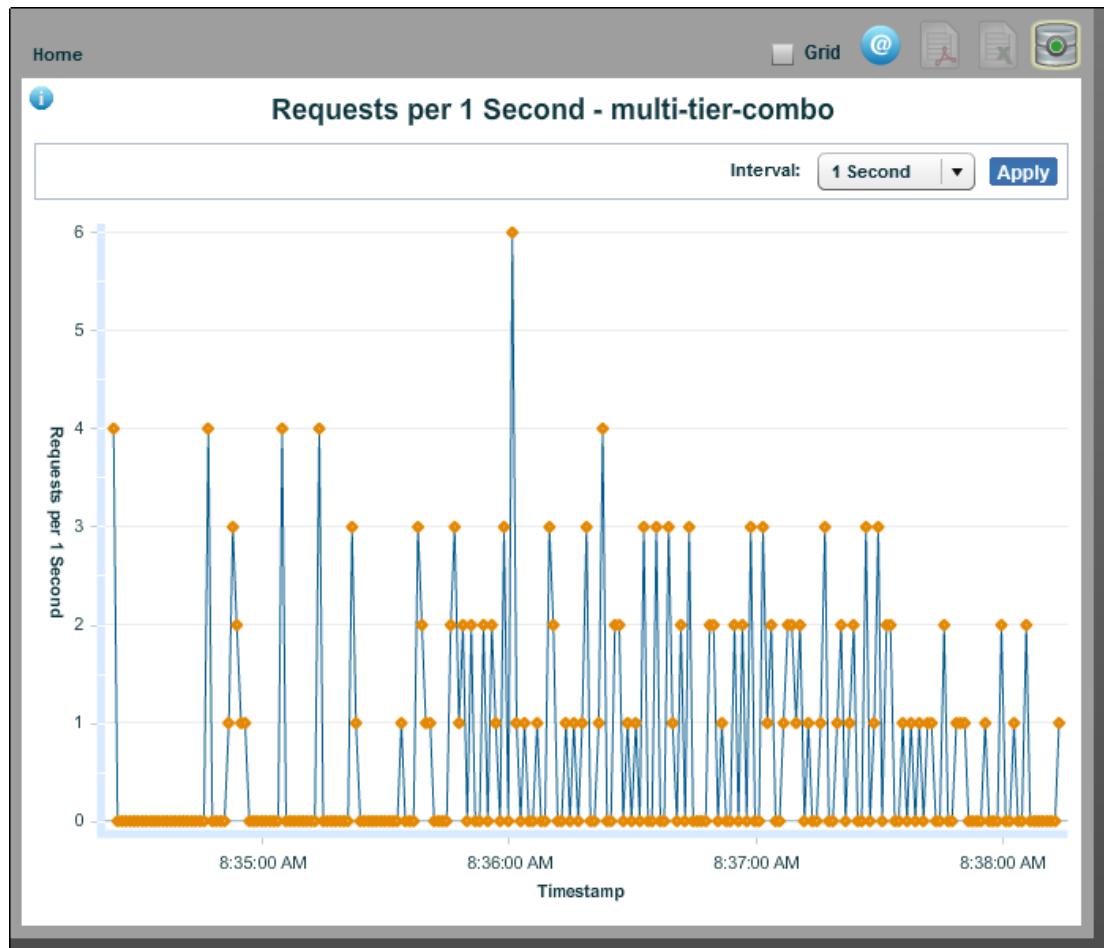
Report: Average Transaction Response

Metrics



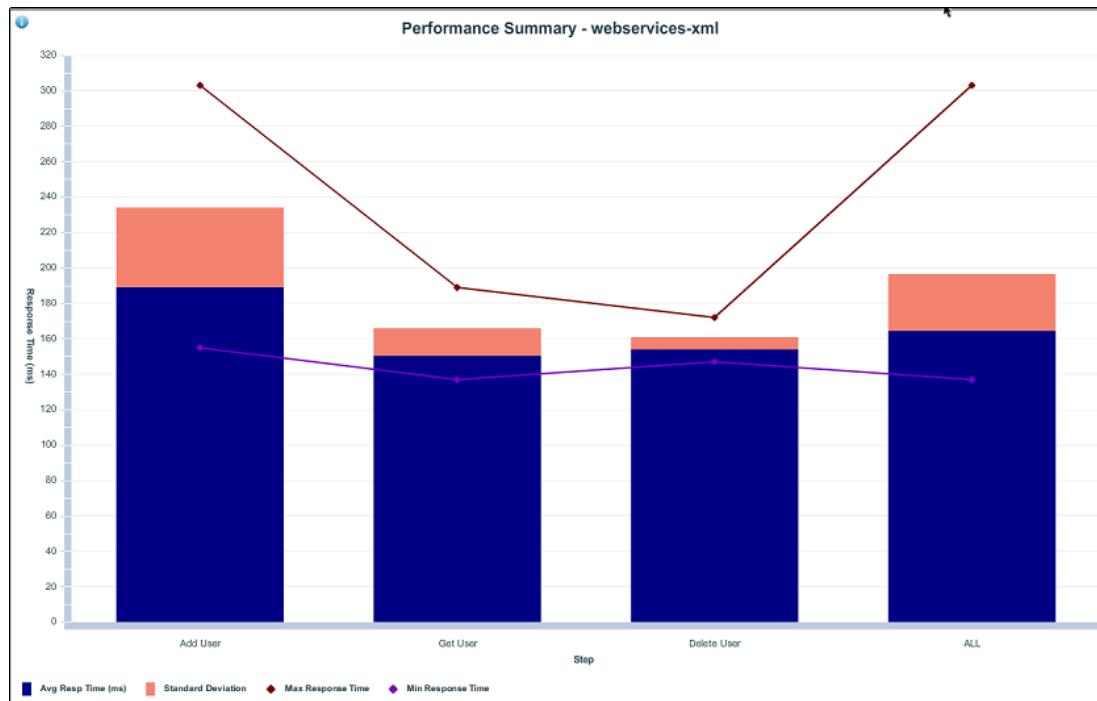
Report: Metrics

Requests/Second



Report: Requests/Second

Performance Summary



Report: Performance Summary

Cycle Performance Summary



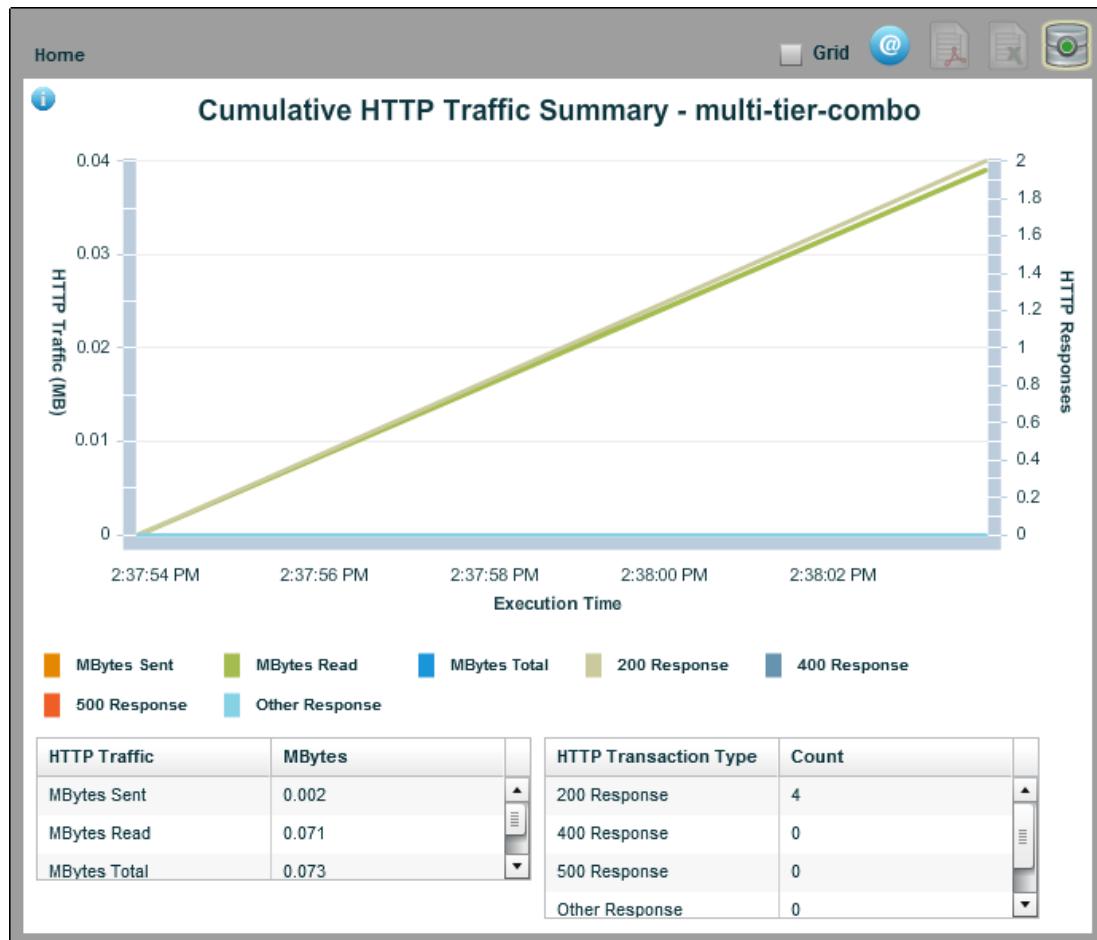
Report: Cycle Performance Summary

HTTP Details

Report: HTTP Details

HTTP Summary

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**.



Report: HTTP Summary

[View Error Report Examples](#)

Error reports report errors and warnings from steps that did not complete error-free. The following graphics are examples of the following error reports:

[Detailed Failures](#)

Home

Grid @

Detailed Failure Report - web-application

Timestamp	Test Case	Cycle	Instance	Step Name	Request	Response	Reason
29 Jul 2011 10:29:05	web-application	0	0	add user	Click for Detail	Click for Detail	add user [assert title]
29 Jul 2011 10:29:07	web-application	0	0	fail	Click for Detail	Click for Detail	
29 Jul 2011 10:29:14	web-application	1	0	add user	Click for Detail	Click for Detail	add user [assert title]
29 Jul 2011 10:29:15	web-application	1	0	fail	Click for Detail	Click for Detail	
29 Jul 2011 10:29:21	web-application	2	0	add user	Click for Detail	Click for Detail	add user [assert title]
29 Jul 2011 10:29:22	web-application	2	0	fail	Click for Detail	Click for Detail	
29 Jul 2011 10:29:29	web-application	3	0	add user	Click for Detail	Click for Detail	add user [assert title]
29 Jul 2011 10:29:30	web-application	3	0	fail	Click for Detail	Click for Detail	
29 Jul 2011 10:29:39	web-application	4	0	fail	Click for Detail	Click for Detail	
29 Jul 2011 10:29:37	web-application	4	0	add user	Click for Detail	Click for Detail	add user [assert title]
29 Jul 2011 10:29:45	web-application	5	0	add user	Click for Detail	Click for Detail	add user [assert title]
29 Jul 2011 10:29:47	web-application	5	0	fail	Click for Detail	Click for Detail	
29 Jul 2011 10:29:55	web-application	6	0	fail	Click for Detail	Click for Detail	
29 Jul 2011 10:29:53	web-application	6	0	add user	Click for Detail	Click for Detail	add user [assert title]
29 Jul 2011 10:30:03	web-application	7	0	fail	Click for Detail	Click for Detail	
29 Jul 2011 10:30:02	web-application	7	0	add user	Click for Detail	Click for Detail	add user [assert title]
29 Jul 2011 10:30:09	web-application	8	0	add user	Click for Detail	Click for Detail	add user [assert title]
29 Jul 2011 10:30:11	web-application	8	0	fail	Click for Detail	Click for Detail	

Details: 112

Report: Detailed Failure

Detailed Errors

Detailed Error Report - web-application							
Timestamp	Test Case	Cycle	Instance	Step Name	Request	Response	Reason
29 Jul 2011 10:29:05	web-application	0	0	add user	Click for Detail	Click for Detail	add user :
29 Jul 2011 10:29:14	web-application	1	0	add user	Click for Detail	Click for Detail	add user :
29 Jul 2011 10:29:21	web-application	2	0	add user	Click for Detail	Click for Detail	add user :
29 Jul 2011 10:29:29	web-application	3	0	add user	Click for Detail	Click for Detail	add user :
29 Jul 2011 10:29:37	web-application	4	0	add user	Click for Detail	Click for Detail	add user :
29 Jul 2011 10:29:45	web-application	5	0	add user	Click for Detail	Click for Detail	add user :
29 Jul 2011 10:29:53	web-application	6	0	add user	Click for Detail	Click for Detail	add user :
29 Jul 2011 10:30:02	web-application	7	0	add user	Click for Detail	Click for Detail	add user :
29 Jul 2011 10:30:09	web-application	8	0	add user	Click for Detail	Click for Detail	add user :
29 Jul 2011 10:30:18	web-application	9	0	add user	Click for Detail	Click for Detail	add user :
29 Jul 2011 10:30:26	web-application	10	0	add user	Click for Detail	Click for Detail	add user :
29 Jul 2011 10:30:35	web-application	11	0	add user	Click for Detail	Click for Detail	add user :
29 Jul 2011 10:30:42	web-application	12	0	add user	Click for Detail	Click for Detail	add user :
29 Jul 2011 10:30:52	web-application	13	0	add user	Click for Detail	Click for Detail	add user :
29 Jul 2011 10:30:59	web-application	14	0	add user	Click for Detail	Click for Detail	add user :
29 Jul 2011 10:31:07	web-application	15	0	add user	Click for Detail	Click for Detail	add user :
29 Jul 2011 10:31:17	web-application	16	0	add user	Click for Detail	Click for Detail	add user :
29 Jul 2011 10:31:23	web-application	17	0	add user	Click for Detail	Click for Detail	add user :

Details: 65

Report: Detailed Errors

Detailed Warnings

The screenshot shows a software interface for DevTest Solutions version 8.4. The title bar reads "DevTest Solutions - 8.4". The main window is titled "Detailed Warning Report - main_all_should_fail". It contains a table with the following columns: Timestamp, Test Case, Cycle, Instance, Step Name, Request, Response, and Reason. A single row is visible, corresponding to the timestamp 01 Aug 2011 7:52, test case main_all_should_f, cycle 0, instance 0, step name Quietly succeed, request Click for Detail, response Click for Detail, and reason Quietly succeed [A]. Below the table, a message says "Details: 1". The top right of the window has icons for Grid, Email, Print, and Save.

Timestamp	Test Case	Cycle	Instance	Step Name	Request	Response	Reason
01 Aug 2011 7:52	main_all_should_f	0	0	Quietly succeed	Click for Detail	Click for Detail	Quietly succeed [A]

Report: Detailed Warnings

Detailed Aborts

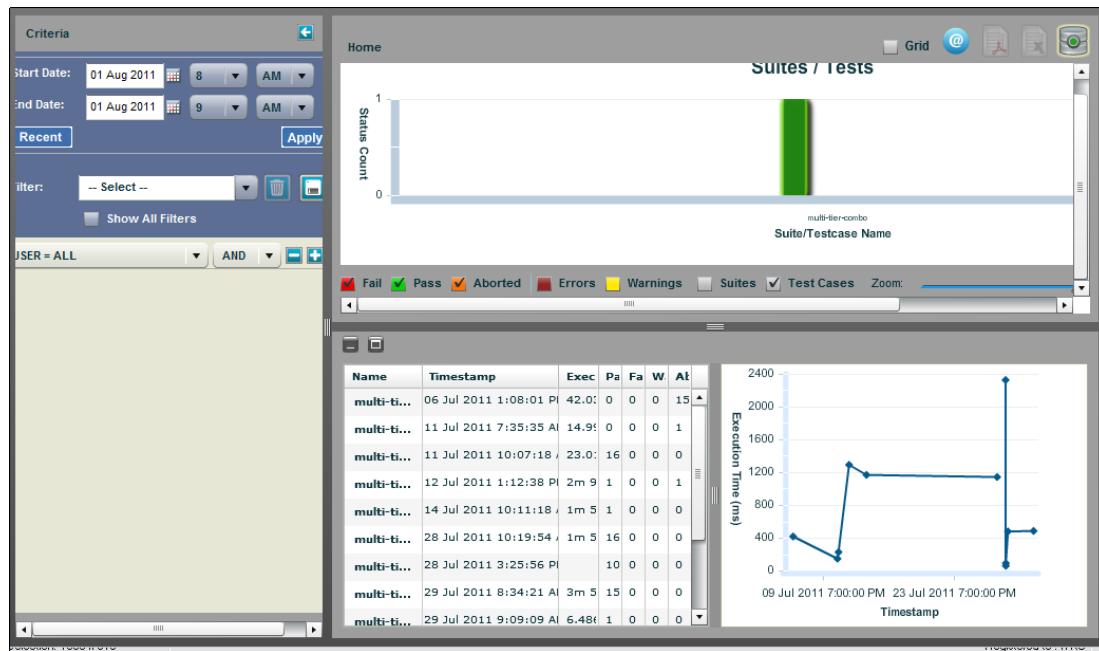
Report: Detailed Aborts

View History Report Example

The View History report shrinks the current graph on to the top half of the window and displays two more reports:

- A grid listing of test cases that were run and their results
 - A line chart showing historic execution times.

This report shows information about previous runs of the same test case.



Report: View History

Reports - Grid View

The grid view displays the same information as the graphical view. The only difference is that it is displayed in a grid format. All report information filters work the same.



Note: You must be in the grid view to export reports to Excel.

To view the results in a grid, select **Grid** at the top of the window.

Suites / Test Cases										
Timestamp	Name	Pass	Pass Percent	Fail	Fail Percent	Aborted	Aborted Percent	Errors	Warnings	Elapsed Time (hr mm)
25 Jan 2013 7:56:12 AM	FastAllTestsSuite	12	63.2	7	36.8	0	0.0	8	0	1m 34.128 s
25 Jan 2013 8:07:51 AM	multi-tier-combo	1	100.0	0	0.0	0	0.0	0	0	51.695 s
25 Jan 2013 8:08:04 AM	webservices	1	100.0	0	0.0	0	0.0	0	0	7.096 s

Suites/Test Cases report, grid view

You can select each of the test cases in the report to find more details for the selected test case.

DevTest Solutions - 8.4

Report: Step report

If you click **Click for Detail** in the **Assert**, **Request**, or **Response** columns, detailed information for each component of the step displays.

Step report, click for detail results

In this view, click to view request and response data. For data that is XML, you can select the **Formatted XML** tab to see formatted text. For non-XML data, the **Formatted** tab shows, "Text is not valid XML."

With the Performance Summary Report in grid view, you can select options from a **Customize** drop-down to indicate the percentile value to display in the last column of the report. The percentile value represents the maximum number of milliseconds for which a specific percentage of step runs were completed. For example, if the 75th longest running cycle of Step A finished in 7.9 seconds, the 75th percentile would be 7900 or greater.

Standard Reports

Four preformatted reports can be generated and exported to PDF.

- [Functional Test Report \(see page 621\)](#)
- [Performance Test Report \(see page 622\)](#)
- [Suite Summary Report \(see page 624\)](#)
- [Metrics Report \(see page 626\)](#)

Important Definitions:

- **Sample size** is based on your staging document. The default is to sample every 1 second and then average every 10 seconds. The sample size can be changed by moving the sliders to change the sample size.
- A **cycle** is a complete run of an entire test case.
- **Avg cycle time** is the average of how long it took to run the test from beginning to end (a cycle).

Functional Test Report

To produce a functional test report:

1. To see the Steps report, double-click a test.



2. Select a step, then select **Export to PDF**.
The **Select a Report** window opens.
3. Select Functional Test Report from the list of available reports for the test and click **OK**.
The **Customize** drop-down lets you produce a detailed summary of this report.

[Customize](#) 

Functional Test Report

rhoan02

Name:	REST Example	Start Time:	18 Jan 2013 8:04:31 AM
C:/Lisa/examples/Tests/REST Example.tst			
C:/Users/rhoan02/lisatmp_7.0/lads/F562526C617711E29649D4BED9102898/Tests/REST Example.tst			
Config:	project.config	End Time:	18 Jan 2013 8:04:37 AM
C:/Lisa/examples/Configs/project.config			
C:/Users/rhoan02/lisatmp_7.0/lads/F562526C617711E29649D4BED9102898/Configs/project.config			
Staging:	QuickStageRun	Duration:	6.090 s
C:/Lisa/examples/StagingDocs/_quick_stage_document_stg			
C:/Users/rhoan02/lisatmp_7.0/lads/F562526C617711E29649D4BED9102898/StagingDocs/_quick_stage_document_stg			

Plan Duration:	N/A	Plan VUsers:	1.00
Load Pattern:	Run N Times	Distribution:	Percent Distribution
Think Time:	100%	Test Pacing:	N/A
Cycles:	1.00	VUsers:	1.00
Avg Cycle (ms):	6,090.0		

Tests Summary	
Cycles Attempted	1
Cycles Started	1
Cycles Passed	1
Pass Percent	100%
Cycles Failed	0
Fail Percent	0%

Report: Functional Test Report

Performance Test Report

To produce a performance test report:

1. Right-click on a test step.

2. Click **Export to PDF**.
The **Select a Report** window opens.

3. Select **Performance Report** from the list of available reports and click **OK**.

Customize  

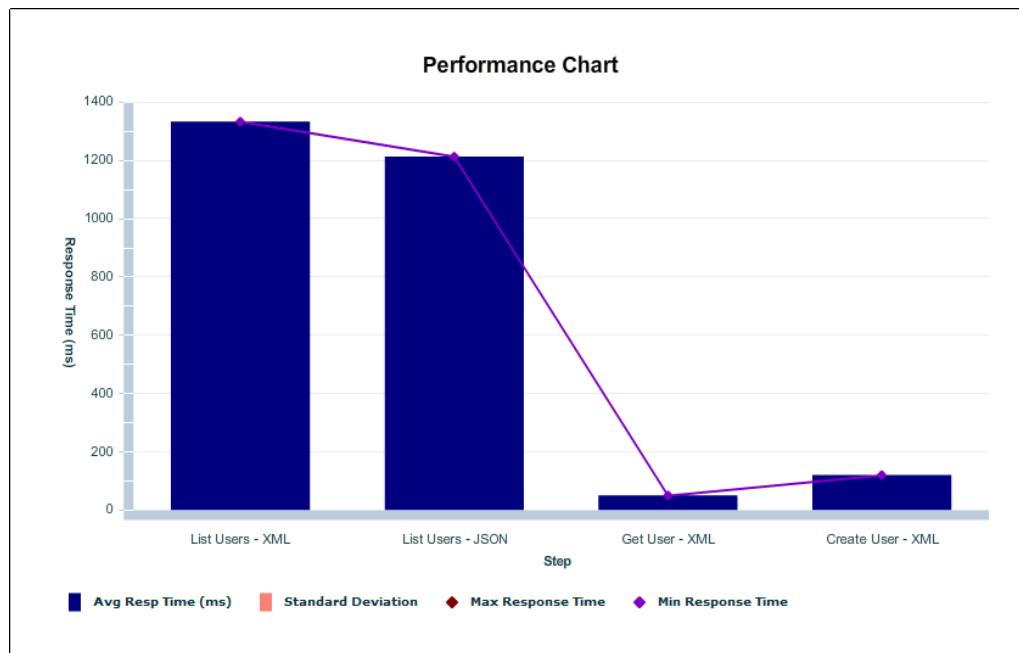
Performance Test Report

rhoan02

Name:	REST Example	Start Time:	18 Jan 2013 8:04:31 AM
C:/Lisa/examples/Tests/REST Example.tst			
C:/Users/rhoan02/lisatmp_7.0/lads/F562526C617711E29649D4BED9102898/Tests/REST Example.tst			
Config:	project.config	End Time:	18 Jan 2013 8:04:37 AM
C:/Lisa/examples/Configs/project.config			
C:/Users/rhoan02/lisatmp_7.0/lads/F562526C617711E29649D4BED9102898/Configs/project.config			
Staging:	QuickStageRun	Duration:	6.090 s
C:/Lisa/examples/StagingDocs/_quick_stage_document_stg			
C:/Users/rhoan02/lisatmp_7.0/lads/F562526C617711E29649D4BED9102898/StagingDocs/_quick_stage_document_stg			

Plan Duration:	N/A	Plan VUsers:	1.00
Load Pattern:	Run N Times	Distribution:	Percent Distribution
Think Time:	100%	Test Pacing:	N/A
Cycles:	1.00	VUsers:	1.00
Avg Cycle (ms):	6,090.0		

Report: Performance Test Report



Report - Performance Test Report chart

Step	Total Time (ms)	Sample Size	Min (ms)	Max (ms)	Avg (ms)	Median (ms)	Std Dev	90th percentile
List Users - XML	1,332	1	1,332	1,332	1,332.00	1,332.00	0.0	1,332.0
List Users - JSON	1,212	1	1,212	1,212	1,212.00	1,212.00	0.0	1,212.0
Get User - XML	50	1	50	50	50.00	50.00	0.0	50.0
Create User - XML	120	1	120	120	120.00	120.00	0.0	120.0

Report - Performance Test Report chart - second page

Suite Summary Report

To produce a suite summary report:

1. Open a test suite report.



2. Click **Export to PDF**.

3. The **Select a Report** window opens.

4. Select **Suite Summary Report** from the list of available reports and click **OK**.

The **Customize** drop-down lets you see details about the types of failed tests. If all tests passed, the Details view is the same as the Summary view.

Customize  

Suite Summary Report

rhoan02@RHOAN02

Name: FastAllTestsSuite **Start Time:** 18 Jan 2013 8:14:26 AM
C:/Lisa/examples/Suites/FastAllTestsSuite.ste

Config: project.config **End Time:** 18 Jan 2013 8:14:55 AM
C:/Lisa/examples/Configs/project.config

C:/Users/rhoan02/lisatmp_7.0/lads/5CDAB3FC617911E29649D4BED9102898/Suites/FastAllTestsSuite.ste
C:/Users/rhoan02/lisatmp_7.0/lads/5CDAB3FC617911E29649D4BED9102898/Configs/project.config

Duration: 29.064 s

Startup Test: N/A
Teardown Test: N/A
Run Mode: Parallel
Message: N/A

Tests Summary	Count
Tests Passed	18
Tests Failed	1
Tests Aborted	0
Total Tests	19



Cycles Summary

Cycles Summary	Count
Cycles Passed	18
Cycles Failed	1
Cycles Aborted	0
Total Cycles	19



Legend:

- Pass (Green)
- Fail (Red)
- Aborted (Orange)

Report: Suite Summary Report

Passed Tests						
Name	Cycles	Pass	Fail	Aborted	Warnings	Errors
AccountControlMD	1	1	0	0	0	0
async-consumer-j	1	1	0	0	0	0
main_all_should_t	1	1	0	0	0	0
webservices-xml	1	1	0	0	0	0
ejb3EJBTest	1	1	0	0	0	0
ip-spoofing	1	1	0	0	0	0
ejb3WSTest	1	1	0	0	0	0
JMS	1	1	0	0	0	0
LISA Config Info	1	1	0	0	0	0
load data from a	1	1	0	0	0	0
log-watcher	1	1	0	0	0	0
multi-tier-combo-	1	1	0	0	0	0
service-validation	1	1	0	0	0	0
REST Example	1	1	0	0	0	0
multi-tier-combo	1	1	0	0	0	0
web-application	1	1	0	0	0	0
ws_attachments	1	1	0	0	0	0
ws_security-xml	1	1	0	0	0	2

Failed Tests						
Name	Cycles	Pass	Fail	Aborted	Warnings	Errors
main_all_should_t	1	0	1	0	0	0

Tests With Errors						
Name	Cycles	Pass	Fail	Aborted	Warnings	Errors
ws_security-xml	1	1	0	0	0	2

Report: Suite Summary Report page 2

Error Messages						

Report: Suite Summary Report page 3

Metrics Report

To produce a Metrics Report:

1. From a test/suite report, right-click on a test or suite.
2. Select **Analyze, Metrics** to display a metrics chart or grid.



3. From that report, click **Export to PDF**.
The **Select a Report** window opens.

4. Select **Metrics Report** from the list of available reports and click **OK**.

The screenshot shows the 'Metrics Report' page for a test named 'rhohan02'. At the top right are 'Customize' and 'Print' buttons. The report title is 'Metrics Report' followed by the test name 'rhohan02'. Below the title, the report summary includes:

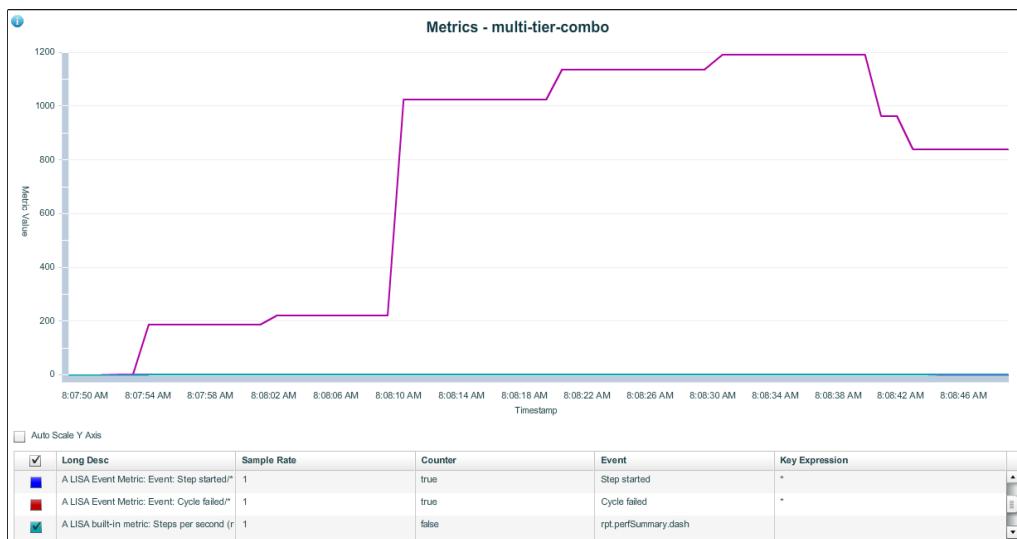
- Name:** REST Example **Start Time:** 18 Jan 2013 8:04:31 AM
- C:/Lisa/examples/Tests/REST Example.tst**
- Config:** project.config **End Time:** 18 Jan 2013 8:04:37 AM
- C:/Lisa/examples/Configs/project.config**
- Staging:** QuickStageRun **Duration:** 6.090 s
- C:/Lisa/examples/StagingDocs/_quick_stage_document_stg**
- C:/Users/rhoan02/lisatmp_7.0/lads/F562526C617711E29649D4BED9102898/StagingDocs/_quick_stage_document_stg**

Performance parameters:

Plan Duration:	N/A	Plan VUsers:	1.00
Load Pattern:	Run N Times	Distribution:	Percent Distribution
Think Time:	100%	Test Pacing:	N/A
Cycles:	1.00	VUsers:	1.00
Avg Cycle (ms):	6,090.0		

Report: Metrics report

When you right-click to see the metrics chart on the window, the legend is interactive and you can select events and the auto-scale parameter. In the PDF report, use the **Customize** drop-down to make selections.



Report: Metrics report graph view

The **Customize** button allows you to select the following options:

- Detailed Summary
- Auto-Scale Y-Axis

- All Metrics
- No Metrics
- Metrics (a selection)

Examples of each customized report follow.

Detailed Summary

Metrics - multi-tier-combo				
Short Desc	Long Desc	Timestamp	Value	Scaled Value
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:07:49 AM	0.00	0.00
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:07:50 AM	0.00	0.00
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:07:51 AM	0.00	0.00
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:07:52 AM	0.00	0.00
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:07:53 AM	0.00	0.00
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:07:54 AM	0.71	0.71
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:07:55 AM	0.71	0.71
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:07:56 AM	0.71	0.71
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:07:57 AM	0.71	0.71
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:07:58 AM	0.71	0.71
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:07:59 AM	0.71	0.71
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:08:00 AM	0.71	0.71
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:08:01 AM	0.71	0.71
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:08:02 AM	0.39	0.39
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:08:03 AM	0.39	0.39
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:08:04 AM	0.39	0.39
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:08:05 AM	0.39	0.39
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:08:06 AM	0.39	0.39
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:08:07 AM	0.39	0.39
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:08:08 AM	0.39	0.39
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:08:09 AM	0.39	0.39
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:08:10 AM	0.27	0.27
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:08:11 AM	0.27	0.27
Steps per second (non-quiet) since the test started	A LISA built-in metric: Steps per second (non-quiet) since the test started	25 Jan 2013 8:08:12 AM	0.27	0.27

Report: Metrics report detailed summary

Auto-Scale Y Axis

Compare this report to the same report shown previously to see the auto-scaled Y axis.



Report: Metrics report autoscaled y axis

All Metrics

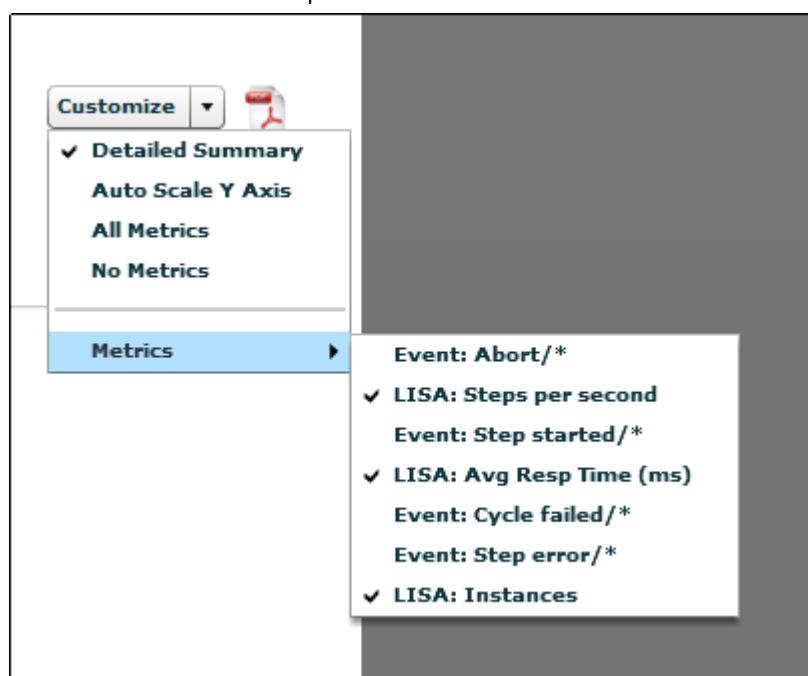
All Metrics is the default selection for the metrics reports. If you have selected **No Metrics** or if you have selected **Metrics** in the **Customize** drop-down, you can select **All Metrics** to display all metrics available for the report.

No Metrics

When you select **No Metrics** from the **Customize** drop-down, all metrics are cleared from the Metrics Chart.

Metrics

You can select which metrics appear on the Metrics Chart by selecting individual metrics that are available in the **Metrics** drop-down.



Reports - Metrics drop-down menu for options

Interpreting Reports

Sometimes, the results of reports are not what you expect, and it could appear that the reporting data is incorrect.

Looping Tests

Loops in test cases can cause unexpected results in the reporting engine.

The workflow engine is designed to flow from the beginning step to the end step, with no loops. One execution of this chain of steps is considered to be one pass or fail or abort event. Looping is designed to be initiated externally to the test case using a staging document. By using a staging document to specify 10 virtual users executing a test case once, you have 10 pass/fail/abort events. When the looping is done in the test case, a pass/fail/abort does not occur until the end step is reached, therefore creating only a single pass/fail/abort event.

Think Time and Reports

Think time is how long DevTest waits to start the execution of a test step. The purpose of think time is to simulate a real user interacting with any system. You can set think time in the step editor or in a staging document.

In the main reporting panel, we display total execution time, which is the length of time from start to finish of the test. Because this is a duration, we include think times by design, to show how long it took for the entire test to run. To exclude think times, set your think time amount to 0 in your staging document or on your test step.

If you are looking for performance numbers, generate a performance report that gives you response times for each step that does not include think time, which is a more meaningful report for performance tests. Think time is not part of the average step performance time.

Exporting Reports

In the Reports viewer, the reports can be exported in three formats:

- XML
- Microsoft Office Excel
- Adobe Acrobat PDF

Export Reports to XML

If your test suite or staging document specified the XML Report, you can export the report data directly to an XML file. To export, right-click on the test or suite and select **Export**. For more information, see [Reports - Graphical View \(see page 604\)](#). After saving your exported data, you can format or manipulate it.

Exporting your report data to XML lets you move reporting data from one registry to another.

Export Reports to Excel

You can export all reports data to an Excel spreadsheet.

Follow these steps:

1. Open the report to export.
2. Select the **Grid** check box.
The report must be in grid view before you can export to Excel.
3. Click **Export to Excel**.
4. Specify the name of the Excel file.
The reporting data is stored in the saved Excel file.

Export Reports to PDF

You can also export the report data to PDF.

Follow these steps:

1. Open the graphical view of the report.
2. Click **Export to PDF**.
The **Select a Report** window opens.
3. Select the report that you want to export.

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.

Troubleshooting Reporting Performance

If you run load tests and 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.

Typically the issue is that DevTest trying to record too much data when doing a load test. On the **Reports** tab of the **Test Suite** editor, try turning off events in this order:

▪ Properties Set / Referenced

This event generates the most number of rows in the database and is almost never useful for a load test. Keep in mind that there can easily be 10 or more property gets and sets for every step executed. In a load test, this adds up quickly to millions of rows in the database. For example, if you have a test case that is 5 steps (with 5 gets and 5 sets for each step) and you run 100 users for 10000 cycles, then you will have 50 million rows in the database.

- **Record All Events**

This event is essentially a recording of the workflow engine. This can also easily generate 5 or more rows for every step.

- **Request / Response**

While this event does not create any extra rows, the payloads are stored as clob because they may be large. This probably creates the largest amount of data but is easier on the database because the tables and indexes do not have to grow as rapidly.

If you still see the reporting engine as the bottleneck, consider enabling the `lisa.reporting.useAsync =false` property. This property tells DevTest to use JMS to send the reporting event. Background threads in the simulators and coordinator write the events to the database asynchronously. This means that your load test will finish before all the events are written to the database, so the report will not appear for some time. Just how long will depend on your test cases and how many events they generate. The simulator queue typically takes the longest to flush; you will get a message at INFO level in the simulator log showing the percentage complete.

The async reporting feature allows the simulator to run faster because it does not slow down writing to the database. It instead puts the data into a JMS queue to be written later.

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 \(see page 632\)](#)
- [Record a Website \(see page 633\)](#)
- [Record a Mobile Test Case \(see page 636\)](#)

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 \(see page 259\)](#). For more information about the Web Service Execution step, see [Web Service Execution \(XML\) Step \(see page 1711\)](#).

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 \(see page 633\)](#)
- [Create the Web Service Client \(see page 633\)](#)
- [Execute the Test Case \(see page 633\)](#)

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 \(see page 635\)](#)
- [Start Recording \(see page 636\)](#)
- [View Recorded Transactions \(see page 636\)](#)
- [View in ITR \(see page 636\)](#)

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

Follow these steps:

1. To start recording the test, 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.

Record a Mobile Test Case

Follow these steps:

1. Verify that the config file that defines the desired mobile asset is active.

2. [Create a test case \(see page 482\).](#)



3. Click **Create steps by recording or templating**.
4. Click **Record Test Case for User Interface, Mobile Recorder**.
The Test Recorder window opens.



Note: If you are using a mobile simulator to record, the mobile simulator window also opens.

5. If you defined multiple mobile assets, select an asset which to connect from the **Choose Mobile asset** drop-down list, then click **OK**.
6. Click **Start Recording** at the bottom of the recorder window.
7. In the Test Recorder window, perform the actions that you want to record for your test case.



Important! Do not perform these actions in the mobile simulator directly. DevTest Workstation sends the actions that you perform in the Test Recorder to the simulator automatically.

The Test Recorder highlights each screen element that you interact with. A red outline indicates a screen position. A green outline indicates the more specific screen element. When you point to a specific screen element, a tooltip window identifies it. When a button or component has multiple possible behaviors, the tooltip displays a hint for your clicking point.

8. To add additional actions or gestures manually while you record, right-click the screen or the specific element in the Test Recorder, then select the action to insert.
For example, you can manually insert a "Shake," change the orientation, or insert a specific assertion during the recording process. For more information about the available actions, see [Modify Mobile Test Steps \(see page 1868\)](#).



Note: One step in a mobile test case contains as many gestures (tapping, swiping) as necessary to complete the step. These substeps, or actions, appear in the Actions table when you double-click a test step.

9. Click **Stop Recording** when you have captured all of the actions for the test.
The recorder window and the mobile simulator close. The new test case is populated with test steps that represent the mobile actions that are captured while recording.
10. To view the details of each step:

- a. Click the test step to review. Each test step contains a screenshot of the action being performed.
- b. To expand the details, click **Mobile testing step** in the element tree on the right. The Mobile Testing Step tab opens and shows a screenshot of the test application. The Actions section at the top of the tab shows the individual actions that are performed in the test step. To highlight the associated element in the screenshot, click an action.
- c. To view the screenshot that is associated with a specific action, click the action in the Actions section.
For more information about modifying a recorded test step, see [Modify Mobile Test Steps \(see page 1868\)](#).
For more information about adding assertions to the test step, see [Add an Assertion to a Mobile Test Step \(see page 430\)](#).

Run a Mobile Test Case in the ITR

Follow these steps:

1. Ensure that the config file with the desired mobile asset is active.
2. [Open the test case \(see page 482\)](#) that you want to run.

3. Click **Start a new ITR**  on the toolbar.



Note: Select whether to start a new ITR run or open a previous ITR run (if you have run the ITR previously).

The Interactive Test Run window opens.

4. Click **Execute**  at the bottom of the ITR window.
The mobile simulator window opens and DevTest runs the test steps. The mobile application is visible on the simulator while the test is running.
A message opens indicating the test is complete. The simulator window closes.
5. Click OK.
For more information about using the ITR, see [Running Test Cases and Suites \(see page 533\)](#).

Run a Mobile Test Case Remotely

Follow these steps:

1. Verify that the config file that defines the expected mobile asset is active.
2. [Open the test case \(see page 482\)](#) to run.

3. Click **ITR**  on the toolbar.



Note: Select whether to start a new ITR run or open a previous ITR run (if you have run the ITR previously).

The Interactive Test Run window opens.

4. Click **Execute**  at the bottom of the ITR window.
The mobile simulator window opens and DevTest runs the test steps.
To view test case details, log in to your cloud provider (for example, SauceLabs.com). You can view the test as a live screencast while it runs on the SauceLabs emulator.
A message in DevTest indicates when the test is complete.
5. Click **OK**.
For more information about using the ITR, see [Running Test Cases and Suites \(see page 533\)](#).

Mobile Testing

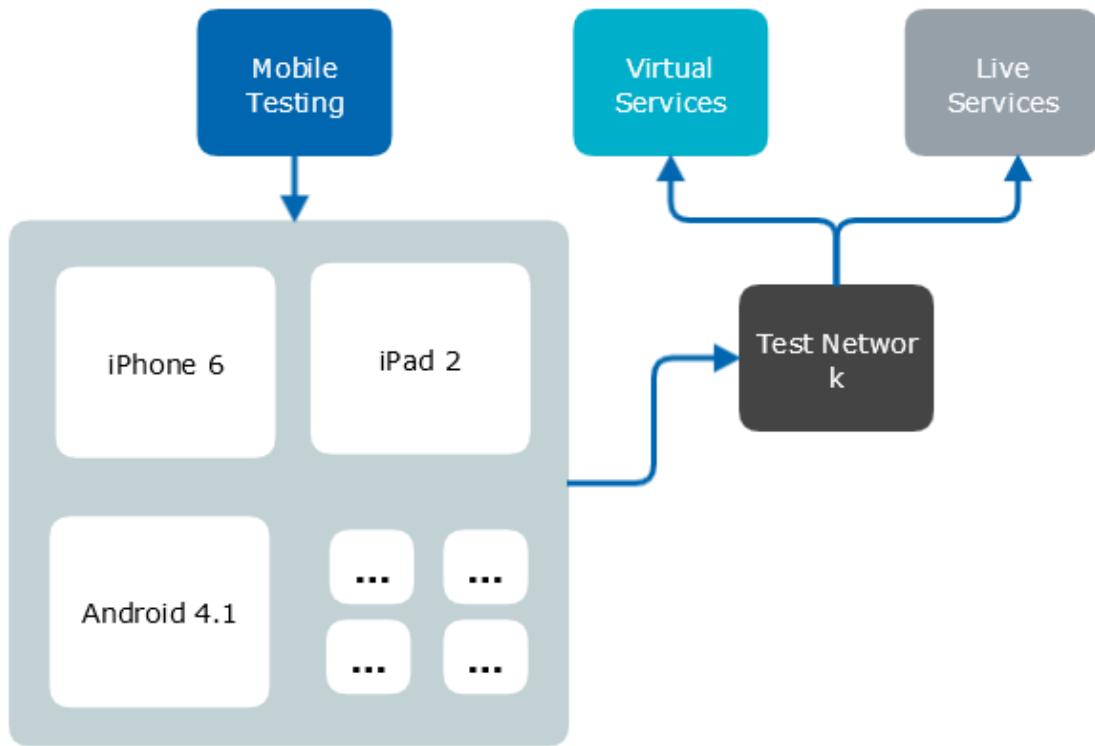
Mobile testing extends basic DevTest functionality for developing, staging, and monitoring tests cases to the testing of mobile iOS, Android, and Hybrid applications. You can easily create test cases by recording interactions with your mobile applications. You can then modify those tests like you would any other DevTest test cases by adding individual actions, assertions, and filters to each step.

The Mobile Test Recorder lets you record in the following ways:

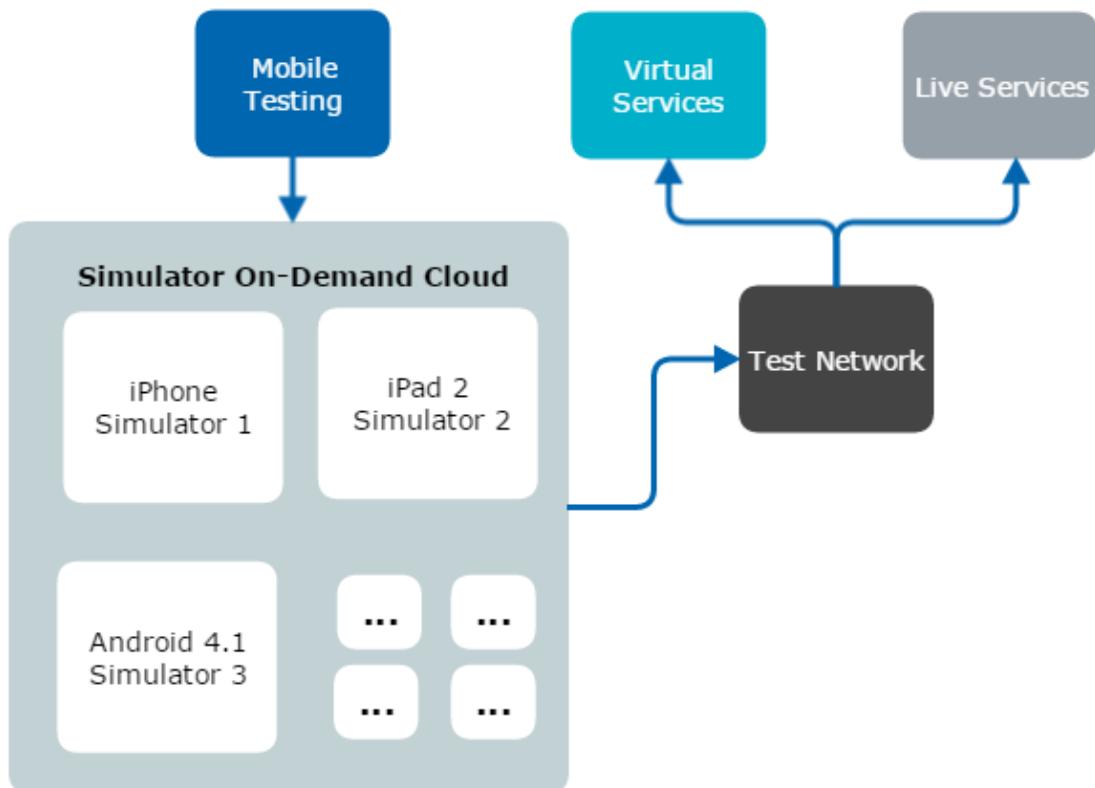
- Record directly from a device that is connected to your machine via the USB port.
- Record using installed Android and iOS simulators.
- Record using cloud-based Android and iOS simulators for a more scalable solution.

Mobile Testing lets you create a single test case that you can then test against multiple devices. This ability, coupled with the collection of available simulators, greatly simplifies the process of developing and testing applications for a variety of mobile devices.

The following illustration shows a mobile testing environment with attached devices and cloud-based simulators.



The following illustration shows a mobile testing environment with installed and cloud-based simulators.





Note: For more information about setting up mobile testing, see [Setting Up the Mobile Testing Environment \(see page 148\)](#).

This section contains the following pages:

- [How to Create and Replay a Mobile Test Case \(see page 641\)](#)
- [Web Browser Testing \(see page 642\)](#)
- [The Mobile Lab \(see page 642\)](#)
- [Automated Tests by Voyager \(see page 645\)](#)
- [Troubleshooting Mobile Test Cases \(see page 649\)](#)
- [Supported Mobile Actions and Gestures \(see page 650\)](#)
- [Work Remotely \(see page 650\)](#)
- [Application Signing \(see page 655\)](#)

How to Create and Replay a Mobile Test Case

To create and replay a mobile test case, complete the following tasks.

Follow these steps:

1. Create one of the following assets:

- **Emulator (Android) or Simulator (iOS) Session asset**
Defines the mobile emulator or simulator that is used for your test.
- **Attached Device Session asset**
Defines the mobile device that is used for your test. This asset is used for physical mobile devices that are connected to your network.
- **SauceLabs Session asset**
Defines the SauceLabs account information for mobile cloud testing.

For more information about each of these assets, see [Mobile Assets \(see page 1519\)](#).

For general information about creating an asset, see [Create Assets \(see page 403\)](#).

2. [Record a mobile test case \(see page 636\)](#).

3. [Modify the test steps as appropriate \(see page 1868\)](#).

4. Run a mobile test case.

- For general information about running a test case, see [Running Test Cases and Suites \(see page 533\)](#).
- For specific information about running a mobile test case in the ITR, see [Run a Mobile Test Case in the ITR \(see page 638\)](#).
- For specific information about running a mobile test case remotely, see [Run a Mobile Test Case Remotely \(see page 638\)](#).

Web Browser Testing

To record a web test, DevTest provides an application with a built-in web browser.

Follow these steps:

1. [Create a new asset \(see page 1519\)](#) that contains the platform and device combination that you want to test your web content in.
2. Load the .app or .apk file that contains an embedded web browser into the Application field of the Mobile Session dialog.



Note: DevTest installs an application with an embedded web browser for both iOS and Android (Mobile Browser.app and MobileBrowser.apk). In the Application field, navigate to: **LISA_HOME\examples\Data\mobile**.

3. [Record a test case \(see page 636\)](#) for the application to ensure that the content looks correct.

The Mobile Lab

The DevTest Mobile Lab lets you take a group of devices and simulators/emulators in your local network make them all available for CA Application Test to interact with.

A single computer running in your environment can have multiple devices that are connected to it. The Mobile Lab makes these devices available to your testers. You can test and manage up to 50 physical devices that are connected through USB or simulators. Both iOS and Android and multiple device versions are supported.

The Mobile Lab includes the same functionality as [DevTest Cloud Manager \(see page 585\)](#), but with the added feature of identifying attached mobile devices and emulators/simulators.

Start a Mobile Lab

When you start a virtual Mobile Lab, you are starting an instance of the lab.

Virtual Mobile Labs lets users sequester mobile application testing to a dedicated hardware pool (for example, dedicated OSX servers for iOS application testing). To launch a virtual Mobile Lab, DevTest requires at least one coordinator server process (per virtual lab instance) paired with one or more simulator server processes that would carry out testing tasks. One simulator instance per server is recommended. While the coordinator server process does not perform actual testing, it is required for correct test scheduling and result reporting.

For more information, see [Start a Lab \(see page 594\)](#).

You can start a lab by invoking the DevTest Server executables, and specifying a server name and a lab name.

Follow these steps:

1. From the command line, launch the Mobile Lab Coordinator Server.

```
CoordinatorServer -l LabName
```

where *LabName* identifies the user-defined lab name. One coordinator process per lab is recommended.

2. Launch the Mobile Lab Simulator.

```
Simulator -l LabName -n LabServerName
```

where *Labname* identifies the user-defined lab name and *LabServerName* is the name of the simulator. One simulator per lab server is recommended.

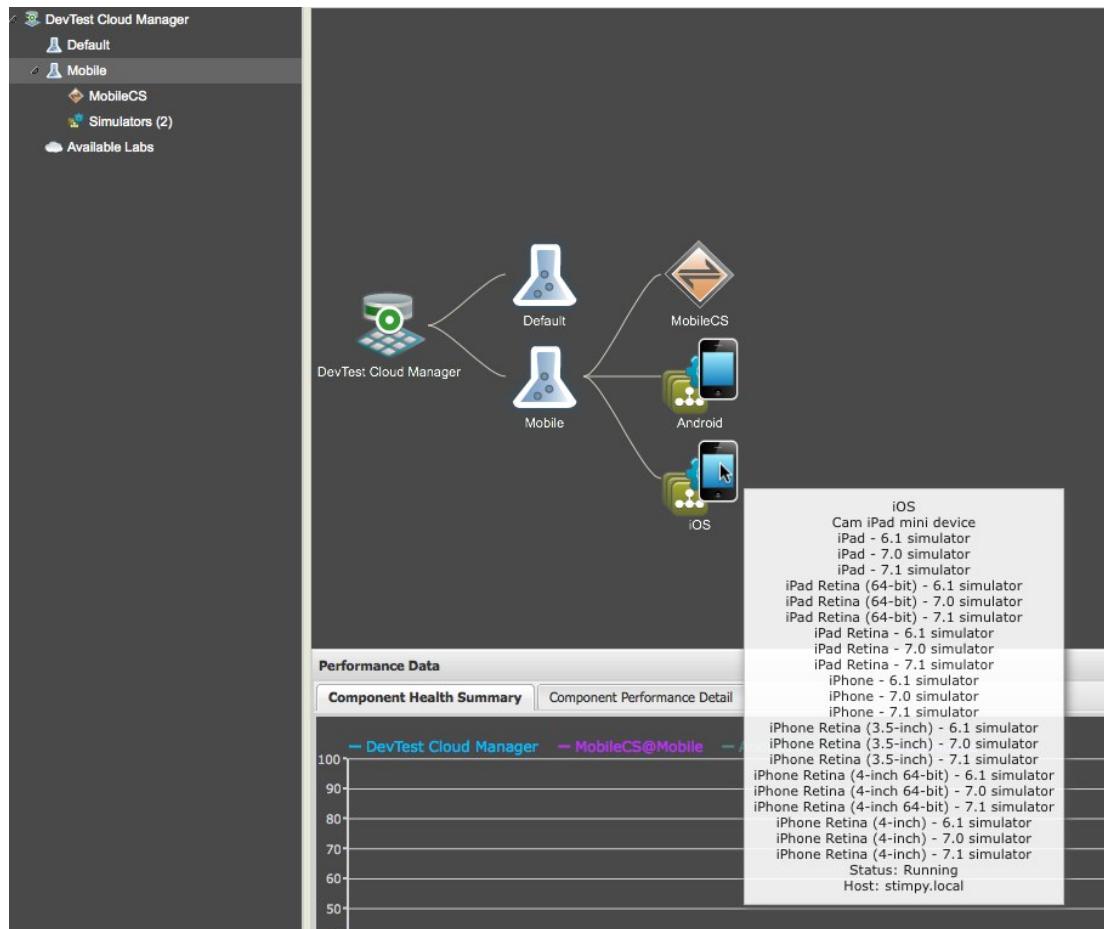


Note: These executables are found in the LISA_HOME\bin directory.

Verify the Mobile Lab

You can verify the existence of the Mobile Lab through the **Server Console**.

Validate the mobile capabilities of the individual machines in the lab by hovering the cursor over the Simulator icon. The capability set lists the attached devices, iPhone simulator images, and the android AVDs.



Mobile Lab with devices attached

Stage a Test in the Mobile Lab

Follow these steps:

1. Locate the test in the Project Panel.
2. Right-click the test, then select **StageTest**.



Note: Make sure you select the Lab coordinator server for staging the test to the lab.

Only the coordinators for the labs that have mobile capability are available for testing in the pre-built list of coordinators.

Stage a Suite of Tests in the Mobile Lab

Follow these steps:

1. Before you stage a suite of tests to the Mobile Lab, ensure that the suite has an assigned default coordinator. If not, you are requested to set one when staging the suite.
 - a. Open the test suite in an editor.
 - b. Click the **Defaults** tab.
 - c. Select the lab coordinator from the list of coordinators.
2. To stage a suite, right-click the suite in the Project Panel.
3. Select **Run with DCM**.
4. Click **Stage**.

Automated Tests by Voyager

Voyager is an application explorer, or "crawler", that looks at your mobile applications and creates test cases. Voyager inspects your application file, then breaks that file apart into all of the vectors - pages, links, gestures, and input - throughout the application. Those vectors are then explored and exercised and data is generated from it.

Voyager lets you:

- **Create a set of automated tests**

Voyager spares you the time that it takes to record tests into the recorder, modify them, and put them in a test suite. Voyager generates all of the vectors across the application and then creates and automates the test cases for you.

- **Create a test case for a specific page of your application**

You can identify a specific area of data from Voyager and then generate a test case for it.

- **Add steps to an already-existing test case using Voyager data**

You can take Voyager data and then use it to add a step to an already-existing test case.

Generate Tests

You can automatically generate test cases using Voyager.

Follow these steps:

1. From the Actions menu, click **Launch Voyager**.
The Voyager configuration dialog opens.
2. In the **Asset** drop-down box, select the mobile asset where you want to run the Voyager test.
A mobile asset must be defined for this drop-down box to display.
3. In the **App** field, enter or select the application that you want Voyager to access.
4. In the **Data Set** field, select an existing data set from the drop-down list, or click **New** to create a new one.

5. If your application requires authentication, enter its credentials in the **Account** and **Password** fields.
6. In the **App Graph** field, enter or select a name for the data file that contains the Voyager data.



Note: The data file for the test can be found under the Data folder in the Project Panel.

7. Click **Auto Generate Tests** to activate the Mobile Test Generator.



Note: Tests are created each time a new page is found, and the database is saved after Voyager is stopped.

8. In the **Output Folder** field, enter a location for your .appgraph file, or accept the default location.
9. Click **Reconfigure SauceLabs** if the SauceLabs account or access key has changed. If SauceLabs is not configured, the SauceLabs dialog opens before starting Voyager.
10. Click **OK** to start Voyager.
The Voyager Dashboard opens. As pages are crawled, the current page displays. The number of unique pages displays at the top.
11. Click **Stop** to stop Voyager.

Create a Page-Specific Test Case

Once you launch Voyager and create a data file to capture the application data, you can identify a specific area of the data and generate a test case for it.

Follow these steps:

1. Create a [new mobile test case \(see page 641\)](#).
2. In the Test Editor, click **Create steps by recording or templating** .
3. Click **Record Test Case for User Interface, Generate Mobile Steps**.
The **Page Selection** dialog opens.



Note: If you are using a mobile simulator to record, the mobile simulator window also opens. This only happens when running Voyager. In this case, the steps are generated from data that has already been recorded. For this dialog to populate, you must have run Voyager at least once.

4. Select **UICatalog** as the application from the drop-down list.
5. Select **Data/Voyager/UICatalog.appgraph** as the Voyager database.
6. Select a page(s) from Voyager to include in your test.

Add Mobile Steps to a Test Case from an Existing Voyager File

You can add a step to an already-existing mobile test case using the data from Voyager.

Follow these steps:

1. Open a mobile test case.
2. Click **Record Test Case for User Interface, Generate Mobile Steps**.
The **Page Selection** dialog opens. Select the application file that you have recorded with Voyager.



Note: For this dialog to populate, you must have run Voyager at least once.

3. Select the .appgraph file that contains the step you want to add to your test.
4. Click on a recorded page.
5. Verify that the pages were added to the database.

Use Data Sets with Voyager

You can use data sets with Voyager to extract specific data from XPath to populate text fields while crawling the app.

Follow these steps:

1. Start the [mobile recorder \(see page 636\)](#).
2. Navigate to the location of the field that will receive input on the device or on the simulator window.
3. Refresh the screen in the recorder.
4. Right-click the input field, then select **Copy XPath** to copy the XPath information.

5. Create a data file that uses the XPaths collected in the previous steps as the key for each data field.

This file can be any file that DevTest Solutions supports. For example, the data file can be an Excel spreadsheet.

6. In the DevTest Workstation, click **Actions, Launch Voyager**.

The Voyager dialog opens.

7. In the Data Set field, select an existing data set from the drop-down list. Or, if one has not been defined, click **New**.

Enter the following information for a new data set:

Data Set Type

Specifies the type of data file you are creating the data set from.

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

Keep the default setting of **Start over**. After all the data is read, this setting specifies to continue reading data from the top of the data set.

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 Excel File data set, enter:

File Location

Enter the full path name of the data file, or browse to file with the browse button. You can use a property in the path name (for example, LISA_HOME).



Note: Data sets are saved into the workspace under Data/Voyager/Datasets with a .dataset extension.

Browse

Specify where to locate the data file.

Select from File System, Class Path, or URL.

Sheet Name

Specifies the Excel sheet (tab) to pull data from.

Open XLS File

Click to open the Excel file with the XPath data. You can make any additional edits.

8. Click **OK** on the data set dialog.

9. Click **OK** on the Voyager dialog to launch Voyager.

The data set is used to populate the text fields while crawling the app.

You can also gather XPath information from an existing test case, either manually recorded or generated by Voyager.

Follow these steps:

1. Open the mobile test.
2. Locate the XPath for the fields that receive input. Click **Mobile testing step** in the element tree on the right to expand the details for the step.
The Mobile Testing Step tab opens and shows a screenshot of the test application.
3. Right-click and select **Copy** to copy the XPath information.

Troubleshooting Mobile Test Cases

- To get more error messages, click System Messages in the bottom left corner of DevTest Workstation. This pane often provides extra information that could resolve the issue.
- For additional debugging information, set the following property in LISA_HOME\logging.properties:
`log4j.logger.com.itko.lisa.mobile=DEBUG`.
- If screen shots do not appear in the Test Recorder during a mobile recording, ensure that hardware acceleration is turned off in the AVD. This issue only applies to AVDs created in the Android SDK.

Follow these steps:

1. Open the Android Virtual Device Manager (see Set Up the Android SDK in the topic [Preinstallation Steps for Mobile Testing \(see page \)](#)).
2. Locate your AVD, then click **Edit**.
3. Clear the **Use Host GPU** check box.
4. Click **OK**.

Limitations

Mobile Testing does not support certain test cases. They include:

- **Recording from a remote Mac**
Local iOS tests must be made with a Mac. They can also be made using a remote Mac or in the Cloud with SauceLabs. Local Android tests can be made on Windows or Mac.
- **Running one test at a time per Mac**
Mobile Testing lets you stage multiple tests at a time to run on Windows. You can run staged mobile tests on a Mac, but need multiple Macs or VMs to run parallel tests.
- **Making a test for a retail-signed iOS app**
Apps that are downloaded from Apple's App Store are retail-signed iOS apps. Mobile Testing cannot make or play back tests with them. Instead, use iOS apps that are debuggable (and not encrypted).

You can use Android apps from the Google Play Store to create and play back tests, however.

- **Creating manual tests from scratch**

Recording a test directly on a device or building test steps manually is not supported. The best way to create a test is to use the Recorder window inside DevTest Workstation.

- **Mobile browser testing**

Mobile browser testing only runs on a web view inside a native app. Tests do not run on external retail browsers such as Safari and Chrome.

- **Secure screens**

You cannot take screen shots of screens that applications mark as secure.

Supported Mobile Actions and Gestures

DevTest Solutions supports the following mobile gestures:

Mobile Gesture	iOS Native App	Android Native App	iOS Hybrid App	Android Hybrid App
Tap	Yes	Yes	Yes	Yes
Long Tap	Yes	Yes	Yes	Yes
Swipe Element	Yes	Yes	Yes	Yes
Swipe Screen	Yes	Yes	Yes	No
Pinch	Yes	Yes	Yes	No
Zoom	Yes	Yes	Yes	N/A
Rotate	Yes	Yes	Yes	N/A
Scroll To	Yes	Yes	Yes	N/A
Change Orientation	Yes	Yes	Yes	Yes
Shake	Yes	Yes	Yes	N/A
Back	No	Yes	No	Yes
Go to Background	Yes	Yes	Yes	Yes
Hide Keyboard	Yes	Yes	Yes	Yes

Work Remotely

Both remote testing and remote recording are supported with Mobile Testing.

What's the difference?

- *Remote testing* tests physical iOS and Android mobile devices that are connected to a computer at one location, which another computer then connects to from a different location. Remote testing logs into a remote Workstation registry. After logging in, the local computer acts as a controller to the remote computer. All apps and assets reside on the remote computer. The local computer tells the remote computer to start the recording on the remote computer.

- *Remote recording* lets you record mobile tests on a remote computer during a local session. The remote recordings can then be run on the local computer (Windows or Mac). The user provides the host address for the remote computer while on the local computer. After logging in, the local computer acts as a controller to the local computer only. All assets and apps reside locally. The recorded test is saved locally but all the actions take place on the remote computer.

Remote Recording

Remote recording lets you record mobile tests on a remote computer during a local session. The remote recordings can then be run on the local computer (Windows or Mac).

When recording a test case remotely, the test case is created and controlled on the local computer. The remote computer is the endpoint, where the device or simulator resides. The availability of these remote devices to record on lets you create a testing strategy that can use various operating system and device combinations.

The local computer must be running the Registry and the Workstation to test remotely. The remote computer must include DevTest Workstation, as well as:

- An AVD and Android SDK if you are recording an Android test.
- XCode must be installed to make a remote iOS test recording.
- A running registry.

Remote recording uses the following process:

1. DevTest Workstation connects to its local registry.
2. The remote computer runs its registry where the physical mobile device or simulator is available.
3. A mobile asset is successfully created and locally verified. An asset cannot be verified remotely.
4. Create a mobile test, select your asset, and then enter the remote host address in the **Mobile Recorder** dialog. A host must be authenticated the first time, whether it is being used for playback or recording.



Note: Previous hosts are saved within the session of DevTest Workstation. When you close DevTest Workstation, you are required to authenticate again and previous hosts are removed.

The recording could take slightly longer than a local test, as the app is copied to the remote computer.

5. After the connection is made, the test is recorded locally while using the remote simulator or device.

6. When the recording is complete, a test step at the beginning of the test displays:

Start remote Appium.

This step starts the Appium service on the remote machine, so that the test can be run using remote devices/simulators. If you play the test back on a local computer (using local simulators or devices), then you must delete this step first.



Note: An iOS test requires a remote Mac for playback and recording. An Android test can play back on both Mac and Windows.

The following procedure explains how to run a remote test for a native application running on an Android device. In this example, the Android device is connected to a remote Mac.

Follow these steps:

1. Start the registry and log in to DevTest Workstation on the local Windows computer.
2. Start the registry on the remote Mac.

3. Click **Create steps by recording or templating** .

4. Click **Record Test Case for User Interface, Mobile Recorder**.

The **Mobile Recorder** window opens on the local Windows computer.



Note: If you are using a remote simulator to record, the mobile simulator windows also opens on the remote computer. If you are using a real device connected to the remote computer, the app opens on the device.

5. Select an asset which to connect from the **Choose Mobile asset** drop-down list, then click **OK**.
6. Enter the **Remote Host** address to connect to, and wait for the recording window to appear.
7. If this is the first time you have connected to the host during the DevTest Workstation session, enter your **User** name and **Password**. Previous hosts are saved in the host field only during that DevTest Workstation session.
8. Click **OK**.
9. Click **Start Recording** at the bottom of the recorder window when it starts up.
10. Click **Stop Recording** when you have captured all the actions for the test.
The recorder window and the mobile simulator close. The new test case is populated with test steps that represent the mobile actions that are captured while recording.

You can play back the test on the local Windows computer while it is connected to the Mac. The test is saved locally but all the actions take place on the remote Mac.



If you use remote recording via TestRunner, you must set the following properties to use authenticate the remote host:

- lisa.mobile.remote.user
- lisa.mobile.remote.password

See [Local Properties File \(see page 1671\)](#) for more information.

Remote Testing

Use remote testing to test physical iOS and Android mobile devices that are connected to a computer at one location, which another computer then connects to from a different location. The availability of remote devices lets you create a testing strategy that can use various operating system and device combinations.

Remote testing lets you run your mobile device tests on various operating system combinations, such as:

- Mac to Windows
- Windows to Mac
- Windows to Windows
- Mac to Mac

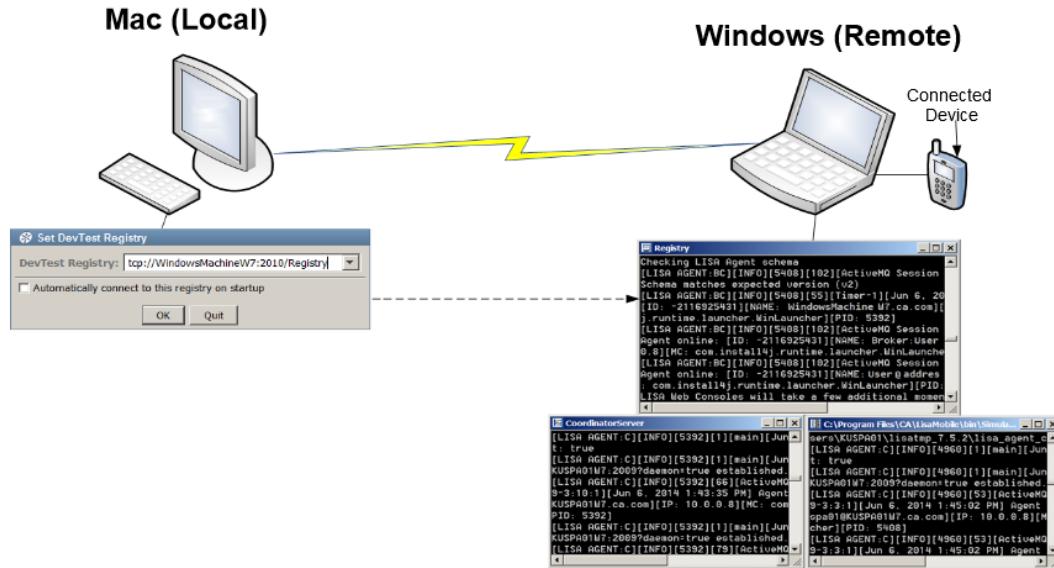
Remote testing with DevTest Workstation uses the following process:

1. Connecting DevTest Workstation from a local machine to the registry, simulator, and coordinator running on a remote machine where a physical mobile device is connected.
2. Create and successfully verify a mobile asset on the local machine before you run remote tests.
3. Create a test case on your local machine.
4. Stage the test case on the local machine, running against the remote machine's registry, simulator, and coordinator that you are connected to.



Note: You can only run tests on Android devices from a Windows machine. You can run both Android and iOS tests from a Mac machine.

The following procedure explains how to run a remote test for a native application running on an Android device. In this example, the Android device is connected to a Windows machine running the registry, simulator, and coordinator. DevTest Workstation on the Mac machine then connects to the registry, simulator, and coordinator on the Windows machine (or the "remote machine"), as shown here:



Follow these steps:

1. Start the registry, simulator, and coordinator on the Windows machine.
 2. Start DevTest Workstation on the Mac machine.
 3. Connect to the registry that is running on the Windows machine. Enter the address of the Windows machine registry in the Set LISA Registry dialog.
 4. Create a new project in DevTest Workstation on the Mac machine.
 5. Copy an .apk file (native Android application) to the Data folder in your LISA project on the Mac machine. You will point your asset Application field to that location for the .apk file. For example:
`Application: <LISA_PROJ_ROOT>/Data/ApiDemos.apk`
 6. Verify that your local computer contains a successfully recorded test case and a verified asset. If your computer already contains these items, continue with step 10.
 7. On the Mac machine, create an Android Mac native real device asset for ApiDemos.
 8. Verify the asset to 100 percent.
 9. Record and play back a test locally, to verify that both actions succeed on the local machine with the device asset that you created in Step 7.
 10. Right-click the test case, then click **Stage Test**.

11. To stage the test, click **Play**.

You can also use the All test suite functionality to perform remote staging.

Follow these steps:

1. Right-click **AllTestSuite**, then **Run with DCM tcp://<Connected Registry>** to stage the test remotely.

The **Run Suite via tcp://<Connected Registry>/** dialog opens.

2. Click **Stage**.

The tests start on the Mac machine, and the test results are rendered. DevTest Workstation on the Mac connects to a remote registry, simulator, and coordinator running on the Windows machine.



Note: Verify that your AllTestSuite file contains the items that you are trying to stage (actual tests, MAR files, suites).

Application Signing

All iOS apps must be code signed and provisioned to launch on a device. *Provisioning* is the process of preparing and configuring an app to launch on devices and use app services. If you did not prepare IPA signing on your iOS app during the Mobile Testing [preinstallation process \(see page 149\)](#) (see Prepare IPA Signing), you can sign the apps when you create an asset. Using this process, when you record tests, iOS applications that need signing are copied to a temporary directory, signed, and then used in the test. The original applications are not modified.

Before this functionality can be used, however, you must set two properties:

- **MOBILE_PROVISION** = *path to iOS_Team_Provisioning_Profile_.mobileprovision*

The MOBILE_PROVISION property must point to the full path to the provisioning file. To run tests on remote simulators, the *.mobileprovision file must reside in the project and its path must be parameterized using {{LISA_PROJ_ROOT}}.

For example: {{LISA_PROJ_ROOT}}/Data/mobile/iOS_Team_Provisioning_Profile_.mobileprovision

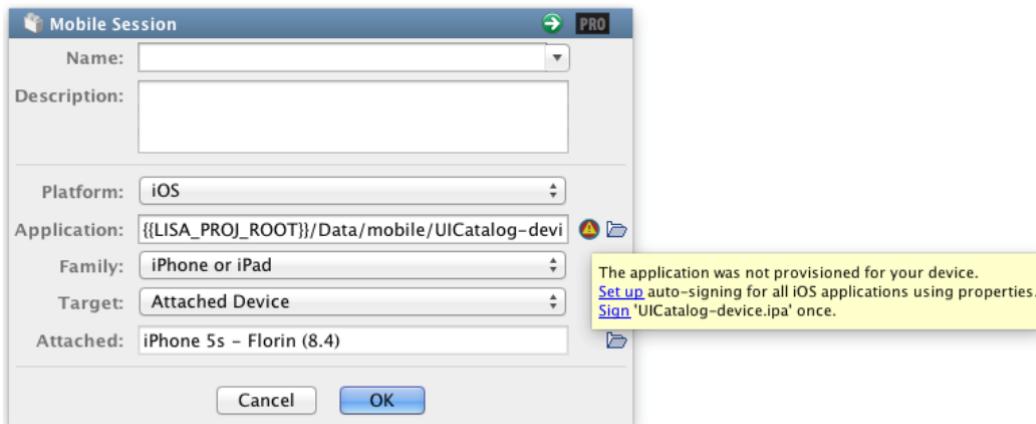
- **IOS_CERTIFICATE** = *the certificate name as shown in Keychain Access (do not use the certificate file name)*

The IOS_CERTIFICATE property must be set to the certificate name. An exact match for the certificate name is not required. For example, you can use "iPhone Developer" and that would be enough for a match. But if you have other iOS certificates, a better idea would be to use the entire certificate name.

When setting up a mobile asset using the Mobile Session asset dialog, you can select an attached device and an application to run on that attached device. That application is then validated, and an error displays if the application is not provisioned to run on the device. If that happens, you can select one of the following options:

- Set up automatic signing by generating the required properties in the current configuration.

- Set up the application once using your certificate and provisioning file. This option only keeps the signed version.



Note: Application signing does not support *.app.zip application archives. Any .zip files must be extracted.

Set up Auto Signing

This option is beneficial for running iOS apps.

Follow these steps:

1. In the Mobile Session dialog, click **Set up auto-signing for all iOS applications using properties**. The **Set Up Auto-Signing** dialog opens.
2. In the **Provisioning** field, select the provisioning file that is associated with the app.
3. In the **Certificate** field, enter the certificate name.
4. Click **OK**.
The file path and the certificate name are validated and the required project properties are generated.

Sign Once

This option works best for tests that are run on a remote Mac simulator that do not have your certificate installed in its keychain.

Follow these steps:

1. In the Mobile Session dialog, click **Sign 'application name' once**. The **Sign Once** dialog opens.
2. In the **Provisioning** field, select the provisioning file that is associated with the app.

3. In the **Certificate** field, enter the certificate name.

4. Click **OK**.

The file path and the certificate name are validated. The invalid app ({{PROJ_ROOT_PATH}} /Data/mobile/[application]) is replaced with a version signed with the provided provisioning file and certificate.

Advanced Features

This section contains the following pages:

- [Using BeanShell in DevTest \(see page 657\)](#)
- [Class Loader Sandbox Example \(see page 660\)](#)
- [Generating DDLs \(see page 661\)](#)

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 \(see page 1873\)](#) test step
- As the interpreter framework for the [Assert by Script Execution \(see page 1499\)](#) 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.](http://www.beanshell.org/) (<http://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 \(see page 1873\)](#) step and the [Assert by Script Execution \(see page 1499\)](#) 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 \(see page 1212\)](#).

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 rfc3339 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> (<http://tools.ietf.org/html/rfc3339#section-5.>)

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, CAI, and CVS, use the following commands:

- **java -jar LisaAgent.jar -ddl oracle** (generate the Oracle DDL for CAI)
- **java -jar LisaAgent.jar -ddl mysql** (generate the MySQL DDL for CAI)

Using CA Service Virtualization

This section describes how to use CA Service Virtualization to virtualize software service behavior and model a virtual service to stand in for the actual service during development and testing.



Note: The DevTest Portal is a web-based application that is intended to become the primary user interface for DevTest Solutions. The Portal provides capabilities for some of the most-used workflows for DevTest products. Over time, CA will enhance the functionality of the DevTest Portal and eventually sunset the other interfaces in the LISA product line, including DevTest Workstation. For a quick summary of the functionality available in the Portal, see [DevTest Portal Functionality \(see page 48\)](#).

[Using the DevTest Portal with CA Service Virtualization \(see page 693\)](#) provides detailed information about using the Portal to virtualize service behavior for supported features.

[Using the Workstation and Console with CA Service Virtualization \(see page 758\)](#) provides detailed information about using the DevTest Workstation and DevTest Console to virtualize service behavior for features that have not yet migrated to the Portal.

Introduction to Virtualization

Virtualization usually refers to hardware virtualization, where the behavior of a physical asset, such as a server or application in a software emulator, is simulated and the emulator is hosted in a virtual environment. The virtual environment provides the same communication with the emulated asset as the physical environment.

Virtualization has the following advantages:

- Better manages physical assets, which eases change and configuration management

- Improves the utilization of physical capacity, which better leverages the capacity of the physical assets
- Improves agility, which avoids the costly delays that happen while waiting for IT to reconfigure or switch servers

CA Service Virtualization provides service virtualization. The process is in principle the same as that for hardware virtualization.

Types of Service Virtualization

CA Service Virtualization has two product configurations, optimized for specific customer applications:

- CA Service Virtualization
- CA Service Virtualization for Performance

CA Service Virtualization is best 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 is specifically for performance testing applications, and is more scalable, only limited by the underlying hardware and network.

CA Service Virtualization mode is set as the default configuration. For information about how to override the default, see [How to Configure CA Service Virtualization \(see page 667\)](#).

Service Virtualization Overview

Service virtualization is the imaging of software service behavior and the modeling of a virtual service to stand in for the actual service during development and testing. Service virtualization is complementary to hardware virtualization and addresses its limitations. The word *virtualization* refers to service virtualization in this documentation.

You may not want or be able to stay connected to the server for your quality assurance tasks. CA Service Virtualization emulates the behavior of the server.

Within CA Service Virtualization, you can use a virtual service environment (VSE) as the client to drive the service to be recorded. However, you will most likely exercise the service for recording with your client (for example, a browser or an in-house application).

High-level Virtualization Steps

Contents

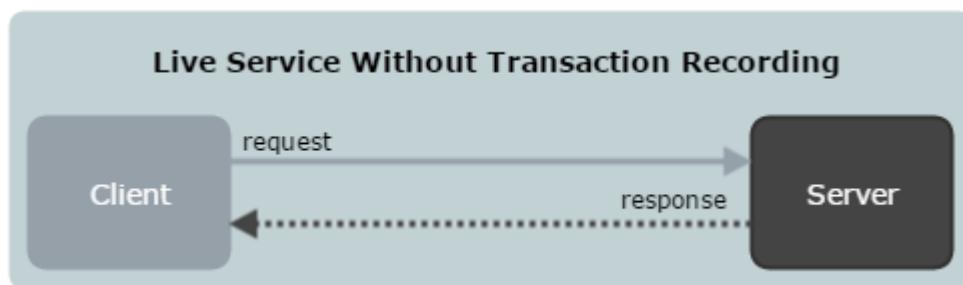
- [Virtualization of Messaging Systems \(see page 664\)](#)

The high-level steps in CA Service Virtualization are:

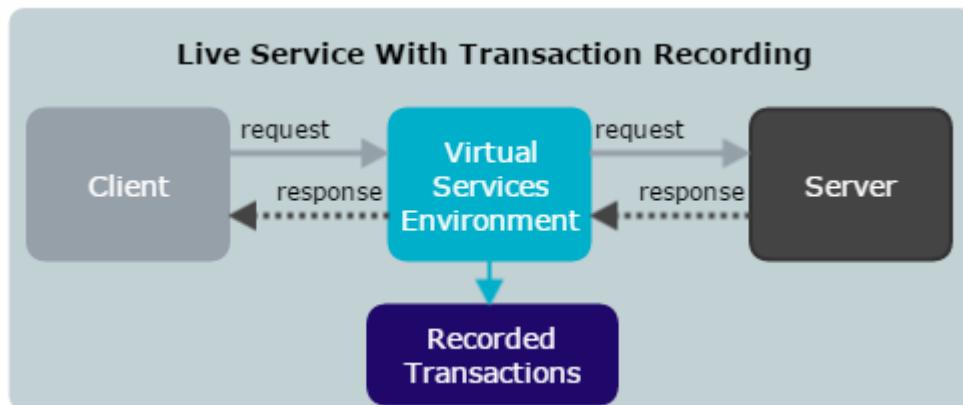
1. Take an image of the service behavior (service image); record transactions that the server handles.
2. Construct the virtual service from the behavior (virtual service model or VSM).
3. Deploy the VSM to the Virtual Service Environment (VSE). The VSM then looks at the captured service images to find the appropriate responses for requests coming in to the VSE.

The following graphics show that when recording the image, VSE acts as the pass through mechanism between the client and server. While VSE passes the requests and responses along, it records the transactions.

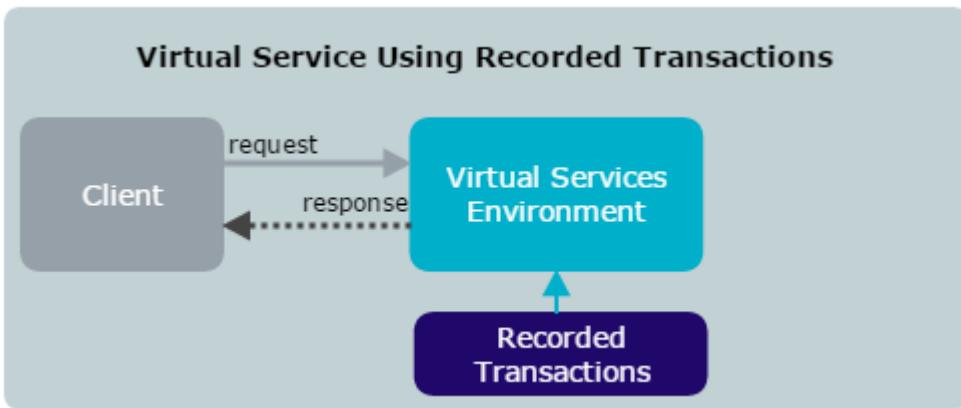
Normal Operation



Recording



At the time of virtualization, in the absence of the server, VSE responds to the client requests by consulting the recorded transactions.



Virtualization of 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:

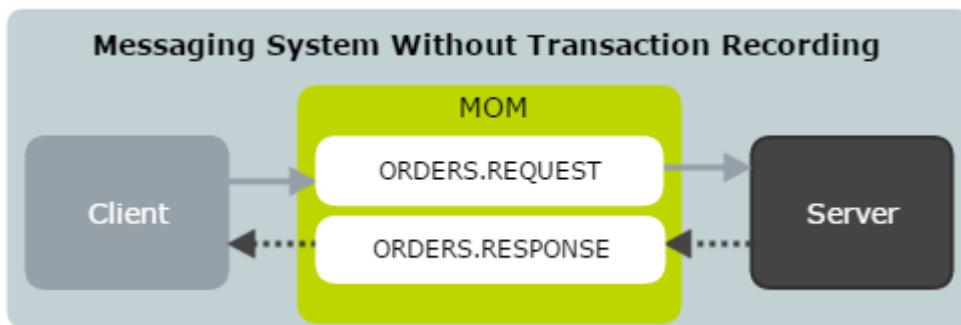
- **Queues**

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.

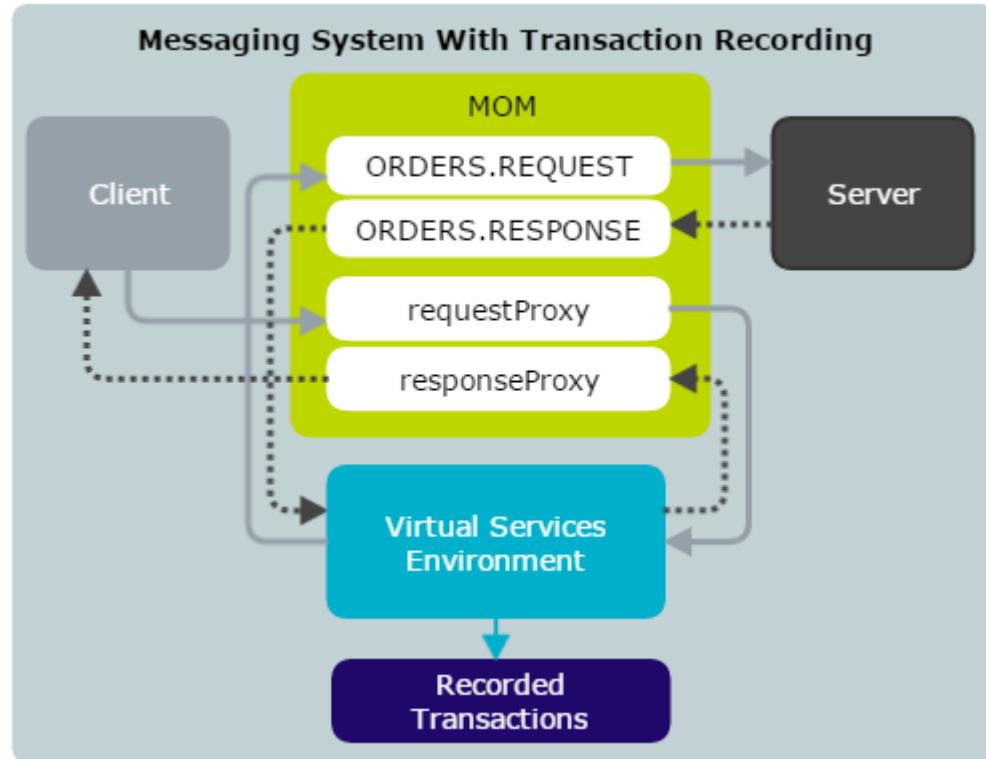
The following graphic shows a simple message-based service. In this scenario, the client adds messages to a queue (**ORDERS.REQUEST**), which the server picks up. The response from the server is in the form of messages added to another queue (**ORDERS.RESPONSE**). The client then picks them up.



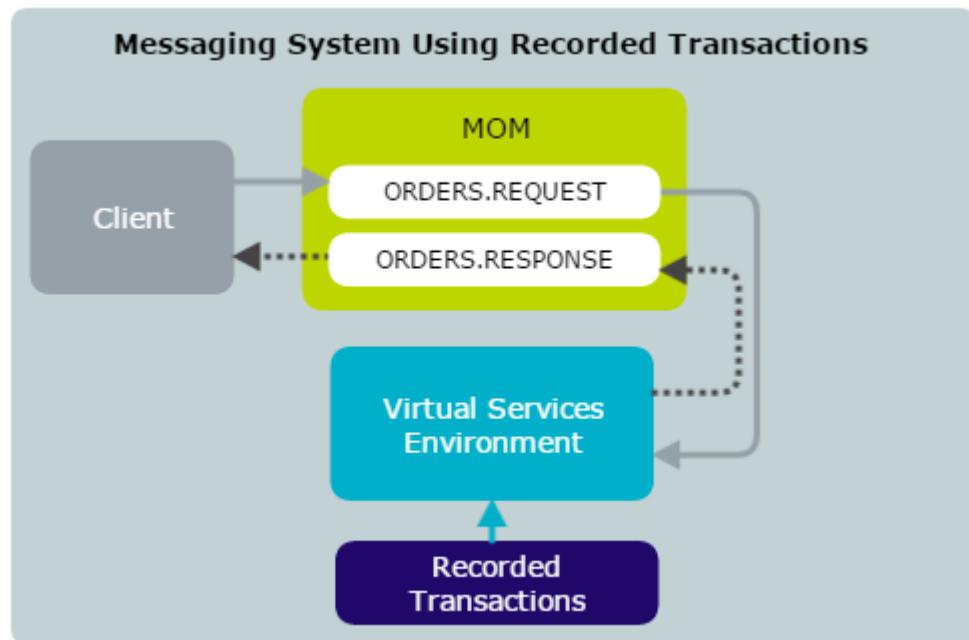
Possible variations include:

- 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.



Installation and Configuration

This section contains the following pages:

- [How to Install CA Service Virtualization \(see page 666\)](#)
- [How to Configure CA Service Virtualization \(see page 667\)](#)
- [Install the Database Simulator \(see page 668\)](#)
- [DDLS for Major Databases \(see page 670\)](#)

How to 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 \(see page 50\)](#).

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.

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 \(see page 50\)](#).

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/vse**
Contains examples relating to VSE.
- **tmp**
Contains the VSE workspace where VSE temporarily stores conversations and stateless transactions.
- **vseDeploy**
Contains deployed VSMs and related data.

How to 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 \(see page 1362\)](#).

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 `local.properties` (optional).

You can optionally set the following extra configurations in `local.properties`:

- **`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.

To change to CA Service Virtualization for Performance mode, perform one of the following actions:

- In `local.properties`, set `lisa.vse.performance.enabled=true`.

- Issue the VSEManager command `--performance on`.

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](#) (see page 703).

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.

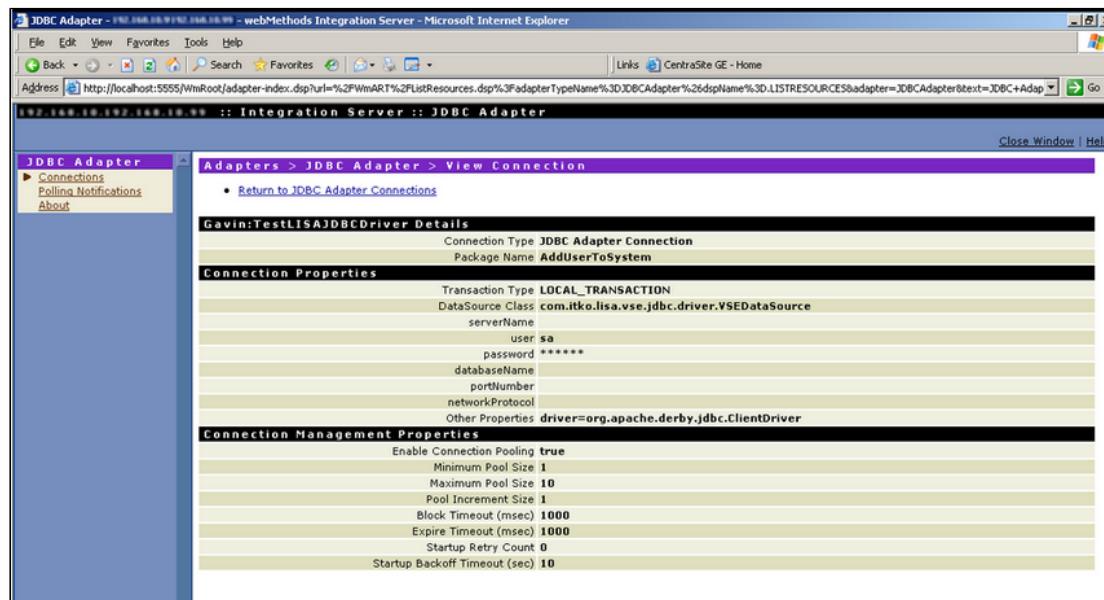
url: The value must be set to the connection URL that the real driver expects. (It must be defined as the last property so that it can contain semi-colons.)



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 CA Continuous Application Insight drivers, make the simulation driver the "outside" driver. Specify the CAI 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.idbc.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 \(see page 671\)](#)
- [EclipseLink Persistence Unit Properties for Schema Generation \(see page 674\)](#)

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.eclip&lt;!--&lt;!--se.persis&lt;!--&lt;!--tence.config.TargetDatabase;import org.eclip&lt;!--&lt;!--se.persis&lt;!--&lt;!--tence.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="0C4J_10_1_3"/>Example: property Mapimport org.eclipse.persistence.config.TargetServer;import org.eclipse.persistence.config.PersistenceUnitProperties;propertiesMap.put(PersistenceUnitProperties.TARGET_SERVER, TargetServer.0C4J_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**.

Understanding CA Service Virtualization

This section contains the following pages:

- [How to Work with CA Service Virtualization \(see page 677\)](#)
- [CA Service Virtualization Components \(see page 677\)](#)
- [Virtual Service Models \(see page 678\)](#)
- [Service Images \(see page 679\)](#)
- [How Virtualization Works \(see page 680\)](#)
- [Magic Strings and Dates \(see page 682\)](#)
- [Understanding VSE Transactions \(see page 687\)](#)
- [Match Tolerance \(see page 690\)](#)
- [Track Transactions \(see page 692\)](#)

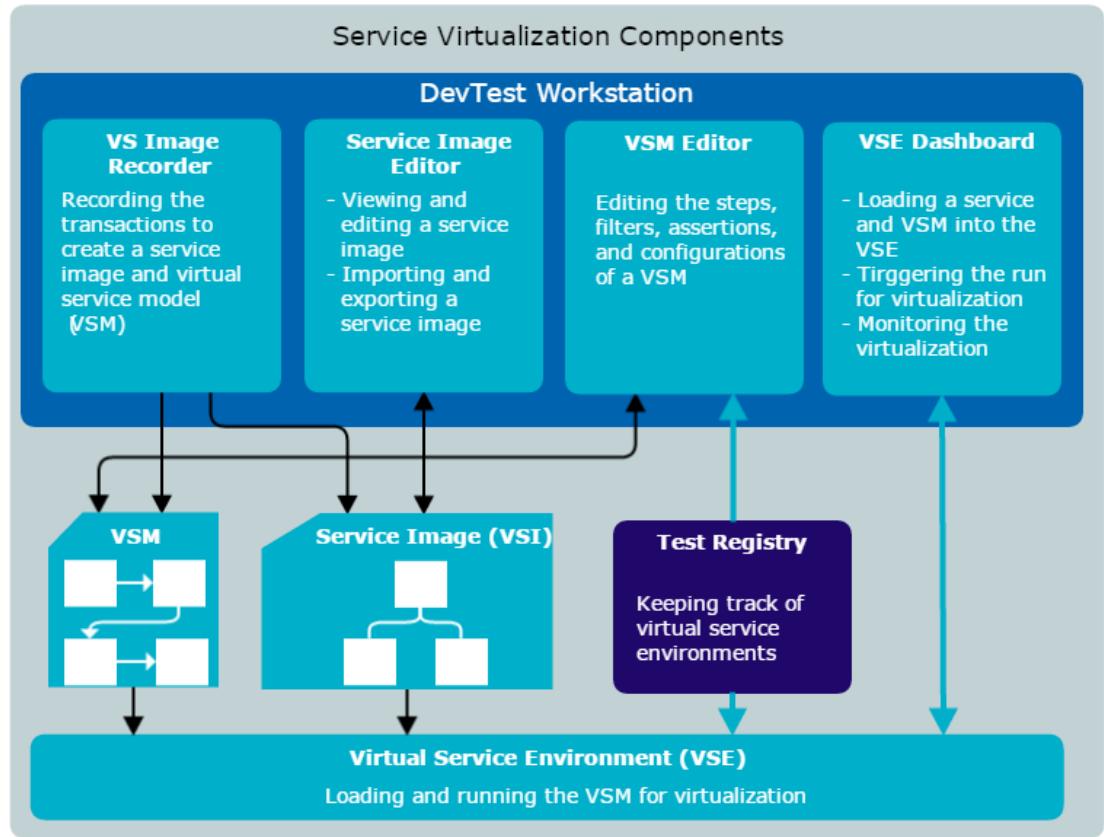
How to Work with CA Service Virtualization

The general process for working with CA Service Virtualization includes the following steps:

1. Start the **Virtual Service Image Recorder**.
2. In the **Virtual Service Image Recorder**, provide basic information about what to record. Select the appropriate protocols. Provide any information that the protocols require.
3. In the **Virtual Service Image Recorder**, start the recording.
4. Exercise the client communication with the server routed through VSE. VSE records the traffic.
5. In the **Virtual Service Image Recorder**, finish the recording.
6. In the VSE Console, deploy the virtual service model and start the virtual service.
7. Run live requests against CA Service Virtualization.

CA Service Virtualization Components

The following graphic shows how the components in CA Service Virtualization relate to each other.



The Virtual Service Image Recorder creates a service image and a VSM. 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 Test Registry service tracks one or more VSEs that are running, and DevTest Workstation uses it to connect to the VSE. The VSE Dashboard is the web UI used to monitor and control the VSMs and service images that are loaded onto the VSEs.

Virtual Service Models

You can conceptualize a virtual service model as a series of steps to be executed when a request is received. The steps of the VSM create and pass back a response to the request. A virtual service model is stored in a .vsm file.

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.

After you record a service image, VSE automatically generates the protocol-specific steps in the VSM. You can modify any of the generated steps. You can also:

- Populate the responses from an 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

After virtualization, a VSM must be deployed to the VSE. 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. In general, VSMs are deployable to VSEs only, while test cases can be staged to coordinators, but not to VSEs.

Service Images

Contents

- [Imported Transactions \(see page 680\)](#)

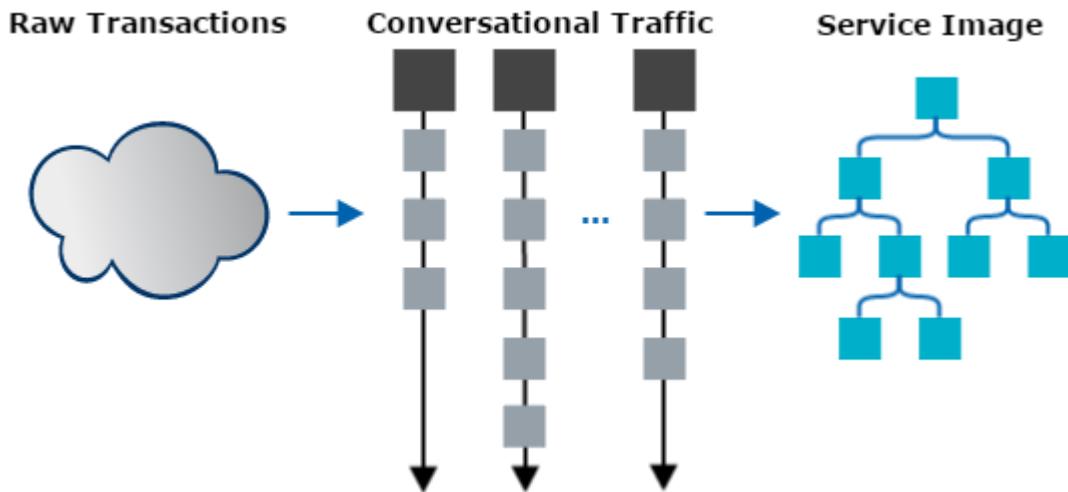
A *service image* is a recording of the interaction between the client and server as created by CA Service Virtualization. A virtual service model references a service image. After a service image is recorded, it is used to deliver the appropriate response to the client in the absence of the server.

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 view, edit, or create a service image with the **Service Image Editor**.

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. This process is known as the virtualization of a server. The process requires loading the service image that a VSM references and running it in the VSE dashboard.

When VSE receives a request, VSE examines the request and tries to match it to an existing conversational state (session) in VSE. For example, a cookie ID or some other session identifier can "tie" requests in stateless protocols such as HTTP. 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 may 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 VSE finds a previous session and VSE is in a conversation, the next transaction match depends on many factors. Those factors can include navigation and match tolerances.

If 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.



More Information:

- [How Conversational Requests are Handled \(see page 681\)](#)

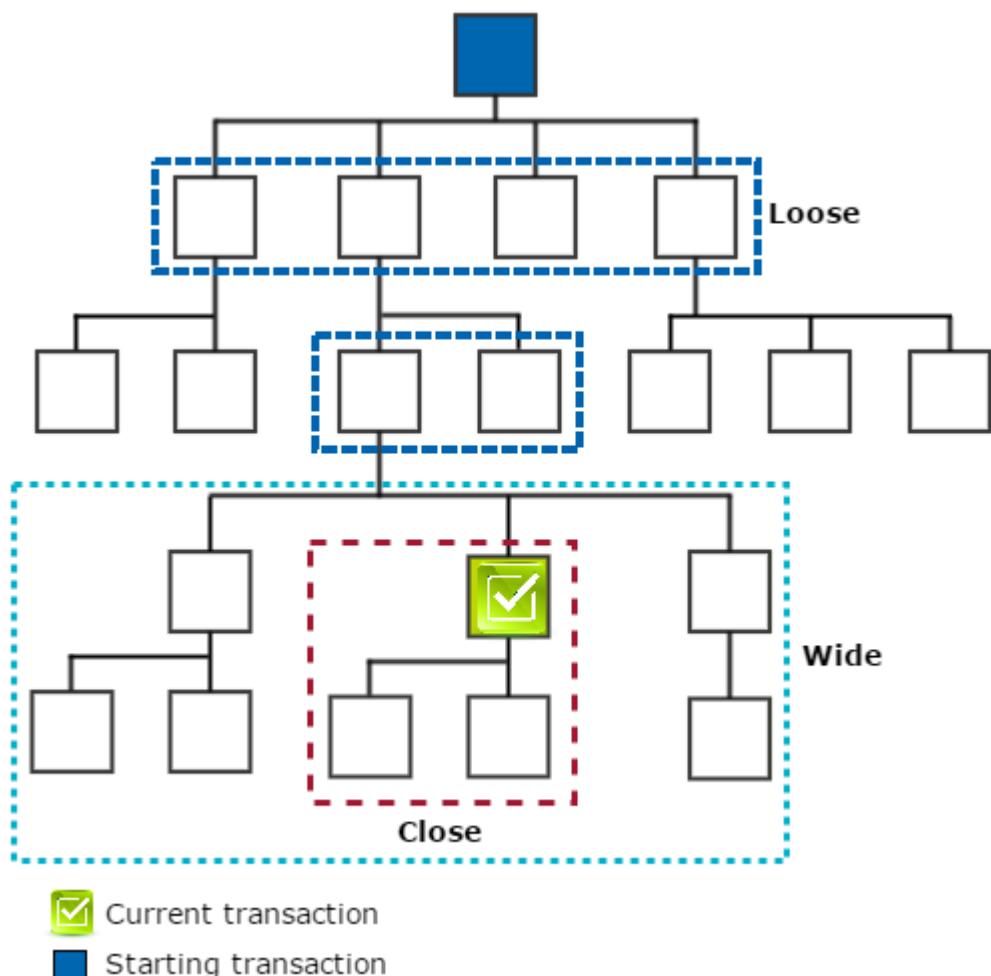
How Conversational Requests are Handled

The navigation tolerance that you can specify for every node in the tree plays an important role in how the VSM handles a conversation request. The navigation tolerance is used to determine where in the conversation tree a VSM searches for a transaction that follows the specified transaction.

Navigation Tolerance Levels

- **Close:** The children of the current transaction are searched.
- **Wide:** A close search, including the current transaction plus siblings and nieces/nephews of the current transactions.
- **Loose:** A wide search, plus the parent and siblings of the current transaction, followed by the children of the starting transaction. If both fail to find a match, then the starting transactions for all conversations are checked, resulting in searching the full conversation.

The following graphic shows how the navigation tolerance affects the transactions to be searched in a conversation tree. A check mark marks the current transaction.



At the time of recording, the VSE recorder allows for initializing the navigation tolerance on transactions the following settings:

- **Default navigation**

Defines the default tolerance on all Meta transactions that have child Meta transactions.

Default: Wide

- **Last**

Defines the default tolerance for Meta transactions that are "leaf" transactions without any child Meta transactions.

Default: Loose

You can change these parameters later for each node through the **Service Image Editor** in DevTest Workstation.

The defaults provide a better match on "right" behavior. VSE responds correctly more often in situations when current run-time sessions restart a conversation without the need to start a new conversation.

Handling 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.

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.

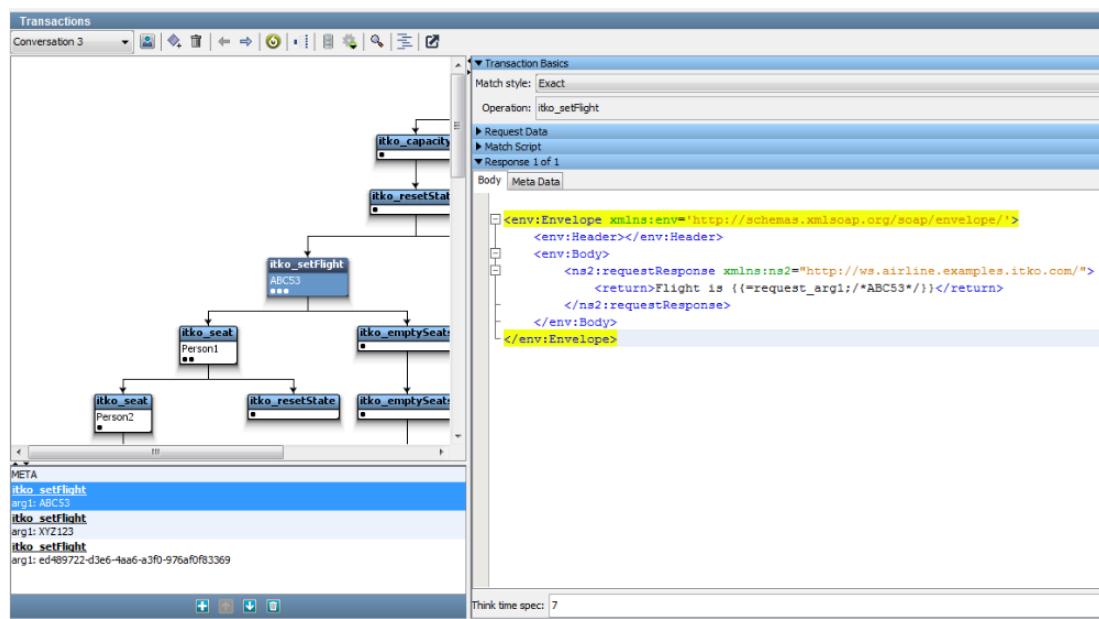
- [Magic Strings \(see page 683\)](#)
- [Magic Dates \(see page 686\)](#)

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.

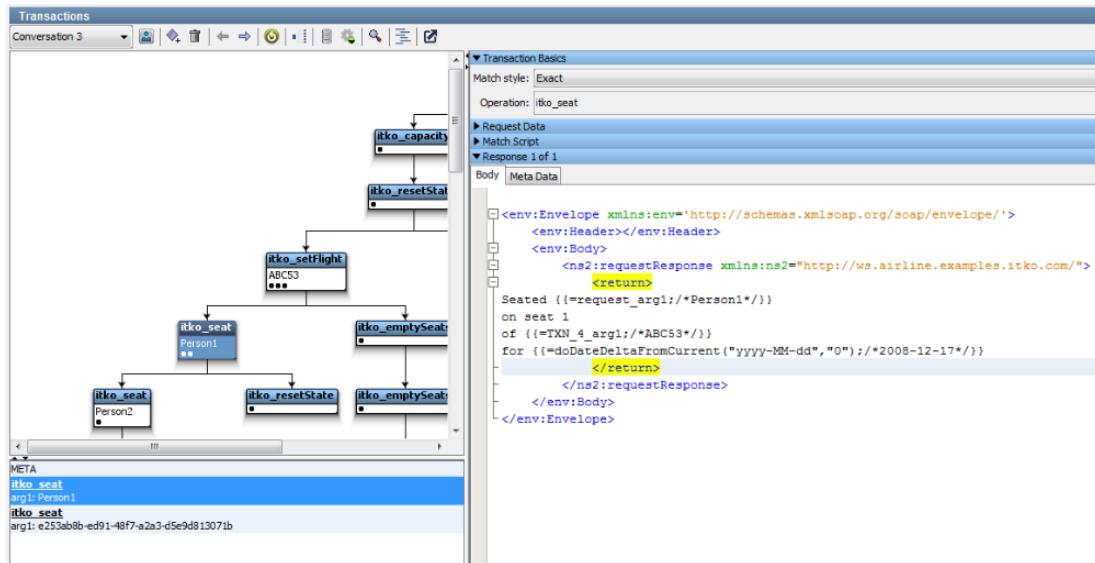
The practical significance is that if a future client requests flight ZZ99, the virtualized service responds with the correct flight number, ZZ99.

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 \(see page 657\)](#)

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 CAI for Java VSE uses. For example:

Request

```
< GetUserRequest>
<userId>lisa&lt;ko</userId>
<includeDetails>true</includeDetails>
</ GetUserRequest>
```

Response

```
< GetUserResponse>
<userId>lisa&lt;ko</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 \(see page 683\)](#), 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.

Understanding VSE Transactions

A *transaction*, as it applies to CA Service Virtualization, is a complete unit of work that a service performs. A transaction includes the request that the client sends to the service and the response that the service sends to the client.

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 in 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 response, completing the transaction.

A transaction typically has only one response (for example, in the case of an HTTP responder). However, the messaging protocol can return multiple responses from a single request.

Stateless and Conversational VSE Transactions

Contents

- [Stateless Transactions \(see page 687\)](#)
- [Conversational or Stateful Transactions \(see page 688\)](#)
- [Conversation Starters \(see page 688\)](#)

VSE transactions are classified as *stateless* or *conversational*.

Stateless Transactions

Stateless transactions include no logical relationships between transactions. For example, HTTP and SOAP are stateless protocols. A stateless transaction always has a static response, regardless of what calls were made previously.

In a stateless conversation, each service call is independent of the others. For example:

- What is the weather like in Dallas, TX? (Operation: weather; arg1-city)

- What is the weather like in Atlanta, GA? (Operation: weather; arg1-city)
- What will the weather be like in Dallas, TX tomorrow? (Operation: weather; arg1-city; arg2-date)

Conversational or Stateful Transactions

Conversational transactions (conversations) 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 always depends on the context that earlier conversations create.

For example, the following stateful conversation involves using an ATM:

1. Connect to the ATM. (Operation: logon)
2. Which account do you want? (arg1-checking)
3. What is my balance? (Operation: balance)
4. Response.
5. Withdraw an amount of money. (Operation: withdrawal)
6. How much? (arg2-amount)
7. What is my balance? (Operation: balance)
8. Response (different based on the amount that was withdrawn in the previous request).
9. End session (Operation: log out)

Conversation Starters

The first transaction in a conversation is the *conversation starter*. When VSE receives an incoming request, it reviews the conversation starters to determine whether the transaction means we are starting a new conversation.

Because all the transactions in a conversation are related to each other, 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.

DevTest supports the following types of conversations:

- **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**

VSE generates the token 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 \(see page 912\)](#).

Navigation Tolerance

In a conversation, VSE follows specific rules to determine how to find the next conversational transaction. The sets of rules, also known as *navigation tolerances*, are:

- **Close**

The transactions must go straight down the tree. The only candidates for the next conversational transaction are the children of the current one.

- **Wide**

The default. Wide tolerance allows navigation to the current transaction, the children of the current transaction, the siblings of the current transaction, and the immediate descendants of the sibling (or "nephews"). The order of precedence is as follows:

1. Children of a current transaction
2. Children of a sibling
3. Sibling of a current transaction

- **Loose**

The most permissive navigation tolerance. VSE first tries the **Close** and **Wide** tolerances, then adds the ability to match the parent of the current transaction, any of the siblings of the parent ("uncles"), the children of the siblings of the parent ("cousins"). If this match fails, navigation is permitted to any transaction in the second or third level of the tree. The order of precedence is as follows:

1. Children of a current transaction (close tolerance)
2. Children of a sibling (wide tolerance)
3. Sibling, or current transaction (wide tolerance)
4. The siblings of the parent transaction ("uncles") but not their children (not cousins)
5. The parent transaction or its siblings (parent or "uncles")
6. The children of the starter transaction for the current conversation (the immediate children of the root of the tree)
7. The starter transactions for all the conversations in the SI

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, then all the specific transactions are asked if they can respond to the request. If none of the specific transactions can respond to the request, then 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 (the specific transaction matches the given request) \{
            return the response from the specific transaction
        \}
    \}
    // No specific transaction found for the given request
    return the response from the meta transaction
\}
\}
```

Further, a physical Meta transaction carries a list of specific transactions and a list of child Meta transactions. The list of child Meta transactions allows Meta transactions to be structured as a decision tree.

Match Tolerance

Contents

- [Argument Match Operators \(see page 690\)](#)
- [Meta Transactions and Specific Responses \(see page 691\)](#)
- [How VSE Selects the Next Response \(see page 692\)](#)
- [How to Debug Match Failures \(see page 692\)](#)

Match tolerance defines how VSE decides whether a specific transaction matches the incoming transaction.

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 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, then the response that is specified for the Meta transaction is used.

How 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 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.

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.

Using the DevTest Portal with CA Service Virtualization

The DevTest Portal is a web-based application that is intended to become the primary user interface for DevTest Solutions. The Portal provides capabilities for some of the most-used workflows for DevTest products. Over time, CA will enhance the functionality of the DevTest Portal and eventually sunset the other interfaces in the LISA product line, including DevTest Workstation. This section provides detailed information about using the Portal to virtualize service behavior for supported features.

- For a quick summary of the functionality available in the Portal, see [DevTest Portal Functionality \(see page 48\)](#).
- [Using the Workstation and Console with CA Service Virtualization \(see page 758\)](#) provides detailed information about using the DevTest Workstation and DevTest Console to to virtualize service behavior for features that have not yet migrated to the Portal.

Opening the DevTest Portal

You open the DevTest Portal from a web browser.



Note: For information about the server components that must be running, see [Start the DevTest Processes or Services \(see page 115\)](#).

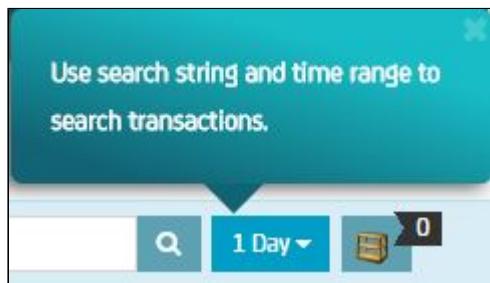
One possible error message indicates that the user cannot be authenticated and that you should make sure the registry service is running. If you receive this message and the registry service is running, the cause might be a slow network. Analyze your network for impaired performance.

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**.

Enabling and Disabling Tool Tips

The DevTest Portal provides tips for new users on how to perform key tasks. The following graphic displays tips for entering search criteria:



Screen shot of the help tip.

You can enable or disable the help tips.

Follow these steps:

1. Open the DevTest Portal.
2. Select **admin, Preferences**.
The Preferences dialog opens.
3. To enable help, select **Enable help for first time user**.
To disable help, clear **Enable help for first time user**.
4. Click **OK**.



More Information:

- [Virtualize Websites \(see page 694\)](#)
- [Editing Virtual Services \(see page 713\)](#)
- [Deploy a Virtual Service \(see page 735\)](#)
- [Video about Recording and Editing Virtual Services in CA Service Virtualization \(https://www.youtube.com/watch?v=0nx6EK0xtM&index=7&list=PLUo5-4tnpYJmxYRsnMkYSWPTdQ-vrKd8\)](#)

Virtualize Websites

The **Virtualize Website (HTTP/S)** window lets you record, configure, and save a virtual service.

To create a virtual service, enter **Recording Information** on the top pane of the CA Service Virtualization Recorder.

Follow these steps:

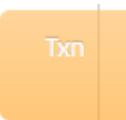
1. Enter the **Recording Name**. To display a list of previously saved recording configurations, click **List**. To clear a selected recording information from the recorder and start with a fresh recording, click **New Recording**.

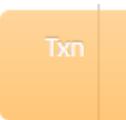
When you display a list of recordings, you can sort the list by name, id, or status. To delete a

recording, highlight it and click **Delete** .

2. Select the **VSE Server** from the drop-down list.

3. Enter a **Description** for this virtual service.



4. Click **Txn**  to display transactions as the virtual service is recording.



5. Click **Status**  to see details about the status of the recording.

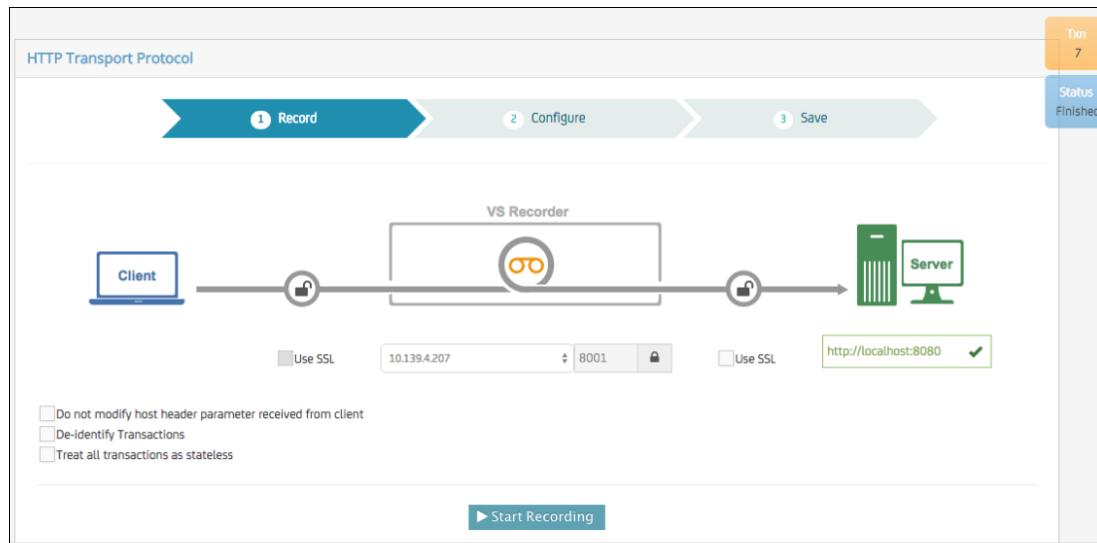


More Information:

- [Record a Website \(HTTP or HTTP/S\) \(see page 695\)](#)
- [Configure a Virtual Service \(see page 699\)](#)
- [Save a Virtual Service \(see page 700\)](#)

Record a Website (HTTP or HTTP/S)

To record a virtual service from an HTTP/S website, enter information about the HTTP transport protocol in the bottom pane of the **Virtualize Website (HTTP/S)** window.



Virtualize Website (HTTP/S) window

Follow these steps:

1. CA Service Virtualization has chosen a default host for the VS Recorder and displays that IP address. 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 of 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 target 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>



Note: As the target URL is typed into this field, VSE validates that the URL format. Once the field is fully typed, and you tab out or enter, VSE makes a call to the target server to ensure that it is reachable. An appropriate error message is shown.

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>**.



Note: As you enter information, watch the top of the window for error messages that highlight invalid entries.



Note: The graphic for the recording changes color as the components are validated. In the preceding graphic, both the VS Recorder and the server are colored, indicating that they have been validated.

The status of the recording changes from "Draft" to "Ready" when all the mandatory information required for recording is entered. The status shows on the blue **Status** button. To get more details for the recording, click **Status**.

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 \(see page 969\)](#). 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. In most cases, 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. Click **Next**.
 - 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**.



Note: If you start a recording at the recording panel, record some transactions, click **Next**, then click **Prev** to go back to the recording panel, you go back to the configuration steps.

11. To see transactions as they record, click **Txn**. To clear the transaction list, either during or after recording, click **Clear**.

12. To work with transactions as they record, click **Txn**. Select a transaction.

From the **Body** tab under Response data, 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 **Next**.

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

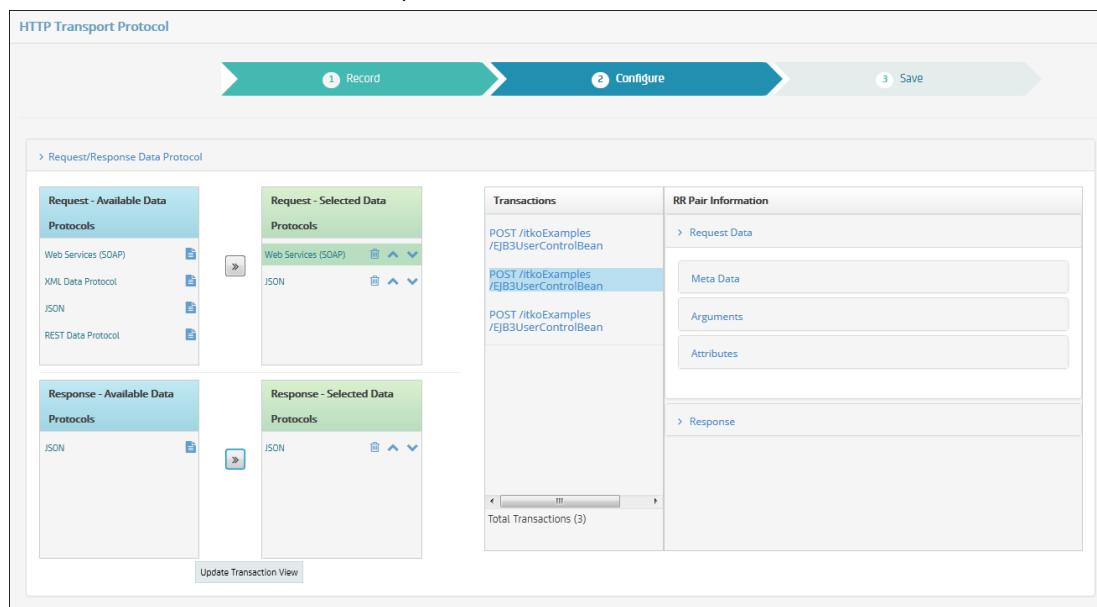
If you select **Use SSL to Client**, an SSL handshake occurs between the client application or test case client and the VSM. The keystore that is provided in the VSM Listen step is used as the Server certificate for one-way authentication. There is no SSL handshake between the client and the VSM. The 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.

Configure a Virtual Service

The **Configure** pane lets you accept, add, or delete data protocols to the recording you completed. You can also see the effect the data protocol has on the transactions.



The column on the left lists the available data protocols for the request and the response side. Data protocols that were auto-detected are highlighted in green and show in the Selected list. All data protocols available for the request and response are displayed.



To add a data protocol, highlight it and click **Move** to move the data protocol into the **Selected Data Protocols** column. You can move multiple data protocols into the selected column. To delete, move up, or move down the selection, use the icons at the top of the column.

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 \(see page 863\)](#).

The updated transactions, with parsed requests and responses, are retrieved from the VSE server and shown in the "preview" **Transactions** pane on the right of the **Configure** pane.



Note: The selected data protocols apply and are saved for the virtual service only when you click **Update Transaction View**. Changes you make in the Available and Selected lists are only UI changes.

From the **Body** tab under Response data, 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

Save a Virtual Service

The **Save** pane of the CA Service Virtualization Recorder displays the following options:

- **Save & Close**
- **Save & Deploy Virtual Service**
- **Save & Edit Virtual Service**

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

To display more options, click **YES** for **Advanced Mode** at the upper right corner of the window. Optionally, complete the following fields:

▪ VSM Style

Specifies whether to generate a VSM including the prepare steps.

Values:

- **More Flexible:** Includes prepare steps (leading to a five-step model).
- **More Efficient:** The prepare steps are absent (leading to a three-step model).

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.

Values:

- Close: The children of the current transaction are searched.
- Wide: A close search, including the current transaction plus siblings and nieces/nephews of the current transactions.
- Loose: A wide search, plus the parent and siblings of the current transaction, followed by the children of the starting transaction. If both fail to find a match, then the starting transactions for all conversations are checked, resulting in searching the full conversation.

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.

Values:

- Close: The children of the current transaction are searched.
- Wide: A close search, including the current transaction plus siblings and nieces/nephews of the current transactions.
- Loose: A wide search, plus the parent and siblings of the current transaction, followed by the children of the starting transaction. If both fail to find a match, then the starting transactions for all conversations are checked, resulting in searching the full conversation.

Default: Loose

Save and Close

Save & Close saves the virtual service and closes the recorder tab.

Save and Deploy Virtual Service

Save & Deploy Virtual Service saves the virtual service and deploys it immediately.

Complete the following fields:

▪ **Group Tag**

Specifies the name of the [virtual service group \(see page\)](#) 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.

VSE allocates a number of 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:

$$\text{Concurrent Capacity} = (\text{Desired transactions per second} / 1000) * \text{Average Think Time in ms} * (\text{Think Scale} / 100)$$

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.

$$(100 \text{ Transactions per second} / 1000) * 200\text{ms} * (100 / 100) = 20$$

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.

The screenshot shows the 'HTTP Transport Protocol' configuration screen. At the top, there's a progress bar with three steps: 1 Record, 2 Configure, and 3 Save. Step 3 is highlighted in blue. Below the progress bar, a message says 'You have three options to choose to Save and Finish.' with three radio button options: 'Save & Close', 'Save & Deploy Virtual Service', and 'Save & Edit Virtual Service'. A 'Select Project' dropdown is set to 'Previous Project'. There's also a checkbox for 'Allow duplicate specific transactions'. A large central panel is titled 'Options for Deploy Virtual Service' and contains three input fields: 'Group Tag' (empty), 'Concurrent capacity' (set to 1), and 'Think Time Scale (%)' (set to 100). At the bottom right of this panel is a green 'Save' button with a right-pointing arrow.

Save virtual service panel

Save and Edit

Save & Edit Virtual Service saves the virtual service and opens it in edit mode in a new editor tab.

Virtualize JDBC Traffic

You can use the DevTest Java Agent with a web-based recorder to virtualize JDBC-based database traffic. This feature is also known as *agent-based JDBC virtualization*.



Note: JDBC is a chatty protocol. For example, a few clicks in a user interface that retrieves data from a database can result in a huge number of JDBC API calls. This behavior can make it difficult to model properly. If you want to capture and virtualize JDBC database traffic, consider doing so at a higher layer.

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 11g JDBC driver to Oracle 11g database (ojdbc7.jar, ojdbc6-11.1.0.2.jar, and ojdbc5.jar)
 - Oracle 11g JDBC driver to Oracle 12c database (ojdbc7.jar, ojdbc6-11.1.0.2.jar, and ojdbc5.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)
- Teradata 14.10

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 \(see page 1017\)](#) 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 \(see page 1253\)](#). For information about deploying the virtual service, see [Deploy a Virtual Service \(see page 735\)](#).

Capture the JDBC Traffic

The **Virtualize JDBC** window in the 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 the DevTest Portal.
2. Select **Create, Virtualize JDBC** in the left navigation menu.
3. Click **Start recording**.
4. 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.
5. (Optional) [Examine the captured queries \(see page 705\)](#).
6. (Optional) [Preview the conversations \(see page 706\)](#).
7. When you have all the queries and conversations that you are interested in, click **Stop recording**.
8. Click **Create VS Artifact**.
9. Select the project where the virtual service will be created.
10. 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:

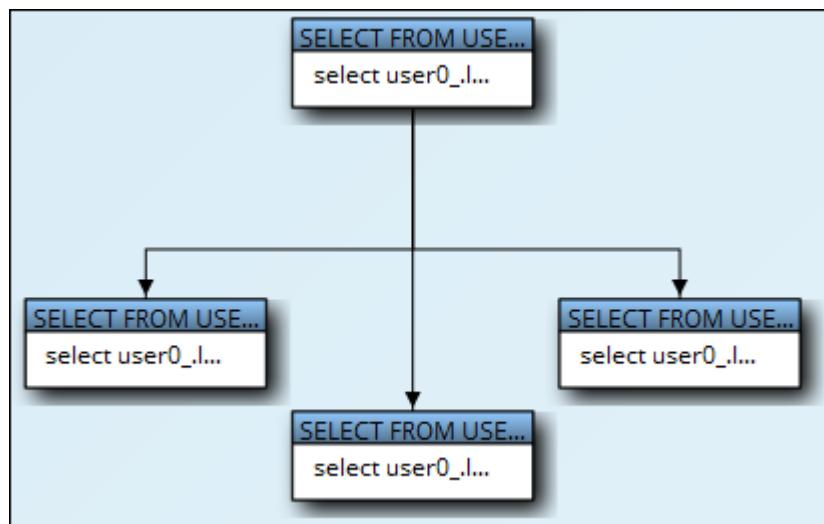
1. To view an instance of a query, select the query in the **Results** pane.
The **SQL Query** pane and the **Transactions** pane are populated.
2. To filter the queries, go to the **Databases** section and select the check box for any database that you want to include.

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.

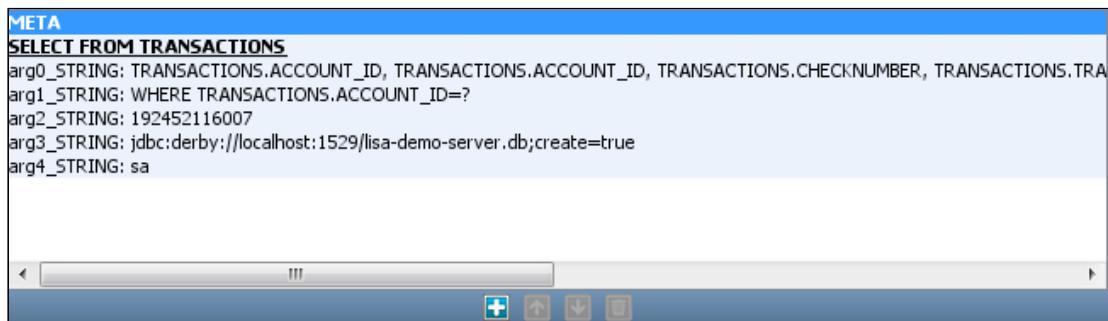
Each transaction in the service image has an operation and a series of arguments.

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 final two arguments are always the connection URL and the database user.

The following graphic shows a transaction. 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.

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, 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.

▼ Response 1 of 1

Body Meta Data

MultiDataSet Document

▼ Results 1 of 1

Update count: 3

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:

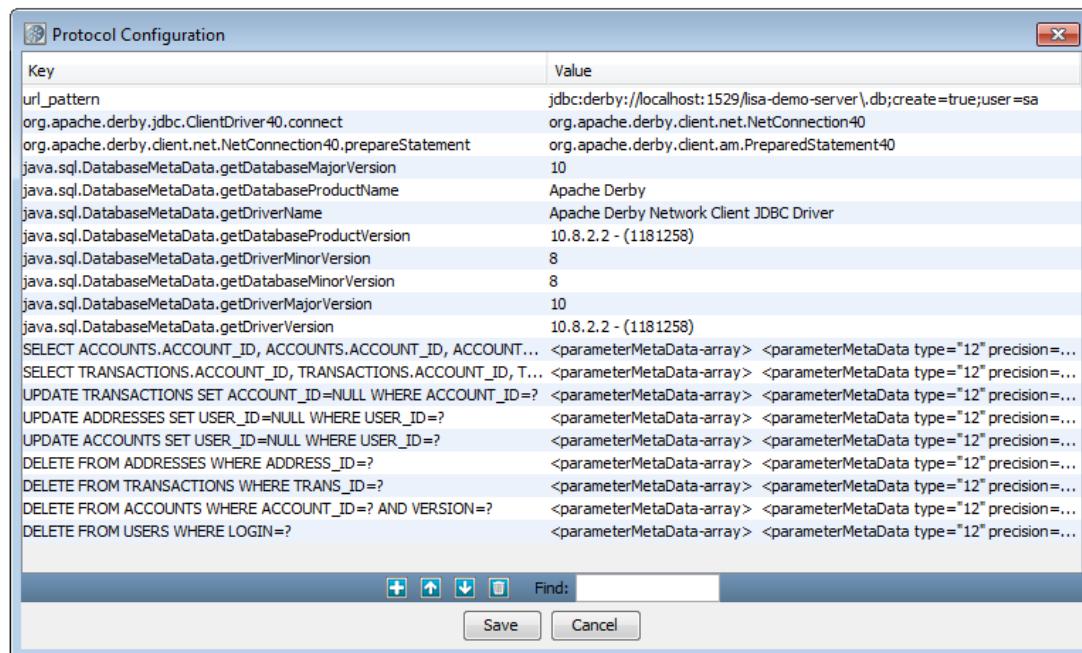
- **Virtual Java Listener**
- **Virtual Java Responder**

To view the protocol configuration information, open the listener step 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 \(see page 1013\)](#) of the DevTest Portal. The property appears in the **Settings** tab.

Synthetic Parents

Certain scenarios can cause CAI 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 the 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 CAI 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.

Database Virtualization Troubleshooting

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 \(see page 1281\)](#) to add the method. For example:

```
<intercept class="com.mycompany.JdbcTestDev" delay="false" method="hitDb"
signature="*" />
```

4. Restart the agent.

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 the VSE Console.

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 **xdb.jar** file and can place it in the classpath of the application.

Virtualize Using Request-Response Pairs

The DevTest Portal lets you quickly create and deploy a virtual service from request-response pairs.

In previous releases, this feature was referred to as VSEasy.

The following modes are supported:

- HTTP protocol
- From VRS file

You can use this feature to create stateless virtual services for HTTP with SOAP, JSON, XML, or HTML payloads.

To configure non-HTTP virtual services, use VRS files.

When using a VRS file, you can chain data protocols. For example, you can use the Generic XML Payload Parser data protocol or the Request Data Manager data protocol to modify your service image.

The VRS file cannot reference external data. For example, the VRS file cannot reference copybooks and certificates for signing.

In the DevTest Portal, you must assign the virtual service to a virtual service group. When you select a VSE server, the available groups appear in the **Service Group** list. The default group is named **vseasy**. To change the name of the default group, update the **lisa.vseeasy.default.group.tag** property in the **lisa.properties** file.

When you click **Create and Deploy**, the DevTest Portal indicates the location of the virtual service. The port number is automatically assigned.



Note: If you deploy a virtual service that has the same name as a virtual service that is already deployed in the VSE server, the existing virtual service is overwritten.

Follow these steps:

1. Go to the home page of the DevTest Portal.
2. Select **Create, Virtualize using RR pairs** in the left navigation menu.
3. Select the mode that you want to use from the **Protocols** list.
4. Follow the instructions in the **Help** panel. The **Help** panel also includes the following sample files:
 - HTTP request/response pair
 - JMS request/response pairs and VRS file



Note: We do not support editing the VSI after importing from Request-Response pairs.

Editing Virtual Services

The [DevTest Portal \(see page 196\)](#) lets you open a virtual service and make changes to various components of the virtual service.

You can open a virtual service from the **Virtual Services** window of the DevTest Portal.

Follow these steps:

1. Go to the home page of the 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

Contents

- [Stateless Signatures \(see page 714\)](#)
- [Request Data Arguments and Response Data \(see page 715\)](#)

To view the stateless transactions, 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

When VSE tries to find a match for an inbound request in the stateless transactions, 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 an action to be performed. An example of an operation is **depositMoney**.

In a virtual service, an argument is a name/value pair that can be included with an operation. An example of an argument is the name **amount** and the value **100.00**.



Note: VSE does not consider the argument values at this stage of the matching process.

Each signature also has a unique identifier. The identifier 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 \(see page 720\)](#).

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**.

The screenshot shows a list titled "Stateless Signatures (7)". The list contains the following items:

- 153 listUsers
- 159 addUserObject** (This item is highlighted with a blue background)
- 165 addUserObject
- 173 deleteToken
- 179 GET /itkoExamples/TokenBean
- 185 GET /itkoExamples/EJB3UserControlBean
- 191 GET /itkoExamples/EJB3AccountControlBean

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 VSE finds a matching signature, it then searches for a *specific transaction* that matches the inbound request.

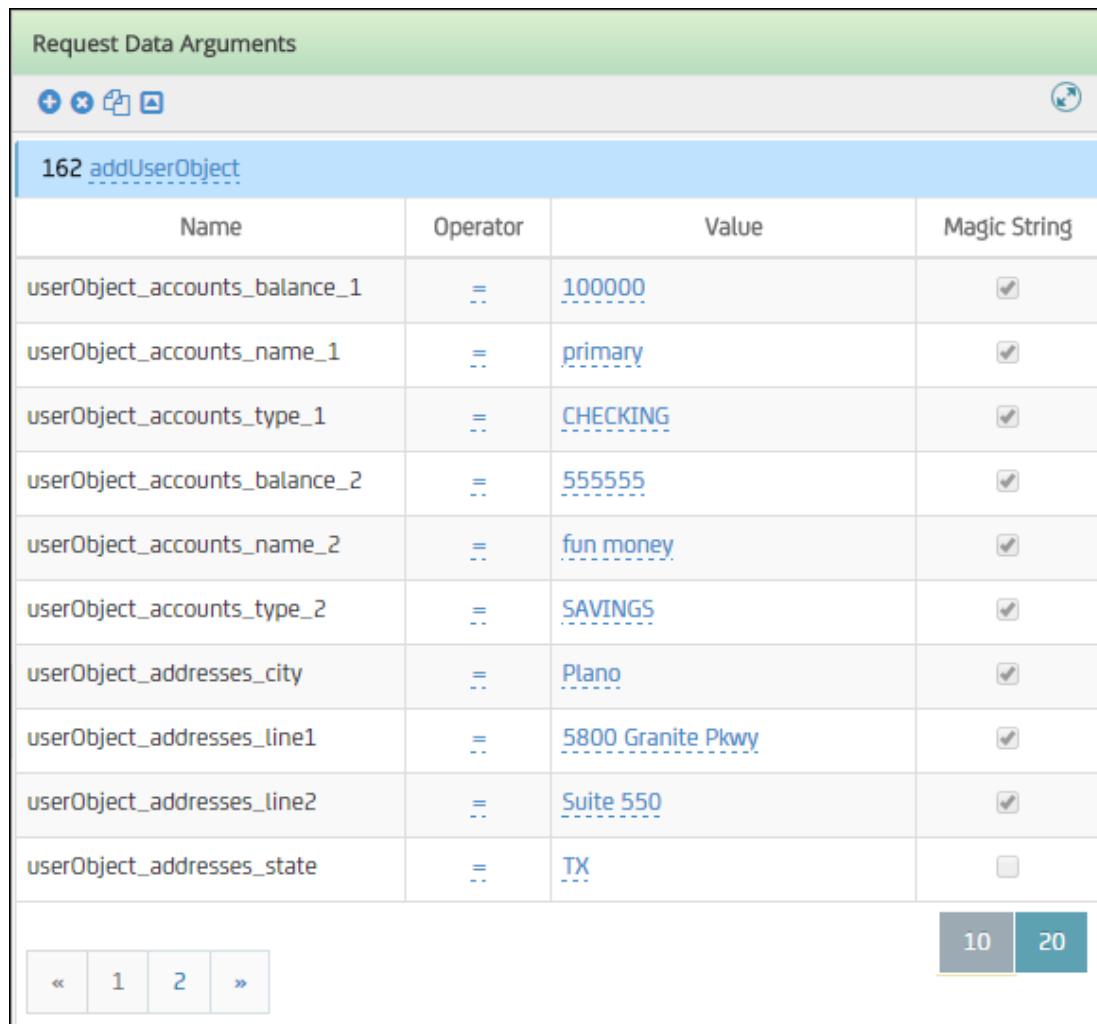
A specific transaction has the following components:

- 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.



The screenshot shows the 'Request Data Arguments' pane with the title '162 addUserObject'. The table has four columns: Name, Operator, Value, and Magic String. The 'Value' column contains several entries with underlines, indicating they are variables or placeholders. The 'Magic String' column contains checkboxes, many of which are checked.

Name	Operator	Value	Magic String
userObject_accounts_balance_1	=	100000	<input checked="" type="checkbox"/>
userObject_accounts_name_1	=	primary	<input checked="" type="checkbox"/>
userObject_accounts_type_1	=	CHECKING	<input checked="" type="checkbox"/>
userObject_accounts_balance_2	=	555555	<input checked="" type="checkbox"/>
userObject_accounts_name_2	=	fun money	<input checked="" type="checkbox"/>
userObject_accounts_type_2	=	SAVINGS	<input checked="" type="checkbox"/>
userObject_addresses_city	=	Plano	<input checked="" type="checkbox"/>
userObject_addresses_line1	=	5800 Granite Pkwy	<input checked="" type="checkbox"/>
userObject_addresses_line2	=	Suite 550	<input checked="" type="checkbox"/>
userObject_addresses_state	=	TX	<input type="checkbox"/>

Below the table are navigation buttons: «, 1, 2, » and page numbers 10 and 20.

Screen capture of specific transaction.

If 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>nU4eI71bcnBGqe00t9tXvY1u5oQ=</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.

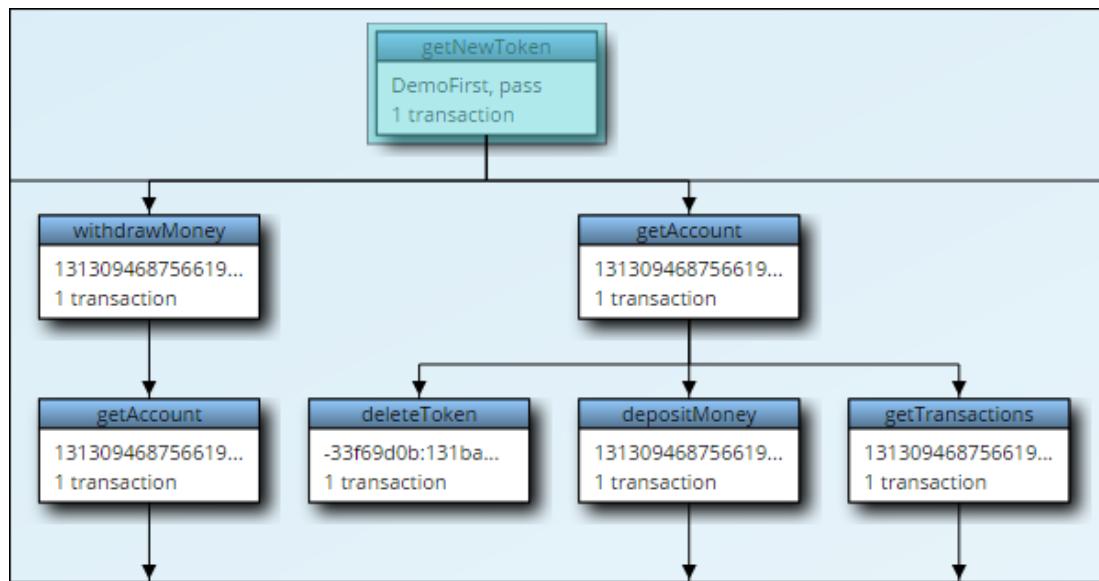
If VSE does not find a matching specific transaction, it sends the response that is associated with the *default transaction*.

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, 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 \(see page 689\)](#).

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 \(see page 715\)](#).

Unknown Responses

When 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 **Service Image Properties** 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.

The screenshot shows a software interface titled "Response Data". At the top, there are navigation buttons (back, forward, search, settings, and refresh) and a status bar indicating "1 / 1". The main area displays an HTML document with line numbers from 1 to 15. The content is as follows:

```

1 <html>
2
3 <head>
4     <title>404 Not Found</title>
5 </head>
6
7 <body>
8     <h1>Not Found</h1>
9     <p>The requested URL was not found on this server.</p>
10    <hr/>
11    <p><i>The LISA VSE service could not match your request to a recorded request.&nbsp</i></p>
12    ; Consider expanding your service image.</i></p>
13    <br/><font size="-2">Produced by a LISA virtualized web server.</font>
14 </body>
15 </html>

```

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
- Service image description
- Signature name
- Signature note
- Specific name
- Unknown response

The following graphic shows the search interface. The results appear in a table. The results are ordered from most significant to least significant.

The screenshot shows a search interface with the following details:

Type	Sub Type	Operation	Snippet
Stateless	Response Body	getUser	pe>SAVINGS</type></accounts><accounts><balance> 100000 .00</balance><id>1433785279732958</id><name>Pri
Stateless	Response Body	getUser	pe>SAVINGS</type></accounts><accounts><balance> 100000 .00</balance><id>1433785278835443</id><name>Pri
Stateless	Response Body	addUserObject	<balance>{#=request_userObject_accounts_balance_2; /* 100000 */}}.00</balance><id>1433785278835443</id>

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 service image description and unknown response.

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 you receive too many or too few results, consider adjusting the search options and filters.

Follow these steps:

1. Open a virtual service.
2. (Optional) Configure the search options:
 - **Whole Word**
Searches must match the entire word.
 - **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 can 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.

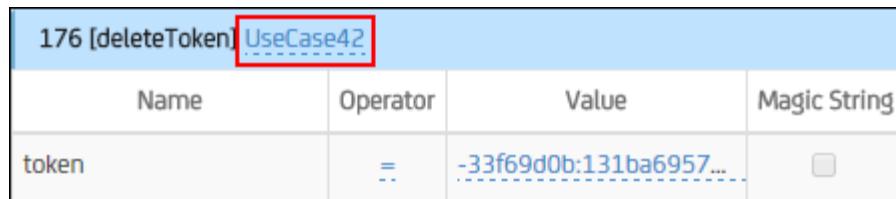
Follow these steps:

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 **Save**.

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.



A screenshot of a table interface. The top row contains the text "176 [deleteToken] UseCase42". Below this is a table with four columns: "Name", "Operator", "Value", and "Magic String". A single row is present with the values "token", "=", "-33f69d0b:131ba6957...", and an empty checkbox. The cell containing "UseCase42" is highlighted with a red rectangular border.

176 [deleteToken]	UseCase42		
Name	Operator	Value	Magic String
token	=	-33f69d0b:131ba6957...	<input type="checkbox"/>

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.

Updating a Virtual Service Manually

You can update a virtual service manually from the DevTest Portal.

The following reasons are some of the reasons to update a virtual service:

- During the recording of the virtual service, you forgot 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 \(see page 759\)](#) the virtual service with another virtual service
- Run the virtual service with the [execution mode \(see page 978\)](#) set to **Image Validation**
- Rerecord the entire virtual service

Add, Change, and Delete Signatures

The 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 \(see page 725\)](#) can also result in a new signature.

When 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 \(see page 726\)](#) 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.

Name	Date Pattern	Case Sensitive	Is Numeric
token	Not set	<input checked="" type="checkbox"/>	<input type="checkbox"/>

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 Down** 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 \(see page 725\)](#) 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

A request/response pair consists of one request file and one or more response files.

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.

The request/response pair must be in text or XML format.

The following list contains an example of the file names in a request/response pair:

- **depositMoney-req.xml**
- **depositMoney-rsp1.xml**
- **depositMoney-rsp2.xml**
- **depositMoney-rsp3.xml**

The request file name must include a prefix followed by the string **-req**.

The response file name must include the same prefix followed by the string **-rsp**. As the preceding example shows, you can provide multiple response files by adding a number after the string **-rsp**. Multiple response files are applicable to messaging scenarios.

You can specify request and response metadata by including one or more sidecar files. For more information, see [Sidecar Files with Request/Response Pairs \(see page 772\)](#).



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		Specific Transaction		
Name	Value	Name	Operator	Value
AccountType	Checking	AccountType	=	Checking
AccountBalance	5000	AccountBalance	>	100
AccountName	Name2	AccountName	Anything	Name1

Disable and Enable Magic Strings

Each argument in a specific transaction can be classified as a [magic string \(see page 683\)](#).

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 [Request Data Editor \(see page 912\)](#).

Editing Request Attributes and Metadata

Edit the Request Attributes

In a virtual service, an *attribute* is a name/value pair that contains information about the request body.

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.

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 contain information that is not part of the request or response body.

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.

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.

Updating Responses

You can update the response that is associated with a specific transaction. You can also 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 service image 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:

- **t:** milliseconds
- **s:** seconds
- **m:** minutes
- **h:** hours

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, 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 \(see page 735\)](#), 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 service image.

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 contain information that is not part of the request or response body.

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. The 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. The 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.

The 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 the 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.

Virtual Service URLs

When you open a virtual service in the DevTest Portal, the URL specifies the path to the virtual service component that is displayed.

The following example shows the URL format of a specific transaction in the **Stateless Transactions** view.

```
http://localhost:1507/devtest/#/main/serviceimageeditor/0/bmS18CcJu-b74PB4tcz
/bmS18CcJu/b74PB4tcz/kioskV6/0?signatureId=153&specificId=156&view=stateless
```

You can bookmark a given URL. You can later open the virtual service to the same component by entering the URL in a browser.

Find and Replace

This release includes a [preview \(see page 204\)](#) of the find and replace feature.

The **Filters** button lets you control which parts of the virtual service are searched. For example, you can limit the search to the request arguments or response think times.

Follow these steps:



1. Click **Replace**.

2. (Optional) Configure the search options:

- **Whole Word**

Searches must match the entire word.

- **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 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 the first result, click **Replace**.
9. To replace the text in all of the results, click **Replace All**.

Data-Driven Virtual Services

Contents

- [Data Set Prerequisites \(see page 732\)](#)
- [Add a Blank Data-Driven Signature \(see page 733\)](#)
- [Map Request Arguments to Data Set Columns \(see page 733\)](#)
- [Map Response to Data Set Columns \(see page 734\)](#)
- [Manage Data Sets \(see page 735\)](#)

This release includes a [preview \(see page 204\)](#) of data-driven virtual services.

This feature lets you maintain the data for a signature definition in an external data set.

For example, the data might be managed by a business analyst who does not have access to the DevTest Portal.



Note: Playback is not supported in this preview.

Data Set Prerequisites

The following types of data sets are supported:

- Microsoft Excel spreadsheet
- Comma-separated value (.csv) file

An Excel file must have the .xlsx extension. Also, 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.

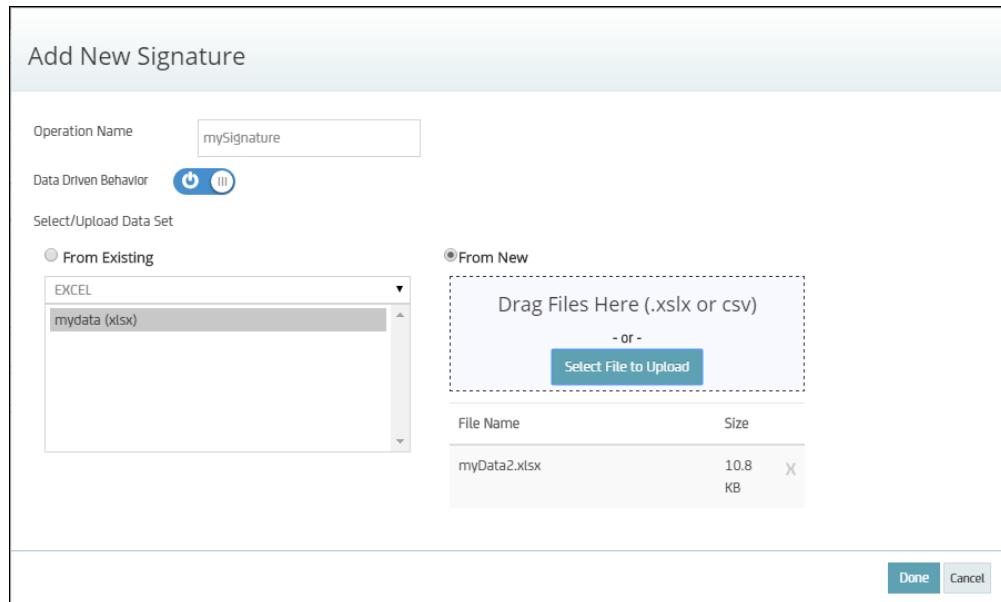
	A	B	C	D	E
1	first	last	city	state	response
2	Bruce	Adams	Los Angeles	CA	<html>Hello</html>
3	Cat	Taylor	Chicago	IL	<html>Hi</html>
4	Heather	Marley	New York	NY	<html>Hey</html>
5	Joe	Smith	Boston	MA	<html>Greetings</html>

Screen capture of sample data set.

Add a Blank Data-Driven Signature

When you [add a signature manually \(see page 722\)](#), you can indicate that the signature is data driven. You then select the data set. The resulting signature does not have any request arguments.

The following graphic shows the **Add New Signature** dialog.



Screen capture of Add New Signature dialog.

Follow these steps:

1. Ensure that the advanced mode is displayed.
2. In the **Stateless Signatures** pane, click the **Add** icon and select **Add New Signature**.
3. Enter the operation name.
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**.
A blank signature is added.

Map Request Arguments to Data Set Columns

Now that you have a blank signature, the next step is to create one or more request arguments manually.

Each argument contains a mapping from the argument name to a column name in the data set.

The following graphic shows a mapping example. This example uses the sample data set that is shown earlier on this page.

Request Data: Arguments Attributes Metadata	
Name	Mapping
first	first
last	last
city	city
state	state

Screen capture of mapping example.

You cannot specify a [comparison operator](#) (see page 726). The only supported operator is the equal sign (=).



Note: The icon that you click in step 1 has the incorrect tooltip **Add Attribute**. The tooltip will be changed to **Add Argument** in a future release.

Follow these steps:

1. Click the **Add Attribute** icon in the **Request Data** pane.
2. To change the default argument name, click the value in the **Name** column and enter the new name.
3. To change the default 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 as needed. You can also delete rows.

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 a column named **first**:

```
<html>
Hello, {{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 name and contents of a data set. You can also download a data set.

Follow these steps:

1. Select the **Data Sets** link.
2. To update a data set:
 - a. Click the **Edit** icon.
 - b. (Optional) Click the name and make your changes, then click  to save.
 - c. Drag or select the updated backing file, and click **Update**.
3. To download a data set:
 - a. Click the **Options** icon and select **Download Data Set**.

Deploy a Virtual Service

You can deploy a virtual service from the DevTest Portal.

Follow these steps:

1. Go to the home page of the 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 then 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 VSM. Capacity here indicates how many threads there are to service requests for this service model.

VSE allocates a number of 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 for the duration of 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:

$$\text{Concurrent Capacity} = (\text{Desired transactions per second} / 1000) * \text{Average Think Time in ms} * (\text{Think Scale} / 100)$$

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.

$$(100 \text{ Transactions per second} / 1000) * 200\text{ms} * (100 / 100) = 20$$

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.

REST Data Protocol Handler

REST is a way of calling web services using the HTTP protocol.

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, VSE uses the rules to virtualize HTTP requests that invoke the same operations.

If VSE does not automatically select the REST data protocol, you can select it manually in the data protocols page of the **Virtual Service Image Recorder** (in DevTest Workstation) or the Configure page in the DevTest Portal Virtualize Website (HTTP/S) window.

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 required. 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

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 (http://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 (http://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@itko.com (http://itko.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 DevTest Portal Virtualize Website (HTTP/S) window include a page where you can review and modify the rules that the **REST** data protocol created.

The upper pane (**URI Rules**) displays the rules. When you select a rule, the lower pane (**Sample of 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.

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, click **Properties**.

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.

Running Live Requests in DevTest Portal



Note: The running live requests capability is a preview feature in this release. See the [Enable and Disable Preview Features \(see page 204\)](#) for more information about how to enable and disable the preview features.

Run live requests against VSE after you deploy the virtual service. If possible, take the live service down and configure the gateway/proxy settings so the client communicates with VSE.

For more information, see:

[View Virtual Service Environments in DevTest Portal \(see page 741\)](#)

[Manage and Monitor Virtual Services in DevTest Portal \(see page 743\)](#)

View Virtual Service Environments in DevTest Portal



Note: The Virtual Service Environments feature is in a preview phase of this release. See [Enable and Disable Preview Features \(see page 204\)](#) for more information about enabling and disabling the preview features.

From the **VSE** window, you can view all the deployed virtual services for a virtual service environment.

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 enable or disable the automatic refresh by selecting or clearing the **Auto Refresh** checkbox. The automatic refresh runs every four seconds.

This window contains the following fields:

- **Name**

Identifies the currently deployed virtual service model. The Name field is searchable by entering criteria in the search box located in the column heading.

- **Resource / Type**

Identifies the port and the type or protocol of the service. The Resource/Type field is searchable by entering criteria in the search box located in the column heading.

- **Status**

Identifies the current state of the virtual service.

- **Up-Time**

Indicates how much time has elapsed since the service was started.

- **Txn Count**

Identifies the number of transactions that were recorded after the service started.

- **Execution Mode**

Displays the [execution mode \(see page 746\)](#) of the virtual service.

- **Group**

Displays the virtual service group of the virtual service. The Group field is searchable by entering criteria in the search box located in the column heading.

- **Errors:**

Displays a red dot to indicate that errors occurred while running the service.

The **Details** panel at the bottom of the window displays details about the service.

This panel contains the following fields:

- **Config Name**

Identifies the name of the currently deployed virtual service model.

- **Execution Mode**

Displays the execution mode for this virtual service. See [Specify How the Selected Model Should Behave \(see page 746\)](#).

- **Last Start**

Displays the date and time this service was last started.

- **Transaction Count**

Displays the number of transactions that VSE recorded after the service started.

- **Current txn/s**

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.

- **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 after you enter the group tag, then click **Update** to update the field.

To delete a group tag from a virtual service, click the **X** next to the group tag name, then click **Update**.

- **Auto-Restart**

Specifies whether to use the auto-restart option for this service. To change this value while the service is running, click this field.

- **Last End**

Displays the date and time this service stopped.

- **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.

To download the backing archive for the virtual service, click the name of the virtual service in the VSE Console. The system prompts you to save the MAR file that is associated with this virtual service.

Manage and Monitor Virtual Service Environments



Note: The managing and monitoring of virtual service environments feature is in a preview phase for this release. See [Enable and Disable Preview Features \(see page 204\)](#) for more information about enabling and disabling the preview features.

To manage and monitor deployed service images, use the **VSE** window. From this window you can do the following actions:

- search for virtual services
- start and stop virtual services
- download a MAR file
- view session and tracking information
- delete a virtual service

Follow these steps:

1. From the navigation menu, select **Monitor, Virtual Service Environments** and select a virtual service environment .
2. Verify that the virtual service is deployed (status is Running).
3. Verify that the service received the requests by viewing the transaction count (**Txn count**)

Search for Virtual Services

From the **VSE** window, you can search for virtual services by entering criteria in the **Search** field located at the top, right of the window. You can search for virtual services by groups and set refinement criteria. Click the filter button at the top, right of the table listing of virtual services to filter by:

- Name
- Resource/Type
- Status
- Txn Count
- Up-time
- Model Behavior
- Group
- Errors

Start a Virtual Service

You can start one or multiple virtual services at once from the **VSE** window.

Follow these steps:

1. Select the checkbox for each virtual service to start. To select all the virtual services, select the checkbox located at the top of the checkbox column.

2. Click **Options**  from the **Actions** column and select **Start**.
A confirmation window displays. The status in the table changes to **Running**.

Stop a Virtual Service

You can stop one or multiple virtual services at once.

Follow these steps:

1. Select the checkbox for each virtual service to stop. To select all the virtual services, select the checkbox located at the top of the checkbox column.

2. Click **Options**  from the **Actions** column and select **Stop**.
A confirmation window displays. The status in the table changes to **Offline**.

Deploy a Virtual Service Model

You can deploy a virtual service from a model archive (MAR).

1. Select the checkbox for each virtual service to start. To select all the virtual services, select the checkbox located at the top of the checkbox column.

2. Click **Options**  from the **Actions** column and select **Download MAR**.

3. Enter the name of a model archive (MAR) to upload.
The MAR must contain a virtual service. When you click **Deploy/Redeploy**, the service loads into the VSE window and the service is available to run.



Note: If a deployed VS model shows no transactions, then the client is not configured properly. Reconfigure the client to reference the virtual model 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.

Inspect Virtual Services

To open the inspection view, click the **Options**  in the **Actions** column and select **Inspection View**. The Inspector dialog displays information that is tailored for virtual services. From the dialog you can view the [Matching](#) (see page 746) and [Request Event Details](#) (see page 746) .

Delete a Virtual Service

You can delete one or multiple virtual services at once.

Follow these steps:

1. Select the checkbox for each virtual service to delete. To select all the virtual services, select the checkbox located at the top of the checkbox column.

2. Click **Options**  from the **Actions** column and select **Delete**.
A confirmation window displays. The virtual service is removed.

VSE Window Matching Tab in DevTest Portal



Note: The matching view is a preview feature in this release. See [Enable and Disable Preview Features \(see page 204\)](#) for more information about enabling and disabling the preview features.

The **matching view** lists recent requests that the virtual services processed. When you select a request, the dialog shows a description of how the request was (or was not) matched. The same information is recorded in the **vse_matches.log file**. If any events were associated with the selected request, they display at the right side of the panel.

VSE Window Request Events Details Tab in DevTest Portal



Note: The Request Events Details tab is a preview feature in this release. See [Enable and Disable Preview Features \(see page 204\)](#) for more information about enabling and disabling the preview features.

The **Request Event Details** tab 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 ITR), 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 for the Selected Virtual Service**

This option lets you see the session tracking information for the virtual service. For more information, see [Session Viewing and Model Healing \(see page 749\)](#).

- **Redeploy the Selected Virtual Service to the Environment**

If you edit the service image or the VSM, save the changes and redeploy the modified service image in the VSE Console by clicking **Deploy/Redeploy**.

- **Specify How the Selected Model Should Behave**

This option lets you set an execution mode for the virtual service. The number of execution modes available for a virtual service depends on the type of test step. For example, if a model does not have a live invocation step, it does not support the **Learning** or **Live Invocation** options.

The available execution modes are:

- **Most Efficient**

The fastest mode; it does not execute the routing or tracking steps. This mode also restricts generated event tracking.

- **Stand In**

Stand In mode 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. You can only enable Stand In mode for a virtual service with a Live Invocation step. Stand In mode does not do any special tracking. It simply allows for a virtual service to fall back on the live service.

- **Transaction Tracking**

This mode 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. Transaction Tracking mode does not show live system responses; it only shows the response from the service image.

- **Live System**

This mode 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.

- **Failover**

Failover mode 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. This mode is the opposite of Stand In mode. You can only enable Failover mode for a virtual service with a Live Invocation step. Failover mode uses the service image to determine a response if the Live Invocation step actually fails (as might happen if the live system were not available).

- **Learning**

Learning 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. When a virtual service is running in Learning mode and it has acquired new knowledge, there is an icon to the left of the virtual service name in the VSE Console. The icon remains visible until you redeploy the virtual service.

- **Image Validation**

This mode 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 VSE provides and a

corresponding live system and, where differences exist, patches or heals the VSE service image to keep in sync with the live system. This mode is also known as "live healing mode." Image Validation is the least efficient of all the modes.

- **Dynamic**

This mode 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.

- **Reset the Transaction and Error Counts for the Selected Virtual Service**

Sets both the transaction count and error count to zero for the selected virtual service.

- **Stop the selected virtual service**

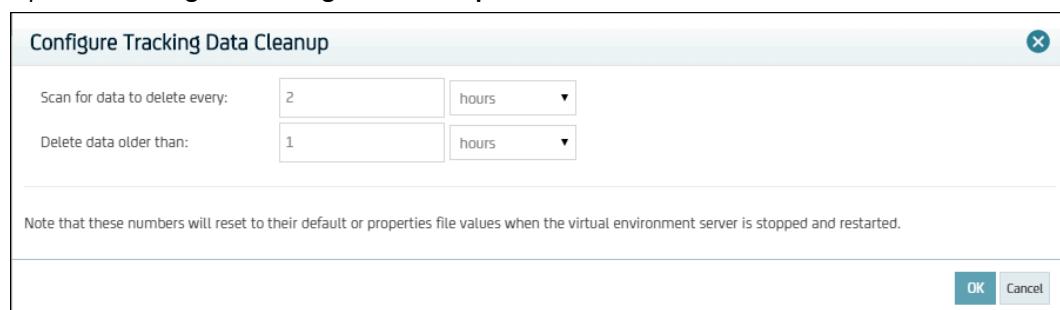
To stop the selected virtual service, click this button. The application displays a confirmation message before the service stops.

- **Remove the selected virtual service from the environment**

To remove the selected virtual service from the console display, click this button. The application displays a confirmation message before it removes the service.

- **Configure tracking data cleanup**

Opens the **Configure Tracking Data Cleanup** window.



Screen capture of Configure Tracking Data Cleanup window

Enter data cleanup values here that 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.

- **Scan for data to delete every**

Defines the interval (in milliseconds, seconds, minutes, hours, days, or weeks) after which to scan for tracking data to delete.

- **Delete data older than**

Defines the age of data (in milliseconds, seconds, minutes, hours, days, or weeks) to delete.

- **Shut down the entire virtual service environment**

To shut down the complete VSE environment, click this button. If you reply affirmatively to the shutdown confirmation message, your VSE shuts down and the corresponding window closes.

Session Viewing and Model Healing in DevTest Portal



Note: The session viewing and model healing feature is in the preview phase for this release. See [Enable and Disable Preview Features \(see page 204\)](#) for more information about enabling and disabling the preview features.

Session viewing lets a VSE user actually see the behavior of current (or recent) sessions on a VSE server. The user 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 only available for virtual services that run with an Execution Mode of Transaction Tracking or Image Validation. Model healing is only available for virtual services that run in Image Validation mode. For more information about execution modes, see [Specify How the Selected Model Should Behave \(see page 746\)](#).

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 from either the VSE dashboard or while editing a virtual service model.

To view the session tracking for a selected virtual service from the VSE window, select the Options icon in the **Actions** column a service and select **View Session/Tracking information**.

When you 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.

The session panel is divided into two panes: **Session Information** and **Response Information**.

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 an icon to indicate the current session status.

A gray ball indicates a session that was recorded in Transaction Tracking mode, or a session that has been healed using model healing.

A red ball indicates a session that was recorded in Image Validation mode and is a candidate for healing.

A green ball indicates a session that was recorded in Image Validation mode where the live and VSE responses match.

- **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 compare request/response between the VSE system and the live system.

In the **Response Information** pane, the following icons represent transactions:

- Green ball: Indicates a signature match on a meta transaction.
- Yellow ball: Indicates a signature match on a meta transaction; the image navigation is successful, but the response body differs between VSE and the live system.
- Red ball: Indicates a conversational transaction that diverged between the live system and 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.

Create Copybook Bundles

Contents

- [Access the Copybook Bundle Page \(see page 751\)](#)
- [Create a Copybook Bundle \(see page 751\)](#)
- [Import a Copybook \(see page 753\)](#)

- [Create a Copybook \(see page 753\)](#)
- [Import a Mapping File \(see page 754\)](#)
- [Create a Mapping File \(see page 754\)](#)
- [Find Your Saved Copybook Bundle \(see page 758\)](#)

The Copybook Bundle page makes it easier to create, import, and manage your copybooks and mapping files. This page provides the following advantages:

- 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 \(see page 867\)](#) and [How to Use the Copybook Data Protocol \(see page 867\)](#).

Access the Copybook Bundle Page

Follow these steps:

1. From the 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 the 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 \(see page 753\)](#) or [create \(see page 753\)](#) the copybooks you want to include in your copybook bundle.

4. [Import \(see page 754\)](#) or [create \(see page 754\)](#) the mapping file for your copybook bundle.

5. Click **Comment**  to add comments or notes to the 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 Codpage 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 the 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 the copybook parsed successfully.
 - A red circle with an X indicates there were errors in the parsing.
 - A black dash indicates the copybook has not been parsed.
6. Make any desired changes or corrections to the copybook.
7. Import any additional copybooks you want to include in this bundle.
8. [Import \(see page 754\)](#) or [create \(see page \)](#) 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 the 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 \(see page 754\)](#).



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 the 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, but 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 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**).

▪ **Value**

The behavior of this attribute (and whether it is required) depends as follows on the Match Type:

- 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.

5. 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** 
 - Click **Add**  to add additional sections.

6. 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.

7. The Copybook Configuration Elements panel let you define additional 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 will be 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 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**

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**

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.

8. Repeat this process for additional 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.

9. 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 \(see page 867\)](#) and [How to Use the Copybook Data Protocol \(see page 867\)](#).

Using the Workstation and Console with CA Service Virtualization

This section provides detailed information about using the DevTest Workstation and DevTest Console to virtualize service behavior for features that have not yet migrated to the Portal.

The DevTest Portal is a web-based application that is intended to become the primary user interface for DevTest Solutions. The Portal provides capabilities for some of the most-used workflows for DevTest products. Over time, CA will enhance the functionality of the DevTest Portal and eventually sunset the other interfaces in the LISA product line, including DevTest Workstation.

- For a quick summary of the functionality available in the Portal, see [DevTest Portal Functionality \(see page 48\)](#).
- [Using the DevTest Portal with CA Service Virtualization \(see page 693\)](#) provides detailed information about using the Portal to virtualize service behavior for supported features.

This section contains the following pages:

- [Creating Service Images \(see page 759\)](#)
- [Editing Service Images \(see page 906\)](#)
- [Editing a VSM \(see page 934\)](#)
- [De-identifying Data \(see page 969\)](#)

- [Virtualizing a Service \(see page 970\)](#)

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 \(see page 759\)](#)
- [Combine Service Images \(see page 759\)](#)
- [Delete a Service Image \(see page 760\)](#)
- [Create a Service Image \(see page 760\)](#)
- [Create a Service Image from Scratch \(see page 762\)](#)
- [Create a Service Image from a WSDL \(see page 762\)](#)
- [Create a Service Image from WADL \(see page 764\)](#)
- [Create a Service Image from RAML \(see page 765\)](#)
- [Create a Service Image from Swagger 2.0 Specification \(see page 767\)](#)
- [Create a Service Image from Layer 7 \(see page 768\)](#)
- [Create a Service Image from Request-Response Pair \(see page 770\)](#)
- [Create a Service Image from PCAP \(see page 773\)](#)
- [Create a Service Image by Recording \(see page 775\)](#)
- [Work with VSMs \(see page 862\)](#)
- [Using Data Protocols \(see page 863\)](#)

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 \(see page 906\)](#).

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](#)**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](#)**Follow these steps:**

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.

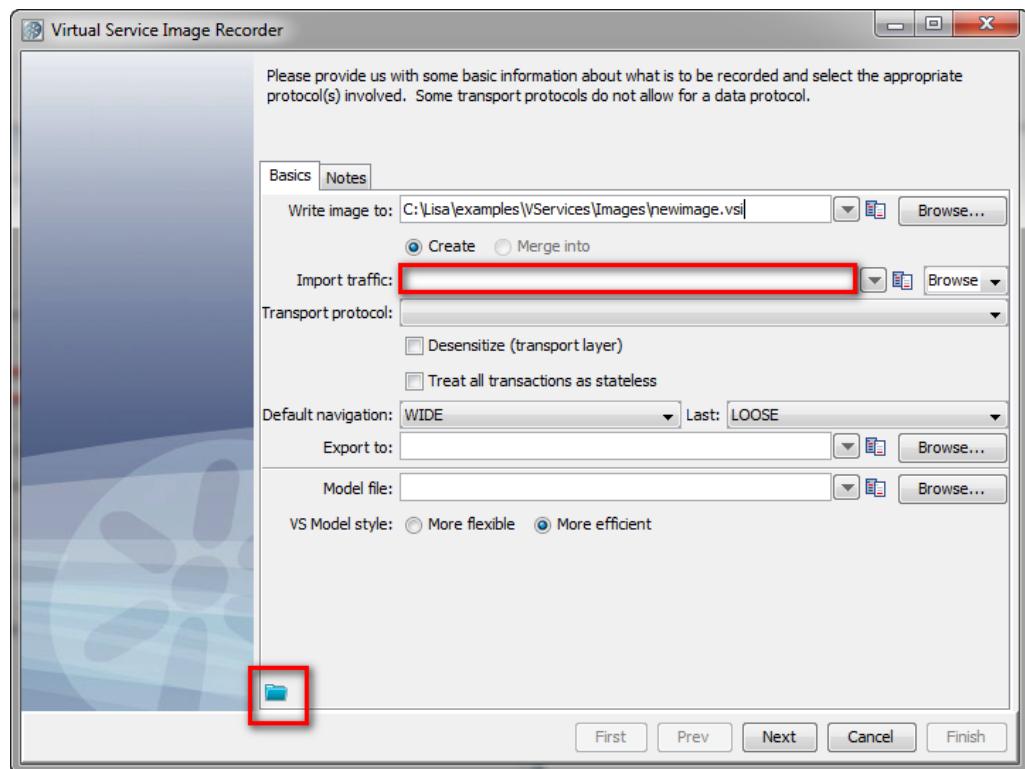


Image of the Virtual Service Image Recorder page, highlighting the Import Traffic field

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 \(see page 762\)](#)
- [Create a Service Image from a WSDL \(see page 762\)](#)
- [Create a Service Image from Request-Response Pair \(see page 770\)](#)

- [Create a Service Image from PCAP \(see page 773\)](#)
- [Work with VSMs \(see page 862\)](#)
- [Using Data Protocols \(see page 863\)](#)

Create a Service Image from Scratch

Follow these steps:

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 in the following ways:

- [From the Quick Start Menu \(see page 762\)](#)
- [From Create New VS Image \(see page 763\)](#)

From the Quick Start Menu

This procedure describes how to create a virtual service image from a WSDL 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 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 \(see page 880\)](#) 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 \(see page 775\)](#).

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` (see page) 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 \(see page 775\)](#).

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.



Note: The parameter `lisa.vse.rest.max.optionalqueryparams` (see page) 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 2.0 Specification

This procedure describes how to create a virtual service image from the Swagger 2.0 Specification. Swagger is a standard, language-agnostic interface to REST APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection.

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 2.0 Specification**.
 - From the **File** menu, select **New, VS Image, From Swagger 2.0 Specification**.

The **Virtual Service From Swagger 2.0 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.

For more information about field descriptions, see [Basics Tab \(see page 775\)](#).

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 **URL** field, enter URL or browse file that contains the Swagger 2.0 specification.
6. Click **Refresh Swagger Cache** .
7. When the Swagger parsing is successful, the Endpoint field and Methods fields are populated with information from the Swagger file. The Endpoint field is disabled and is for information only.
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 for this VSI from the Swagger file 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` (see page 1638) specifies the maximum number of optional query parameters to process per method in a Swagger 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 Layer 7

The **Virtual Service From Layer7** 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 \(see page 880\)](#) 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 Pair

You can generate a virtual service image from request/response pairs in the following ways:

- [From the UI \(see page 770\)](#)
- [From the Command Line \(see page 772\)](#)

You can use [sidecar files \(see page 772\)](#) 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.

The request/response pairs should be named with a unique identifier, then 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. Using the previous example, the following file names would create one request with three responses.

- **addUserObject-req.xml**

- addUserObject-rsp1.xml
- addUserObject-rsp2.xml
- addUserObject-rsp3.xml

VSE generates a transaction for each request/response pair in the directory 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 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.

Sidecar Files with Request-Response Pairs

When VSE creates a service image from request/response pairs, the system can use another file with the request/response pairs named a sidecar file. A sidecar file is a properties file in which you can add key/value pairs that need to be added to the metadata of certain requests, responses, or both. You can add some files that are generic for all requests and responses.

Example:

A directory that is named **soap** has request/response pair files that are named **abc-req.xml** and **abc-rsp.xml**. You add the sidecar files with the naming convention of **abc-req-meta.properties**, **abc-rsp-meta.properties**, and so on. If you intend to add a property to the meta of all requests and responses in that directory, use the file names **meta-req.properties** and **meta-rsp.properties**. The entries in the transaction-specific sidecar files (**abc-req-meta.properties** and **abc-rsp-meta.properties**) override anything that is found in the global sidecar files (**meta-req.properties** and **meta-rsp.properties**). Therefore, you can have the set of all the defaults in the global files and then override the defaults for specific transactions with transaction-specific sidecars.

For example, if you have three requests/responses (**abc1-req.xml/abc1-rsp.xml**, **abc2-req.xml/abc2-rsp.xml**, and **abc3-req.xml/abc3-rsp.xml**), you can have a common sidecar file, **rsp-meta.properties**, that has:

`Content-type=text/plain`

and an abc3-specific sidecar file, **abc3-rsp-meta.properties**, that has:

`Content-type=text/html`

In the service image, the metadata for responses **abc1** and **abc2** has the Content-type meta property set to "text/plain", but the **abc3** response has the value "text/html".

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. Download the appropriate binary package for your operating system from <http://jnetpcap.com/download>.
2. 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.
3. 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 \(see page 787\)](#).

- **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

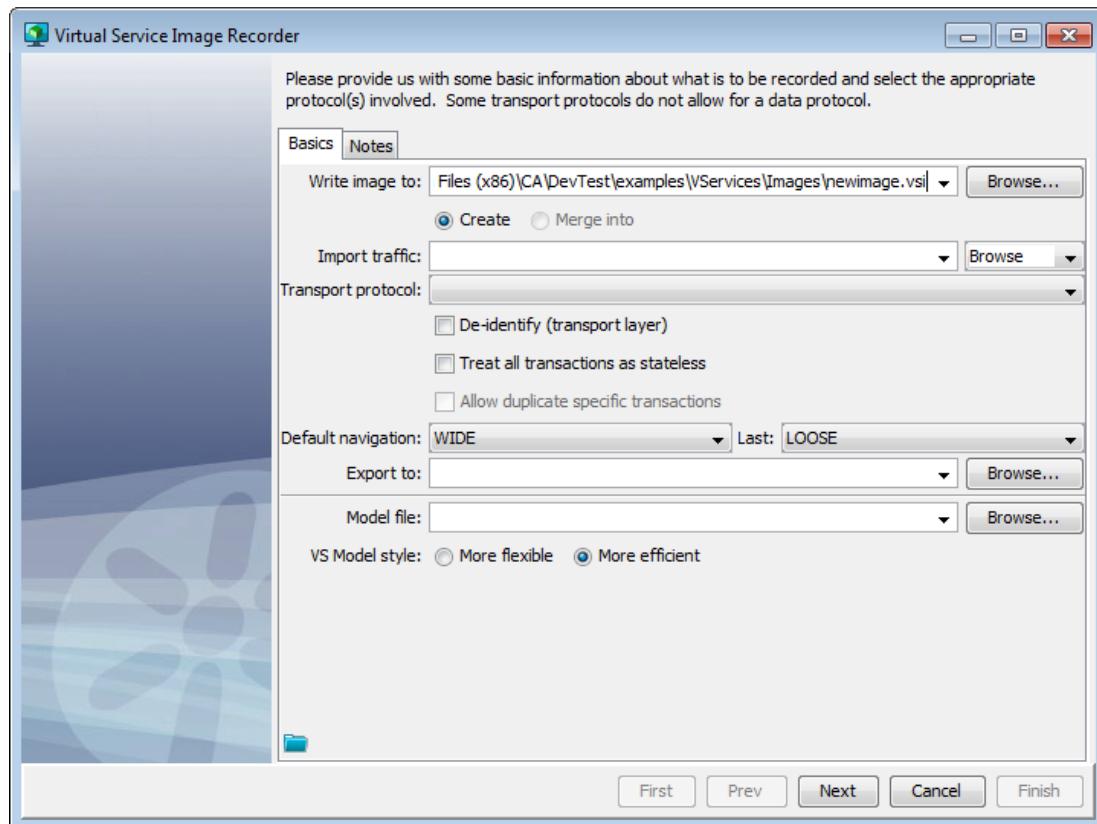
The **Virtual Service Image Recorder** generates service images, which pretend to be what you recorded.

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**.
2. Complete the following **Virtual Service Image Recorder tabs**, as appropriate:
 - [Basics \(see page 775\)](#)
 - [Data Protocols \(see page 778\)](#)
 - [Recording by Transport Protocol \(see page 781\)](#) (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 provides 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 "[Virtual Service Image Recorder - Transport Protocols \(see page 781\)](#)".



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 \(see page 781\)](#)".

■ De-identify

Specifies whether to try to recognize sensitive data and substitute random values during the recording. For more information, see "[De-identifying Data \(see page 969\)](#)".

■ 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.

Values:

- CLOSE: The children of the current transaction are searched.
- WIDE: A close search, including the current transaction plus siblings and nieces/nephews of the current transactions.
- LOOSE: A wide search, plus the parent and siblings of the current transaction, followed by the children of the starting transaction. If both fail to find a match, then the starting transactions for all conversations are checked, resulting in searching the full conversation.

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.

Values:

- CLOSE: The children of the current transaction are searched.
- WIDE: A close search, including the current transaction plus siblings and nieces/nephews of the current transactions.
- LOOSE: A wide search, plus the parent and siblings of the current transaction, followed by the children of the starting transaction. If both fail to find a match, then the starting transactions for all conversations are checked, resulting in searching the full conversation.

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

The second window in the **Virtual Service Image Recorder** wizard lets you enter information about data protocols for the virtual service.

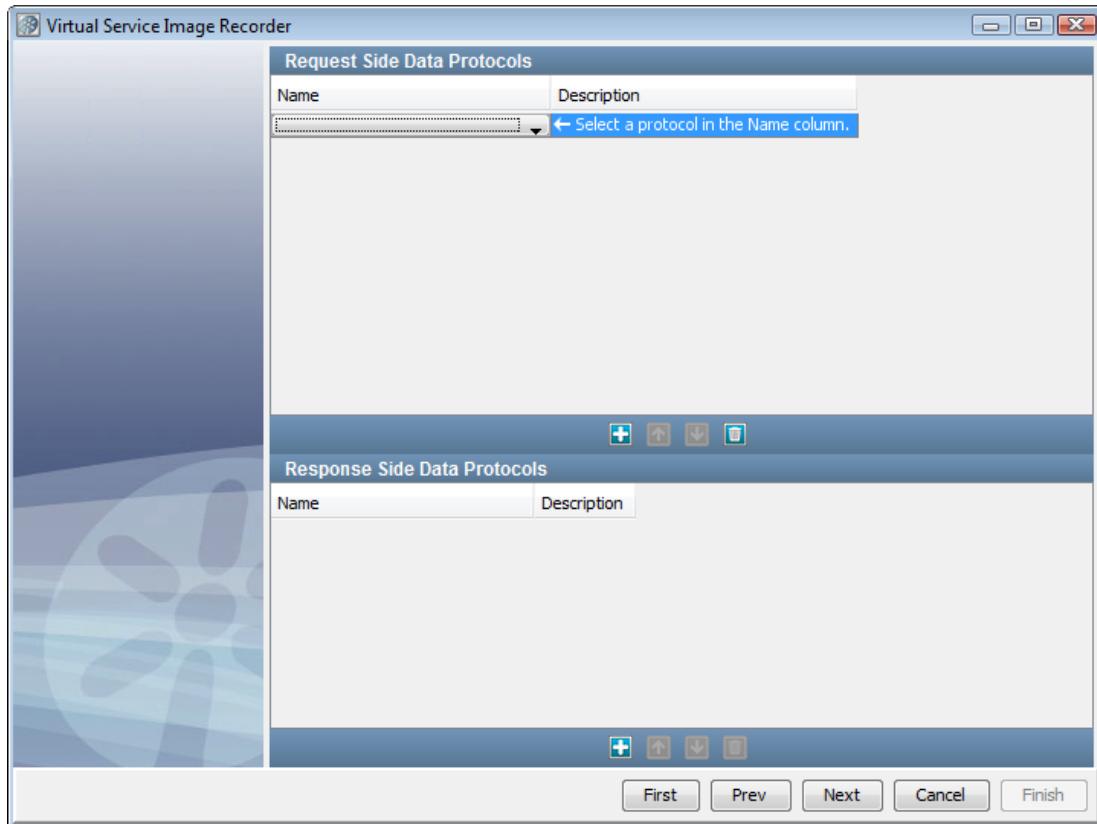


Image of the Data Protocols page on the Virtual Services Image Recorder

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-identifying Data \(see page 969\)](#).

- **Delimited Text Data Protocol**

Converts delimited strings to XML.

- **DRDA Data Protocol**

Converts binary DRDA payloads to XML during recording to facilitate alignment with native DevTest functionality, readability, and dynamic data support. Converts responses back to their native format on playback.

- **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.

- **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 \(see page 863\)](#)".

For more information about using a dynamic data protocol, see "[Generic XML Payload Parser \(see page 886\)](#)".

Transport Protocols

Each available transport protocol is described in the following sections:

- [HTTP/S Transport Protocol \(see page 781\)](#)
- [IBM MQ Native Transport Protocol \(see page 789\)](#)
- [IBM WebSphere MQ Transport Protocol \(see page 794\)](#)
- [JMS Transport Protocol \(see page 798\)](#)
- [Standard JMS Transport Protocol \(see page 808\)](#)
- [Java Transport Protocol \(see page 812\)](#)
- [JDBC Transport Protocol \(see page 814\)](#)
- [TCP Transport Protocol \(see page 821\)](#)
- [CICS \(LINK DTP MRO\) Transport Protocol \(see page 826\)](#)
- [CICS Transaction Gateway \(ECI\) Transport Protocol \(see page 845\)](#)
- [IMS Connect Transport Protocol \(see page 847\)](#)
- [SAP RFC via JCo Transport Protocol \(see page 850\)](#)
- [JCo IDoc Transport Protocol \(see page 851\)](#)
- [Opaque Data Processing Transport Protocol \(see page 854\)](#)

HTTP/S Transport Protocol

Contents

- [Virtualize Two-way SSL Connections \(see page 787\)](#)

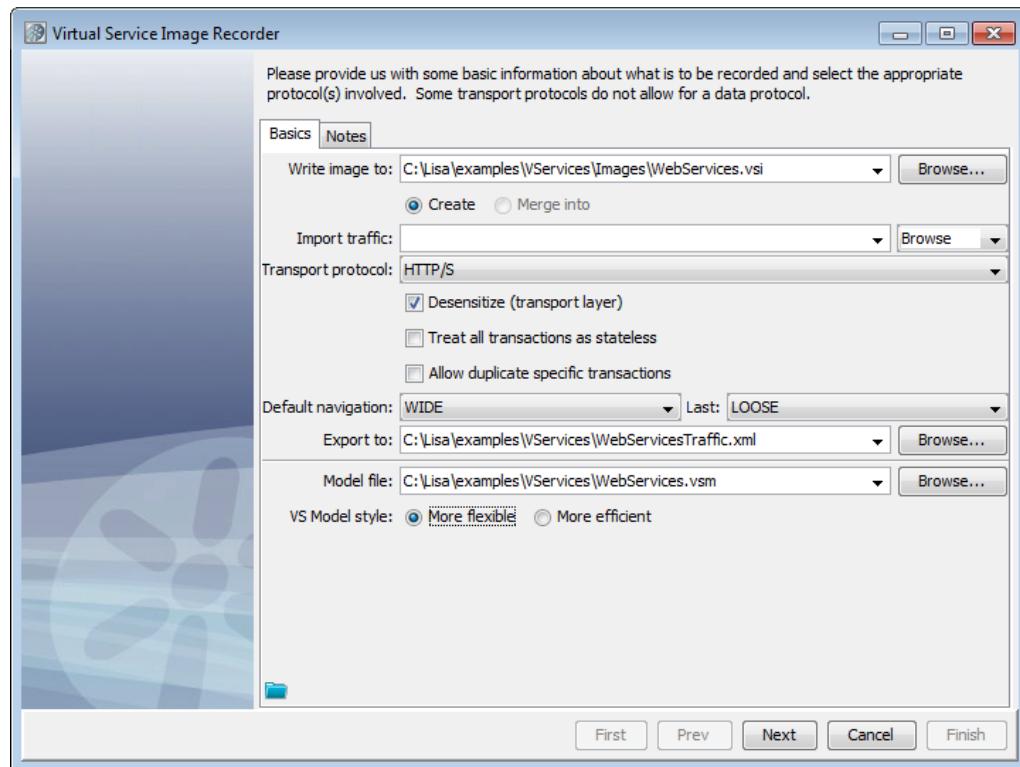
Follow these steps:

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 \(see page 775\)](#) 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.

- **Target host**

Defines the name or IP address of the target host where the server runs. Leave blank if you are going to select a Proxy pass through style.

- **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 \(see page 787\)](#).

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 service 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.

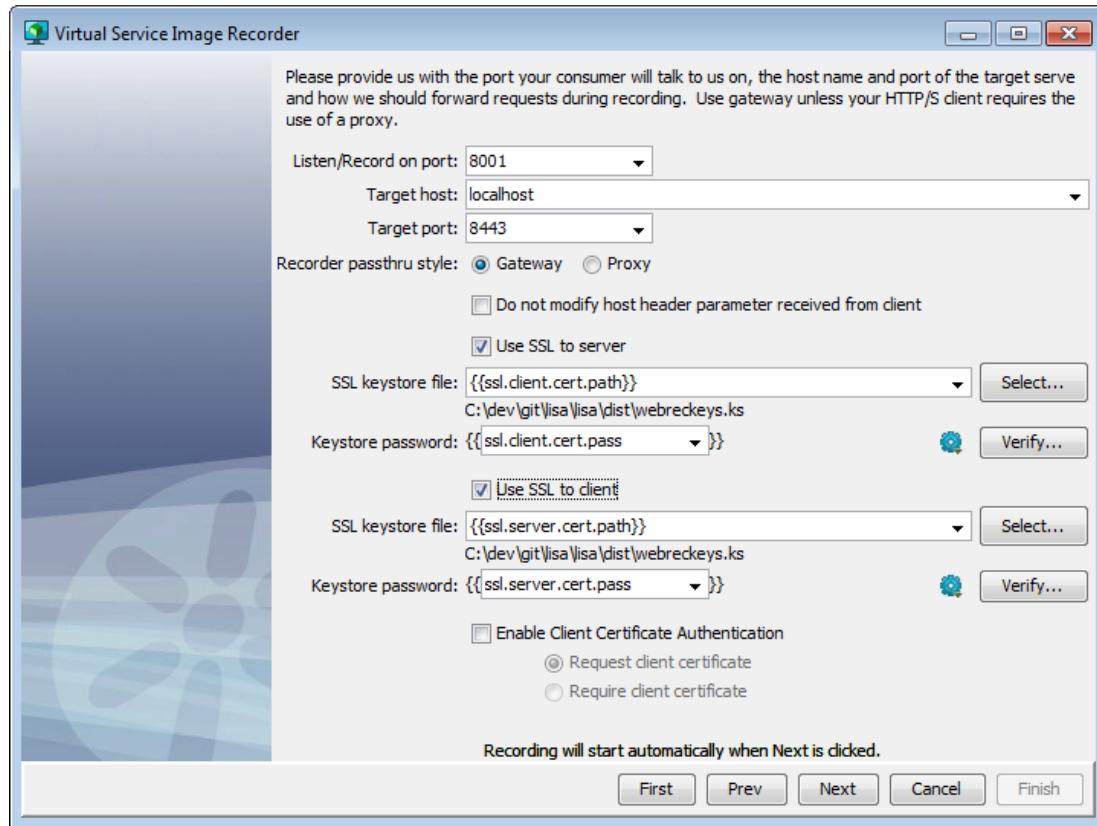
Virtualize Two-way SSL Connections

To virtualize a two-way SSL connection, DevTest must have information for both the client-side keystore and server-side keystore.

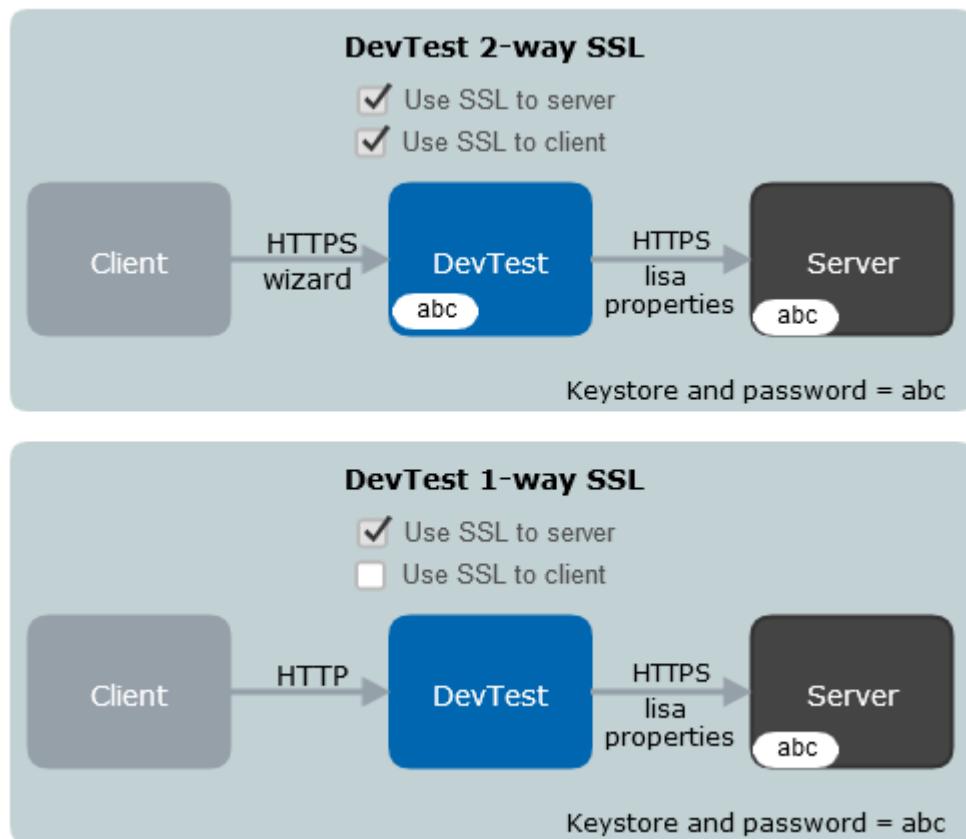


Note: To use the default DevTest keystore (see **webrekeys.ks** in the installation directory) as the server-side keystore, extract the DevTest certificate from the DevTest keystore and add it to the client truststore. The DevTest certificate is a self-signed certificate and not a certificate that a 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 \(see page 798\)](#).

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 \(see page 1380\)](#).

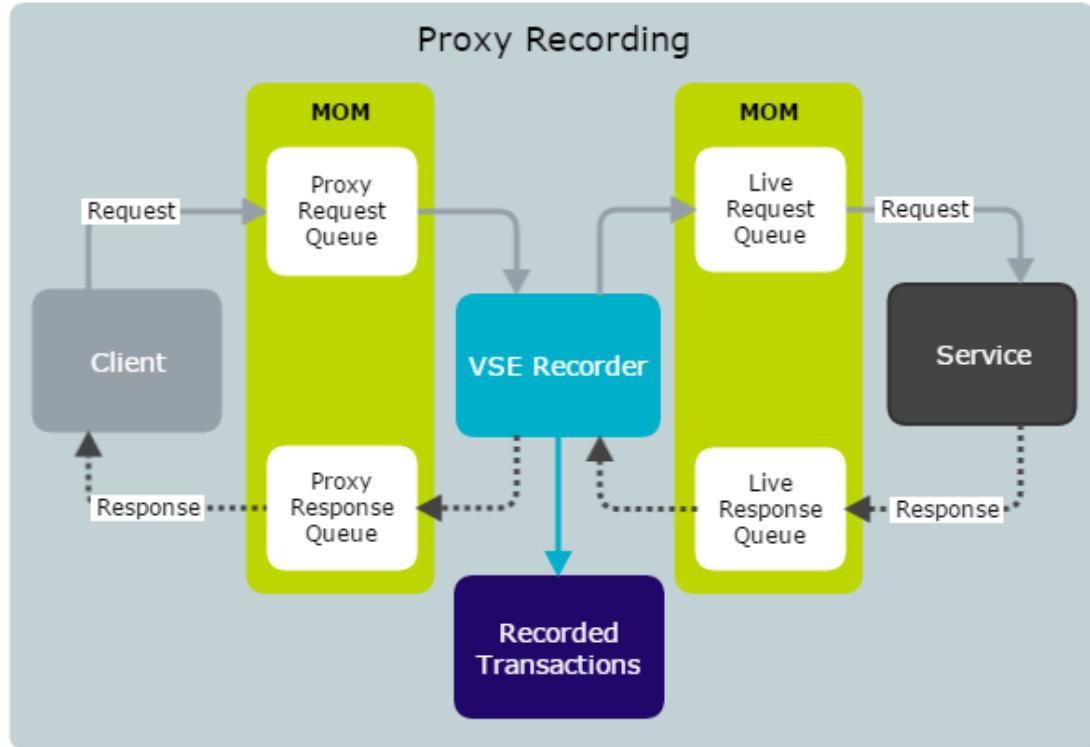
Proxy recording is a common method for recording a messaging application.

The following graphic shows the main components of proxy recording:

- The client
- The service
- The proxy queues

- The live queues
- The VSE recorder

The message-oriented middleware (MOM) is the platform on which messages are exchanged.



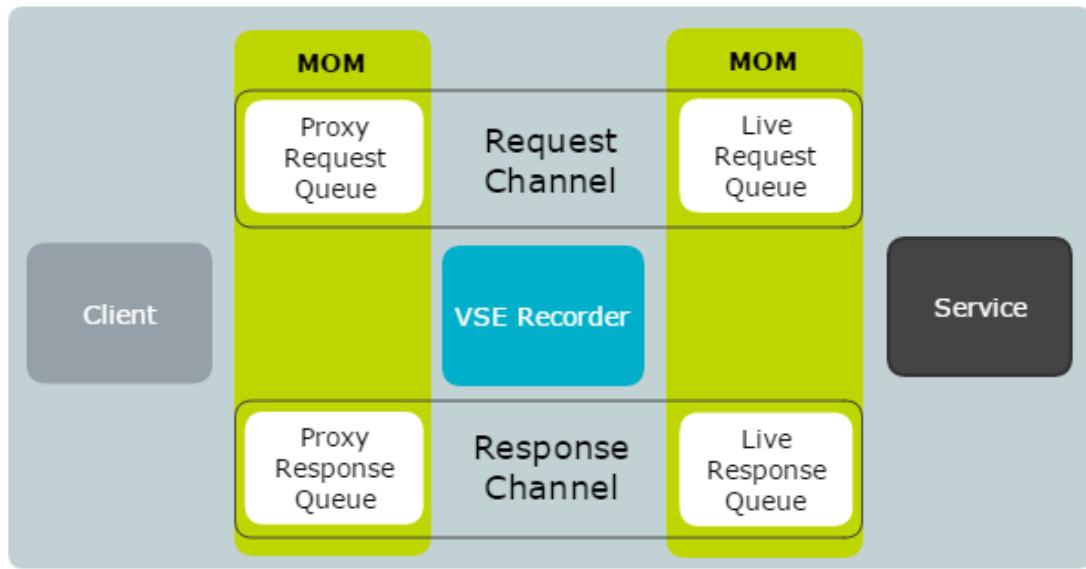
The VSE recorder is inserted into the message flow between the client and the service. Each request message goes through the recorder before it reaches the service. Each response message goes through the recorder before it reaches the client.

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.

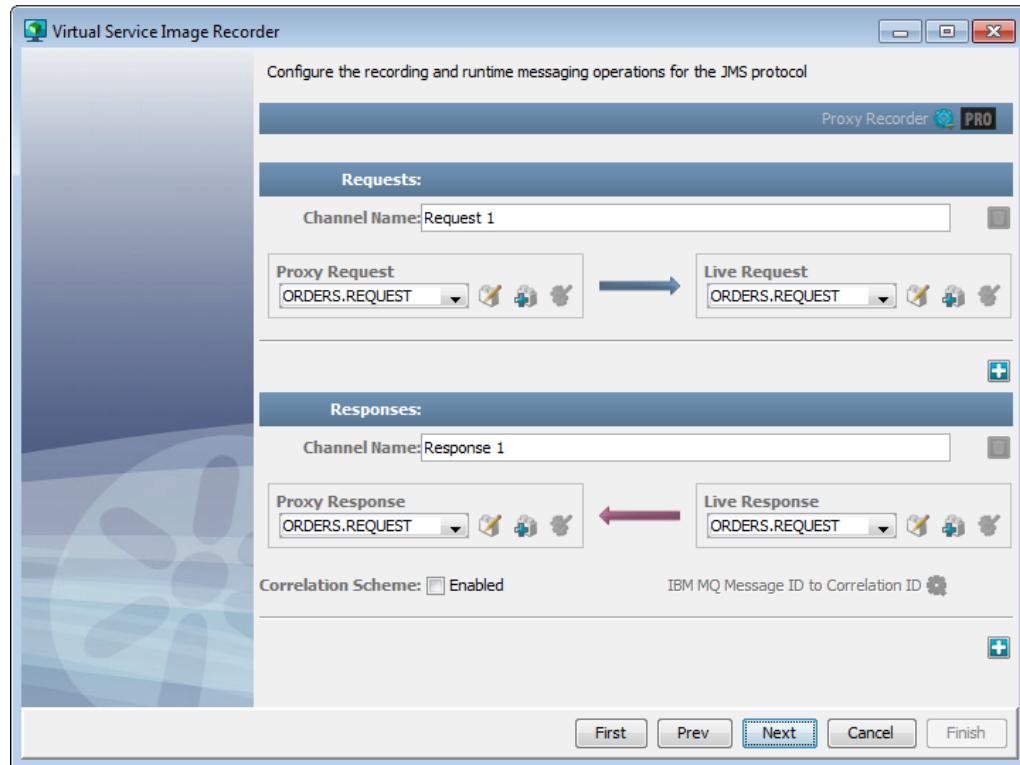


A proxy recording can have multiple request channels and multiple response channels.

In this procedure, you select an [asset \(see page 398\)](#) 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.

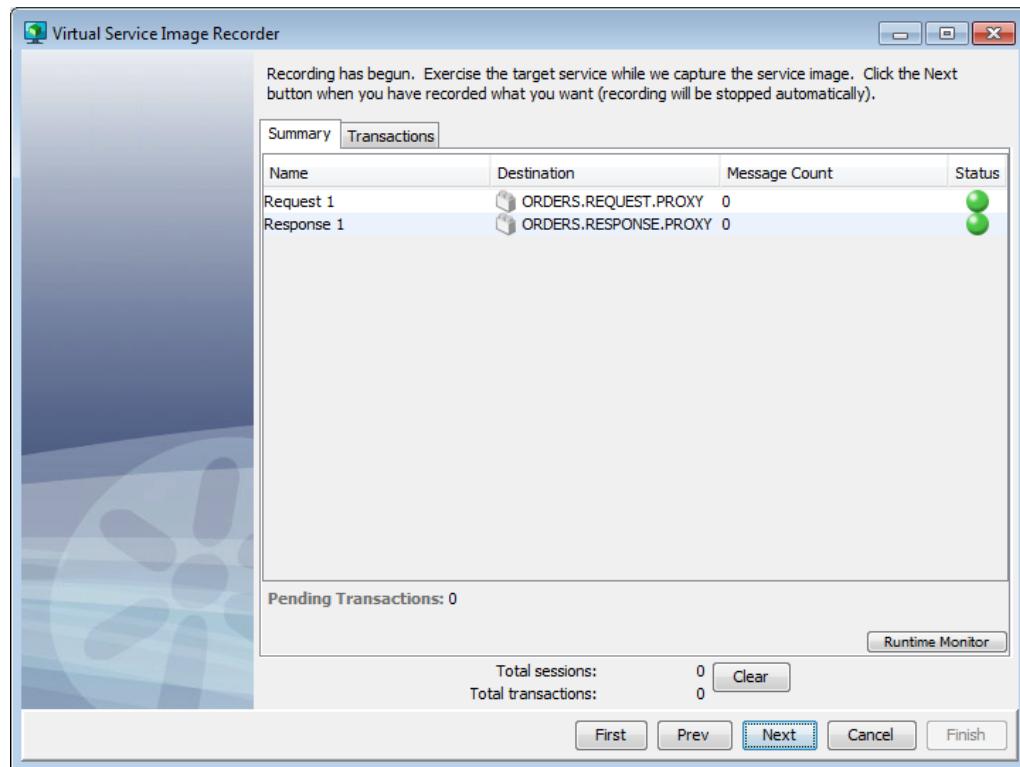


Screen capture of proxy recorder setup page.

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 \(see page 1555\)](#) 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 \(see page 1851\)](#).

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 \(see page 933\)](#) and the [virtual service model \(see page 961\)](#) are created.

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 \(see page 1380\)](#).

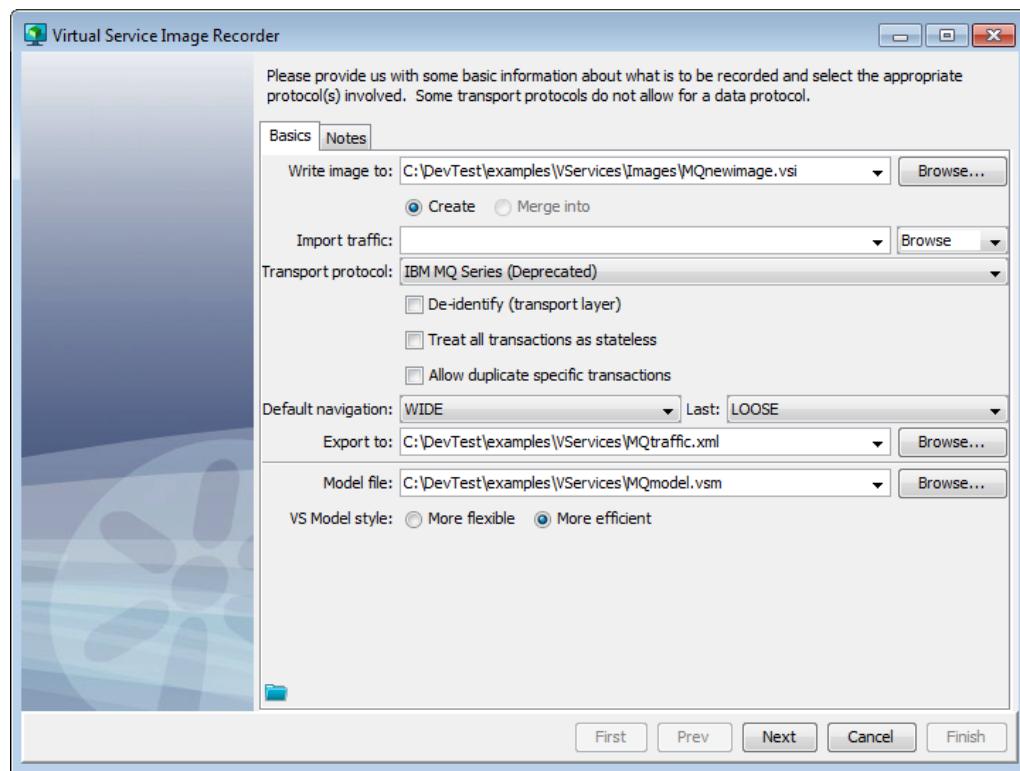
Follow these steps:

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** (see page 775) 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 CAI, this option is cleared. CAI autodetects queues and transactions. If the raw traffic file does not come from CAI, 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 \(see page 892\)](#)". 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 \(see page 799\)](#)
- [Live VSM Type \(see page 803\)](#)
- [TIBCO Monitor Recorder \(see page 803\)](#)
- [JMS Topic Recorder \(see page 805\)](#)

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 \(see page 1380\)](#).

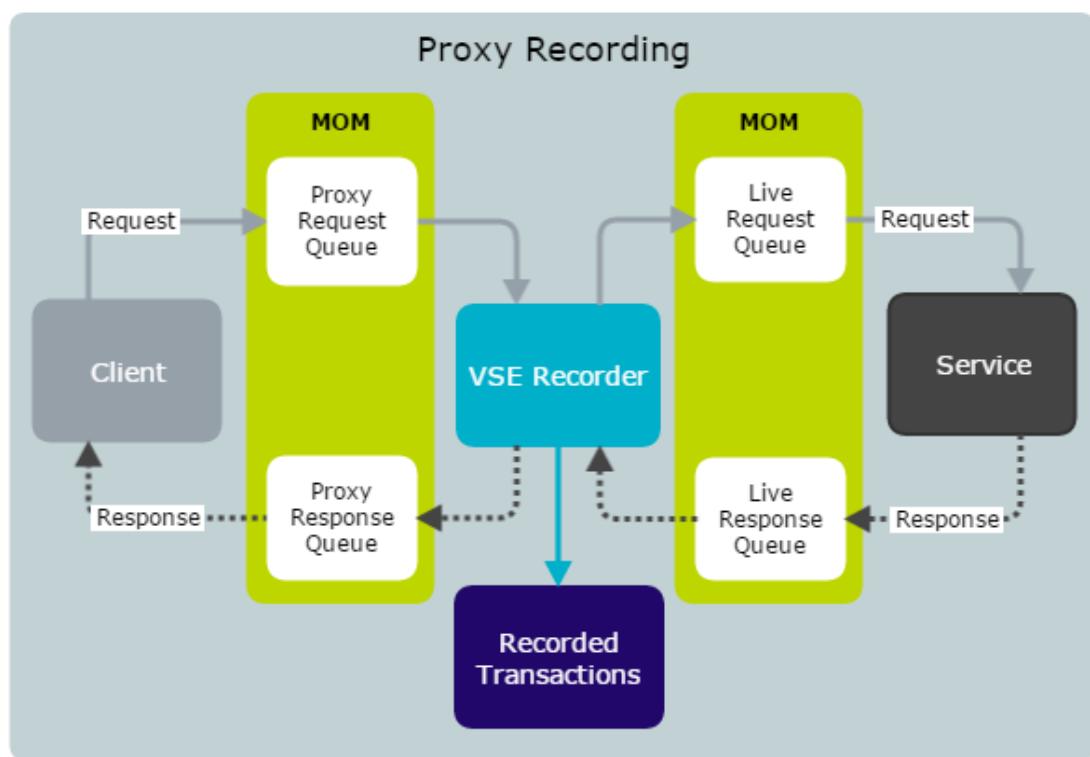
Basic Proxy Recording

Proxy recording is a common method for recording a messaging application.

The following graphic shows the main components of proxy recording:

- The client
- The service
- The proxy queues
- The live queues
- The VSE recorder

The message-oriented middleware (MOM) is the platform on which messages are exchanged.



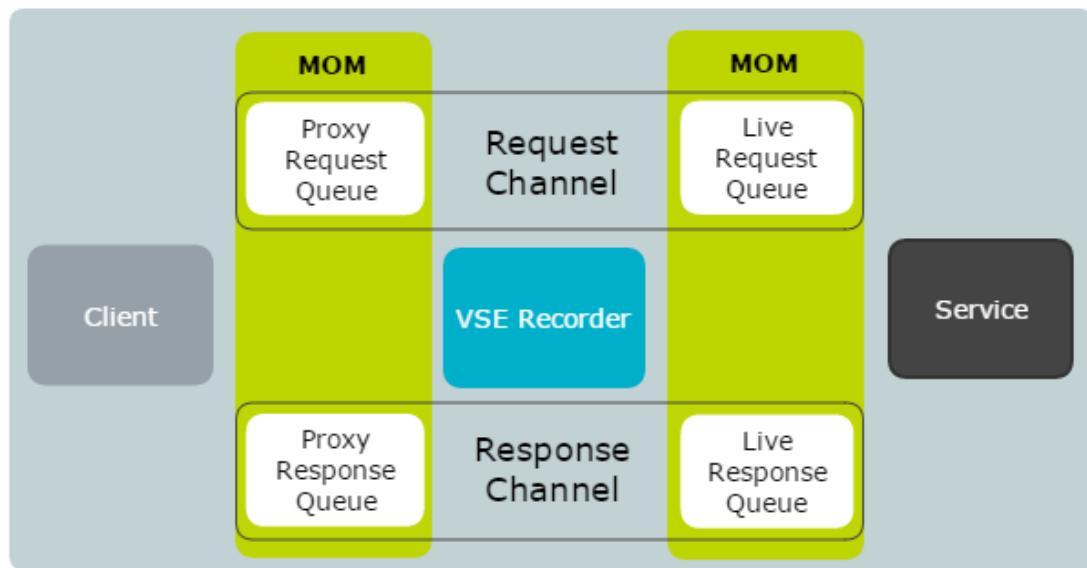
The VSE recorder is inserted into the message flow between the client and the service. Each request message goes through the recorder before it reaches the service. Each response message goes through the recorder before it reaches the client.

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.

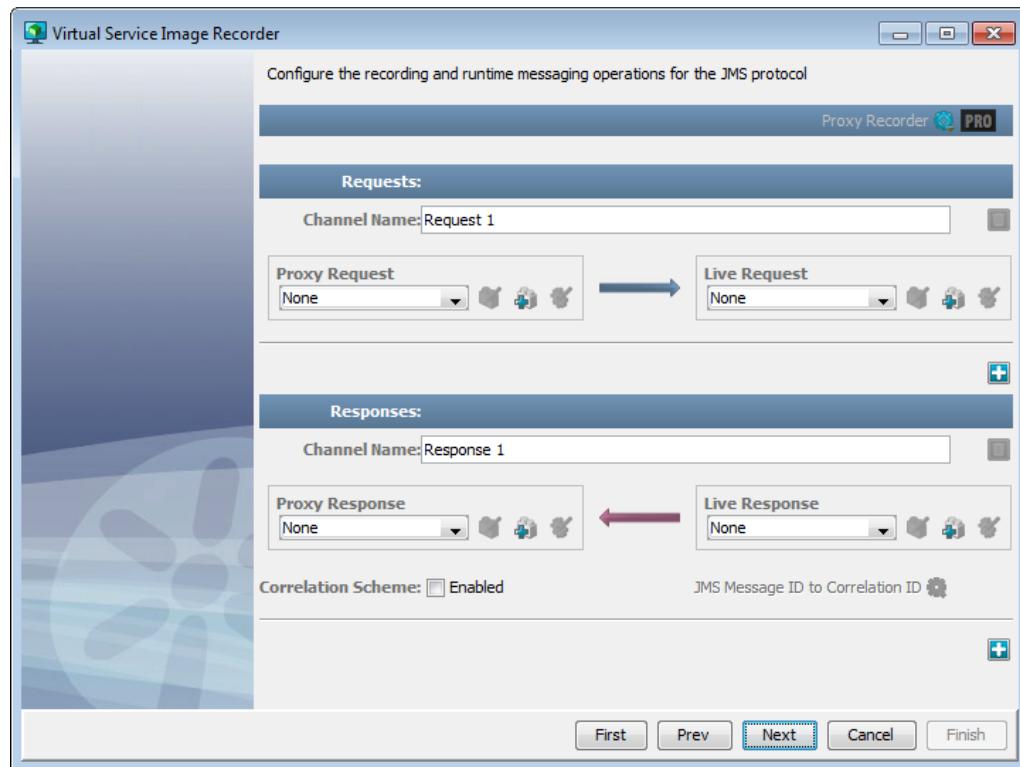


A proxy recording can have multiple request channels and multiple response channels.

In this procedure, you select an [asset \(see page 398\)](#) 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.

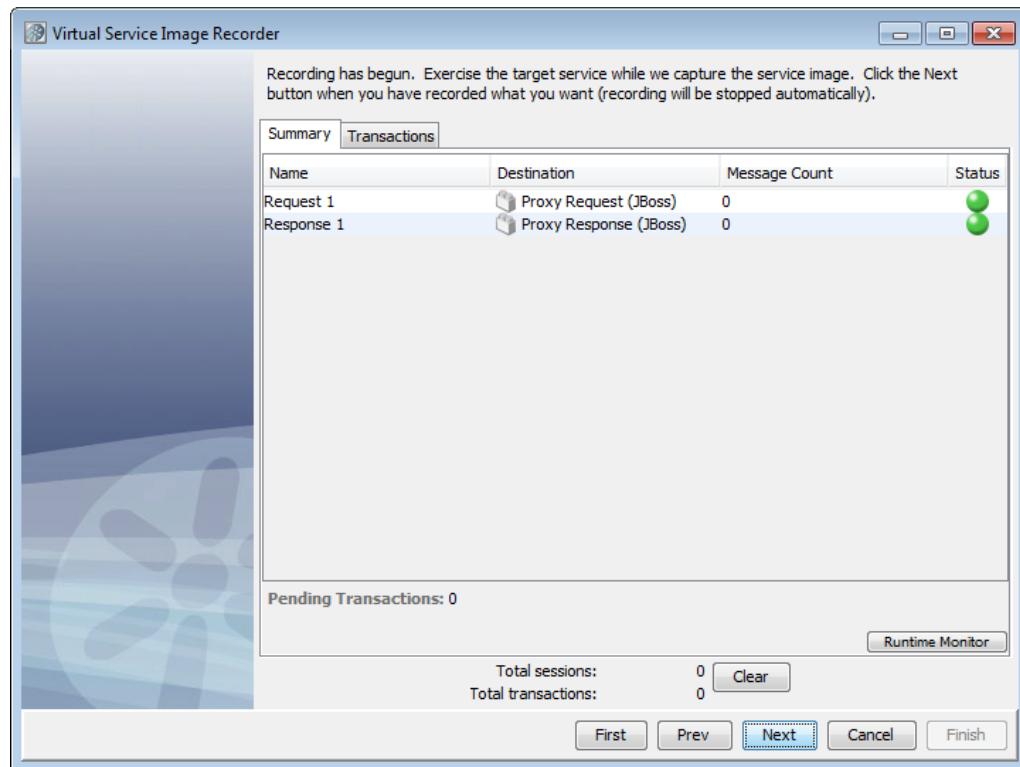


Screen capture of proxy recorder setup page.

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 \(see page 803\)](#) 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 \(see page 1555\)](#) 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 \(see page 1803\)](#).

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 \(see page 933\)](#) and the [virtual service model \(see page 961\)](#) are created.

Live VSM Type

The [proxy recorder \(see page 799\)](#) includes a field called **VSM Type**.

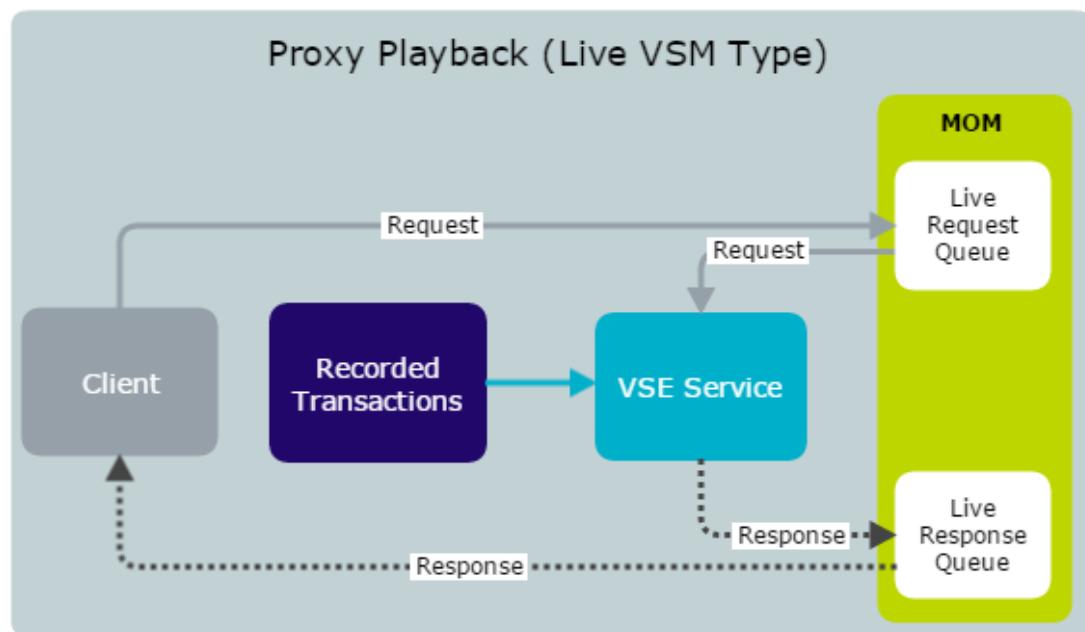
The default value is **Proxy With Live Invocation**. The recorder generates a virtual service model that operates on the proxy queues and supports the live invocation mode.

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

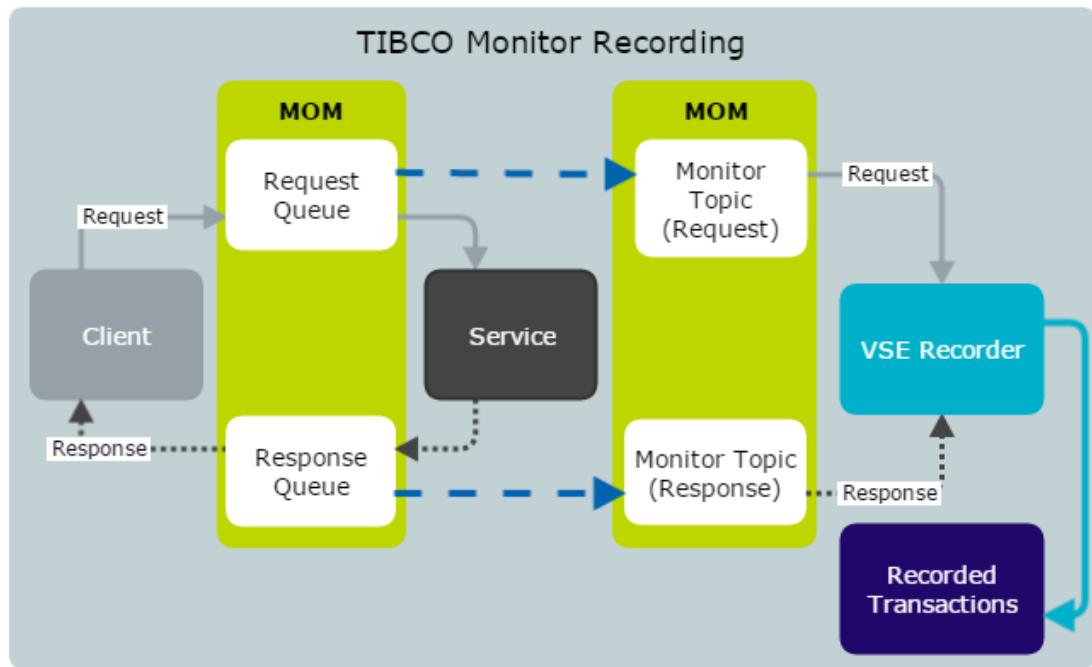
With every queue on a TIBCO EMS server, a user with administrator privileges can connect to a special monitor topic. While subscribed to this monitor topic, an administrator can receive a copy of every message that is sent over the corresponding queue. This feature allows the VSE recorder to be outside 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:

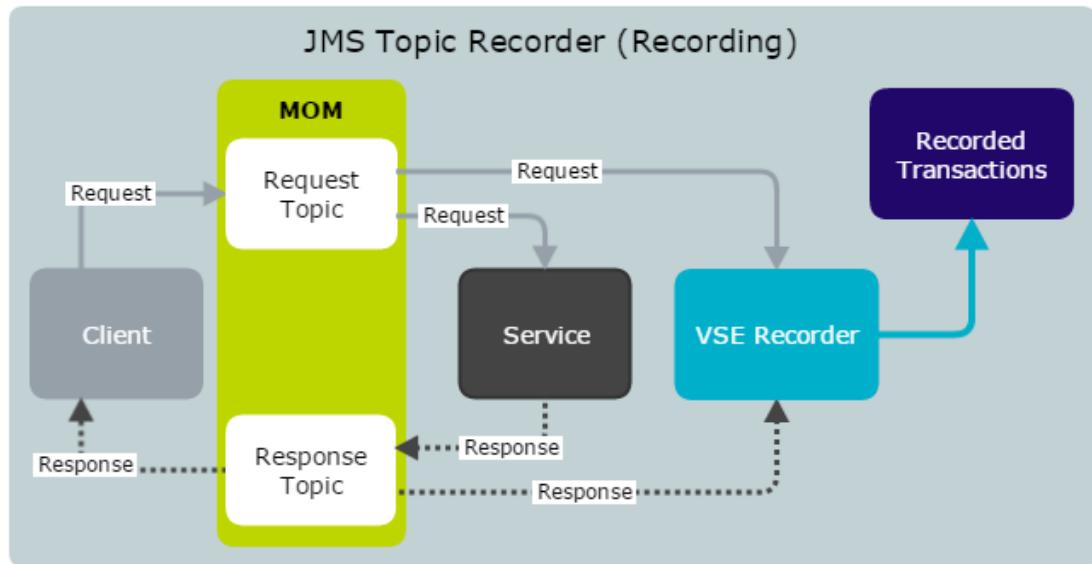
1. In the **Basics** tab of the **Virtual Service Image Recorder**, set the transport protocol to **JMS**. Be sure to configure the service image and virtual service model.
2. Click **Next**.
The proxy recorder setup page opens.

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 \(see page 1555\)](#) 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 \(see page 1803\)](#).
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.

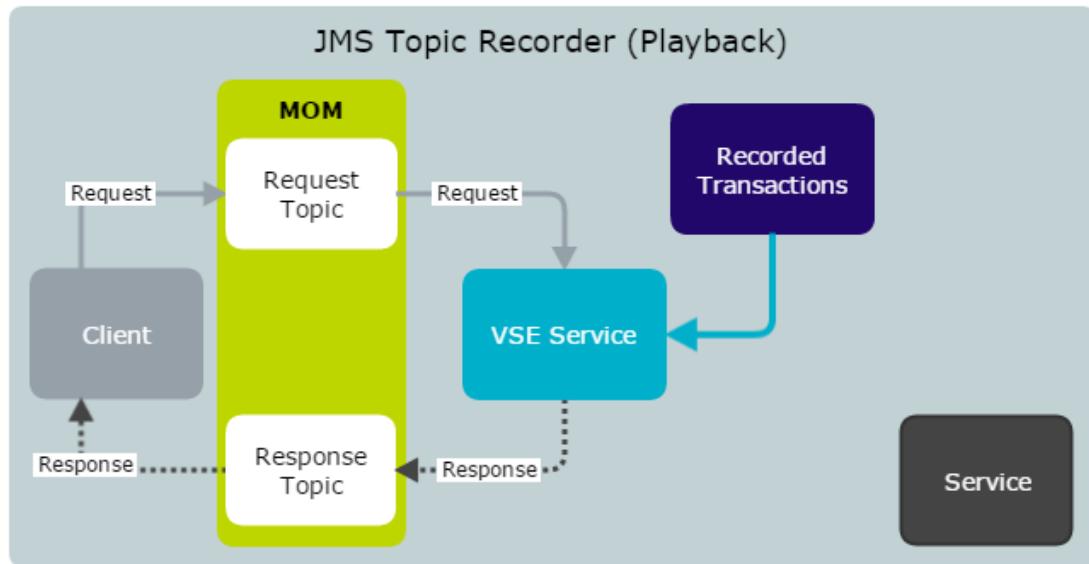
A request channel is composed of the following parts:

- 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 \(see page 1555\)](#) 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 \(see page 1803\)](#).

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

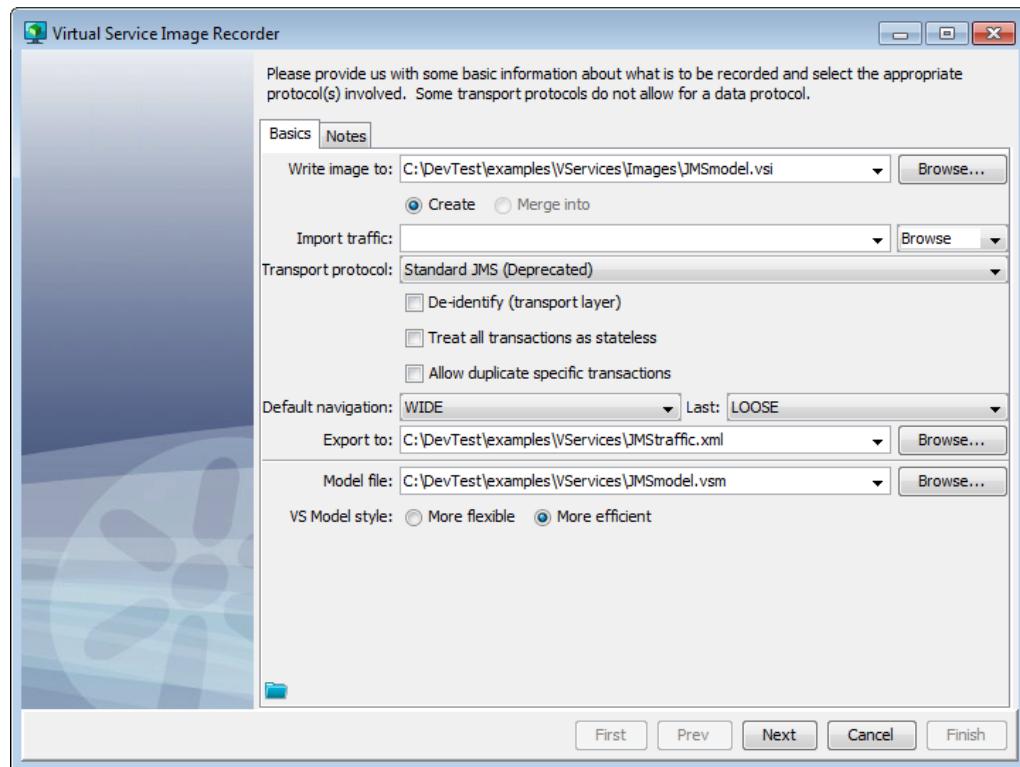
This page contains the detailed instructions for recording a virtual service image using the Standard JMS transport protocol. **Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see [Third-Party File Requirements \(see page 1380\)](#).

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 \(see page 775\)](#) 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 CAI, this option is cleared. CAI autodetects queues and transactions. If the raw traffic file does not come from CAI, 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 \(see page 863\)](#).

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.

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 \(see page 1253\)](#).

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 \(see page 775\)](#) 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 \(see page 863\)](#).
14. When you have configured the data protocols, click **Next**.
While it prepares 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 \(see page 703\)](#).

For more information about prerequisites and preparatory steps, see [Preparing to Virtualize JDBC \(see page 818\)](#).

Follow these steps:

- 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.

- Complete the **Basics** (see page 775) tab as in the following graphic:

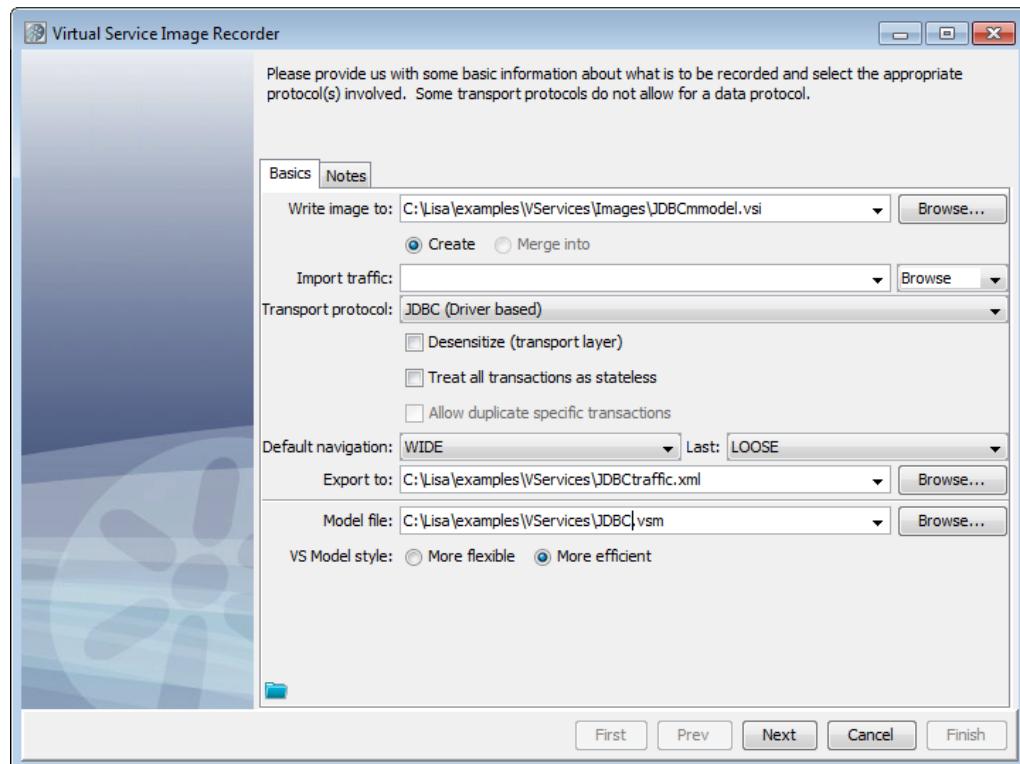


Image of the Basics tab on the Virtual Service Image Recorder for JDBC transport protocol

- Click **Next**.

The endpoint configuration window opens.

- 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.
- 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_MODIFIED, 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 \(see page 668\)](#). 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 \(see page 703\)](#).

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.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.

For more information about prerequisites and preparatory steps, see [Virtualize TCP \(see page 824\)](#).

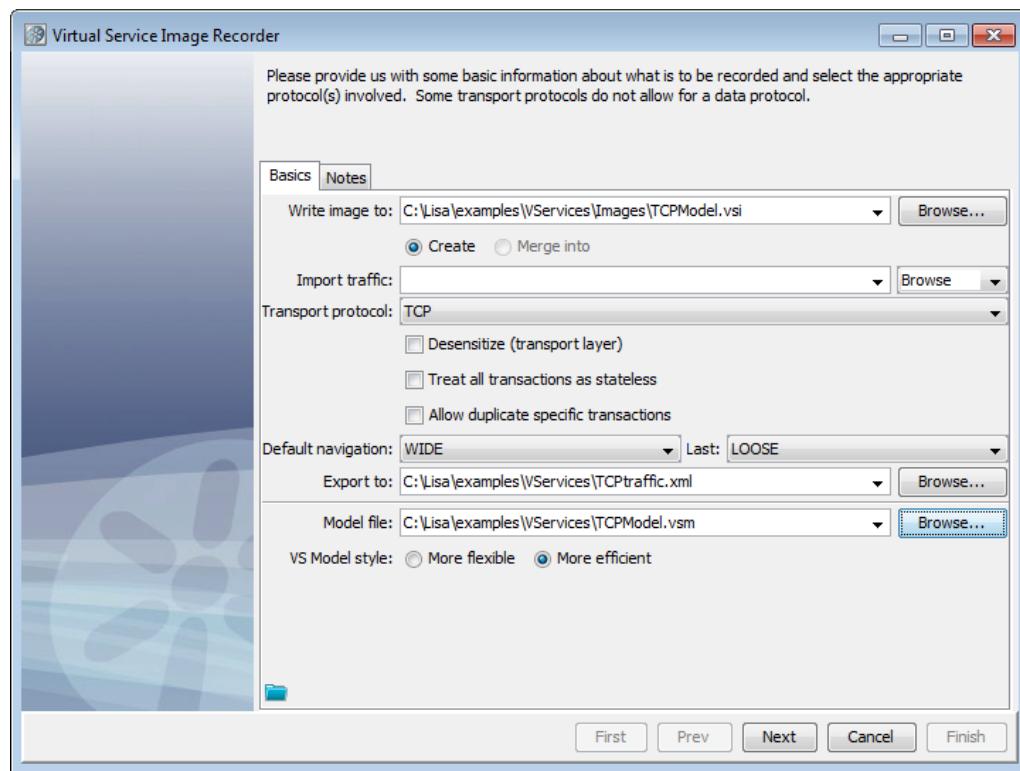
Follow these steps:

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 (see page 775) 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 \(see page 824\)](#).
 - 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 \(see page 824\)](#).

- **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.

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

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 says it is not complete.
4. A response is considered complete when request data arrives, even if the response delimiter (if any) says 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 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 in most cases), back-to-back sets of delimiters are ignored.

If you do not specify a delimiter when recording, the recorder will attempt to determine the delimiter.

Treat Request as Text and Treat Response as Text

The **Treat Request as Text** and **Treat Response as Text** 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 Request as Text**

Specifies whether to use the request as the body of the VSE request.

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 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.

CICS (LINK DTP MRO) Transport Protocol

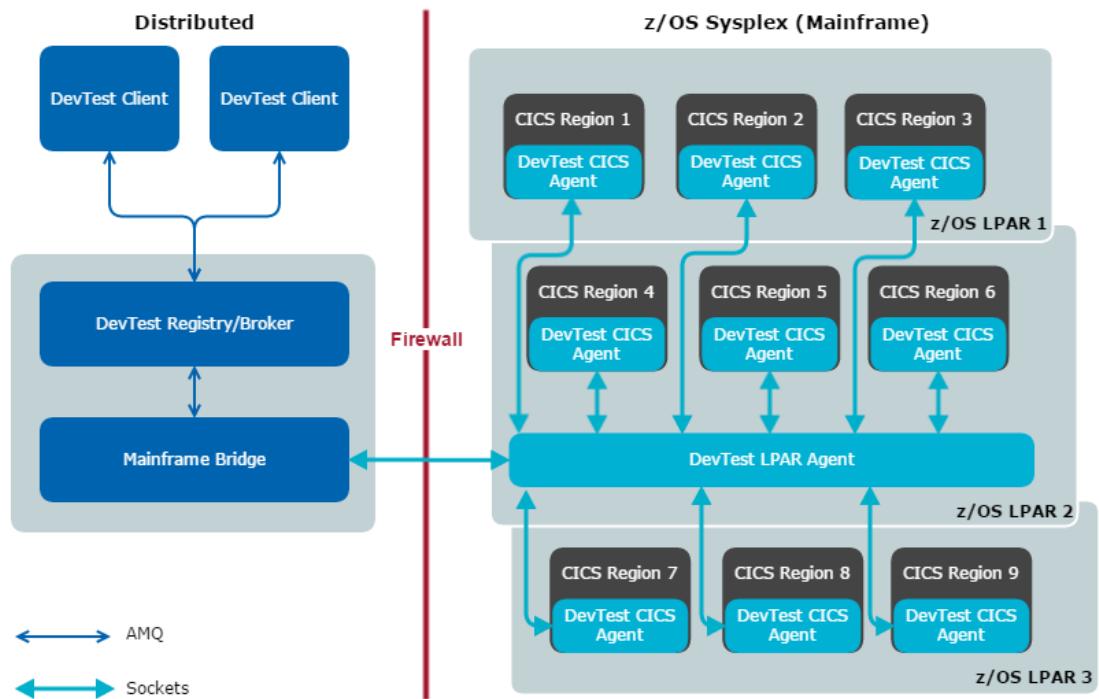
This section contains the detailed instructions for recording a virtual service image using the CICS transport protocol. The section contains the following pages:

- [Recording IBM CICS Overview \(see page 826\)](#)
- [Recording IBM CICS Example \(see page 827\)](#)
- [Using the CICS Programs to Virtualize Panel \(see page 837\)](#)

CICS Overview

To allow recording IBM CICS service images, DevTest has a mainframe bridge that is a part of the registry. That 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, and 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 mode**
The mainframe bridge initiates the connection to the LPAR agent.
- **Server mode**
The mainframe bridge waits for the LPAR agent to initiate the connection.

The [lisa.properties \(see page 1638\)](#) file contains properties to enable the mainframe bridge. For more information, see "Mainframe Properties".

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.

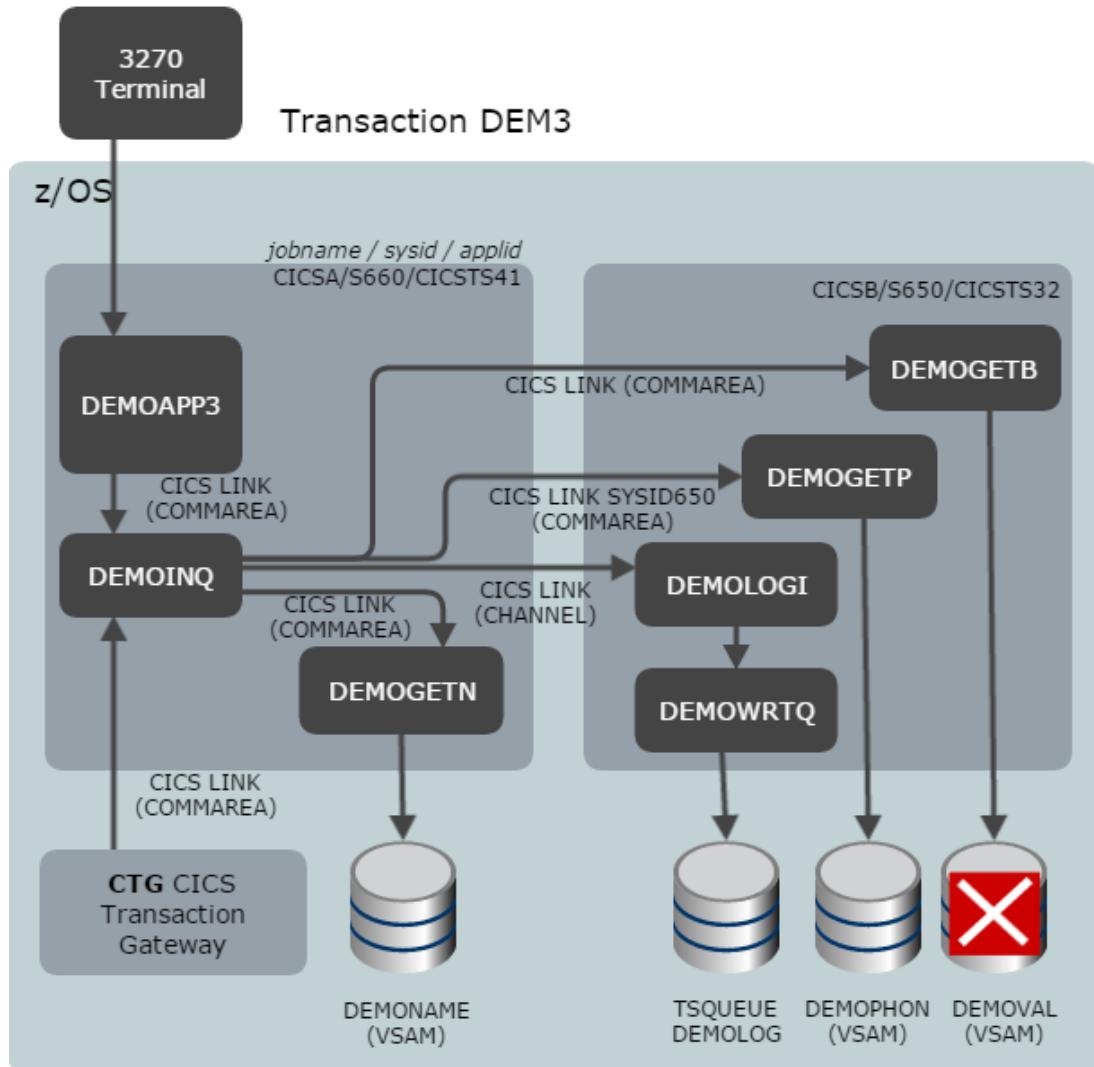
The following graphic shows the CICS transaction (DEM3) that is running. The transaction runs from a 3270 terminal, but it could run from the CICS Transaction Gateway.

Running the transaction initiates DEMOAPP3, which performs a CICS LINK to DEMOINQ. DEMOINQ then runs the following CICS LINKS:

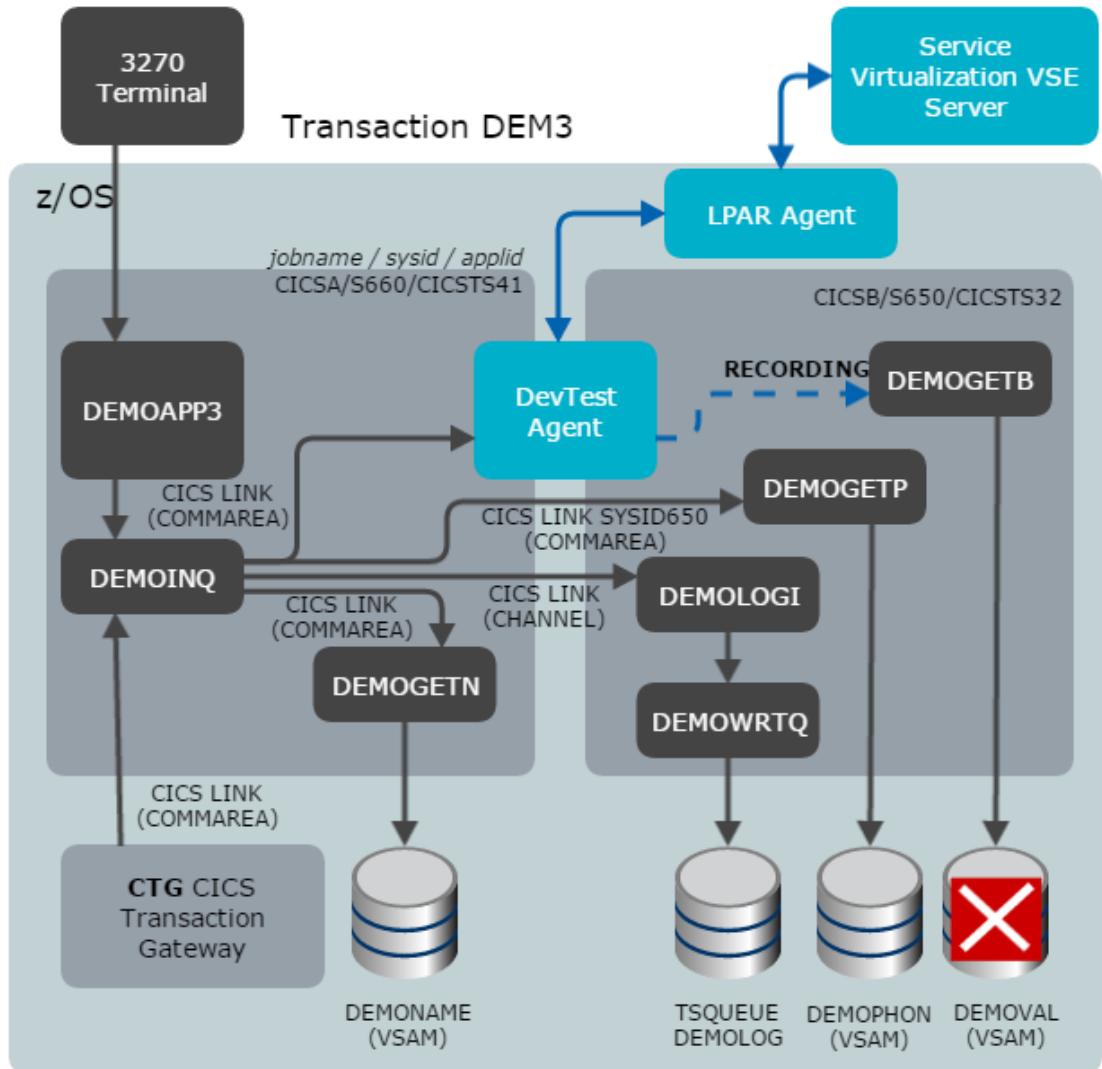
1. A local LINK passes a COMMAREA to DEMOGETN to get the customer name and address from a VSAM.
2. A CICS LINK to CICSB passes a COMMAREA to DEMOGETB. DEMOGETB reads the VSAM file DEMOGETBAL, which is not available in the example.
3. A CICS LINK passes a COMMAREA to DEMOGETP to get the telephone number 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



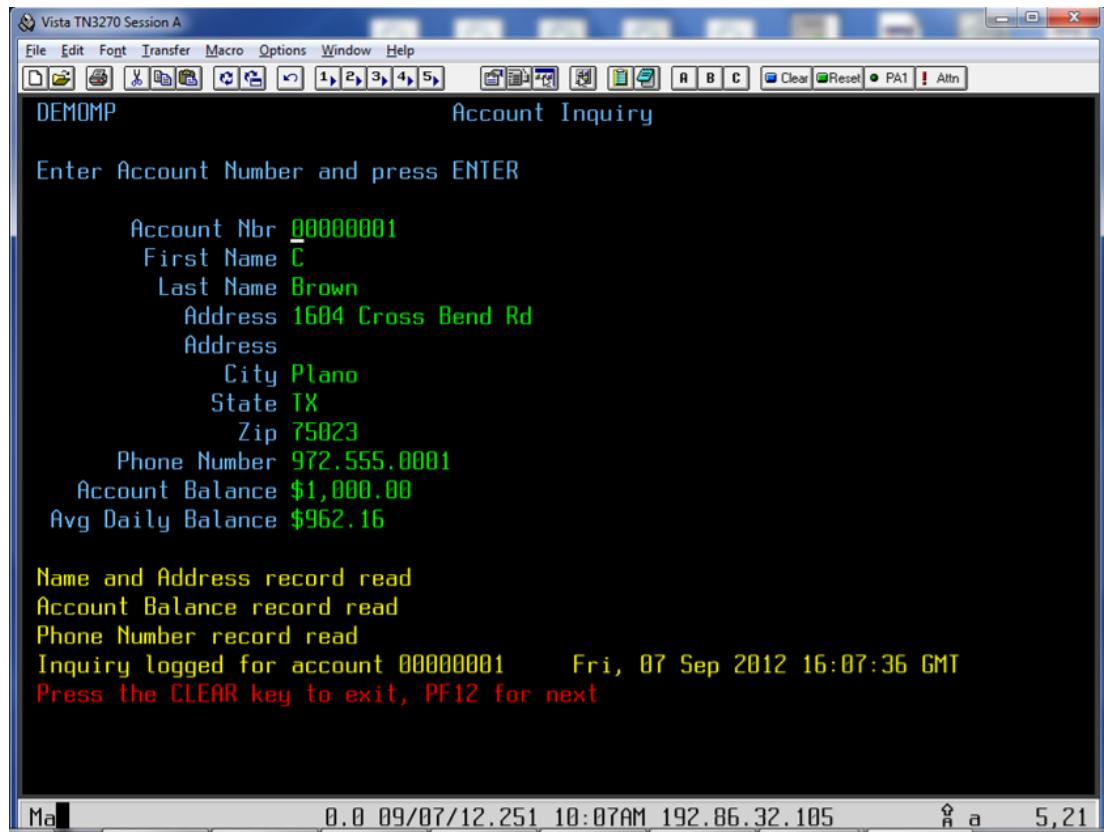
This example virtualizes DEMOGETB on CICSTS41. The example uses the DevTest agent, running in CICSA, communicating with the LPAR agent, communicating with the VSE server. We first run the CICS transaction to record the CICS LINK to DEMOGETB. We then use that recording to virtualize the CICS LINK.



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.

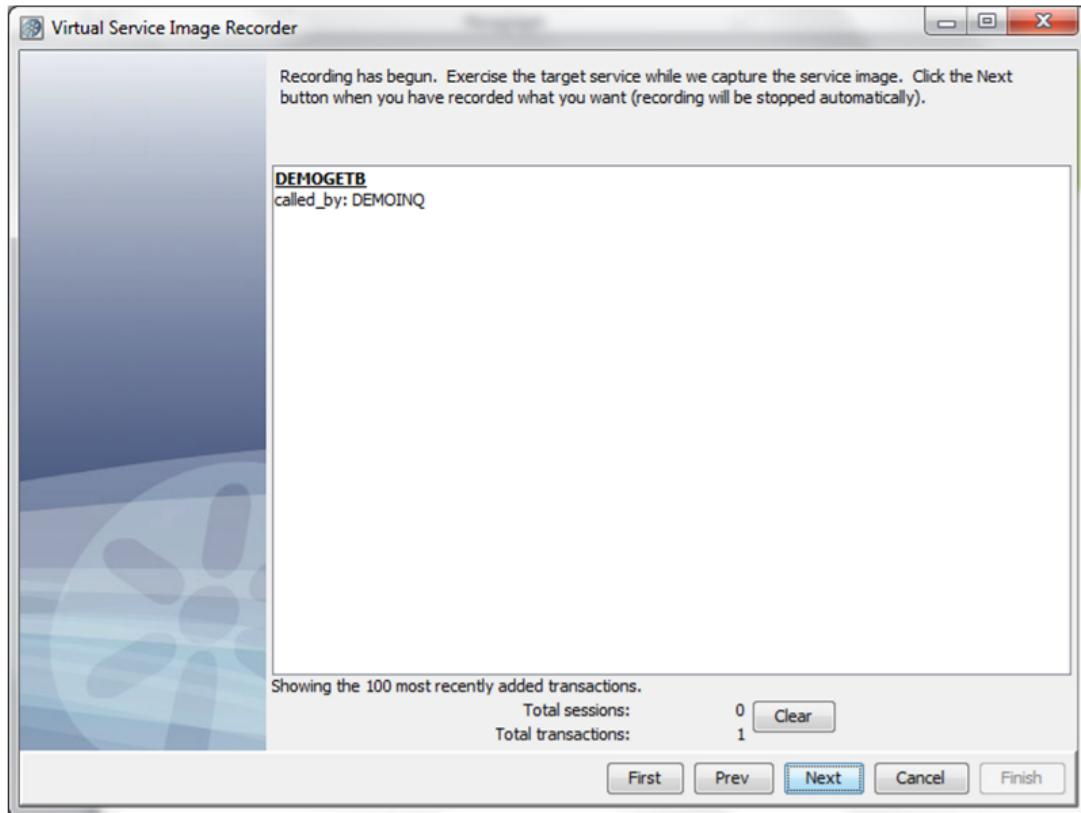
The **CICS Programs to Virtualize** panel displays next. For detailed information about this panel, see [Using the CICS Programs to Virtualize Panel \(see page 837\)](#).

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

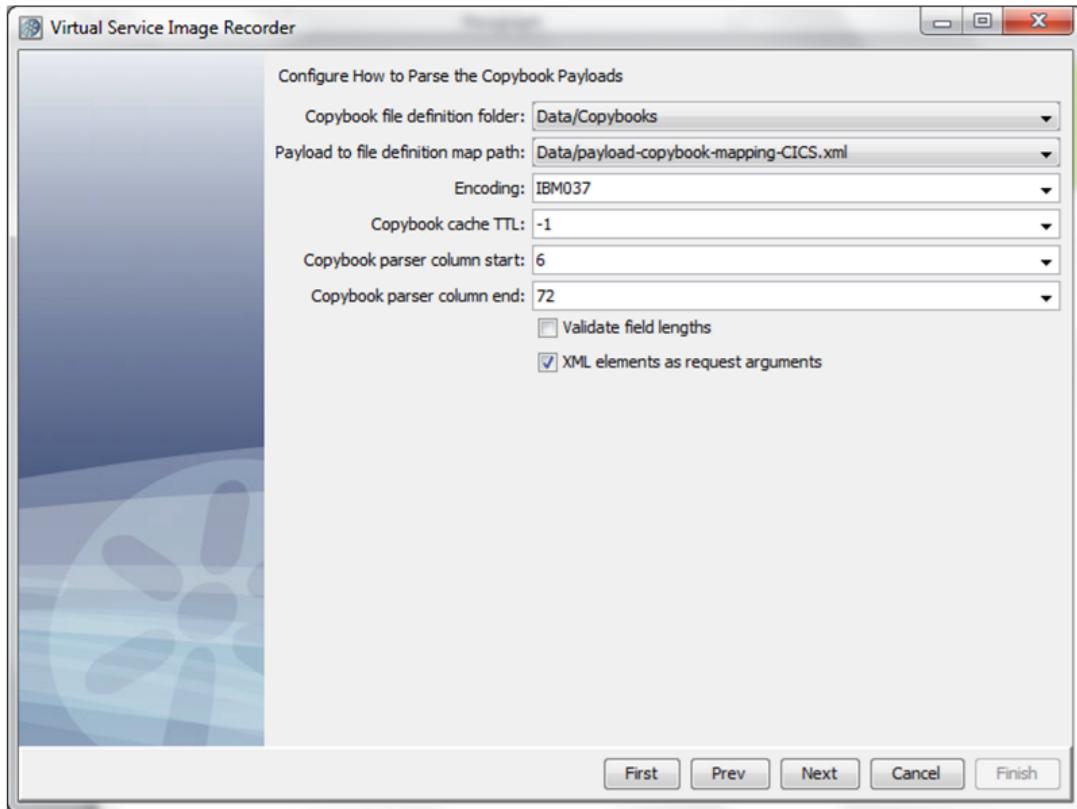
```
<?xml version="1.0" encoding="UTF-8"?>
<payloads>
    <!-- DEMOGETB -->
    <payload name="DEMOGETB" type="request" matchType="metaData"
        key="DEMOGETB" value="DEMOGETB" >
        <section name="Body">
            <copybook key="">DEMOGETB.txt</copybook>
        </section>
    </payload>
    <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:

Name	Name in Session	Comparison Operator	Value	Magic String	Date Pattern
called_by	=		DEMOINQ	<input type="checkbox"/>	
Body_DEMOGETB-COMMAREA_DEMOGETB-RETURN	=			<input type="checkbox"/>	
Body_DEMOGETB-COMMAREA_DEMOGETB-MESSAGE	=			<input type="checkbox"/>	
Body_DEMOGETB-COMMAREA_BALANCE-REC_BALANCE-ACCOUNT-NBR	=		00000001	<input checked="" type="checkbox"/>	
Body_DEMOGETB-COMMAREA_BALANCE-REC_BALANCE-FILLER_1	=			<input type="checkbox"/>	
Body_DEMOGETB-COMMAREA_BALANCE-REC_BALANCE-BALANCE	=			<input type="checkbox"/>	
Body_DEMOGETB-COMMAREA_BALANCE-REC_FILLER_2	=			<input type="checkbox"/>	
Body_DEMOGETB-COMMAREA_BALANCE-REC_BALANCE-AVERAGE	=			<input type="checkbox"/>	

Screenshot of the request data arguments for the CICS LINK recording example.

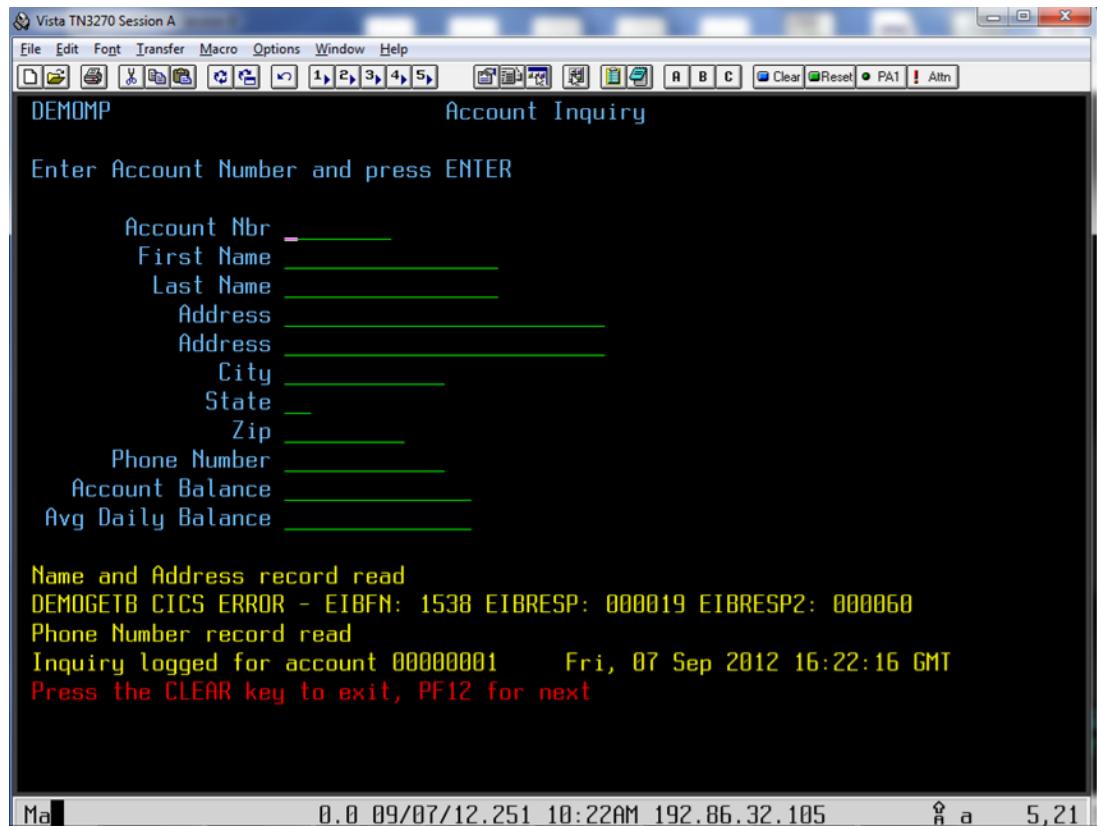
The copybook has also formatted the response. The following graphic shows the response to return when using virtualization:

Node	Type	Occurs	Nil	Nillable	Value
copybook-payload					
Body					
DEMOGETB-COMAREA					
DEMOGETB-RETURN					
left-pad-length					0
origin-length					1
DEMOGETB-MESSAGE					
left-pad-length					0
origin-length					70
BALANCE-REC					
BALANCE-ACCOUNT-NBR					{{<request_Body_DEMOGETB_COMMAREA_BALANCE_REC_BALANCE_ACCOUNT_NBR;/\"00000001\";}}
left-pad-length					0
origin-length					8
FILLER					
left-pad-length					0
origin-length					1
BALANCE-BALANCE					\$1,000.00
left-pad-length					0
origin-length					14
FILLER					
left-pad-length					0
origin-length					1
BALANCE-AVERAGE					\$962.16
left-pad-length					0
origin-length					14

Screenshot of the response to send back when using virtualization for the CICS LINK recording example.

Example

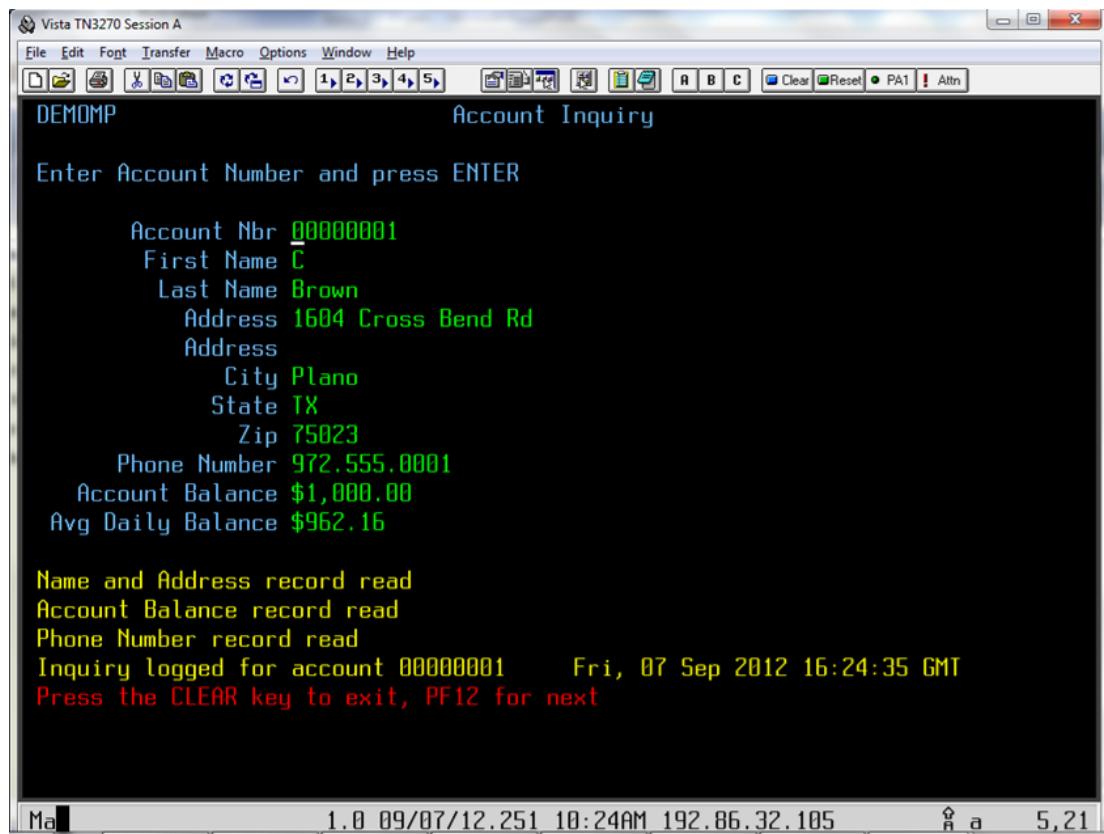
In this example, the VSAM file DEMOBAL is made unavailable, and program DEMO3 is run. DEMO3 fails because of the unavailability of DEMOBAL.



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.



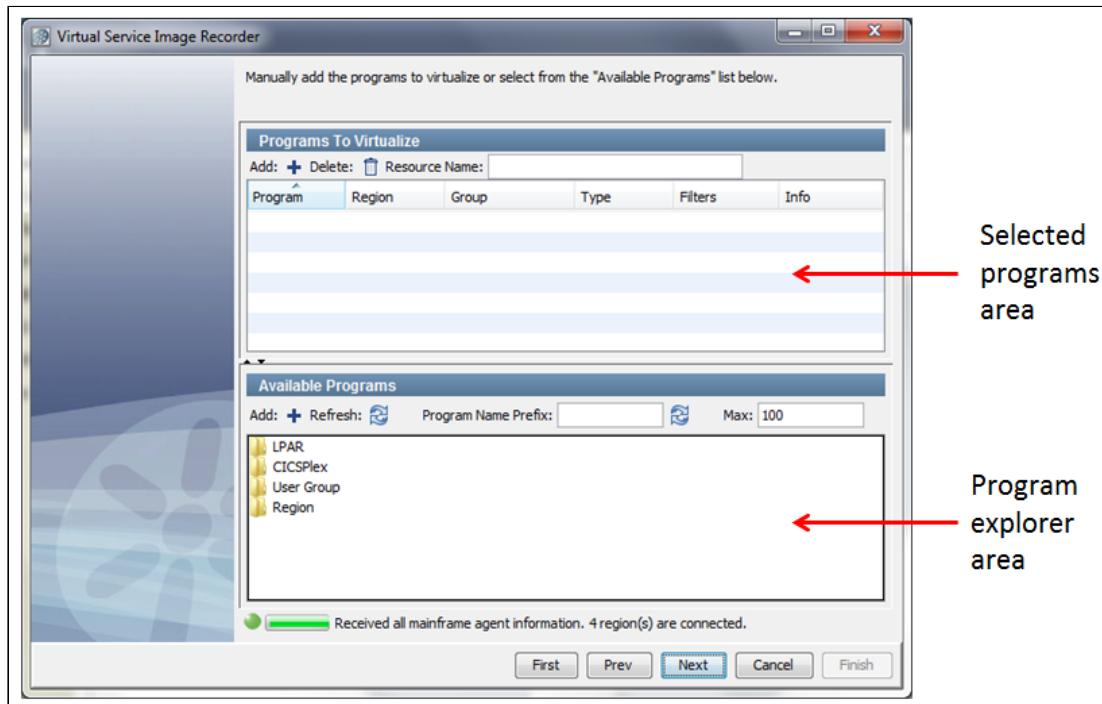
Screenshot of the CICS screen for the CICS LINK recording example, with success having VSE virtualize the SUT.

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

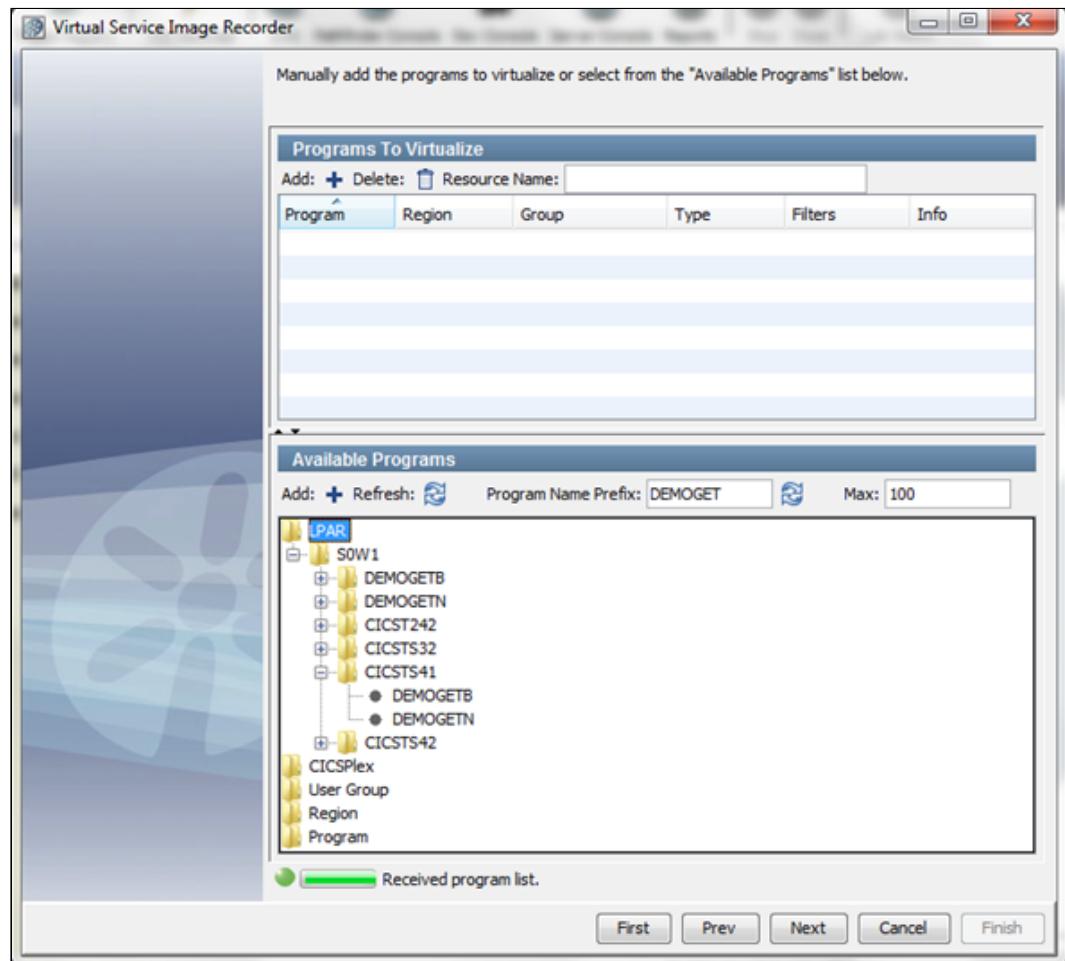
The **Available Programs** area lets you explore CICS regions that are connected to DevTest with the DevTest CICS Agent and see the programs that are defined there.

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.



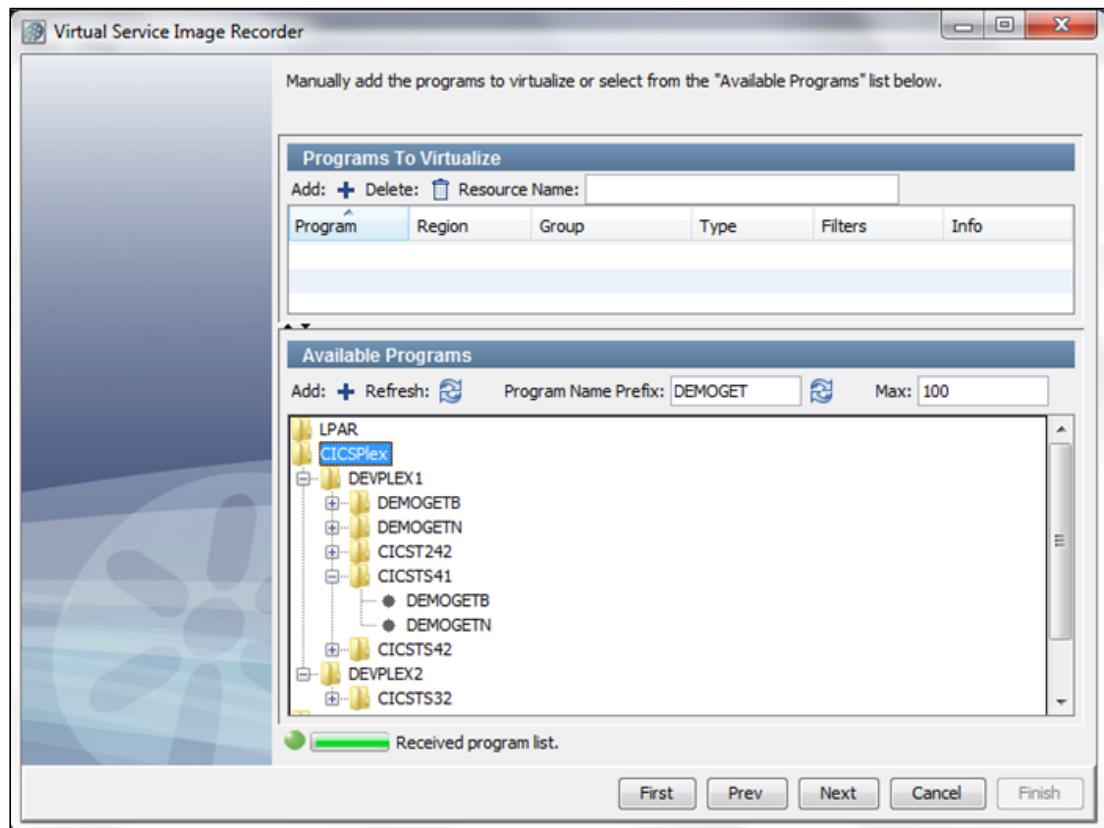
The CICS Programs to virtualize selection panel with the CICS LINK recording example environment shown.

Clicking LPAR shows the known LPARs (in this case, SOW1).

Clicking the plus sign next to SOW1 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 CICSplexes and reveals DEVPLEX1 and DEVPLEX2.

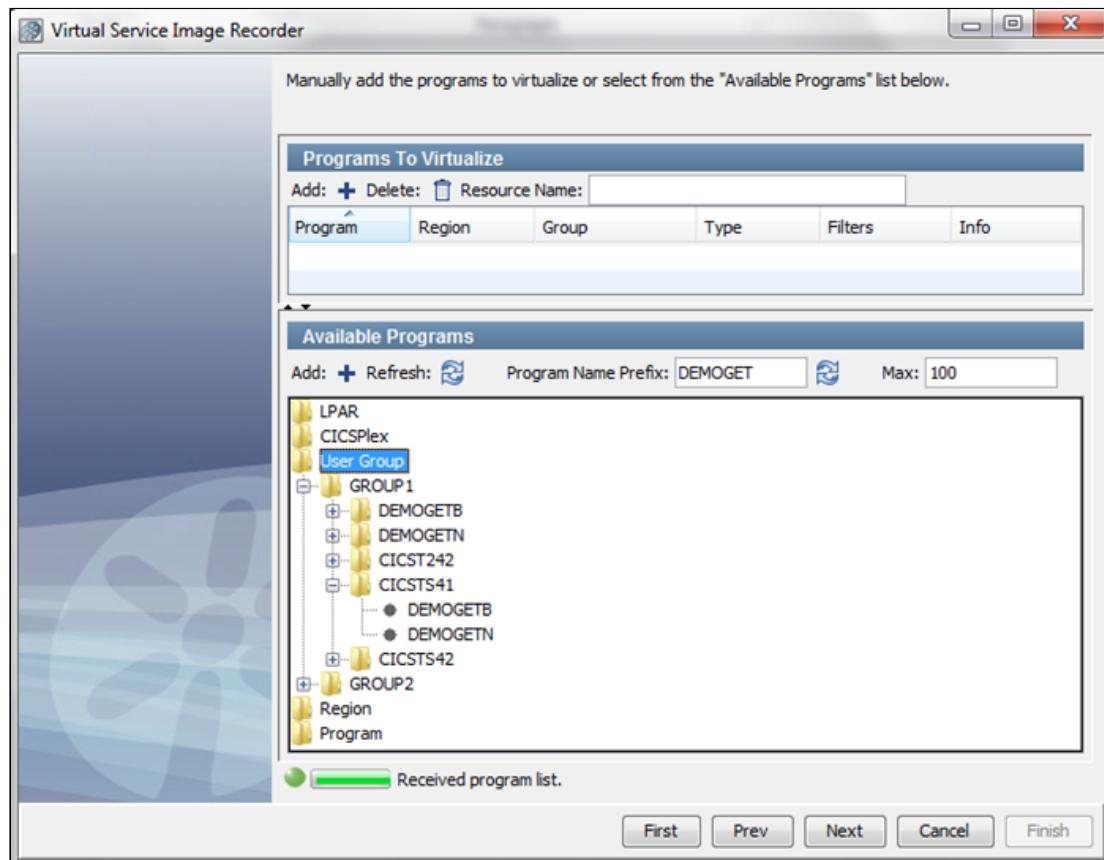
Clicking the plus sign next to DEVPLEX1 shows the known CICS regions: CICST242, CICSTS41, and CICSTS42.

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 \(see page 1325\)](#).



The CICS Programs to virtualize selection panel with User Groups expanded.

Clicking **User Group** shows the known user groups and reveals GROUP1 and GROUP2.

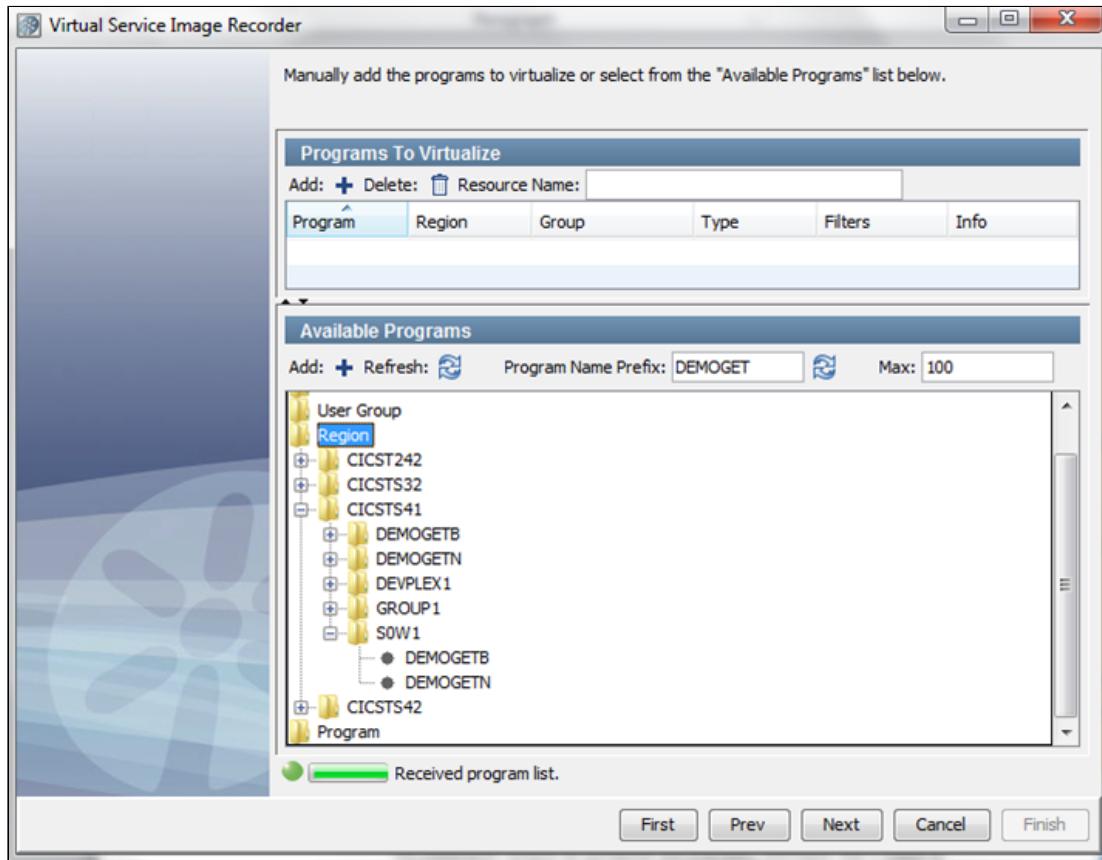
Clicking the plus sign next to GROUP1 shows the known CICS regions: CICST242, CICSTS41, and CICSTS42.

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 \(see page 1325\)](#).



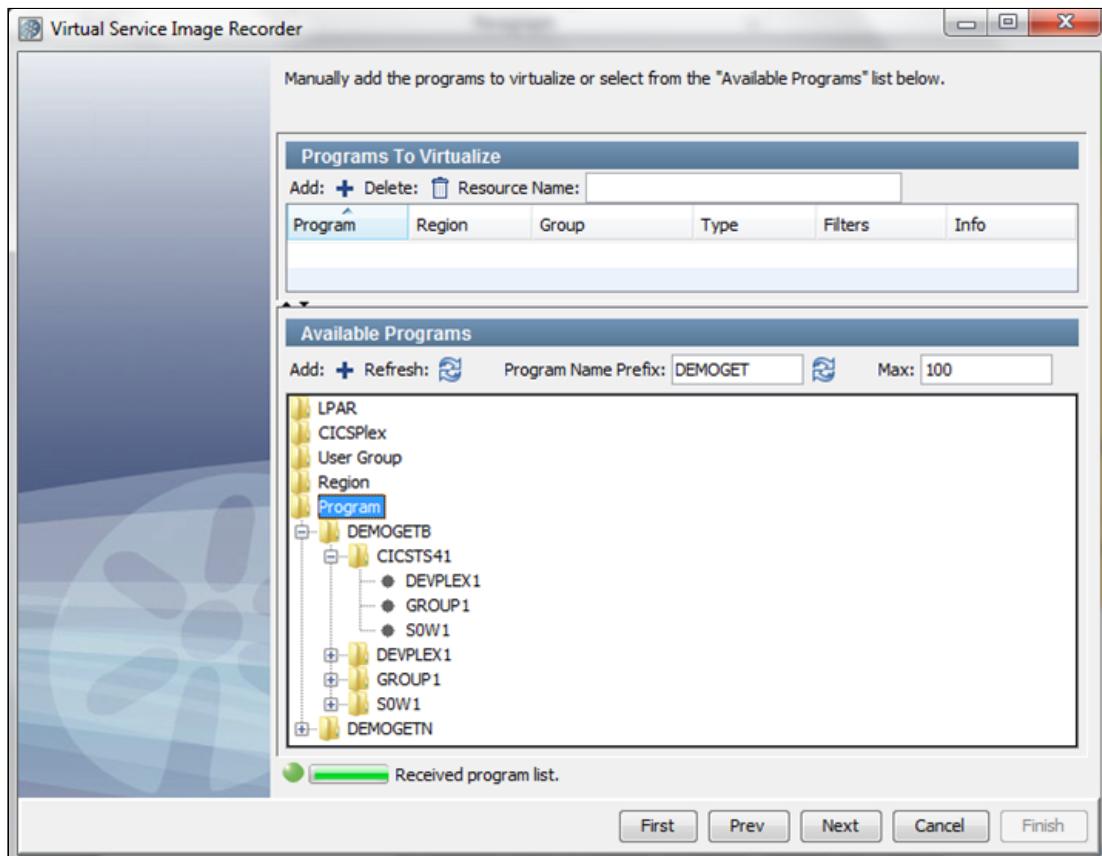
The CICS Programs to virtualize selection panel with Region expanded.

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

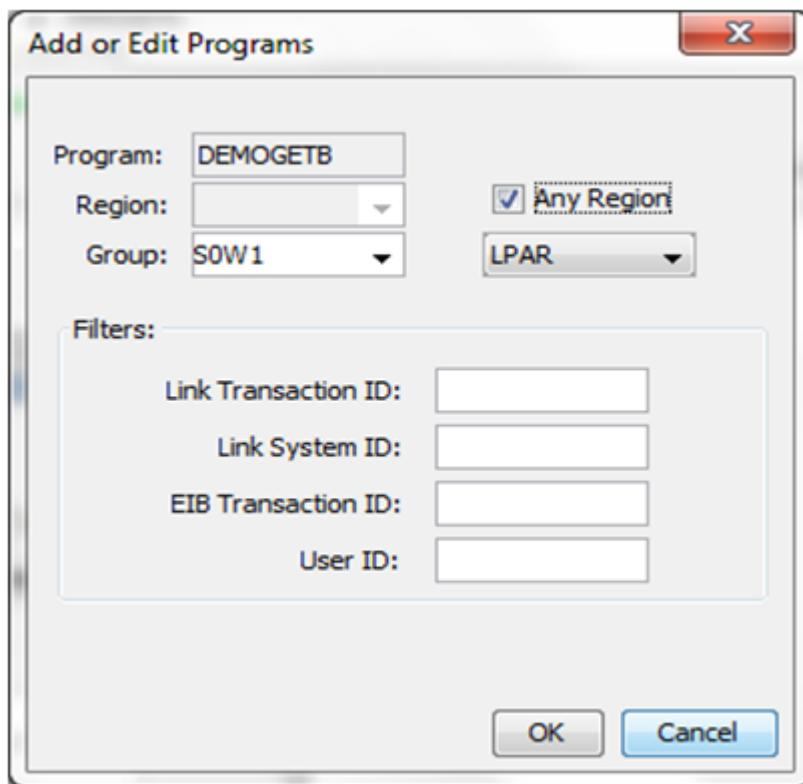
To add programs, double-click them and they appear in the **Programs to Virtualize** panel. Or, click

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.

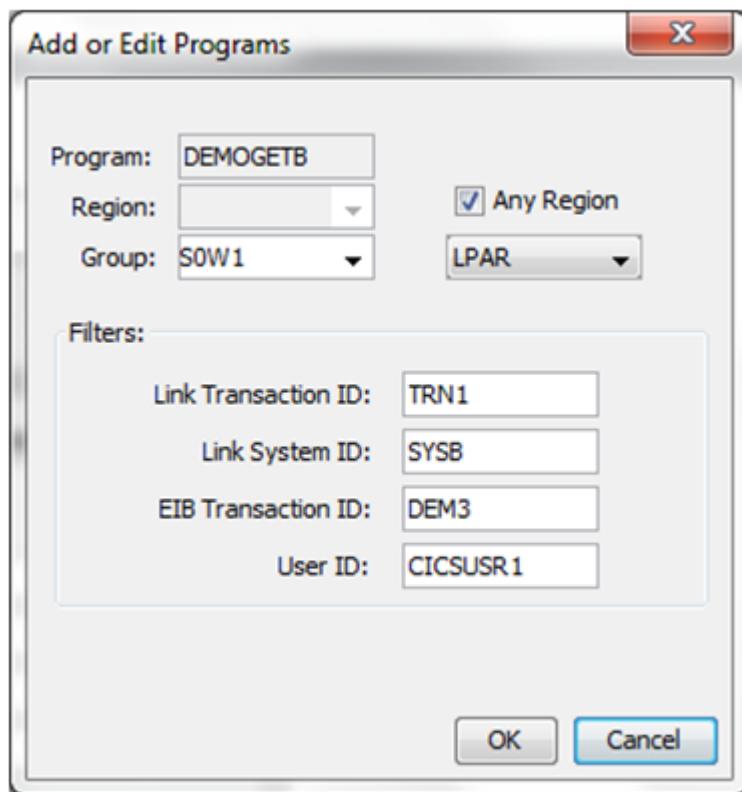
In this example, program DEMOGETB is virtualized in CICS region CICSTS41. Group SOW1 and LPAR are ignored unless the Any Region check box is selected.

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.

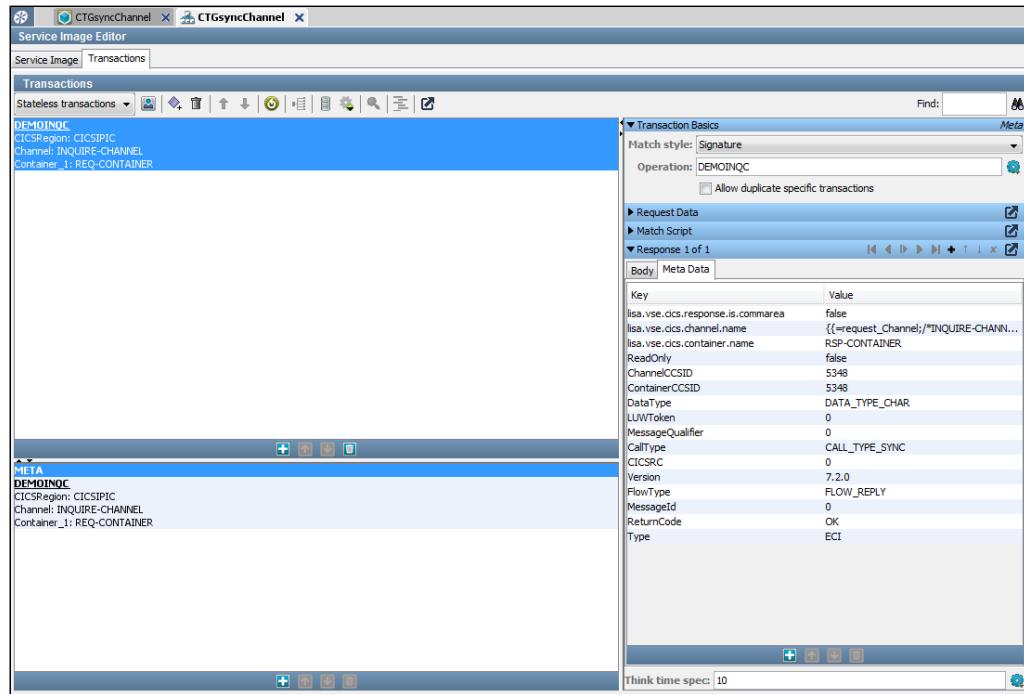
CICS Transaction Gateway (ECI) Transport Protocol

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.



Service Image Editor with CTG transactions.

IMS Connect Transport Protocol

Contents

- [IMS Connect Customized Format Files \(see page 849\)](#)

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.

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 applicatio
ns
# that communicate with IMS TM.
#-----

RequestHeaderCommon {
    LLLL           int;          #total message length I
    RM + user data
    IRMFixedHeader {
        LL           short;       #IRM_LEN, total length
        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_RSNODE
        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 won't
#-----

----- ResponseMessageHeader {
    LLLL          int;                                # followed by multiple R
    esponsePayloadSegment
}

#-----
----- # 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

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.

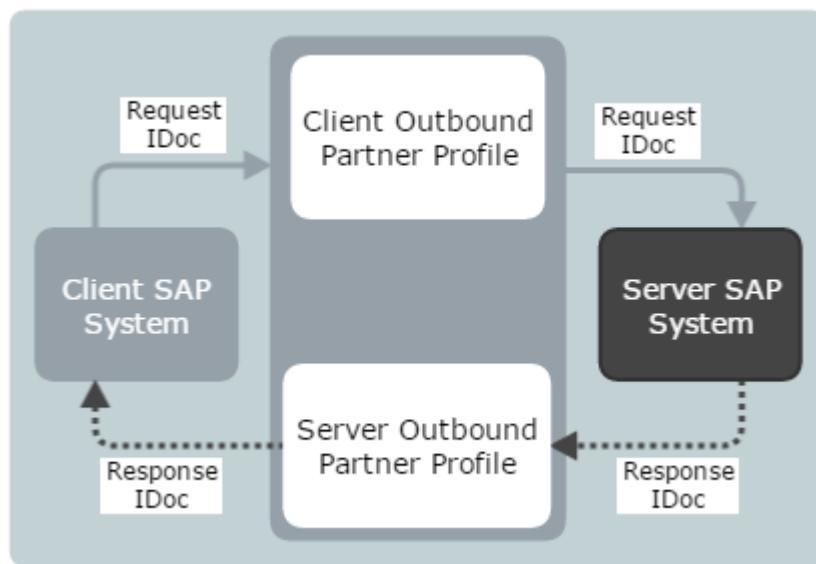
The SAP JCo IDoc transport protocol allows you to create virtual services that use the SAP JCo IDoc protocol.

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 \(see page 1380\)](#).

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.

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 \(see page 824\)](#).

Follow these steps:

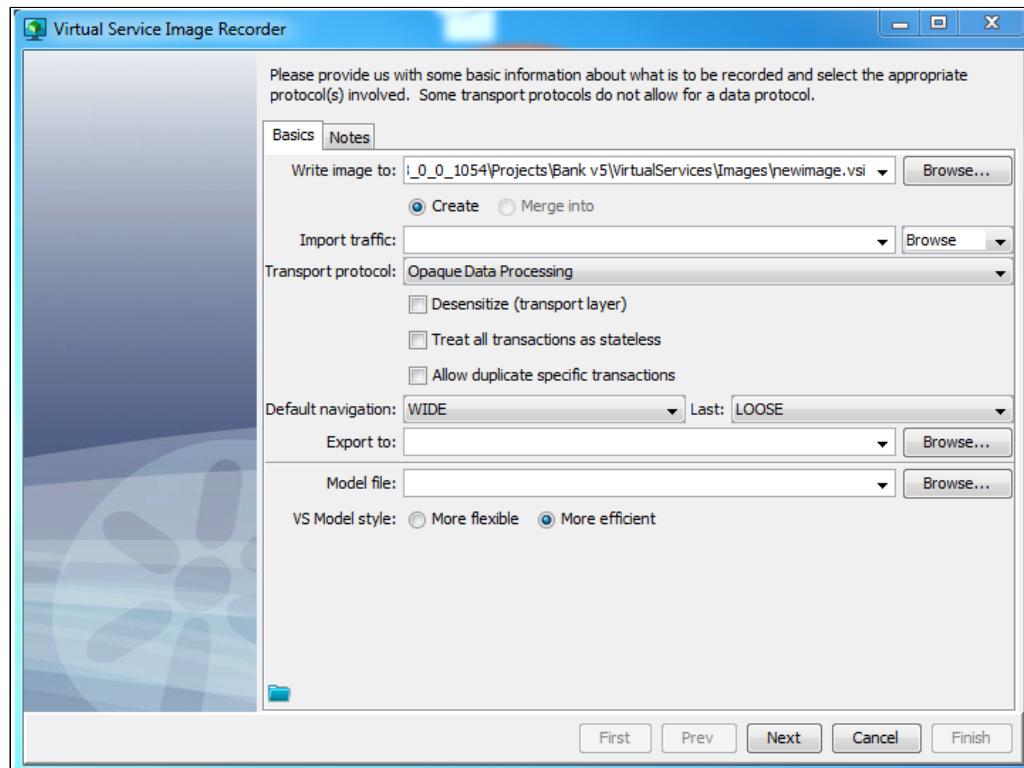
1. Open DevTest Workstation.

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**.

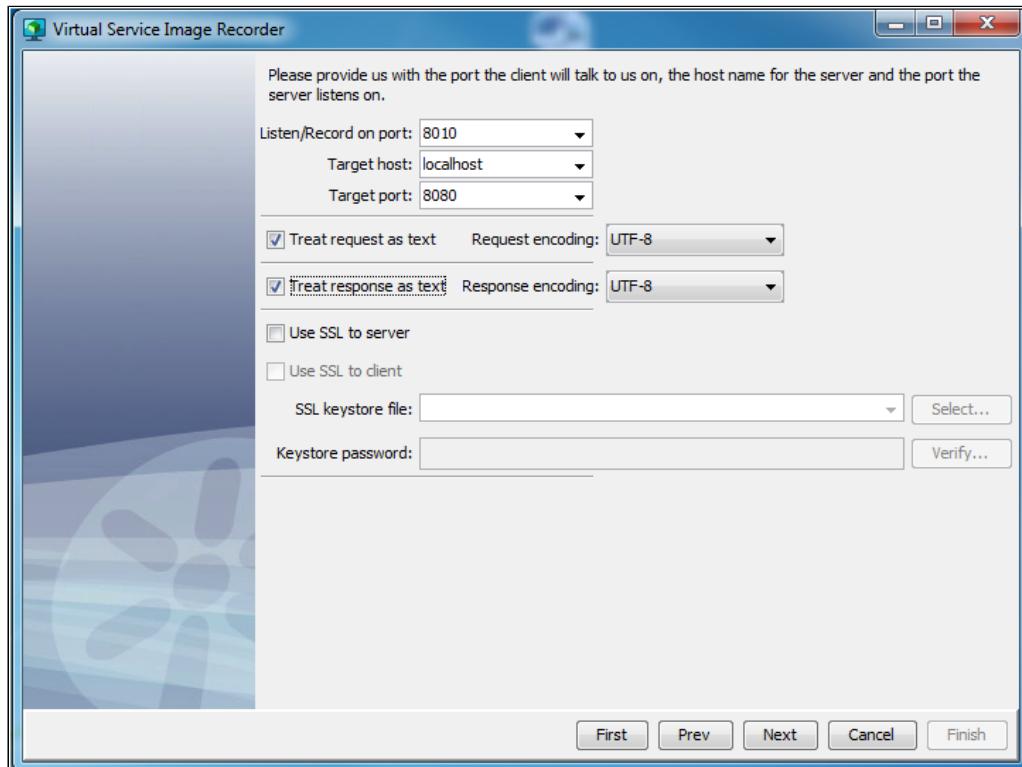
The **Virtual Service Image Recorder** opens.

3. Complete the **Basics** (see page 775) 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.

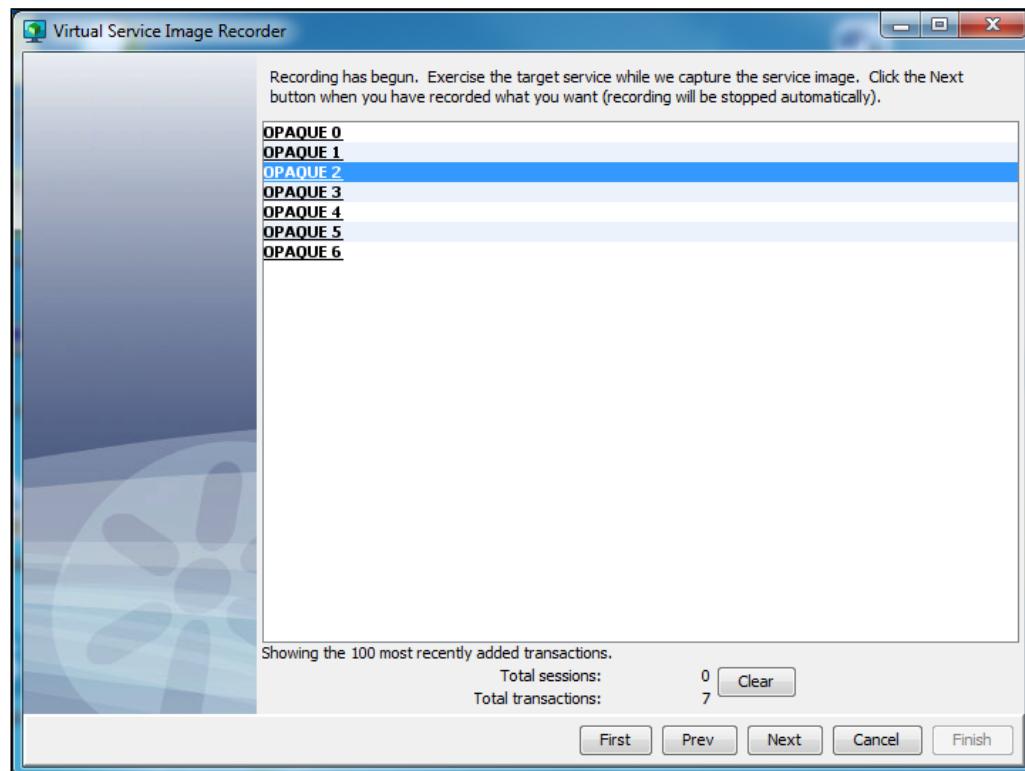


Client and Server Data for ODP

- **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 \(see page 824\)](#).
- **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 \(see page 824\)](#).
- **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.

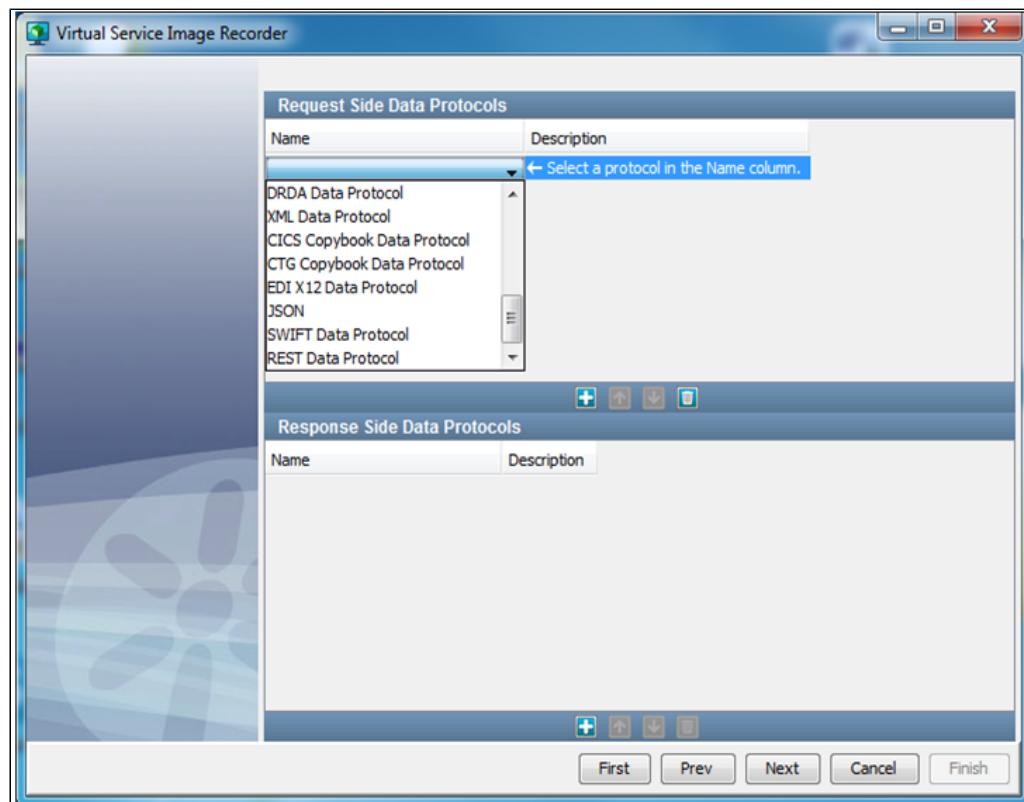
6. Click **Next** to start recording.



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.

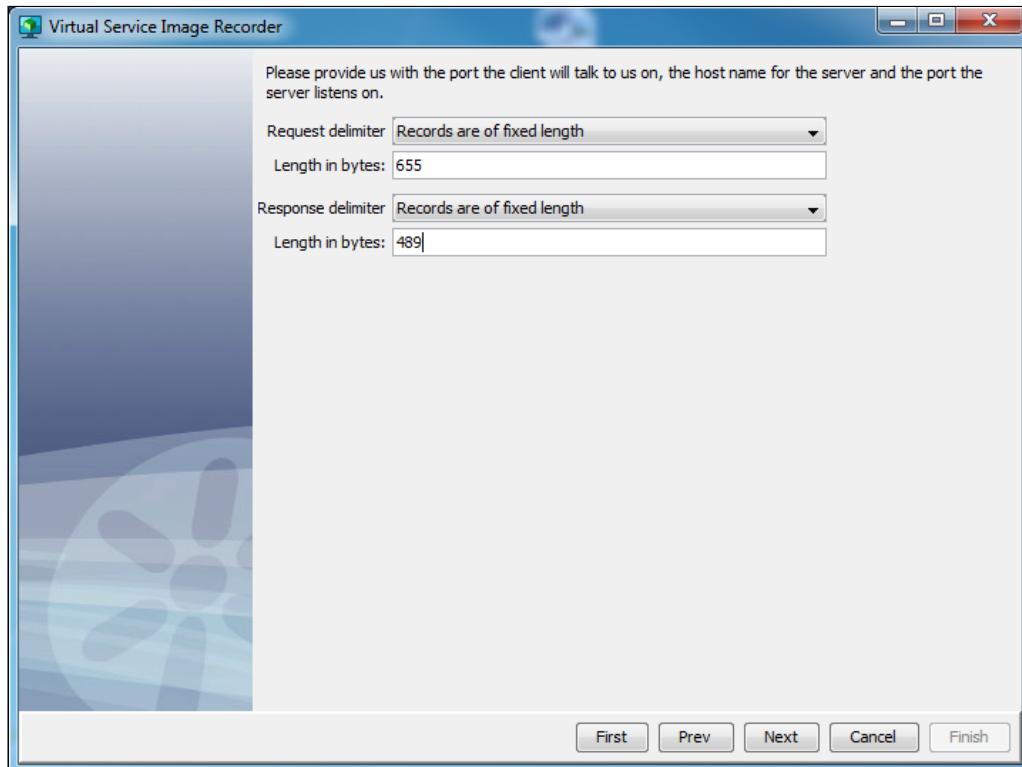


Selecting data protocols for ODP recording

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 delimiters 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.

More information:

- Video about [Opaque Data Processing, Part 1](https://www.youtube.com/watch?v=C-AJBF5GkiA&index=3&list=PLUo5-4tnpYJmxYRsnMkYSWPTdQ-vrKd8) (<https://www.youtube.com/watch?v=C-AJBF5GkiA&index=3&list=PLUo5-4tnpYJmxYRsnMkYSWPTdQ-vrKd8>)
- Video about [Opaque Data Processing, Part 2](https://www.youtube.com/watch?v=qtQRSmI_x5w&index=4&list=PLUo5-4tnpYJmxYRsnMkYSWPTdQ-vrKd8) (https://www.youtube.com/watch?v=qtQRSmI_x5w&index=4&list=PLUo5-4tnpYJmxYRsnMkYSWPTdQ-vrKd8)

SSL with VSE Recording

During VSE recording, the SSL Server application is the System Under Test.

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 (<http://lisa.net>) .trustStore**
The full path to the custom trust store file to use
- **lisa.net (<http://lisa.net>) .trustStore.password**
The password for the custom trust store file. This property is automatically encrypted and replaced by the lisa.net (<http://lisa.net>).trustStore.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}}/webreckey.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}}/webreckey.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}/webrekeys.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}/webrekeys.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

Contents

- [Create a VSM \(see page 862\)](#)
- [Open a VSM \(see page 862\)](#)

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.

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. 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 payload of the request. One transport protocol can have many data protocols. Some transport protocols (for example, JDBC) do not allow a data protocol.



Note: A data protocol is also known as a data handler or a data protocol handler.

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 Delimited Text data protocol, the Generic XML Payload Parser data protocol, and the Request Data Manager data protocol together.

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

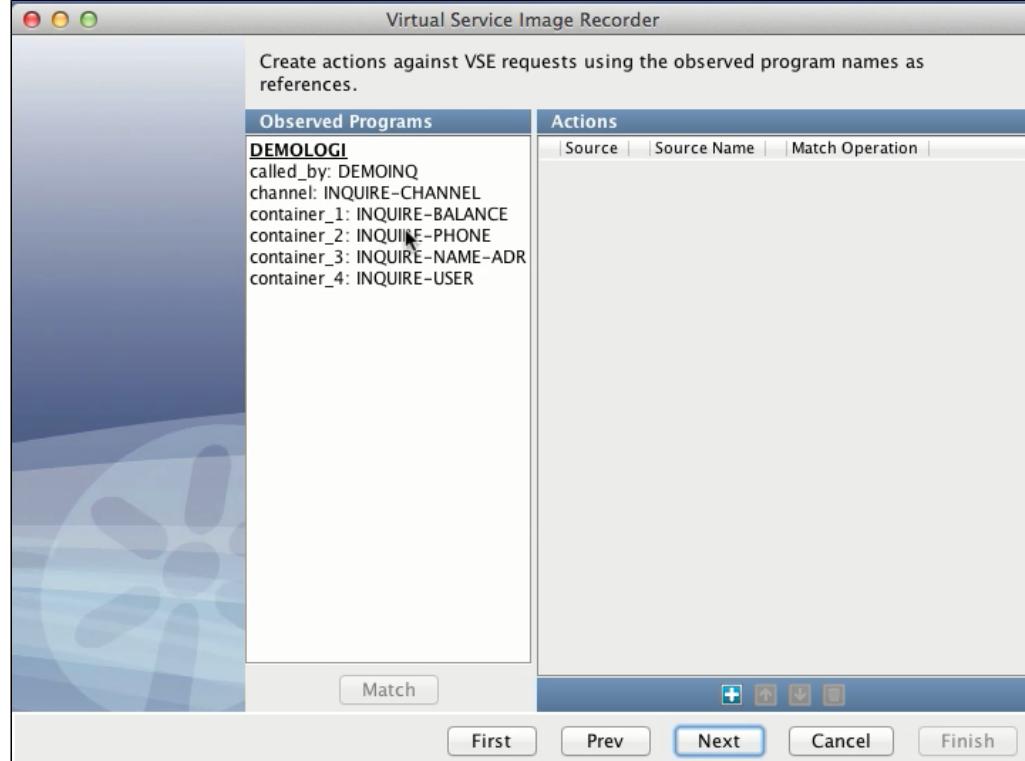
The CICS Copybook data protocol splits the recorded request into its respective container chunks. Each chunk is then sent to the Copybook data protocol and aggregated with the corresponding XML. The resulting XML is the aggregated XML of the containers.

This example demonstrates the CICS Copybook data protocol usage.

Follow these steps:

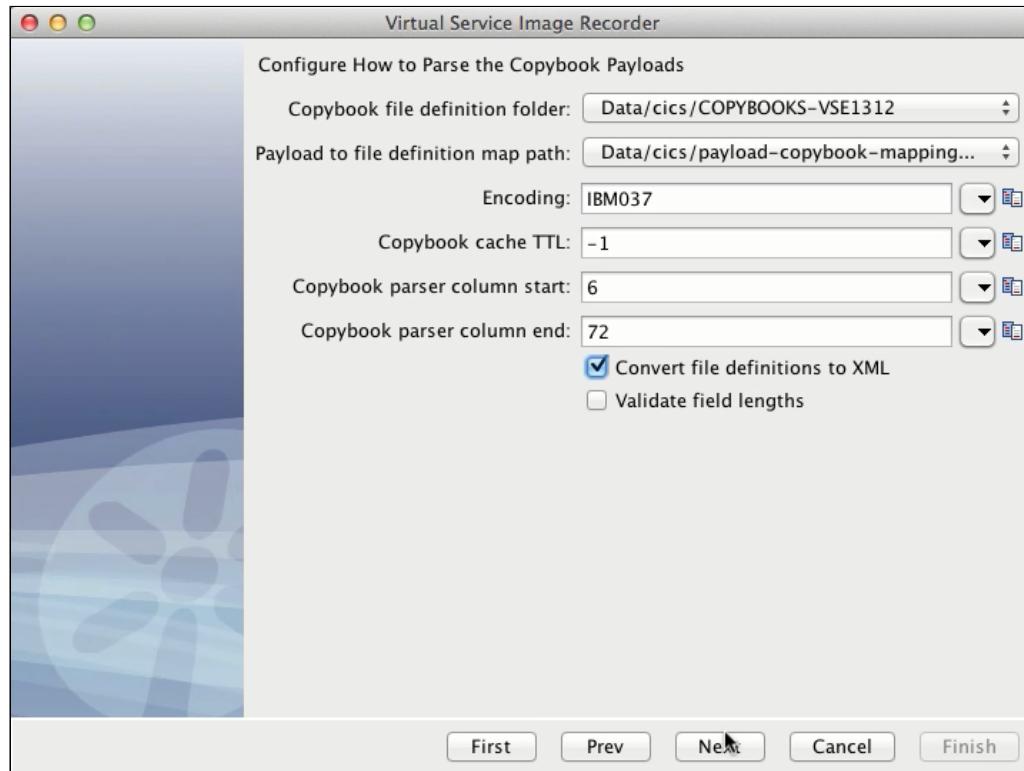
1. Complete the required fields on the **Virtual Service Image Recorder Basics** (see page 775) tab.
2. Click **Next**.
3. Enter an IP address and port, then click **Next**.
The **Data Protocol selection** window opens.
4. Select the CICS Request Data Access data protocol on the request side.
5. Select 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**.

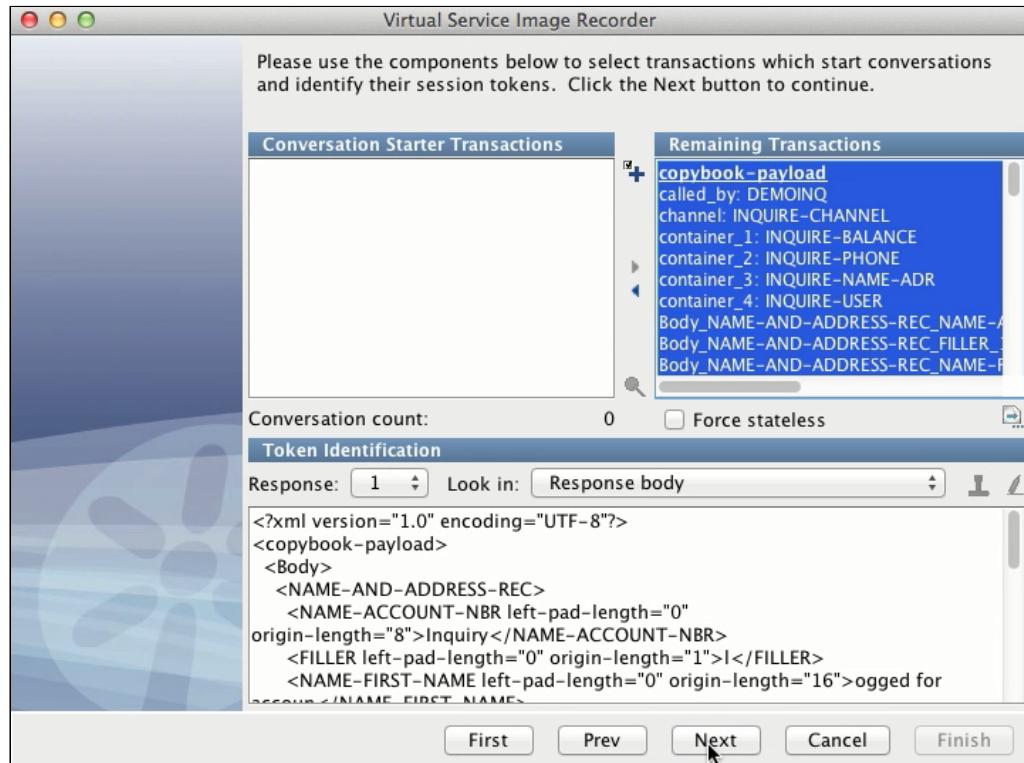


Screenshot of CICS Copybook data protocol How to Parse screen.

7. Enter the following parameters on the request side definitions:

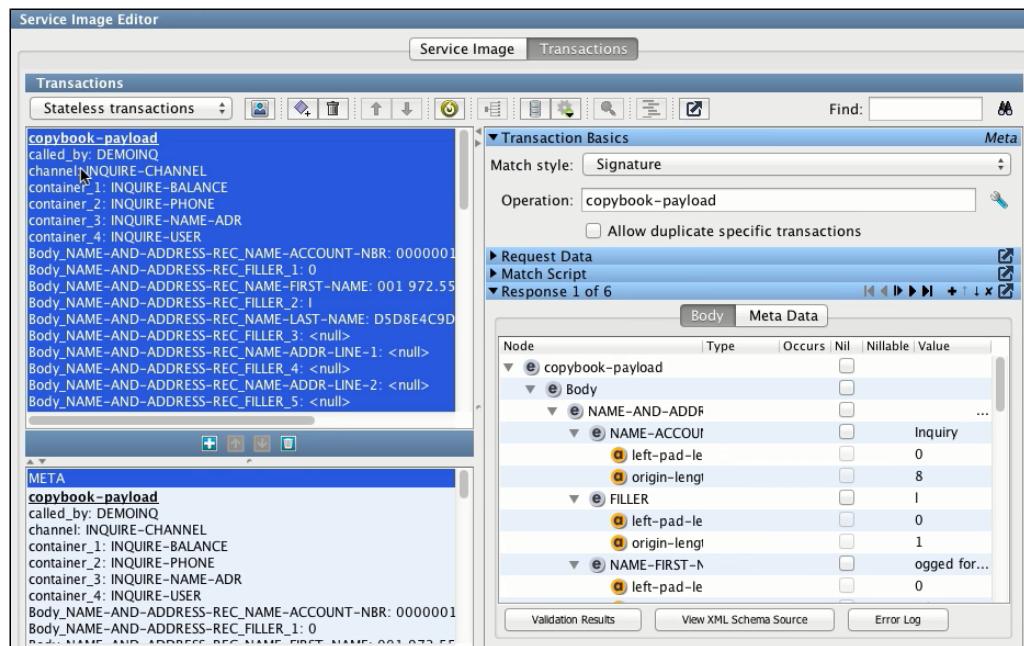
- **Copybook file definition folder**
Specifies the folder for the copybook file definition.
- **Payload to file definition map path**
Specifies the path for the payload to file definition map.
- **Encoding**
Specifies the encoding information.

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 \(see page 870\)](#).
- 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 \(see page 750\)](#).

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](http://docs.oracle.com/javase/1.4.2/docs/api/java/nio/charset/Charset.html) (<http://docs.oracle.com/javase/1.4.2/docs/api/java/nio/charset/Charset.html>). 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 \(see page 886\)](#).

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>
```

At Re Description

tri qu
bu ire
te d
/O
pti
on
al

na Re Defines a unique name to identify the type of request described here. **Name** must be

m qu unique among the set of payloads of the same type (that is, request or response).

e ire
d

ty Re Specifies the payload type.

pe qu
ire **Value:** One of the following:
d
request

response.

m Op The matchType attribute applies differently to payload definitions of type "response." For **at** tio example, responses do not contain arguments or operation names. If a payload definition

ch nal of type "response" sets this attribute to **argument**, **attribute**, or **operation**, it will never

Ty match anything (unless the matching request sets the definesResponse attribute to **true**,

pe 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

ke Re The behavior of this attribute depends as follows on the matchType:

y qu

ire If matchType is **operation** or **payload**, the value of this attribute is used for matching.
d

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.

va Op The behavior of this attribute (and whether it is required) changes depends as follows on

lu tio the matchType:

e nal

/R If the matchType is **operation** or **payload**, the value of this attribute is ignored and can be eq excluded.

uir

ed 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**).

ke Op Designates the position where the search for the key should start in the payload (1-based
yS tio index). The search is **inclusive** of this value.

ta nal

rt 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.

ke Op Designates the position where the search for the key should end in the payload (1-based
yE tio index). The search is **exclusive** of this value.

nd nal

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.

he Op 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.

By

te

s

fo Op 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.

er

By

te

s

sa Op 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.

He

ad **Values:**

er

Fo true: Persists the stripped header and footer bytes.

ot

er false: Does not persist the stripped header and footer bytes.

Default value: false.

de Op 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**.

es

Re **Default value:** false.

sp

on

se

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>
...

```

Attri Req Description**bute uire****d**

nam Req 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.

d**■ <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>
...
```

A R Description**tt e****ri q****b u****u ir****t e****e d**

k R 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.

e**d**

o O 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.
I However, this affects performance because each copybook has to be checked.

Default: 0

m O 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".

x ti**o****n****a****l**

p O Defines a value to override the record name (that is, the root level in the copybook).

ti

n o

a n 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.

e l

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.

le O Defines how to split the payload so that the next record search begins in the correct place.**n p**

g ti If this attribute is not present, the processor attempts to determine the length of the **t o** copybook from the definition. If, for some reason, it cannot figure out the length, the **h n** processor assumes that the rest of the payload applies to this copybook, and ends processing - **a** after applying this copybook. The processor ignores this field if it is not an unsigned Display **fi l** numeric field.

el**d**

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.

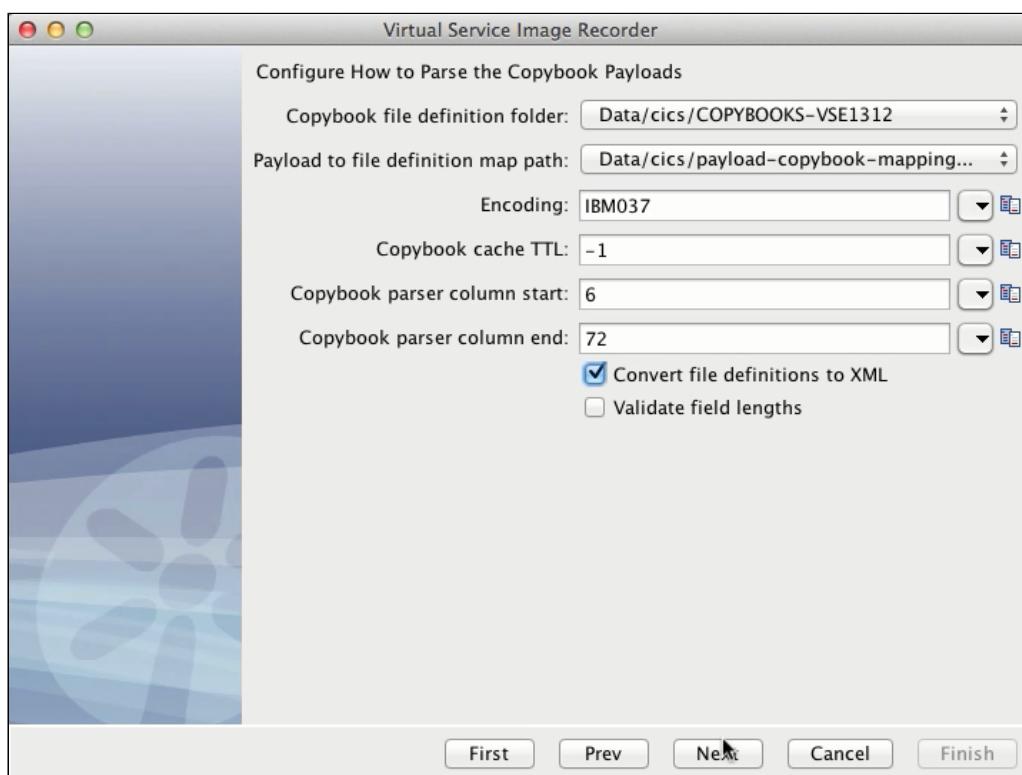
CTG Copybook Data Protocol

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** (see page 775) 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**.

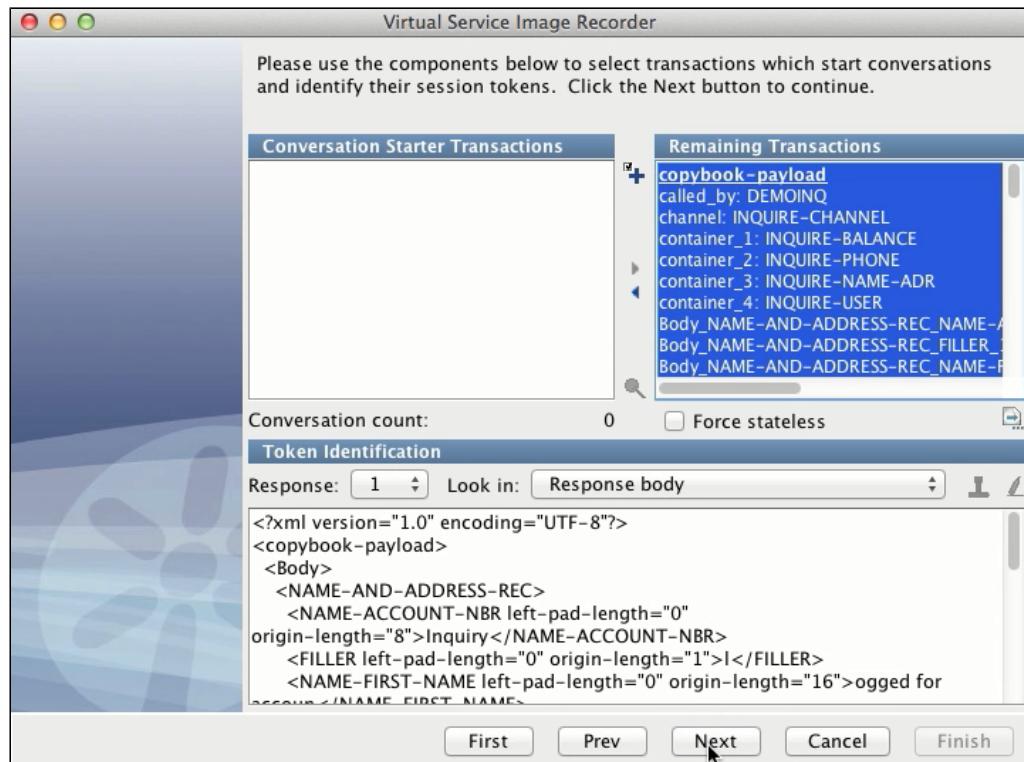


Screenshot of CICS Copybook data protocol How to Parse screen.

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 \(see page 775\)](#) tab of the **Virtual Service 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](#) (see page 886) (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:

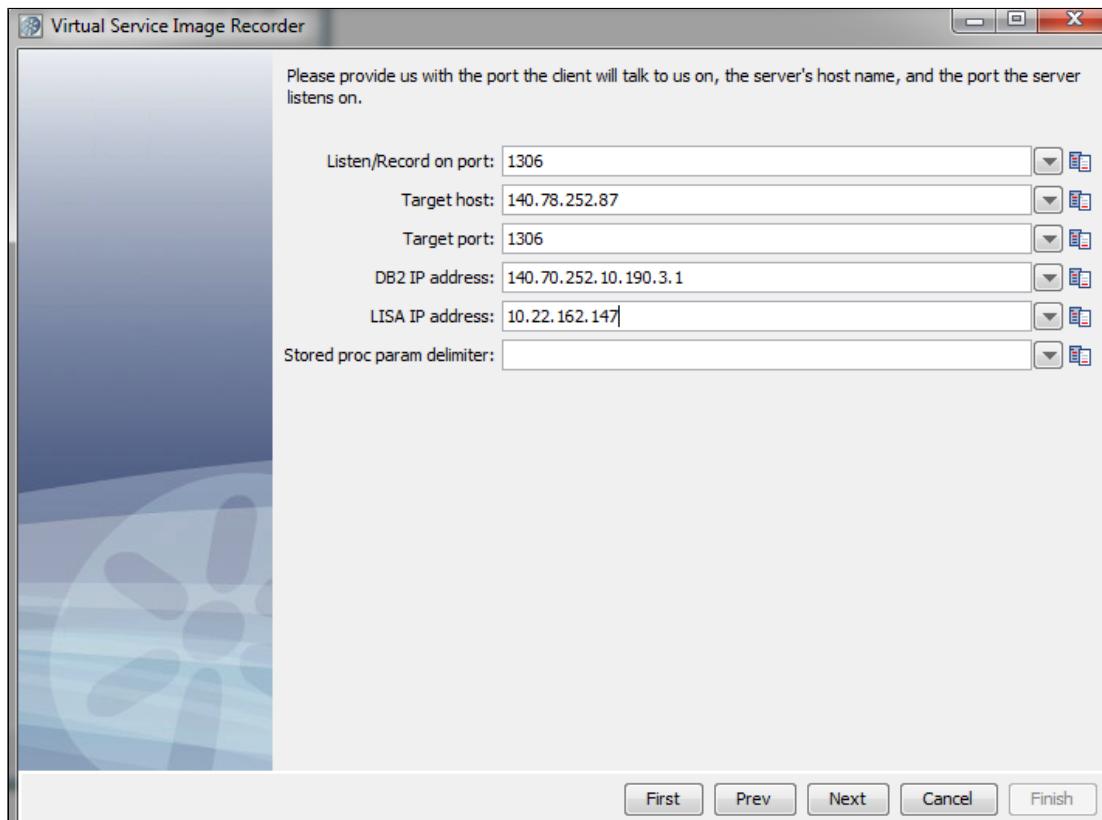
- [Generic XML Payload Parser Data Protocol \(see page 886\)](#)

DRDA Data Protocol

The DRDA data protocol converts binary Distributed Relational Database Architecture (DRDA) payloads to XML during recording. This protocol facilitates alignment with native DevTest functionality, readability, and dynamic data support. Responses are converted back to their native format on playback.

When you select the DRDA data protocol, the Request-side and Response-side data protocols fields are automatically populated with the DRDA data protocol selections.

The next window of the recorder lets you specify communications information.



Screenshot of DRDA data protocol communications screen.

Enter the communications parameters for the service image.



Note: If the payload contains non-ASCII characters, the payload is displayed in binary in the Service Image Editor.

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*P0*0011223456*999999999*950120*0147*5*X*003040$  
ST*850*000000001$  
BEG*00*SA*95018017***950118$  
N1*SE*UNIVERSAL WIDGETS$  
N3*375 PLYMOUTH PARK*SUIT 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$  

P01*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*0000000001$  

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="P01">
        <segment Id="P01">
            <element Id="P0101">001</element>
            <element Id="P0102">4</element>
            <element Id="P0103">EA</element>
            <element Id="P0104">330</element>
            <element Id="P0105">TE</element>
            <element Id="P0106">IN</element>
            <element Id="P0107">525</element>
            <element Id="P0108">VN</element>
            <element Id="P0109">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>
            </segment>
        </loop>
    </loop>

```

```

        <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 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](#) (see page 880), [Copybook Data Protocol](#) (see page 867), and [DRDA Data Protocol](#) (see page 882), 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.

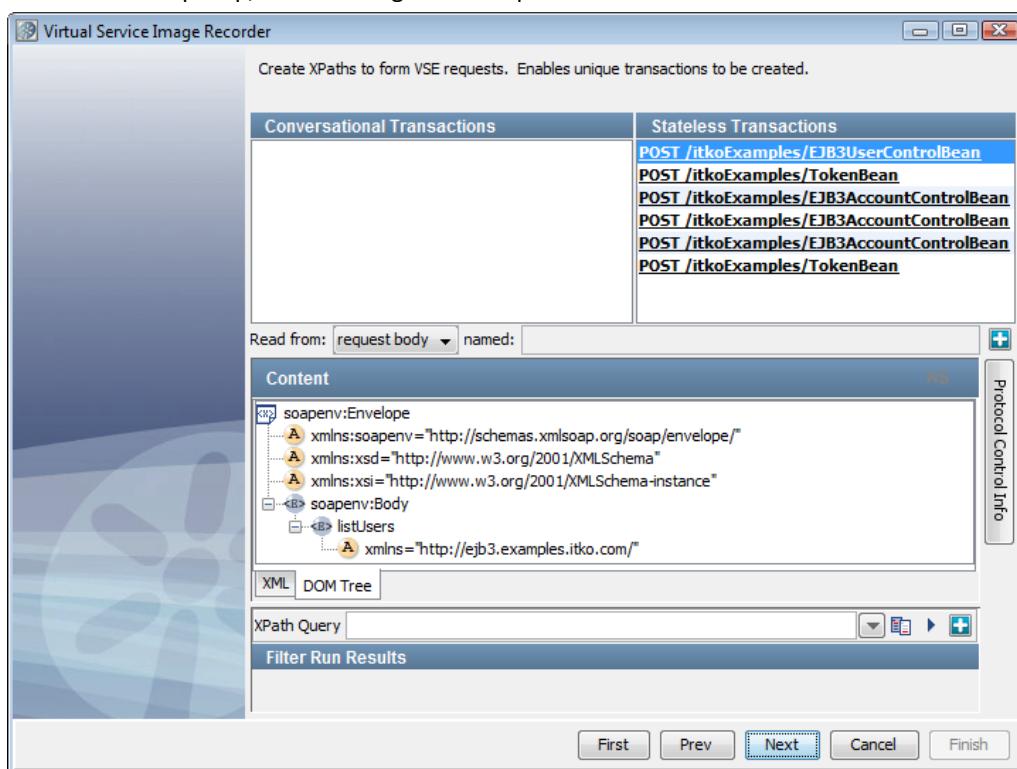
Using the Generic XML Payload Parser is a technique to help VSE inspect the body of the recorded messages (payload) and extract meaningful information from them to help identify the transactions. This technique can be the only way to get meaningful conversation information, especially for opaque protocols like 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:

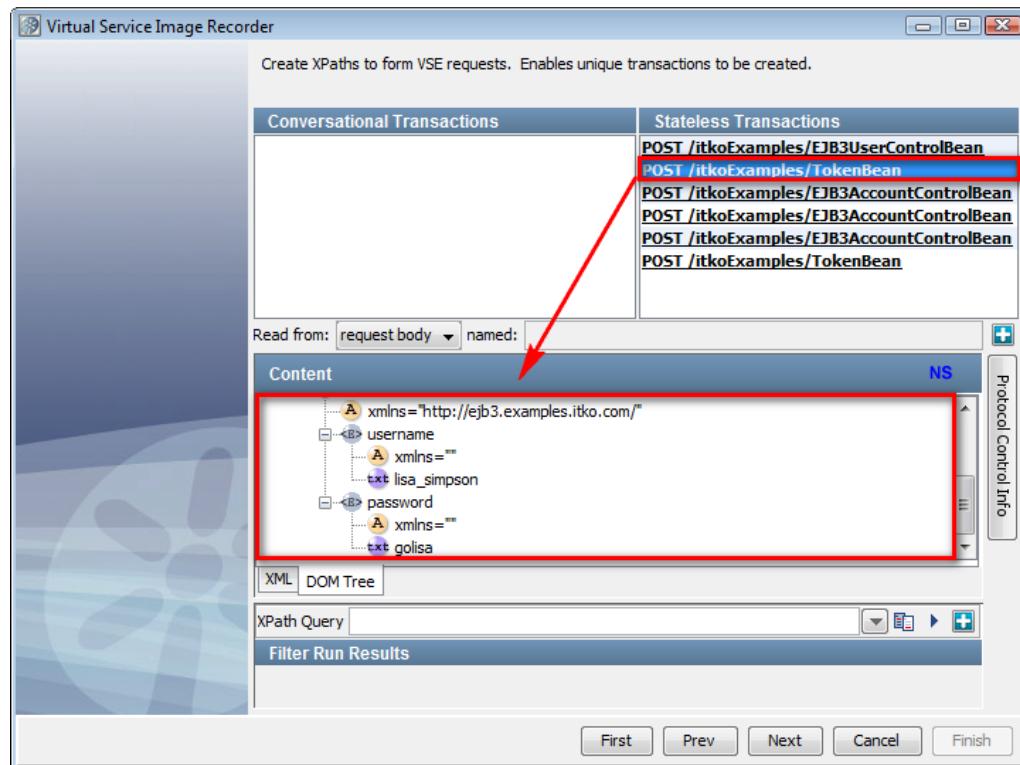
1. To use a dynamic data protocol, in the **Data Protocols** tab of the **VSI Recorder**, select **Generic XML Payload Parser**, if the payload is a well-formed XML.
2. Go through the rest of the steps, up to the cleanup step.
After the cleanup step, the following window opens:



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.



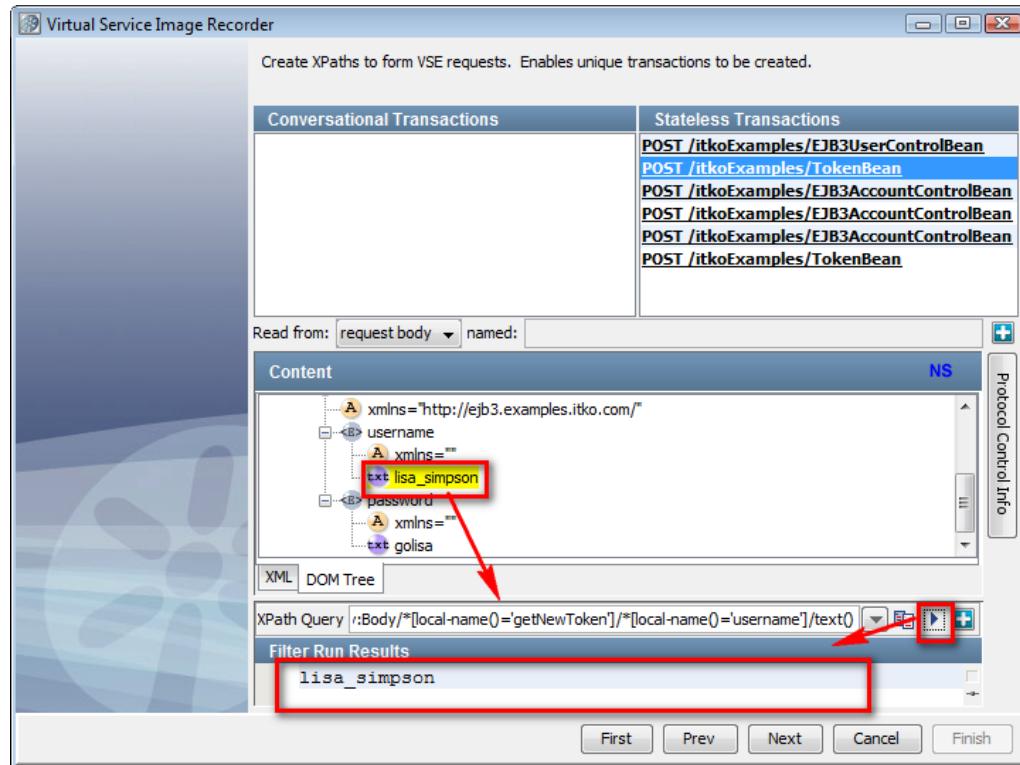
XML data protocol Create XPaths screen, marked up.

The XML tab in the **Content** area shows the classic XML view. The **DOM Tree** tab shows the payload in the tree format.



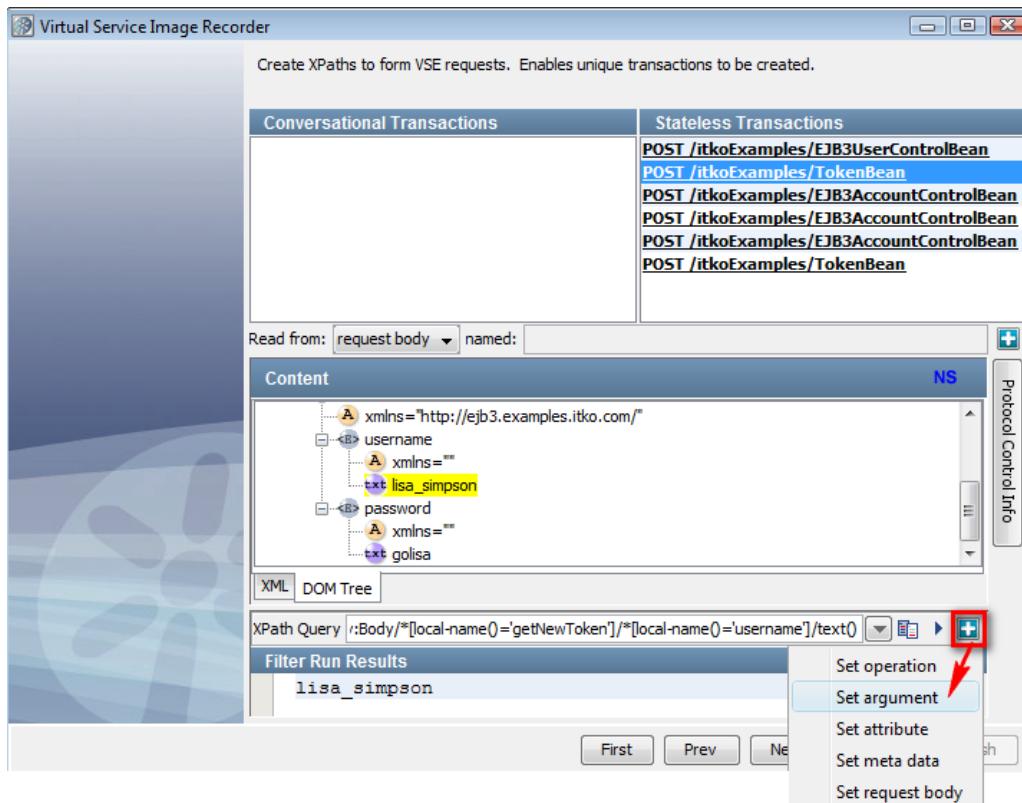
Note: This portion of the window is similar to the panel that appears when you create an XML XPath filter. As described in [XML XPath Filter \(see page 1620\)](#), you can use the `lisa.xml.xpath.computeXPath.alwaysUseLocalName` property to ignore the namespace.

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. Click **Next** when you are satisfied with your choice of variables. You are then 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. 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",  
   "unitNbr": 1000000000000000000}
```

```

    "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"}}
```

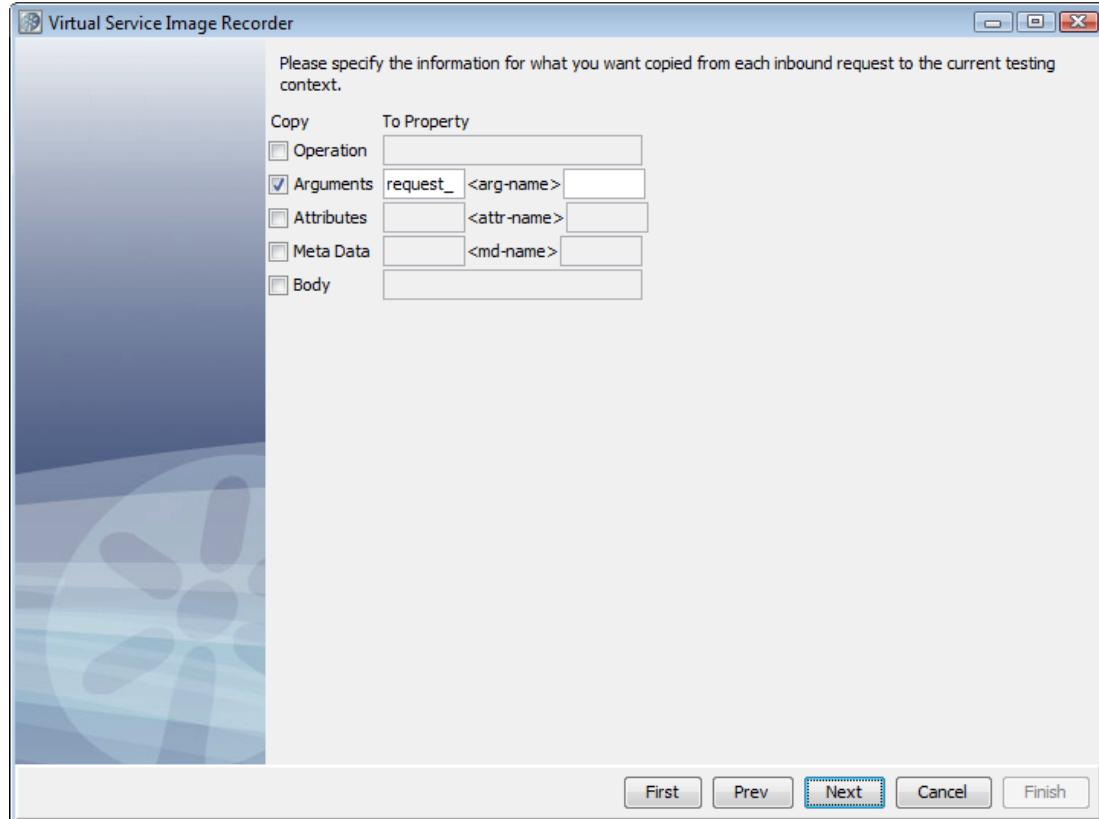
The JSON data protocol handler reorders the key/value pairs in an object so they are alphabetical by key name.



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

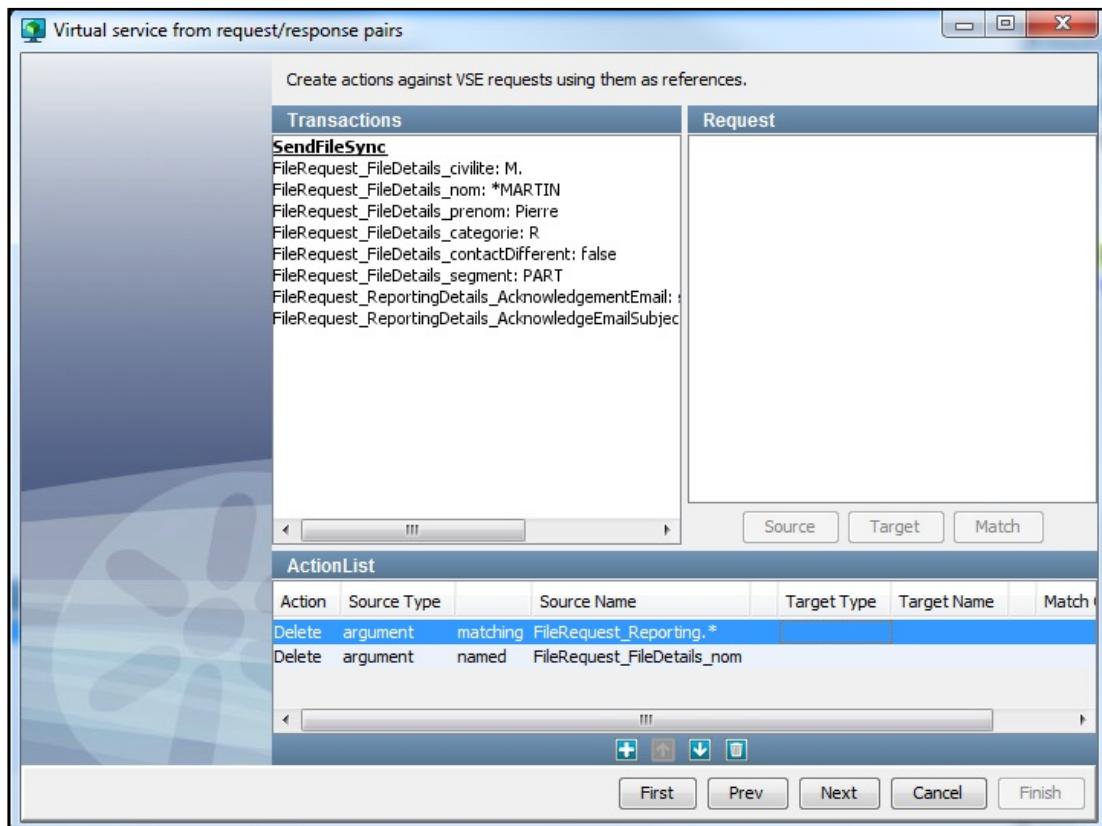
The Request Data Copier lets you copy data from the current inbound request into the current testing context. This capability is useful to more easily examine request information when custom behavior is required before the VSE response lookup step in the VS model.



Screenshot of the Request Data Copier data protocol copy request screen.

Request Data Manager Data Protocol

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

The Request Data Manager protocol lets you alter VSE requests during recording or playback.

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.

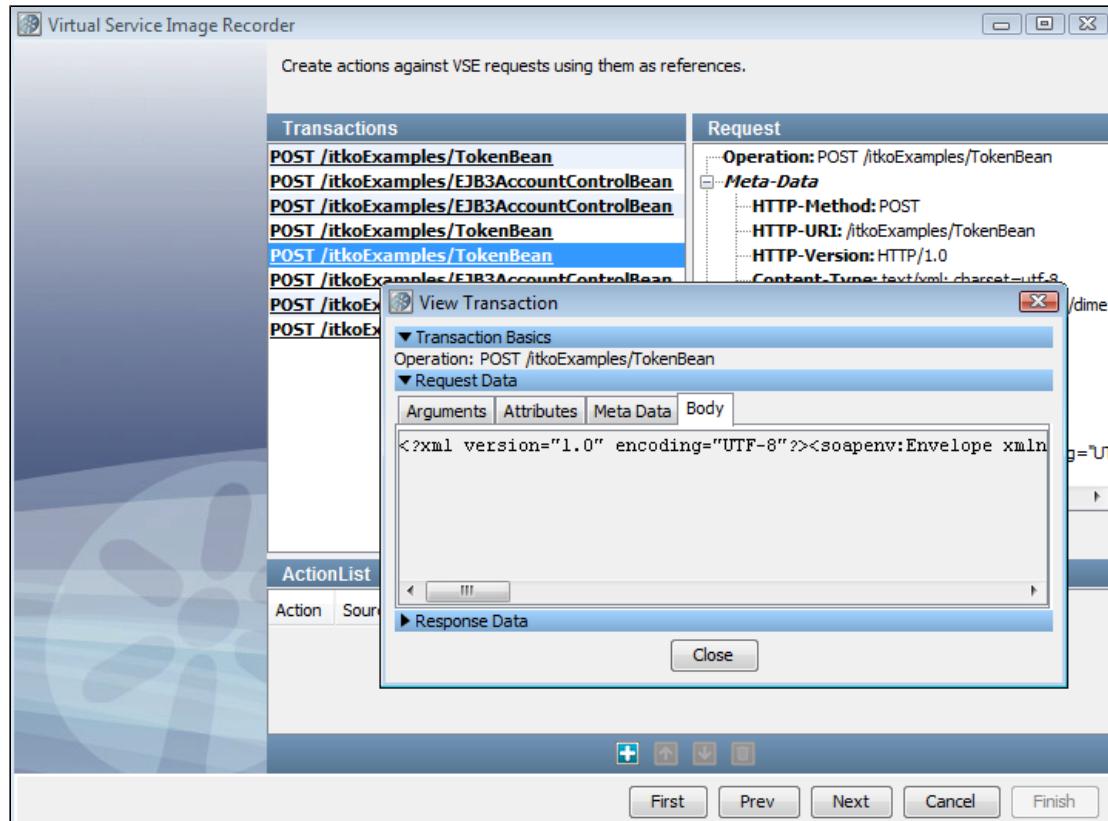


Note: The **Keep** action is most meaningful for arguments, attributes, and metadata. If you specifically keep a value from one of these three groups, any other value in that group that is not referenced by any action in the list for the data protocol is deleted. If you keep one argument, other arguments 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 DPH, 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 DPH 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.

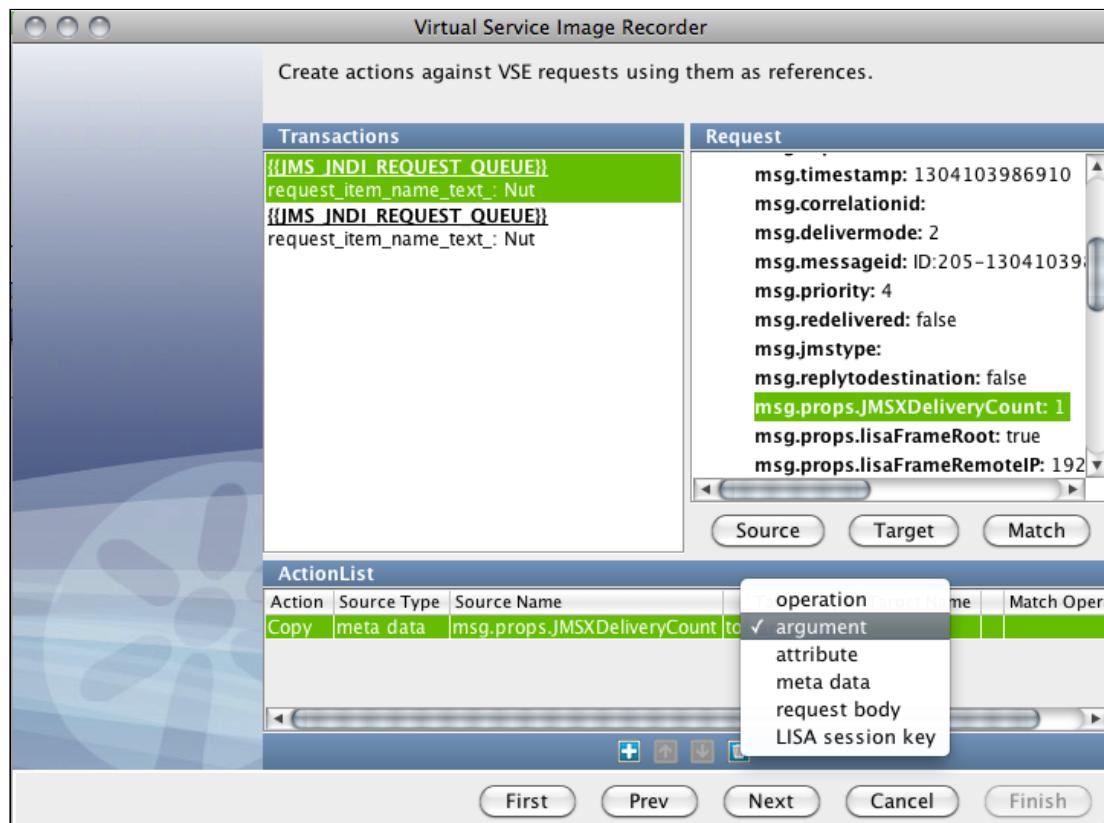
From this list of transactions, double-click a transaction to open a dialog showing the content of the transaction.



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, like 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.

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 \(see page 737\)](#).

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

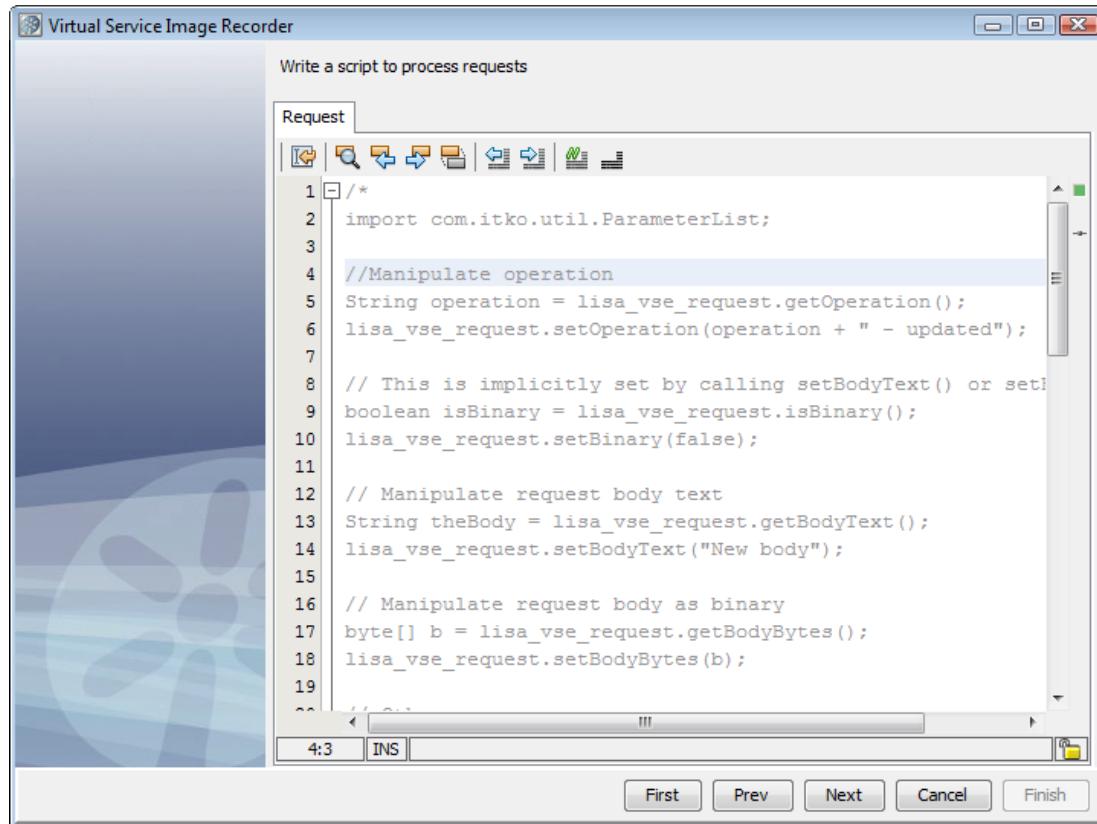
Format:

`%language%`

Default: beanshell

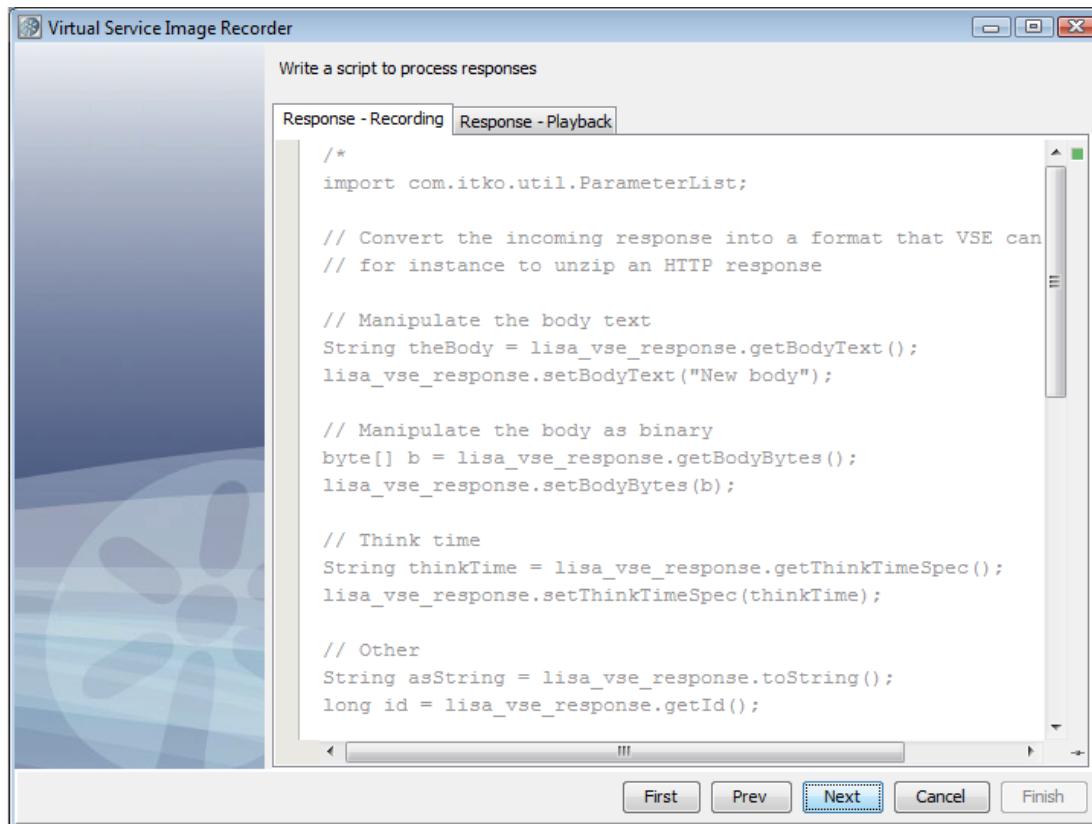
To use additional scripting languages, see "[Enabling Additional Scripting Languages \(see page 1385\)](#)."

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}{2:01031609050901BANKDEFXAXXX89549829458949811609N}{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>
  <MIRSessonNumber>8954</MIRSessonNumber>
  <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**

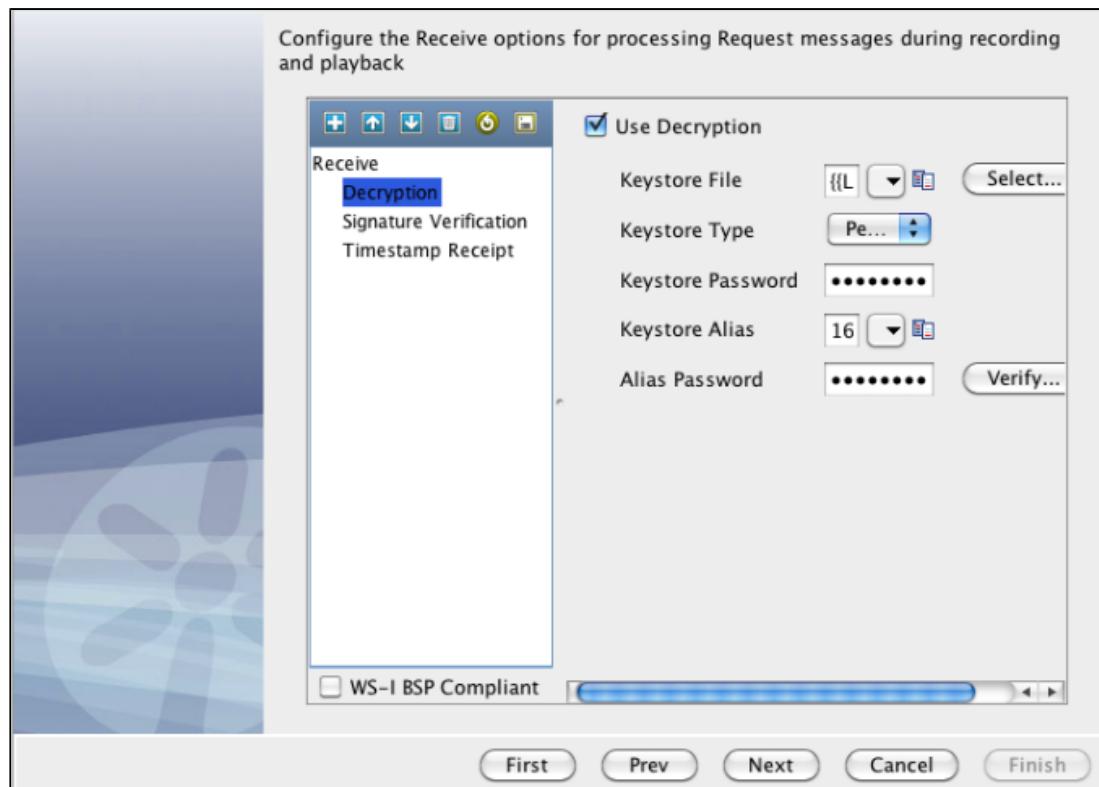
To use encryption, enter the following information:

- **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)**

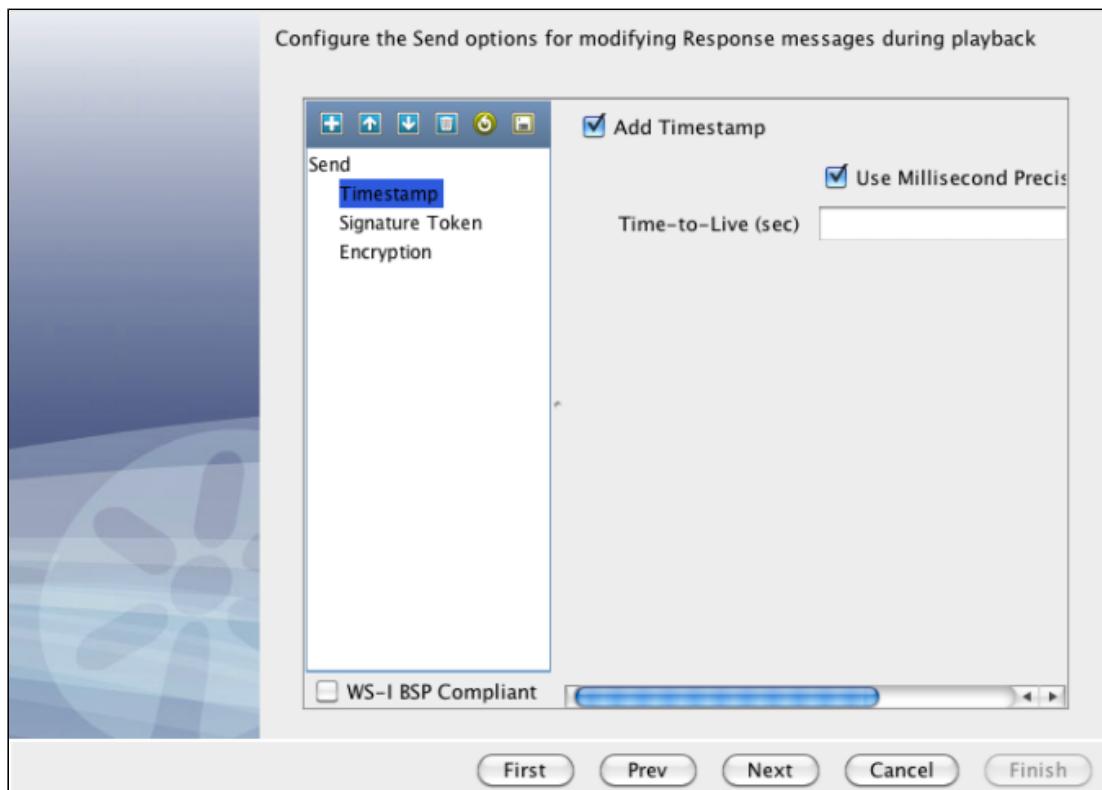
Defines the lifetime of the message in seconds. To avoid including an Expires element, enter **0**.

- **Use Millisecond Precision in Timestamp**

Specifies whether to output the timestamp in milliseconds.

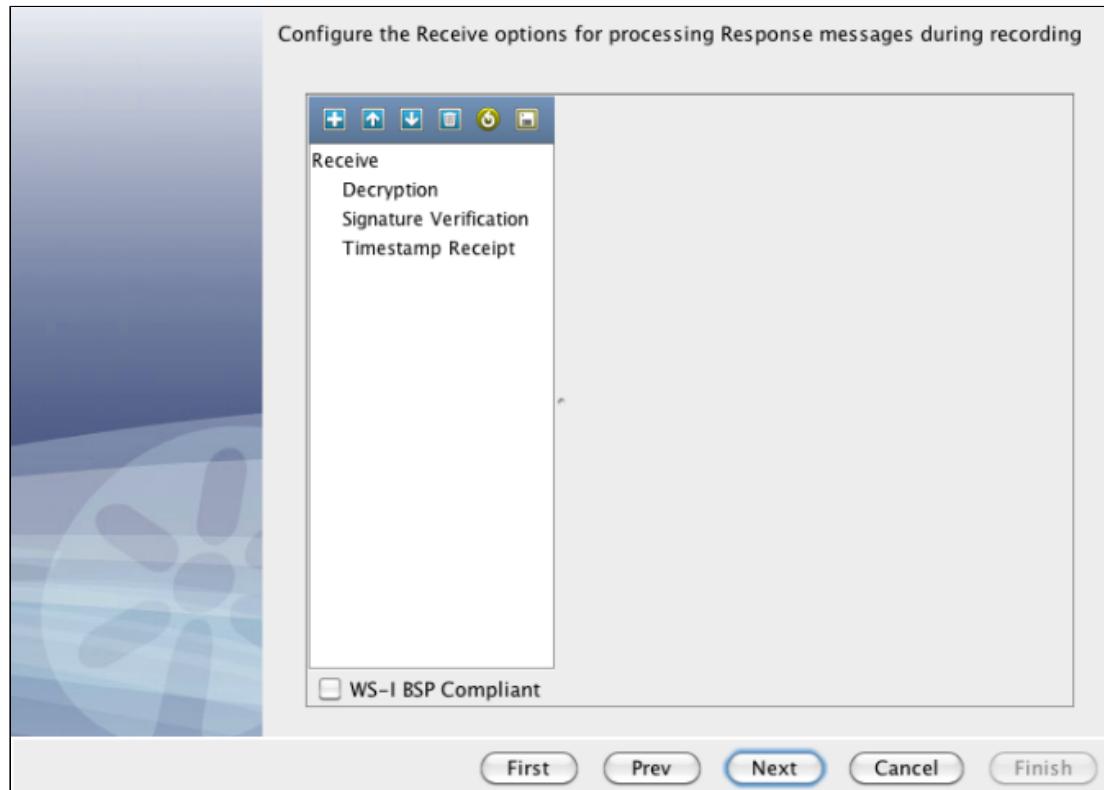


Note: Some web services (for example .NET 1.x/2.0 with WSE 2.0) do not comply with standard timestamp date formatting, and do not allow milliseconds. For these web services, clear the **Use Millisecond Precision in Timestamp** check box.



WS-Security Request data protocol Send options for modifying Response messages during playback

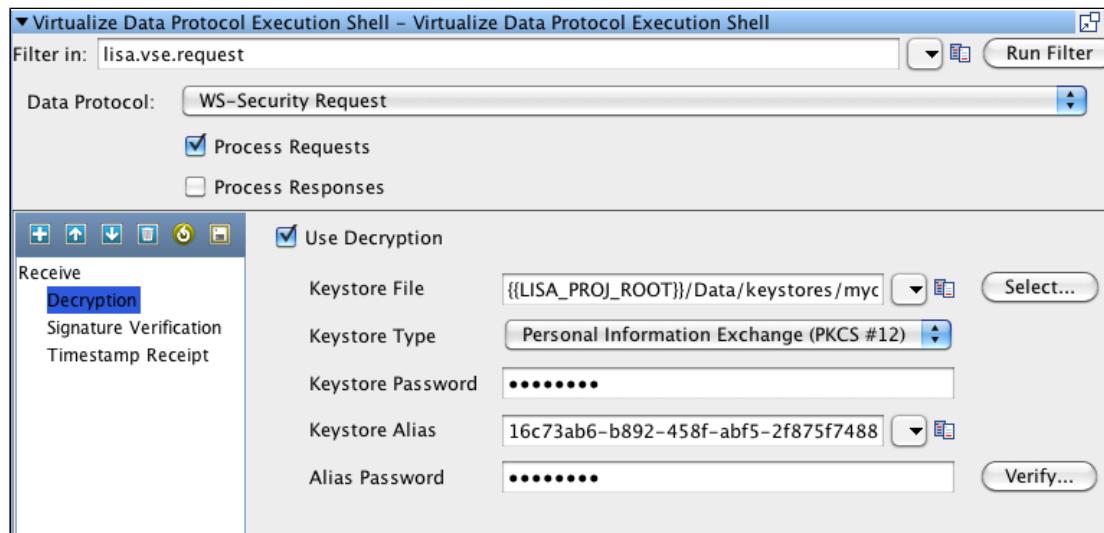
During playback, you must process the message as the server would send it. The SOAP message from the VSM has no Security headers and this configuration applies the Security header.



WS-Security Request data protocol Receive options for processing Response messages during recording

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 \(SOAP\) \(see page 900\)](#) data protocol. The data protocol requires no configuration information and so does not present a window in the recording wizard.

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 \(see page 906\)](#)".

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 icon 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.

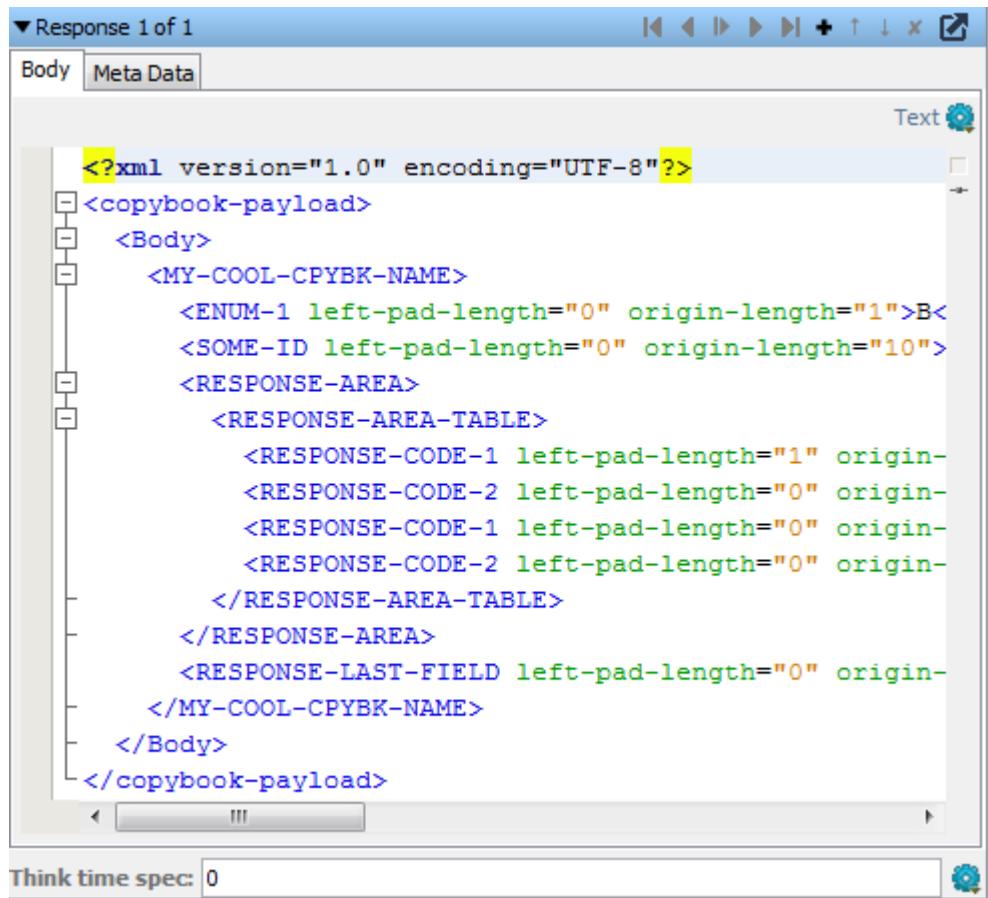


Note: A step subtracts its own processing time from the think time to have consistent pacing of test executions.

Customize the Response Editor



To customize the response editor, click  at the lower right corner of the panel.



The screenshot shows the DevTest Solutions Response Editor interface. The title bar says "Response 1 of 1". The menu bar includes icons for back, forward, search, and file operations. The main window has two tabs: "Body" (selected) and "Meta Data". The "Body" tab contains XML code:

```

<?xml version="1.0" encoding="UTF-8"?>
<copybook-payload>
  <Body>
    <MY-COOL-CPYBK-NAME>
      <ENUM-1 left-pad-length="0" origin-length="1">B<
      <SOME-ID left-pad-length="0" origin-length="10">
      <RESPONSE-AREA>
        <RESPONSE-AREA-TABLE>
          <RESPONSE-CODE-1 left-pad-length="1" origin-
          <RESPONSE-CODE-2 left-pad-length="0" origin-
          <RESPONSE-CODE-1 left-pad-length="0" origin-
          <RESPONSE-CODE-2 left-pad-length="0" origin-
        </RESPONSE-AREA-TABLE>
      </RESPONSE-AREA>
      <RESPONSE-LAST-FIELD left-pad-length="0" origin-
    </MY-COOL-CPYBK-NAME>
  </Body>
</copybook-payload>

```

Below the code is a "Think time spec: 0" input field. The status bar at the bottom shows "Text" and a gear icon.

Response editor body tab

Options on the response editor menu include:

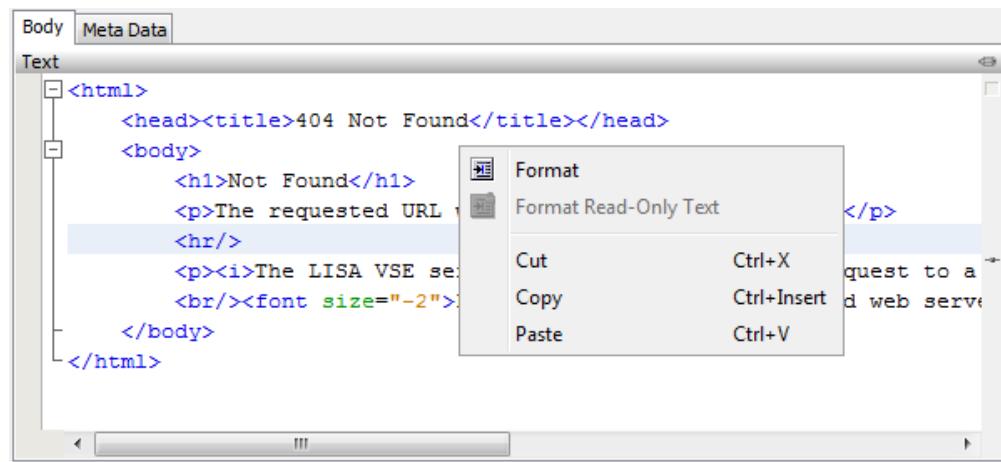
- Set reference protocol to
No Specific Protocol or JDBC (Driver based).

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.

```

DMG*D8*19430501*M~
NM1*PR*2*BLUE CROSS BLUE SHIELD OF NORTH DAK*****PI*~
N2*OTA~
CLM*26462967*75***11++1*Y*A*Y*Y*C~
DTP*454*D8*20010320~
DTP*431*D8*20010120~
DTP*453*D8*20010331~
DTP*455*D8*20010401~
REF*D9*17312345600006351~
HI*BK+0340*BF+V7389~
NM1*82*1*SMITH*ANDREW****24*788893334~
PRV*PE*ZZ*103T00000N~
LX*1~
SV1*HC+90801*75*UN*1***+1**N~
DTP*472*D8*20010201~
SE*31*0021~
GE*1*1~
IEA*1*000000905~

```

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.

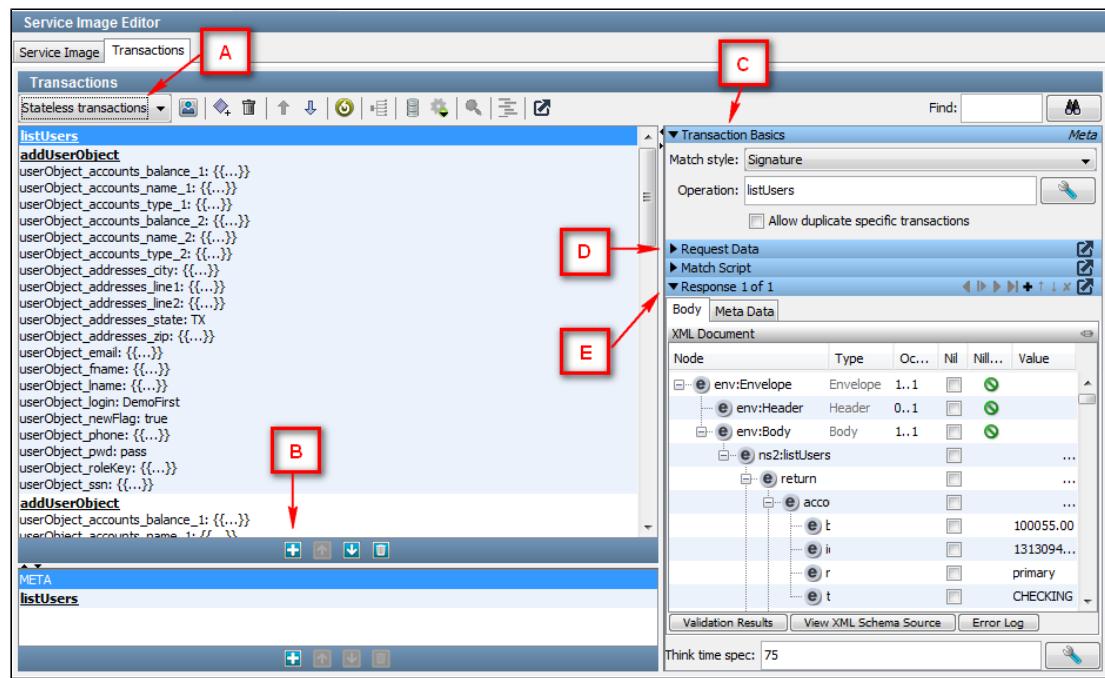


More Information:

- [Transactions Tab for Stateless Transactions \(see page 912\)](#)
- [Transactions Tab for Conversations \(see page 920\)](#)
- [Conversation Editor \(see page 922\)](#)

Transactions Tab for Stateless Transactions

When viewing a stateless transaction, 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.

- **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								
Arguments		Attributes		Meta Data				
Name	Name in Session	Comparison Operator	Value	Magic String	Date Pattern	Case Sensitive	Is Numeric	
username			DemoFirst	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>	
password			pass	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>	

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. \(see page 689\)](#) 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 \(see page 690\)](#) 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.

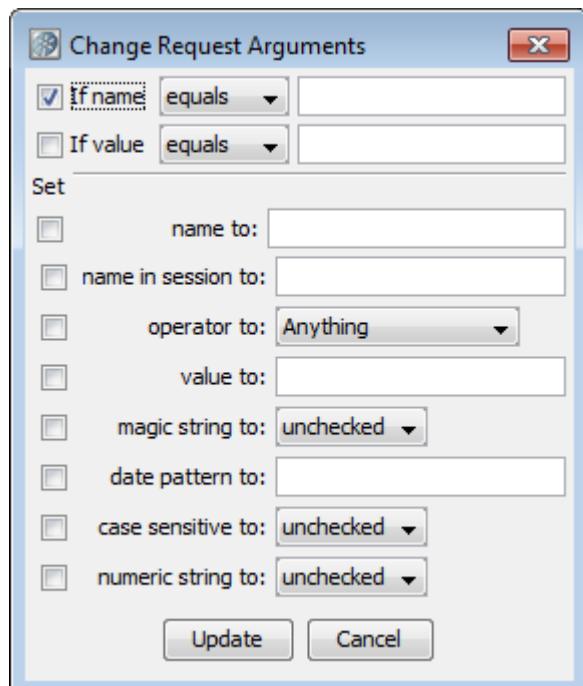
Values:

- **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.

```

17 }
18
19     bnsMatch = incomingReque
20         atch) {
21             ngValue = incomingReques
22                 ngValue = sourceRequest.ge
23
24         if(incomingValue.substring(0, 3).equa
25             // true means these arguments match
26             return true;
27         }
28     }
29 // false means no match
30 return false;
    !!!
19:1 INS
 Do not use the script

```

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 (http://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:

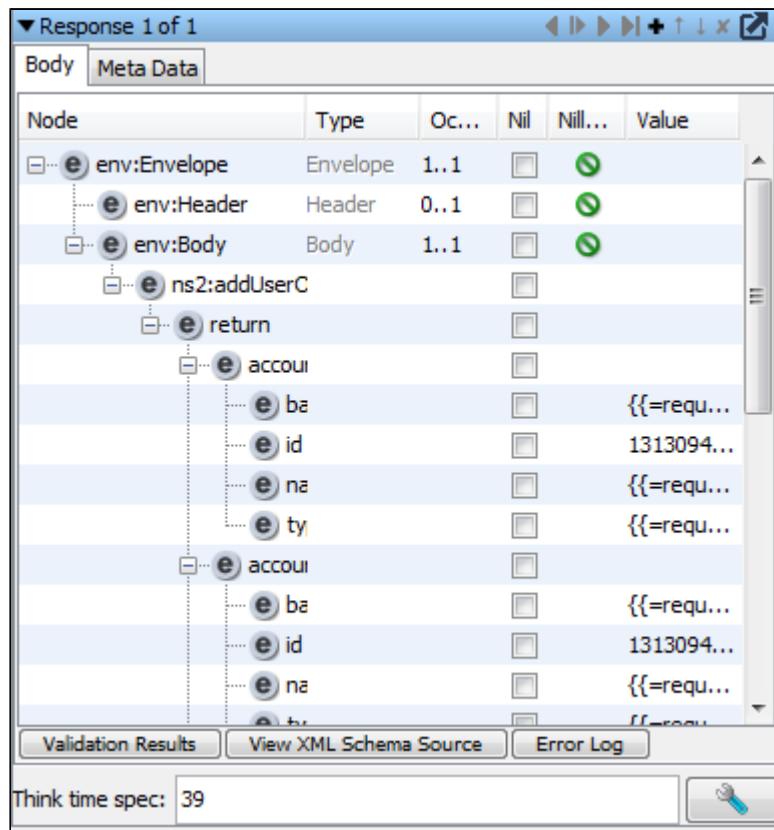
- | | |
|--|---|
| | Returns you to the last edit that was made |
| | Finds the next occurrence of the selected text |
| | Finds previous occurrence |
| | Finds next occurrence |
| | Toggles the highlight search |
| | Shifts the current line to the left four spaces |

-  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 \(see page 907\)](#).
- To inspect the Validation Results, view the XML schema source, and see the error log, use the buttons at the bottom of the panel.



The screenshot shows the DevTest Solutions Response Data Editor interface. The main window displays a hierarchical tree of XML nodes under the 'Body' tab. The tree structure includes an 'env:Envelope' node (Type: Envelope, Occurrences: 1..1), which contains an 'env:Header' node (Type: Header, Occurrences: 0..1) and an 'env:Body' node (Type: Body, Occurrences: 1..1). The 'env:Body' node contains an 'ns2:addUserC' node, which in turn contains a 'return' node. The 'return' node has two child nodes: 'accour' and 'accour'. Each 'accour' node has three child nodes: 'ba', 'id', and 'na'. The 'ba' node under the first 'accour' has a value of '{{=requ...}'. The 'id' node has a value of '1313094...'. The 'na' node has a value of '{{=requ...}'. The 'ba' node under the second 'accour' also has a value of '{{=requ...}'. The 'id' node has a value of '1313094...'. The 'na' node has a value of '{{=requ...}'. At the bottom of the editor, there are three buttons: 'Validation Results', 'View XML Schema Source', and 'Error Log'. Below these buttons, a status bar displays 'Think time spec: 39'. On the far right, there is a small icon of a wrench and screwdriver.

Response Data Editor stateless

Edit the **Think Time Spec** field as necessary.

Key	Value
HTTP-Response-Code	200
HTTP-Response-Code-Text	OK
Server	Apache-Coyote/1.1
X-Powered-By	Servlet 2.4; JBoss-4.2.1....
Content-Type	text/xml;charset=utf-8
Date	Wed, 17 Dec 2008 17:07:...

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.

The Service Image Editor shows the Transactions tab. Key components include:

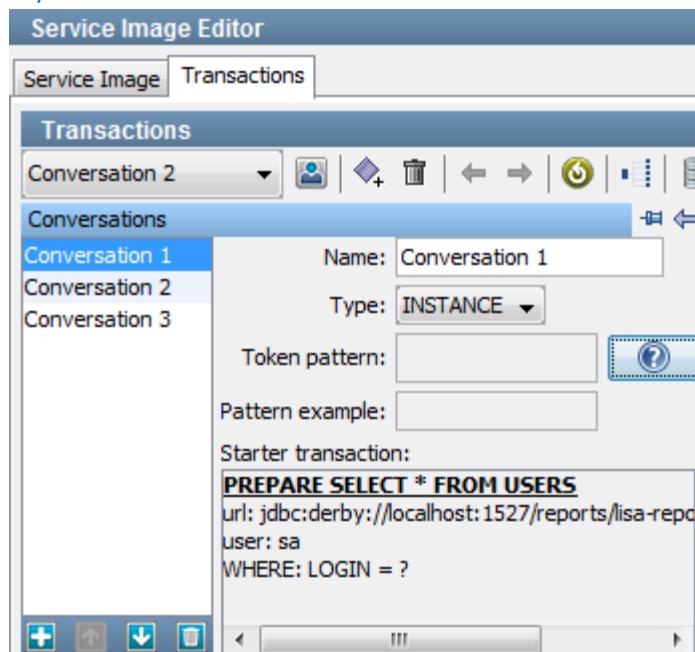
- Conversations List:** Shows Conversation 1, Conversation 2 (selected), Conversation 3, and Stateless transactions. A red box labeled **A** points to Conversation 2.
- Conversation Tree Editor:** Displays a tree structure for Conversation 2. A red box labeled **B** points to the tree node.
- Transaction Basics Tab:** Contains settings for Navigation (WIDE), Match style (Signature), Operation (T INTO USERS (LOGIN, PWD)), and other options. A red box labeled **D** points to the tab.
- Request Data Tab:** Shows a prepared statement: PREPARE INSERT INTO USERS (LOGIN, PWD). A red box labeled **E** points to the tab.
- Response Tab:** Shows Response 1 of 1. A red box labeled **F** points to the tab.
- XML Document Tab:** Shows XML structure for the response. A red box labeled **G** points to the tab.
- META Panel:** Shows connection details: url: jdbc:derby://localhost:1527/reports/lisa-reports.db;create=true, user: sa, VALUES: (? , ?).
- Bottom Buttons:** Includes standard application icons for file operations.

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](#) (see page 922) 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** (see page 912) 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** (see page 912).
- **E:** To enter the data for conversational requests during playback, use the **Transaction Request Data** (see page 912) pane.
- **F:** To enter and edit a script to return actions that are based on specified matching conditions, use the **Match Script Editor** (see page 912).
- **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** (see page 912) 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

Contents

- [Conversation Editor Toolbar \(see page 922\)](#)
- [Conversation Editor Graph View \(see page 924\)](#)
 - [Node Display Status \(see page 925\)](#)
- [Conversation Editor Tree View \(see page 926\)](#)
 - [Search and Replace Action \(see page 927\)](#)
 - [Test from Here Action \(see page 928\)](#)
 - [Highlight Navigation \(see page 930\)](#)
 - [Viewing Close Navigation \(see page 930\)](#)
 - [Viewing Wide Navigation \(see page 931\)](#)
 - [Viewing Loose Navigation \(see page 931\)](#)
 - [Restructure Conversation \(see page 932\)](#)

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

The **Conversation Tree editor** toolbar varies slightly, depending on the display.

Graph View Tools



Conversation Editor toolbar - Graph View

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		Toggles the display of the panel for managing a list of conversations. You can specify name, type, token pattern, pattern example, or starter transaction. display icon
Create New Transaction		Adds a new transaction. Create new transaction icon
Delete Selected Transaction		Deletes a selected transaction. Delete icon
Up Arrow		Tree View: moves the selected node earlier in the sibling list. Move up icon
Down Arrow		Tree View: moves the selected node later in the sibling list. Move down icon
Right Arrow		Graph View: moves the selected node later in the sibling list. Move right icon
Left Arrow		Graph View: moves the selected node earlier in the sibling list.



Move left
icon

Regenerate		Regenerates magic strings and date variables for all transactions.
	Refresh icon	

View Navigation		Opens a menu to select menu navigation highlights for stateful transactions (conversations). You can select the following:
View navigation icon		<ul style="list-style-type: none"> ▪ No navigation highlight ▪ Highlight on transaction's tolerance ▪ Highlight as if close ▪ Highlight as if wide ▪ Highlight as if loose

Toggle		Toggles the display of transaction IDs for debugging.
	Toggle icon	

Match		Pulls a match description from the clipboard and highlights the relevant information in the service image.
	Match icon	

Zoom		Opens a zoom menu.
	Zoom icon	

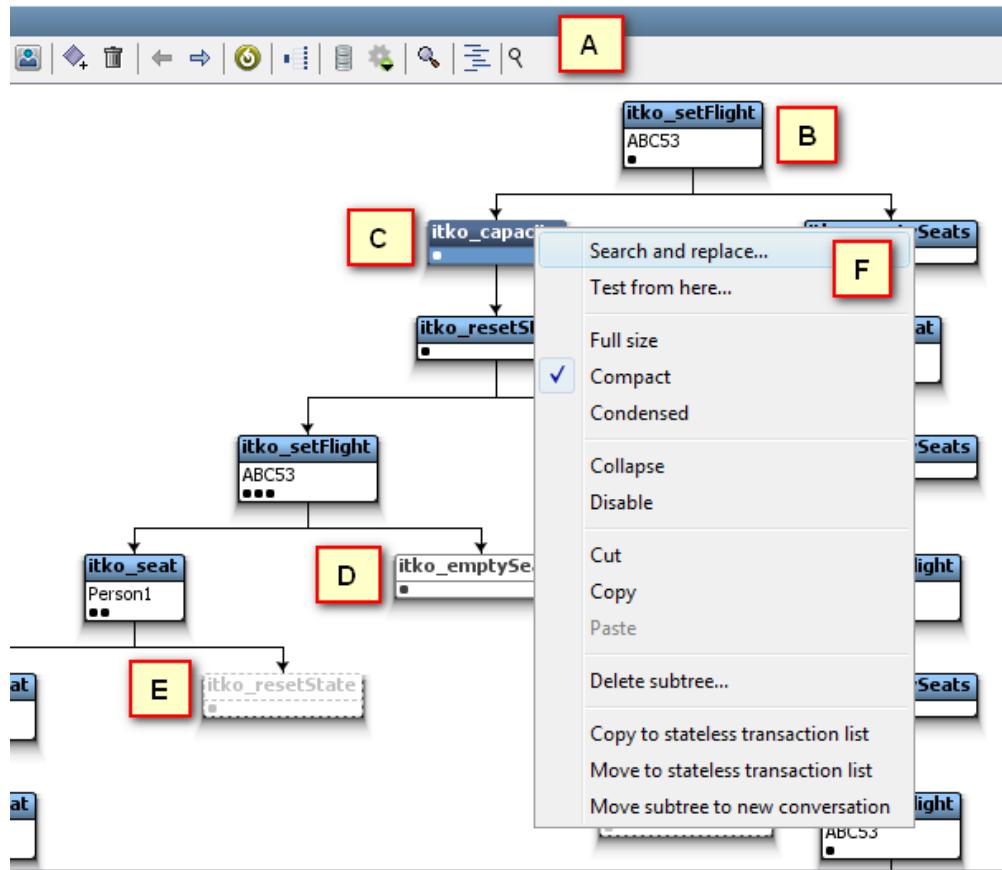
Display		Toggles the conversation display to a tree display.
	Display icon	

Display		Toggles the conversation display to a graph display.
	Display icon	

Zoom Panel		Zooms this panel to its largest size.
	Zoom panel icon	

Conversation Editor Graph View

The Conversation Editor displays nodes according to status. It has the following components:



Conversation Editor graph view components labelled.

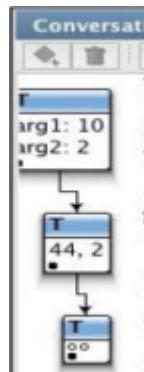
- **A:** Toolbar. For more information, see Conversation Editor Toolbar.
- **B:** A standard node.
- **C:** A selected node.
- **D:** A collapsed node. Child nodes are not displayed. To view child nodes, expand the node.
- **E:** A disabled node. This node and related child nodes (not displayed) are ignored during run time.
- **F:** The shortcut menu.

Node Display Status

You can display nodes in the following styles:

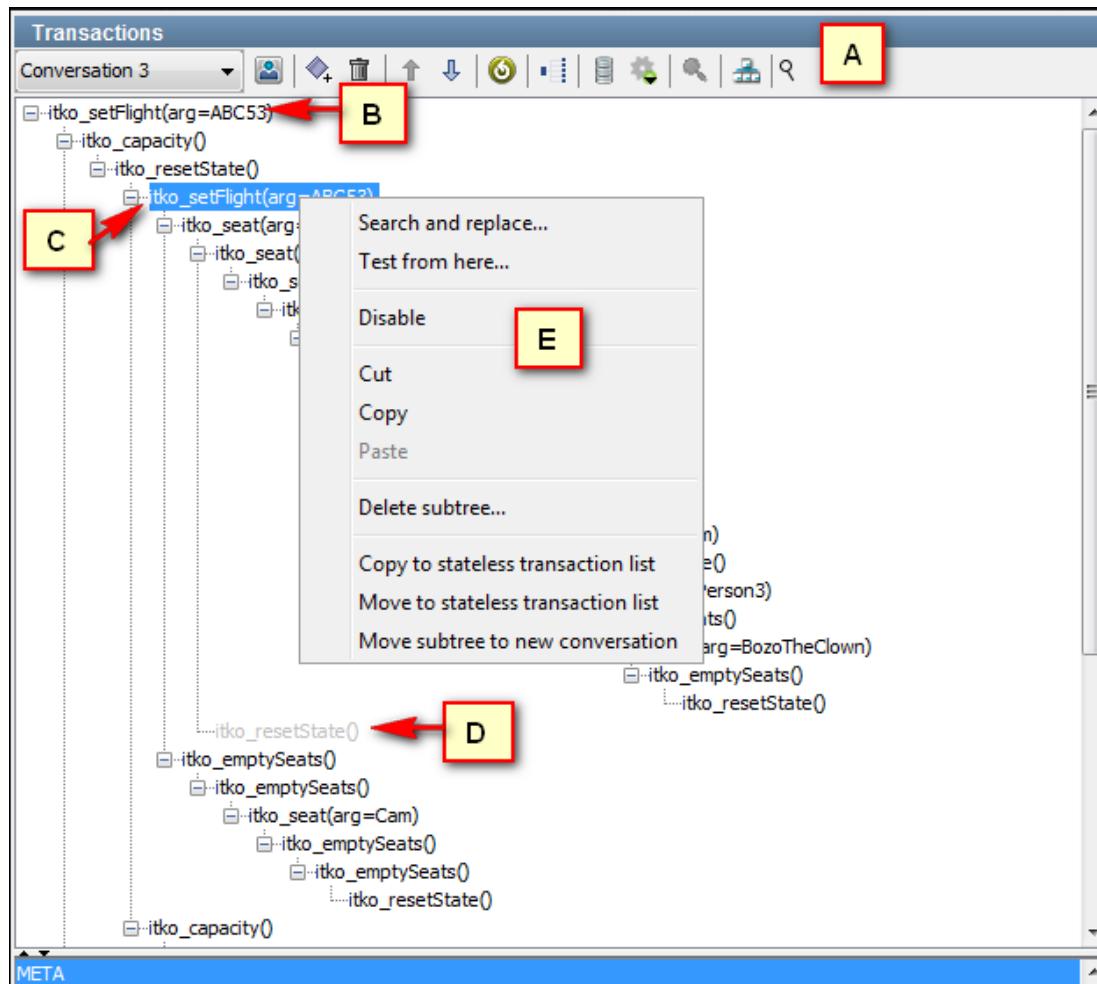
- Full size
- Compact (default)
- Condensed

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:

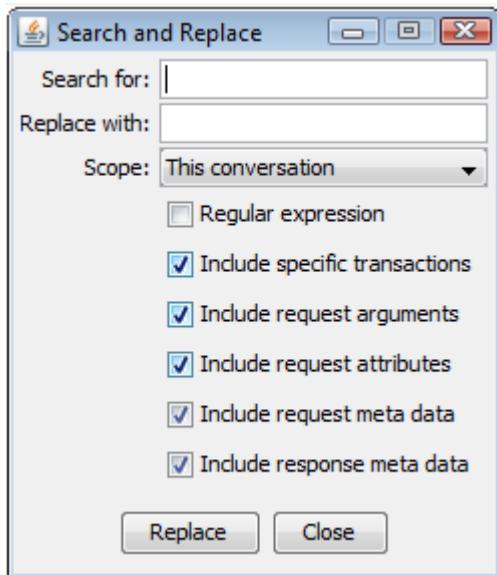


Conversation Editor tree marked up

- **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.

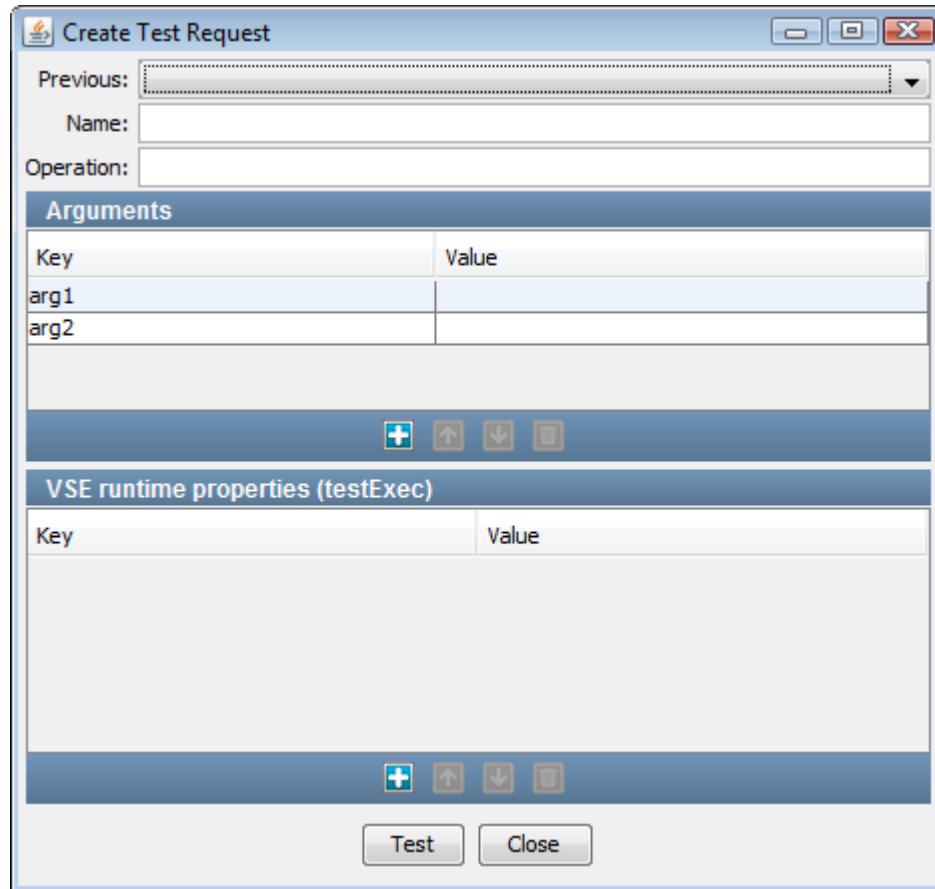
You can also use the check boxes to specify which pieces of the transactions to include.

Test from Here Action

To test for an expected response from a selected transaction in both graph and tree views, use the Conversation Editor.

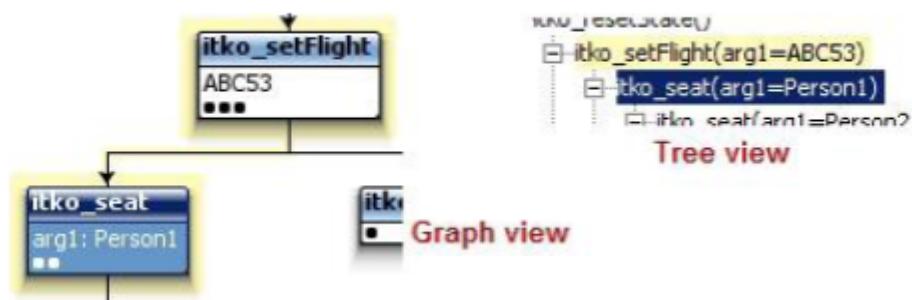
Follow these steps:

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:

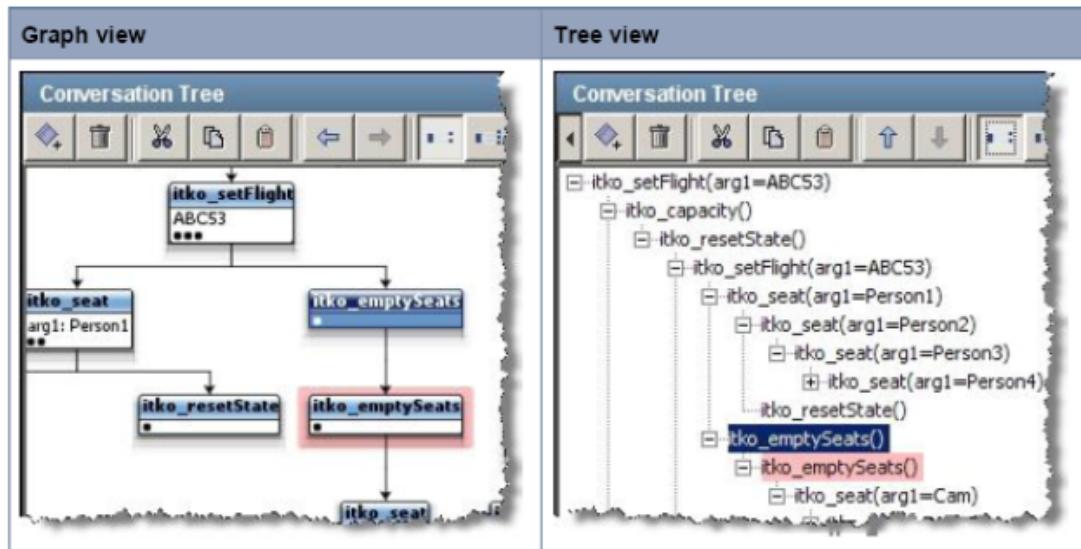
- Close
- Wide
- Loose



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

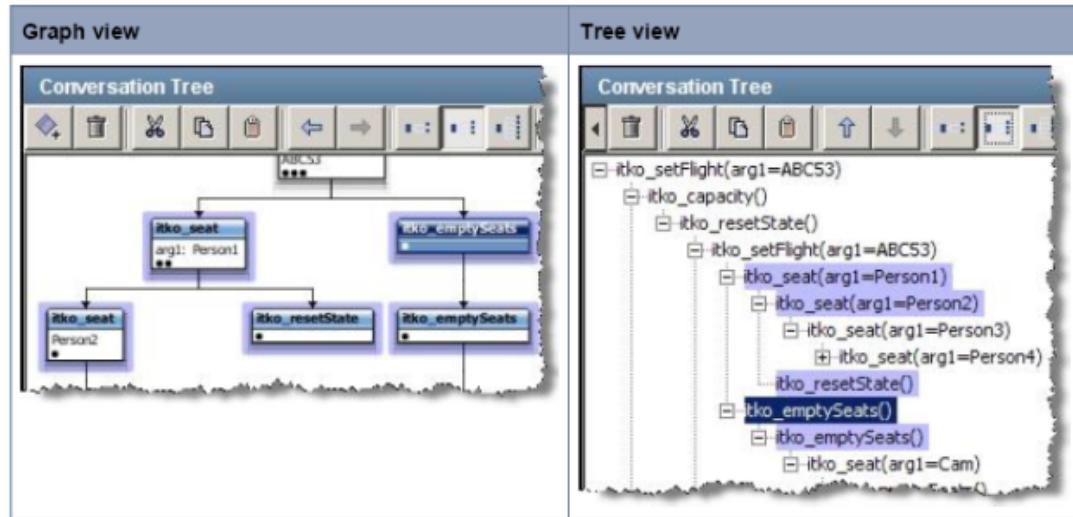
Select a transaction and click **View Navigation**. The transactions searched in the selected transaction subtree are highlighted in red.



Viewing Close Navigation, graph view and tree view

Viewing Wide Navigation

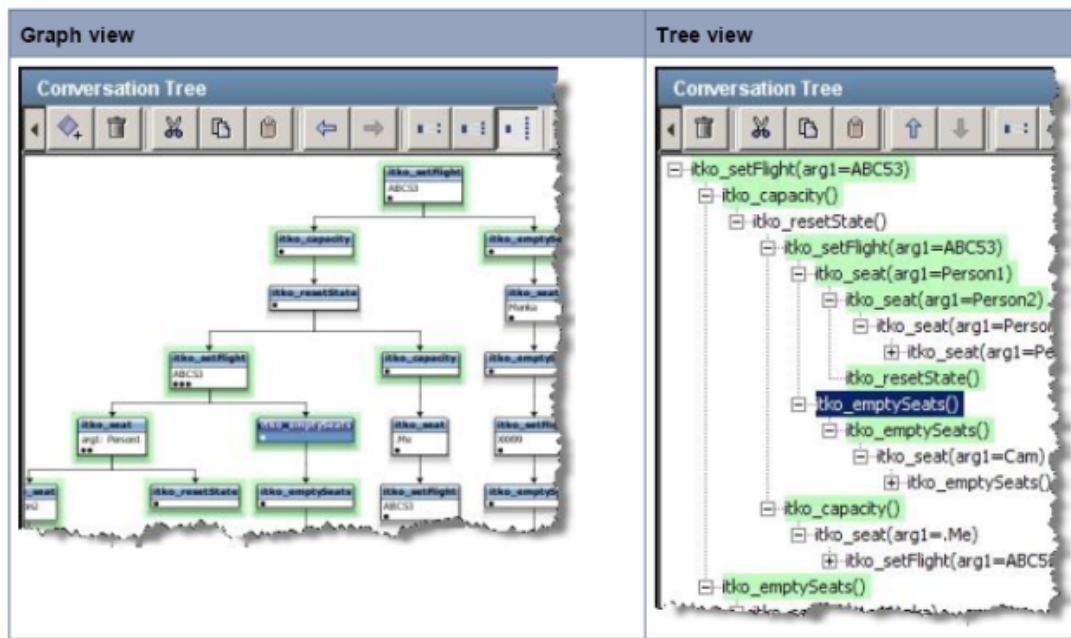
Select a transaction and click **View Navigation**. The transactions that are searched in the selected transaction subtree and sibling subtrees are highlighted in blue.



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.



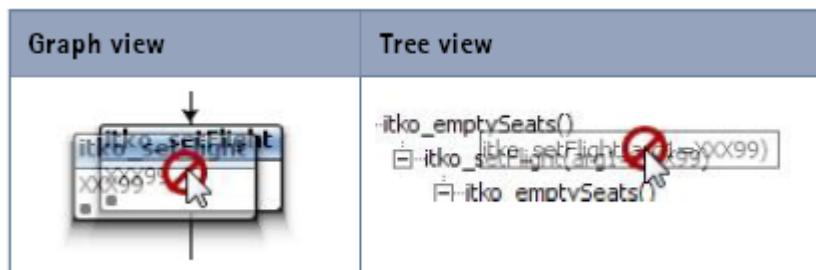
Viewing Loose Navigation, graph view and tree view

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 specific characteristics of service images that are created using the [JMS transport protocol](#) (see page 798) and the [IBM MQ Native transport protocol](#) (see page 789).

The queue and connection data in the metadata is purely for informational purposes. The data is not used at playback time.

The following items apply to the request side:

- By default, the operation name for the transaction is set to the request channel name defined in the request step in the virtual service model.
- The metadata properties that are not grouped with a prefix contain the queue and connection information for the proxy request queue.
- 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 **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 following items apply to the response side:

- The metadata properties that are not grouped with a prefix contain the queue and connection information for the proxy response queue.
- The metadata properties that start with **msg** are used to reconstruct the response message and its properties. The only required property is **msg.type**, which is included in service images that are created using the JMS transport protocol. The **msg.type** property indicates the type of message to send.

- 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 **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 \(see page 935\)](#)
- [Virtual Service Tracker Step \(see page 935\)](#)
- [Virtual Conversational/Stateless Response Selector Step \(see page 936\)](#)
- [Virtual HTTP/S Listener Step \(see page 936\)](#)
- [Virtual HTTP/S Live Invocation Step \(see page 938\)](#)
- [Virtual HTTP/S Responder Step \(see page 940\)](#)
- [Virtual JDBC Listener Step \(see page 941\)](#)
- [Virtual JDBC Responder Step \(see page 942\)](#)
- [Socket Server Emulator Step \(see page 942\)](#)
- [Messaging Virtualization Marker Step \(see page 944\)](#)
- [Compare Strings for Response Lookup Step \(see page 945\)](#)
- [Compare Strings for Next Step Lookup Step \(see page 946\)](#)
- [Virtual Java Listener Step \(see page 947\)](#)
- [Virtual Java Live Invocation Step \(see page 948\)](#)
- [Virtual Java Responder Step \(see page 949\)](#)
- [Virtual TCP/IP Listener Step \(see page 949\)](#)
- [Virtual TCP/IP Live Invocation Step \(see page 951\)](#)
- [Virtual TCP/IP Responder Step \(see page 953\)](#)
- [Virtual CICS Listener Step \(see page 954\)](#)
- [Virtual CICS Responder Step \(see page 954\)](#)
- [CICS Transaction Gateway Listener Step \(see page 954\)](#)
- [CICS Transaction Gateway Live Invocation Step \(see page 956\)](#)
- [CICS Transaction Gateway Responder Step \(see page 957\)](#)
- [Virtual DRDA Listener Step \(see page 957\)](#)

- [Virtual DRDA Response Builder Step \(see page 958\)](#)
- [Virtual DRDA Live Invocation Step \(see page 958\)](#)
- [IMS Connect Listen Step \(see page 959\)](#)
- [IMS Connect Live Invocation Step \(see page 960\)](#)
- [Virtual IMS Connect Responder Step \(see page 961\)](#)
- [Steps for Messaging Transport Protocols \(see page 961\)](#)
- [JCo IDoc Listener Step \(see page 965\)](#)
- [JCo IDoc Live Invocation Step \(see page 966\)](#)
- [JCo IDoc Responder Step \(see page 967\)](#)
- [JCo RFC Listener Step \(see page 967\)](#)
- [JCo RFC Live Invocation Step \(see page 968\)](#)
- [JCo RFC Responder Step \(see page 968\)](#)

Virtual Service Router Step

This 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 this 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 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 \(see page 1671\)](#) 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

This 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 this 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 this 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

This 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...**

Specify 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 this 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 this 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

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 this 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 this 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 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

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 \(see page 837\)](#).

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 this 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

se this 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.

Virtual DRDA Listener Step

Use the Virtual DRDA Listener step to virtualize DRDA traffic over TCP/IP. The step intercepts incoming DRDA data and converts it to a standard virtual request format.

Complete the following fields as described:

- **Listen/Record on port**

- **Target host**

- **Target port**

- **DB2 IP address**

- **LISA IP address**

- **Stored proc param delimiter**

- **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 DRDA Response Builder Step

This step decomposes DRDA requests into individual DRDA commands to mitigate the complexity of VSE receiving DRDA commands in different groupings. For example, DRDA sends the same set of commands in sets of one, two, three, or four for each request. Matching each command and building a composite response makes our models as flexible as possible for playback.

Virtual DRDA Live Invocation Step

Use the Virtual DRDA Live Invocation step to make a real DRDA call to a real server in the context of a virtualized DRDA service. Recording and virtualizing DRDA traffic typically creates the step. The step performs the real request in regard to the current VSE request in play.

Complete the following fields as described:

- **Listen/Record on port**
- **Target host**
- **Target port**
Enter the name of the port on which the request is made.
- **DB2 IP address**
- **LISA IP address**
- **Stored proc param delimiter**
- **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.

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

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 this 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](#) (see page 798):

- **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](#) (see page 789):

- **IBM MQ Native VSE Listen**

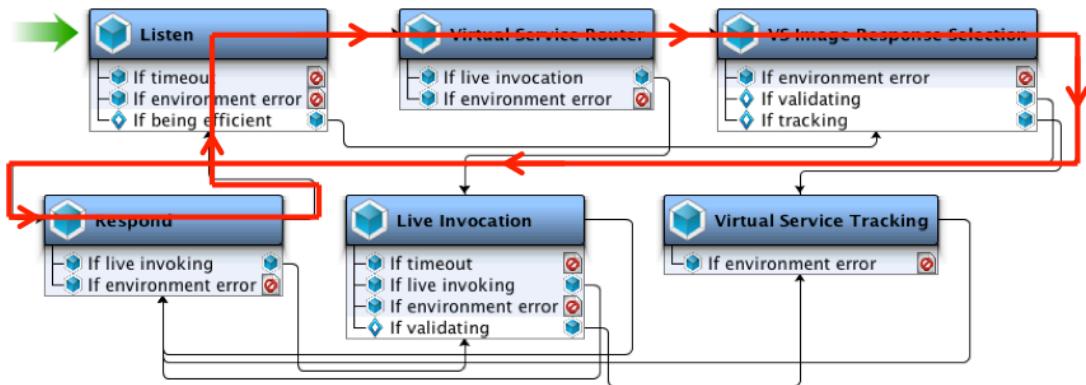
- **IBM MQ Native VSE Respond**

- **IBM MQ Native VSE Live Invocation**

Each step has basic and advanced parameters. To display the advanced parameters, click **PRO** at the top of the editor.

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.

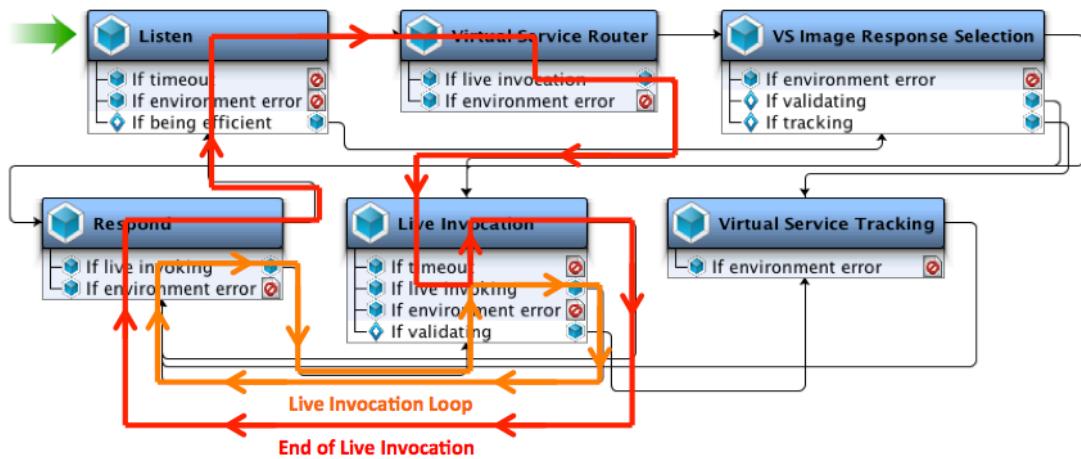
During live invocation, the execution flow of the virtual service model is more complicated.

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.

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

JCo IDoc Listener Step

Use this 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.

▪ 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.

- **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

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

This 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 this 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

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

This 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

In VSE, *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

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 \(see page 879\)](#).

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 \(see page 999\)](#).

Deploy and Run a Virtual Service

Follow these steps:



1. On the VSE Console, click **Deploy a new virtual service to the environment**. The **Deploy Virtual Service** window opens.
2. Enter the name of a model archive (MAR) to upload, then click **Deploy**.
 - The MAR must contain a virtual service. When you click **Deploy**, the service loads into the VSE Console and the service is available to run.
 - Alternatively, go to the project panel in DevTest Workstation and right-click a virtual service model (VSM) to open the **Deploy Virtual Service** window.

The **Deploy Virtual Service** window opens.

3. Modify the fields as necessary:

- **Name**

Displays the name of the virtual service that the VSM you selected references.

- **VS model**

Displays the virtual service model that you selected.

- **Configuration**

(Optional) Lets you select an alternate configuration file.

- **Group Tag**

Specifies the name of the [virtual service group \(see page\)](#) 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.

VSE allocates a number of 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 for the duration of 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:

$$\text{Concurrent Capacity} = (\text{Desired transactions per second} / 1000) * \text{Average Think Time in ms} * (\text{Think Scale} / 100)$$

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.

$$(100 \text{ Transactions per second} / 1000) * 200\text{ms} * (100 / 100) = 20$$

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.

- **Start the service on deployment**

Specifies whether to deploy and start the service immediately.

Values:

- **Selected:** The service deploys and starts immediately.
- **Cleared:** The service deploys, then you start it manually from the VSE Console.

- **If service ends, automatically restart it**

Specifies whether to keep the service running even after an emulation session reaches its end point.

Values:

- **Selected:** Continues running the service after the emulation service ends.
- **Cleared:** Stops the service when the emulation service ends.

Default: Selected

4. Click **Deploy**.

The VSE Console displays the virtual service status as loaded.



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.

Running Live Requests

Run live requests against VSE after you deploy the virtual service. If possible, take the live service down and configure the gateway/proxy settings so the client communicates with VSE.

For more information, see:

- [Access the VSE Console \(see page 973\)](#)
- [Toolbar \(see page 974\)](#)
- [Services Tab \(see page 975\)](#)
- [Matching Tab \(see page 977\)](#)
- [Request Events Details Tab \(see page 978\)](#)

Access the VSE Console

To manage and monitor deployed service images, use the VSE console. From this console, you can deploy, start, view, stop, redeploy, and remove virtual services.

Follow these steps:

1. From DevTest Workstation, click **Server Console** to open the DevTest Server Console.
2. Select the VSE service and double-click it to open the VSE console.
3. In the VSE Console, verify that the virtual service is deployed (status is Running).

- In the VSE Console, verify that the service received the requests by viewing the transaction count (**Txn count**).



Note: If a deployed VS model shows no transactions, then the client is not configured properly. Reconfigure the client to reference the virtual model instead of the real system. If another service is using that port, stop that service or change the port setting to remove the conflict.

VSE Console Toolbar

The VSE Console has a specific toolbar, which consists of the following buttons. You can also access these functions by pointing at a service and right-clicking.



VSE Console toolbar

The VSE Console toolbar contains the following commands:

Deploy		Deploys or redeploys a virtual service to the environment.
---------------	--	--

/Redeploy



Stage Quick Test icon

Start		Starts a virtual service that you selected.
--------------	--	---

LISA--

icon_image_rightarrowing

reencircle

Show		Displays the inspection view for the selected virtual service.
-------------	--	--

Icon of diagonal arrow

View		Displays session/tracking information for the selected virtual service.
-------------	--	---

Virtualized database icon

Set		Specifies how the selected virtual service should behave.
------------	--	---

Blue Gear Icon

Reset		Resets the transaction and error counts for the selected virtual service.
--------------	--	---

twolefthalfarrows

Stop		Stops the selected virtual service.
-------------	--	-------------------------------------



X icon

Remove Removes the selected virtual service from the environment.

con_image_whitetrashcan
bluebackground

Configure Configures tracking data cleanup.

Wrench icon

Shut down Shuts down the entire virtual service environment.

Icon - image of a cube
and a circle

Groups Selects whether to display services associated with all groups, with no groups, or with one group.

VSE Console Group
dropdown

- **Deploy a new virtual service to the environment**

To deploy a virtual service from a model archive (MAR), select this option.
Enter the name of a model archive (MAR) to upload. The MAR must contain a virtual service.
When you click **Deploy/Redeploy**, the service loads into the VSE Console and the service is available to run.

- **Start the selected virtual service**

Starts a virtual service that you have selected.
A confirmation window opens to indicate that the service has started. Click **OK**.

- **Show the inspection view for the selected virtual service**

The **Show Inspection View** button opens a tab for an inspector panel that is tailored for virtual services. This panel has two tabs, [Matching \(see page 977\)](#) and [Request Event Details \(see page 978\)](#).

VSE Console Services Tab

In the **Services** tab, 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.

This tab contains the following fields:

- **Name**

Identifies the currently deployed virtual service model.

- **Resource / Type**

Identifies the port and the type or protocol of the service.

- **Status**

Identifies the current state of the virtual service.

- **Up-Time**

Indicates how much time has elapsed since the service was started.

- **Txn Count**

Identifies the number of transactions that were recorded after the service started.

- **Execution Mode**

Displays the [execution mode \(see page 978\)](#) of the virtual service.

- **Group**

Displays the [virtual service group \(see page \)](#) of the virtual service.

- **Errors:**

Displays a red dot to indicate that errors occurred while running the service.

The **Virtual Service Details** panel at the bottom of the window displays details about the service.

This panel contains the following fields"

- **Model Name**

Identifies the name of the currently deployed virtual service model.

- **Execution Mode**

Displays the execution mode for this virtual service. See [Specify How the Selected Model Should Behave \(see page 978\)](#).

- **Last Start**

Displays the date and time this service was last started.

- **Transaction Count**

Displays the number of transactions that VSE recorded after the service started.

- **Current txn/s**

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.

- **Group Tag**

Displays the name of the [virtual service group \(see page \)](#) 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 after you enter the group tag, then click **Update** to update the field.

To delete a group tag from a virtual service, click the **X** next to the group tag name, then click **Update**.

- **Config Name**

Identifies the name of the configuration file that this service uses.

- **Auto-Restart**

Specifies whether to use the auto-restart option for this service. To change this value while the service is running, click this field.

- **Last End**

Displays the date and time this service stopped.

- **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.

To download the backing archive for the virtual service, click the name of the virtual service in the VSE Console. The system prompts you to save the MAR file that is associated with this virtual service.

VSE Console-Matching Tab

You can see the **Matching** tab when viewing a virtual service with the inspection view in the VSE Console.

The **Matching** tab lists recent requests that the virtual services processed. When you select a request, the tab shows a description of how the request was (or was not) matched. The same information is recorded in the **vse_matches.log file**. If any events were associated with the selected request, they display at the right side of the panel.

The screenshot shows the 'Virtual Service Environment VSE@Default' interface with the 'kioskV5 Inspector' tab selected. The 'Recent Requests' section displays three log entries:

- 2011-08-10 08:07:16,405 [deleteToken](#)
token: -8588995a:2557022286c:-5257
- 2011-08-10 08:07:13,441 [getNewToken](#)
username: admin
password: admin
- 2011-08-10 08:07:01,911 [getUser](#)

A 'Refresh' button is located below the requests. To the right, there's a 'What Happened' panel with service information and a 'Events' table:

Timestamp	Event	Short Info
Events		

At the bottom, tabs for 'Matching' and 'Request Event Details' are shown, along with 'Refresh' and 'Auto Refresh' buttons.

Screenshot of VSE Console Matching tab

VSE Console Request Events Details Tab

You can see the **Request Event Details** tab when viewing a virtual service with the inspection view in the VSE Console.

The **Request Event Details** tab 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 ITR), select a step.

The screenshot shows the 'Virtual Service Environment VSE@Default' interface with the 'kioskV5 Inspector' tab selected. In the 'Errored Requests' section, two entries are listed:

- 2011-08-12 13:41:36,985 **deleteToken**
token: -4655d680:656078c197c:-6627
- 2011-08-12 13:41:35,225 **getNewToken**
username: itko

A 'Refresh' button is located below these entries.

The 'Steps Executed' column on the left lists the following steps:

- HTTP/S Listen
- Prepare Request
- VS Image Response Selection
- Prepare Response
- HTTP/S Respond

The 'Events -- Prepare Request' table on the right shows the following log entries:

Timestamp	Event	Short Info
2011-08-12 13:41:36,978	Property set	lisa.vse.request
2011-08-12 13:41:36,978	Step response	Prepare Request
2011-08-12 13:41:36,978	Assertion fired	Prepare Request [If being efficient]

At the bottom of the interface are buttons for 'Matching' (selected), 'Request Event Details', 'Refresh', and 'Auto Refresh'.

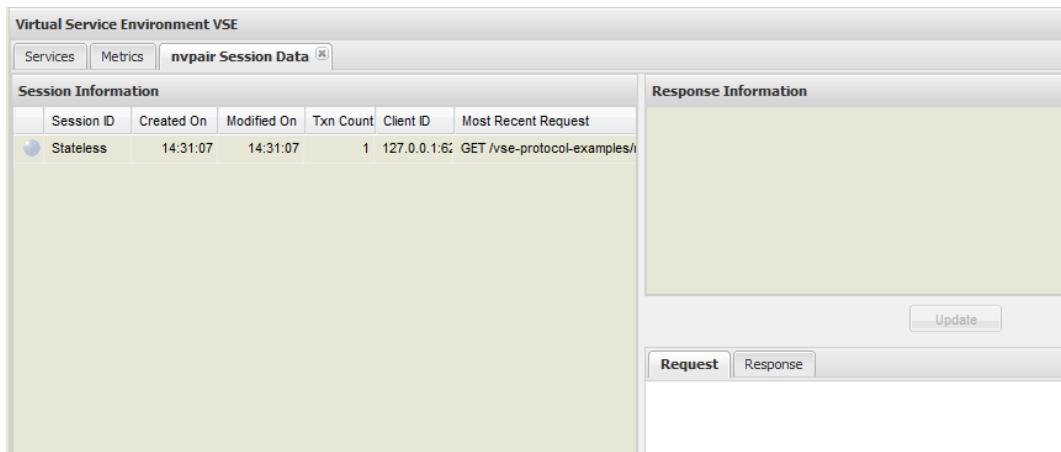
Screenshot of VSE Console - Request Events Details tab



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 for the Selected Virtual Service

This option lets you see the session tracking information for the virtual service. For more information, see [Session Viewing and Model Healing \(see page 983\)](#).



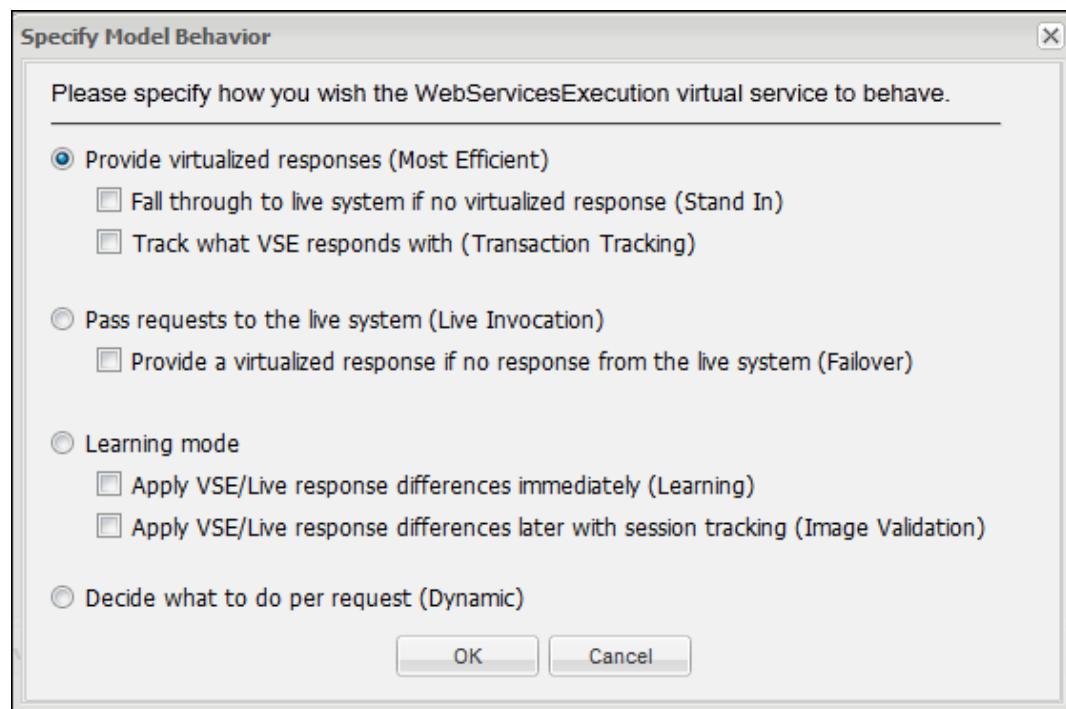
VSE Console - Request Events Details tab

- **Redeploy the Selected Virtual Service to the Environment**

If you edit the service image or the VSM, save the changes and redeploy the modified service image in the VSE Console by clicking **Deploy/Redeploy**.

- **Specify How the Selected Model Should Behave**

This option lets you set an execution mode for the virtual service. The number of execution modes available for a virtual service depends on the type of test step. For example, if a model does not have a live invocation step, it does not support the **Learning** or **Live Invocation** options.



Set Execution Mode dialog

The available execution modes are:

- **Most Efficient**

The fastest mode; it does not execute the routing or tracking steps. This mode also restricts generated event tracking.

- **Stand In**

Stand In mode 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. You can only enable Stand In mode for a virtual service with a Live Invocation step. Stand In mode does not do any special tracking. It simply allows for a virtual service to fall back on the live service.

- **Transaction Tracking**

This mode 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. Transaction Tracking mode does not show live system responses; it only shows the response from the service image.

- **Live System**

This mode 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.

- **Failover**

Failover mode 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. This mode is the opposite of Stand In mode. You can only enable Failover mode for a virtual service with a Live Invocation step. Failover mode uses the service image to determine a response if the Live Invocation step actually fails (as might happen if the live system were not available).

- **Learning**

Learning 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.

When a virtual service is running in Learning mode and it has acquired new knowledge, there is



an icon to the left of the virtual service name in the VSE Console. The icon remains visible until you redeploy the virtual service.

- **Image Validation**

This mode 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 VSE provides and a corresponding live system and, where differences exist, patches or heals the VSE service image to keep in sync with the live system. This mode is also known as "live healing mode." Image Validation is the least efficient of all the modes.

- **Dynamic**

This mode 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.

- **Reset the Transaction and Error Counts for the Selected Virtual Service**

Sets both the transaction count and error count to zero for the selected virtual service.

- **Stop the selected virtual service**

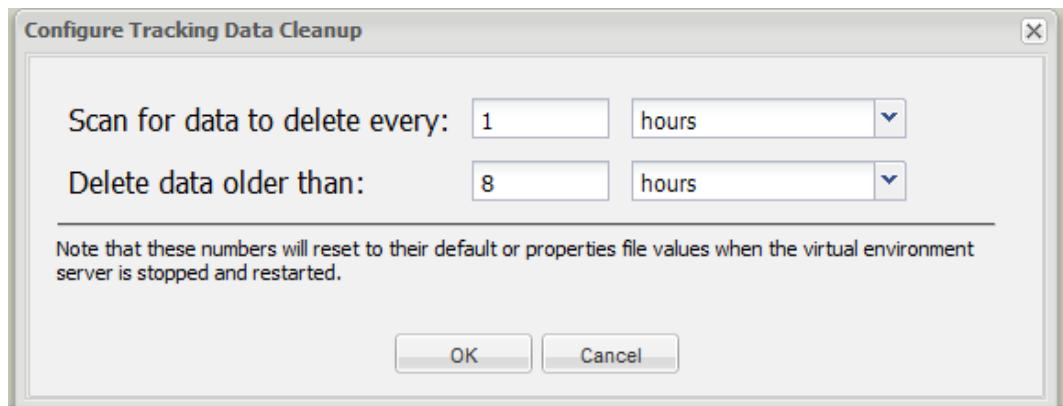
To stop the selected virtual service, click this button. The application displays a confirmation message before the service stops.

- **Remove the selected virtual service from the environment**

To remove the selected virtual service from the console display, click this button. The application displays a confirmation message before it removes the service.

- **Configure tracking data cleanup**

Opens the **Configure Tracking Data Cleanup** window.



Configure Tracking Data Cleanup dialog

Enter data cleanup values here that 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.

- **Scan for data to delete every**

Defines the interval (in milliseconds, seconds, minutes, hours, days, or weeks) after which to scan for tracking data to delete.

- **Delete data older than**

Defines the age of data (in milliseconds, seconds, minutes, hours, days, or weeks) to delete.

- **Shut down the entire virtual service environment**

To shut down the complete VSE environment, click this button. If you reply affirmatively to the shutdown confirmation message, your VSE shuts down and the corresponding window closes.

Session Viewing and Model Healing

Session viewing lets a VSE user actually see the behavior of current (or recent) sessions on a VSE server. The user 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 only available for virtual services that run with an Execution Mode of Transaction Tracking or Image Validation. Model healing is only available for virtual services that run in Image Validation mode. For more information about execution modes, see [Specify How the Selected Model Should Behave \(see page 978\)](#).

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 from either the VSE dashboard or while editing a virtual service model.

Session ID	Created On	Modified On
Stateless	8:25:13	8:28:23
-92ac754b:4289c860a61:-2d54	8:28:21	8:28:21
-10c3833c:4474830fb53:-3878	8:25:10	8:25:10

Response Information	
8:50:16	getNewToken(username=lisa_simpson, password=golisa)

Request		Response	
Virtual		Live	
[3] getNewToken		getNewToken	
username	demo1	username	lisa_simpson
password	pass	password	golisa

Session Information tab in the VSE Console

To view the session tracking for a selected virtual service, from the VSE Console **Services** tab, select a service and click **Session/Tracking Information**

When you click the icon, 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.

The session panel is divided into two panes: **Session Information** and **Response Information**.

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 an icon to indicate the current session status.

A gray ball indicates a session that was recorded in Transaction Tracking mode, or a session that has been healed using model healing.

A red ball indicates a session that was recorded in Image Validation mode and is a candidate for healing.

A green ball indicates a session that was recorded in Image Validation mode where the live and VSE responses match.

- **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 ball: Indicates a signature match on a meta transaction.

- Yellow ball: Indicates a signature match on a meta transaction; the image navigation is successful, but the response body differs between VSE and the live system.
- Red ball: Indicates a conversational transaction that diverged between the live system and 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.

VSE Metrics

Contents

- [View VSE Metrics \(see page 985\)](#)
- [Define Metrics Collection \(see page 985\)](#)
- [Server Chart Metrics \(see page 986\)](#)
- [Service Chart Metrics \(see page 989\)](#)

View VSE Metrics

The **Metrics** tab on the DevTest Console allows you to view VSE metrics.

Follow these steps:

1. To display a list of active services, click **Select VSE Service**.
2. Select the service to display.
3. To see the available charts for the selected service, click **Select Chart**.
4. Select a chart and click **Generate Chart**.
To refresh the display, click **Refresh**. Use the **Zoom** slider and the **Scroll** slider to move the chart to make your data more visible.



Note: To view the metrics charts, the Adobe Flash plugin must be installed on the browser that is used to access the server console.

Define Metrics Collection

To define the parameters of VSE metric collecting, update the following properties in the **lisa.properties** file:

- lisa.vse.metrics.collect
- lisa.vse.metrics.txn.counts.level
- lisa.vse.metrics.sample.interval
- lisa.vse.metrics.delete.cycle
- lisa.vse.metrics.delete.age



Note: For more information, see [DevTest Property File \(see page 1638\)](#).

Server Chart Metrics

Server chart metrics apply to all activity on a specific virtual server, or VSE.

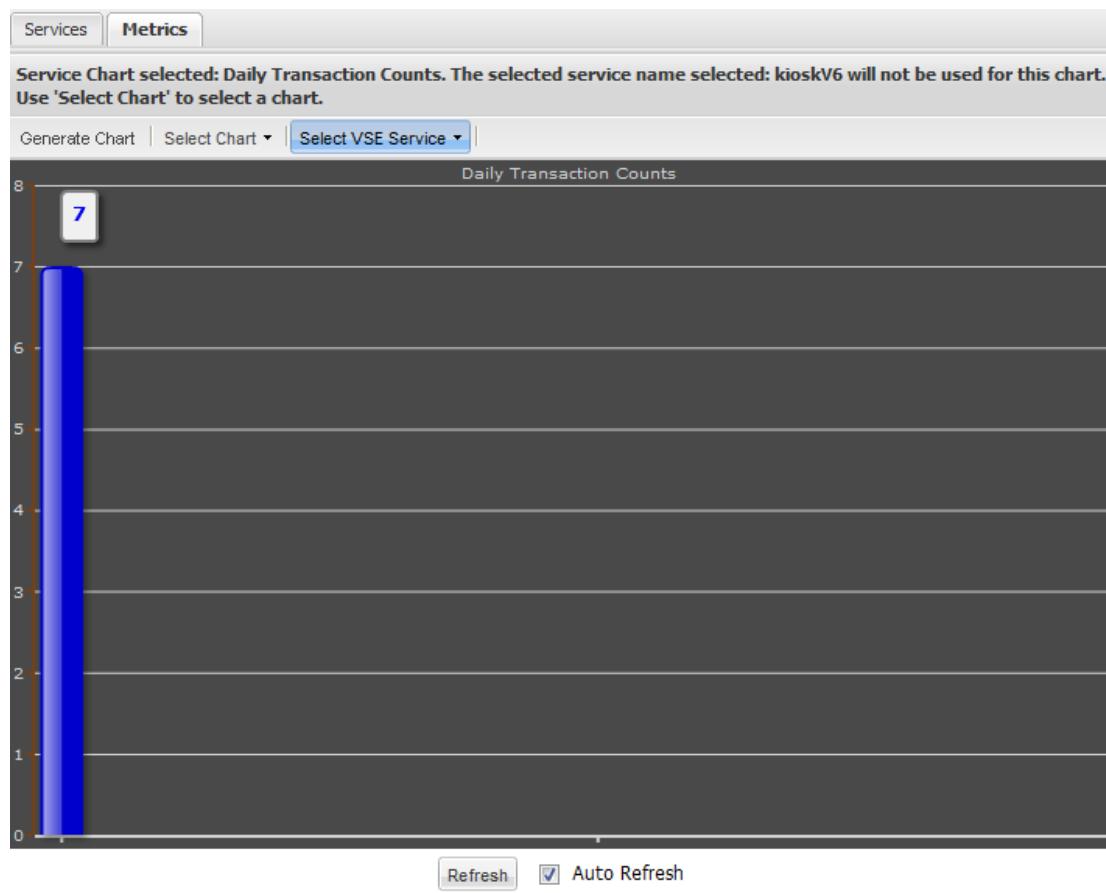


Note: If you select a server chart, the panel header indicates that the selected virtual service is not used for this chart. These charts apply to the entire server, not only the selected service.

Server Availability

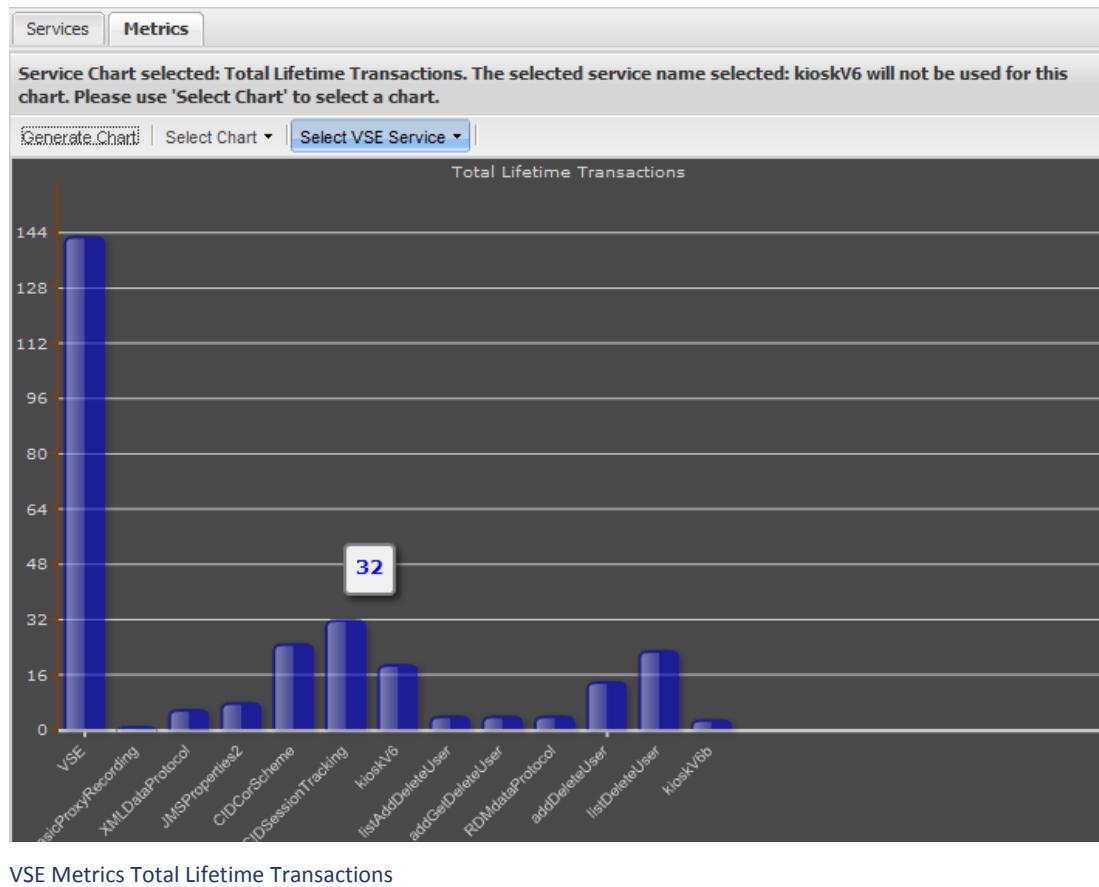


Daily Transaction Counts



VSE Metrics Daily Transaction Counts

Total Lifetime Transactions

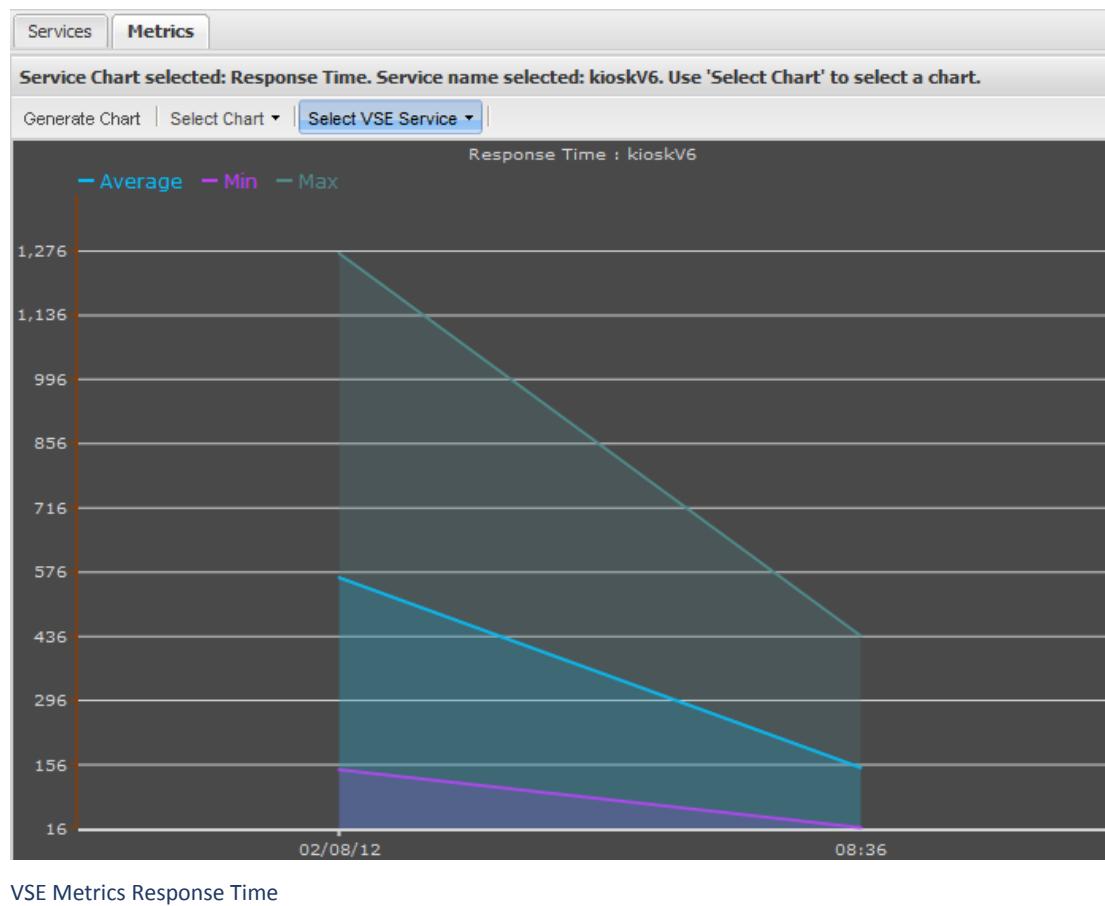


VSE Metrics Total Lifetime Transactions

Service Chart Metrics

Service chart metrics apply to each virtual service. On the right pane, there is a list of virtual services that were deployed in this environment. You can select one of them, then select a chart from the list.

Response Time



Forced Delay



VSE Metrics Forced Delay

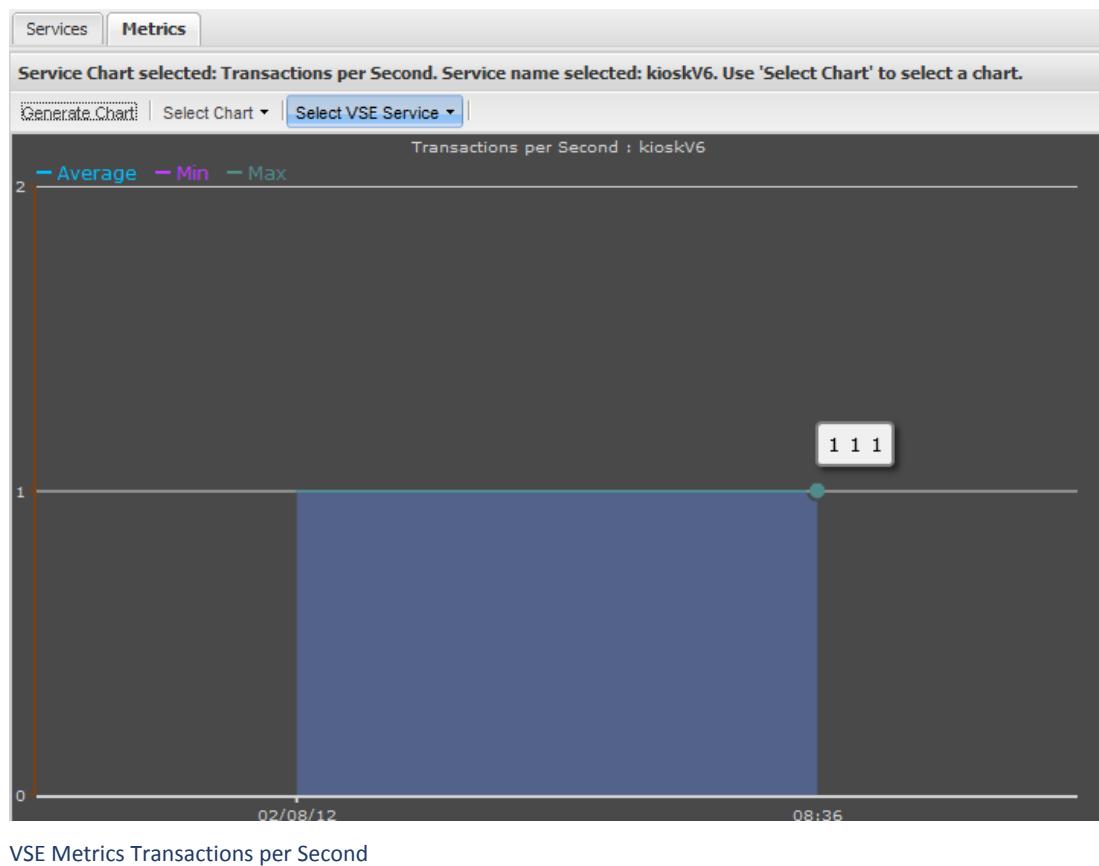
- **Forced Delay**

Indicates the number of milliseconds before VSE sent a result.

A positive number means VSE deliberately waited to send a result because of the specified think time. For example, if think time was 10ms and the time to listen and respond was 8ms, VSE waits 2ms.

A negative number means VSE took longer to generate the result than the think time. For example, the think time was 10ms but it took 12 ms to listen and respond was 12 ms. The report shows -2.

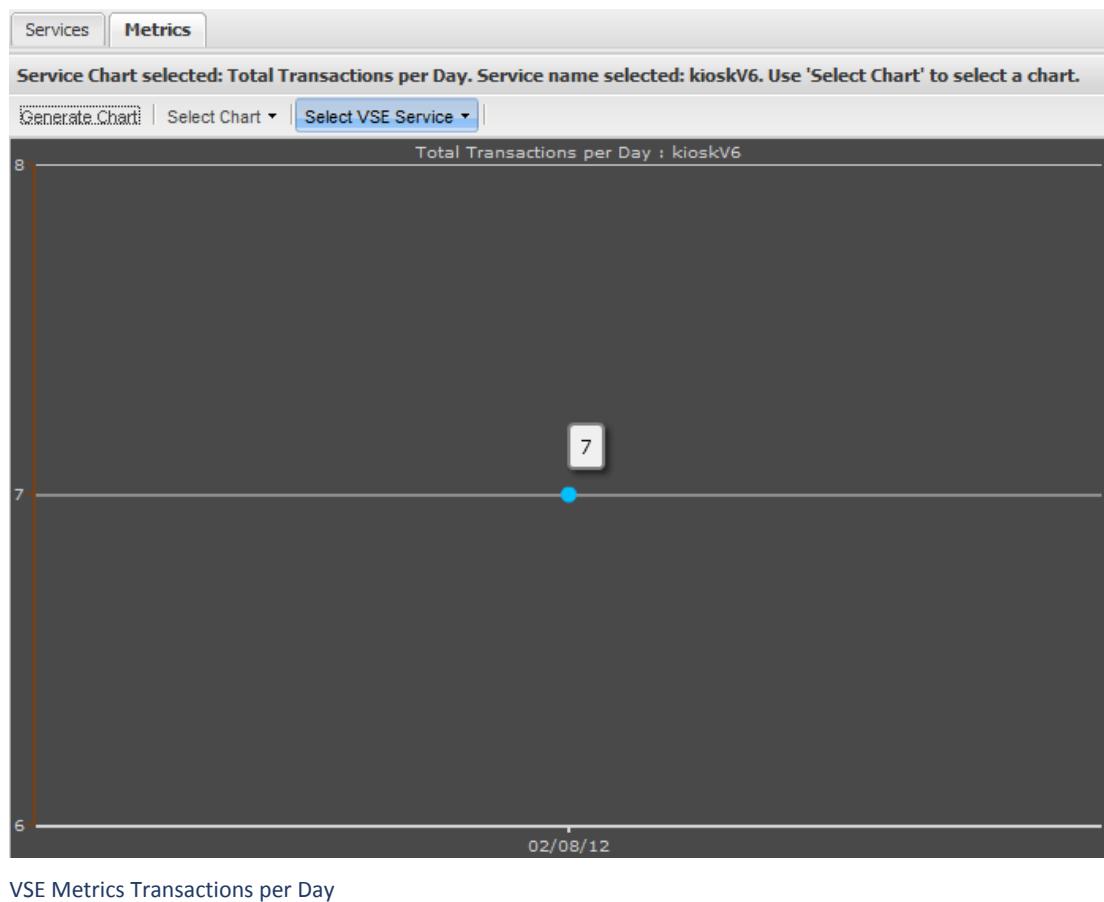
Transactions Per Second

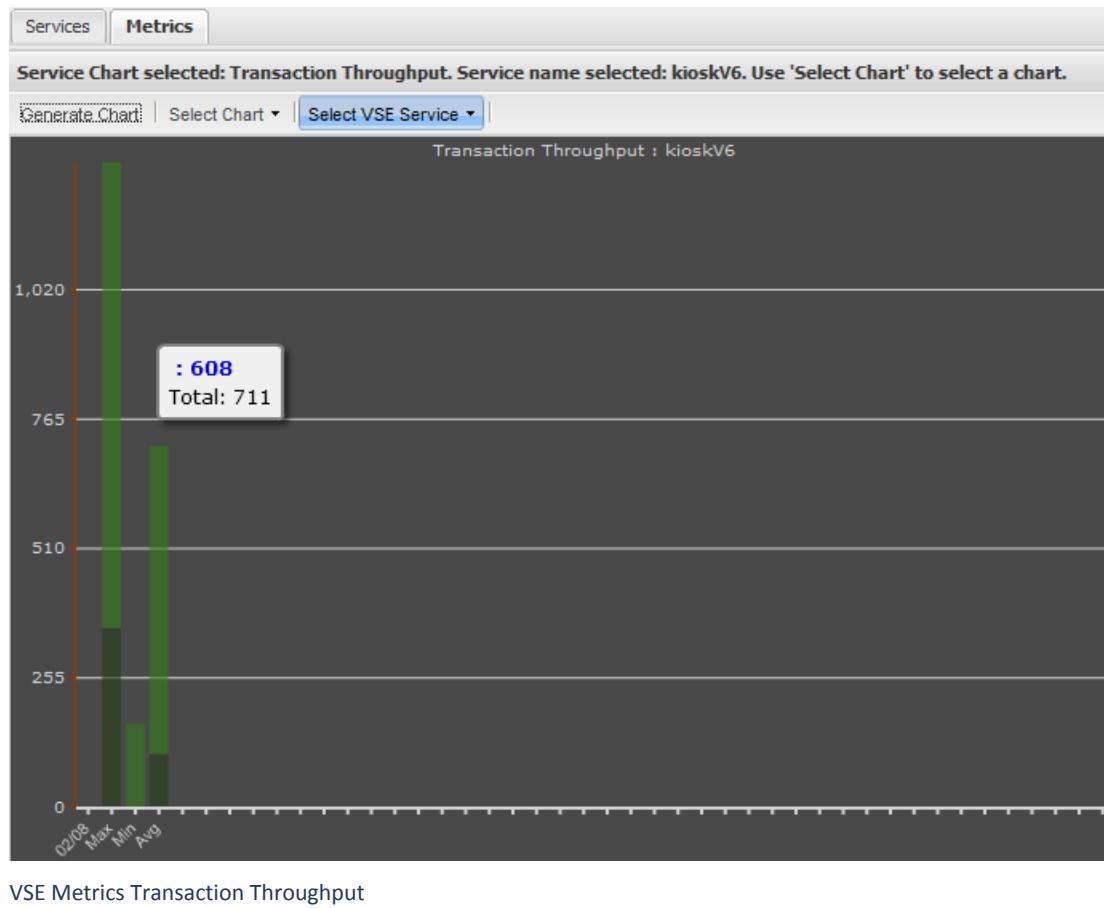


Transaction Hits and Misses



Total Transactions per Day





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.

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/8.0/updates/>
3. Select the VSE Manager UI feature, then click Next.
The Install Details window opens.

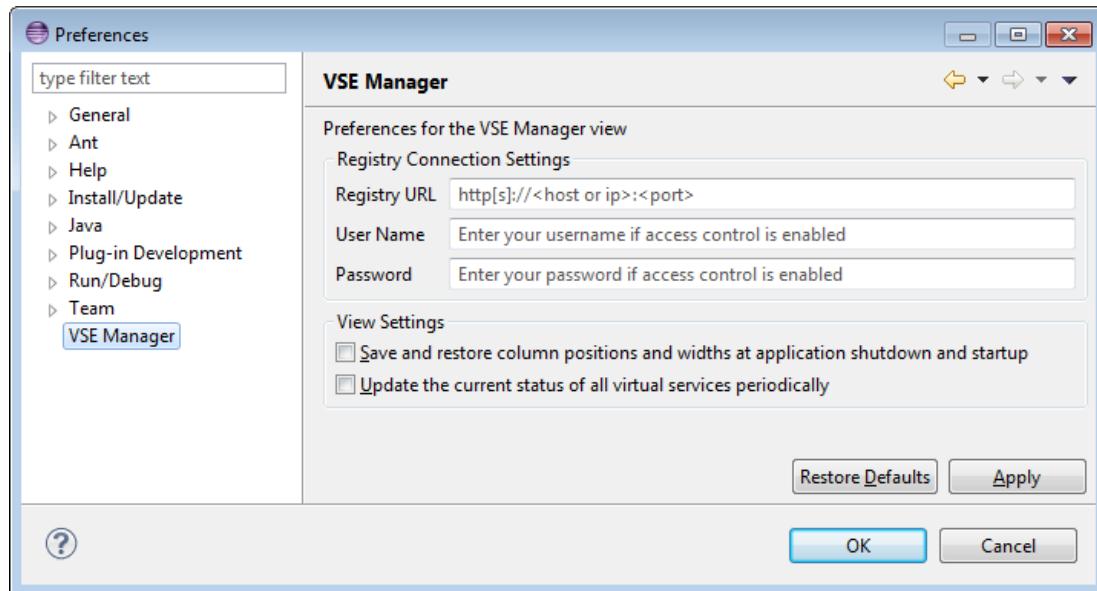
4. Review the install details, then click Next.
The Review Licenses window opens.

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 [Administering \(see page 1362\)](#).

- 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

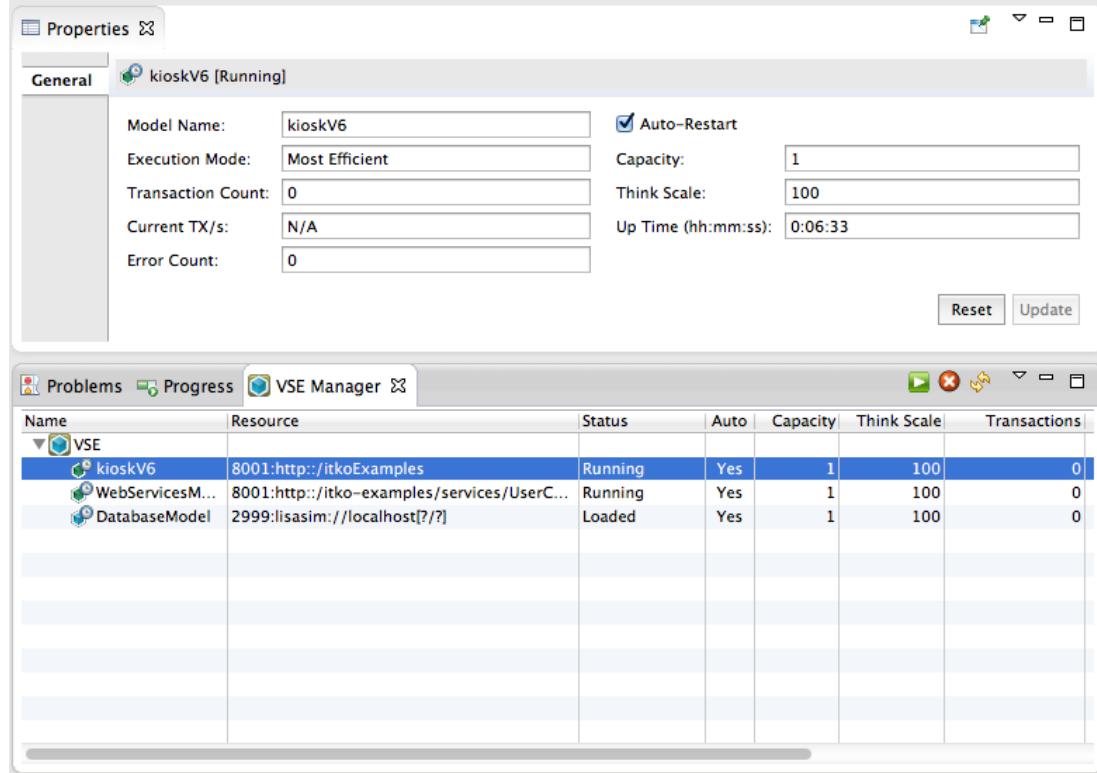
VSE Manager View panel

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.

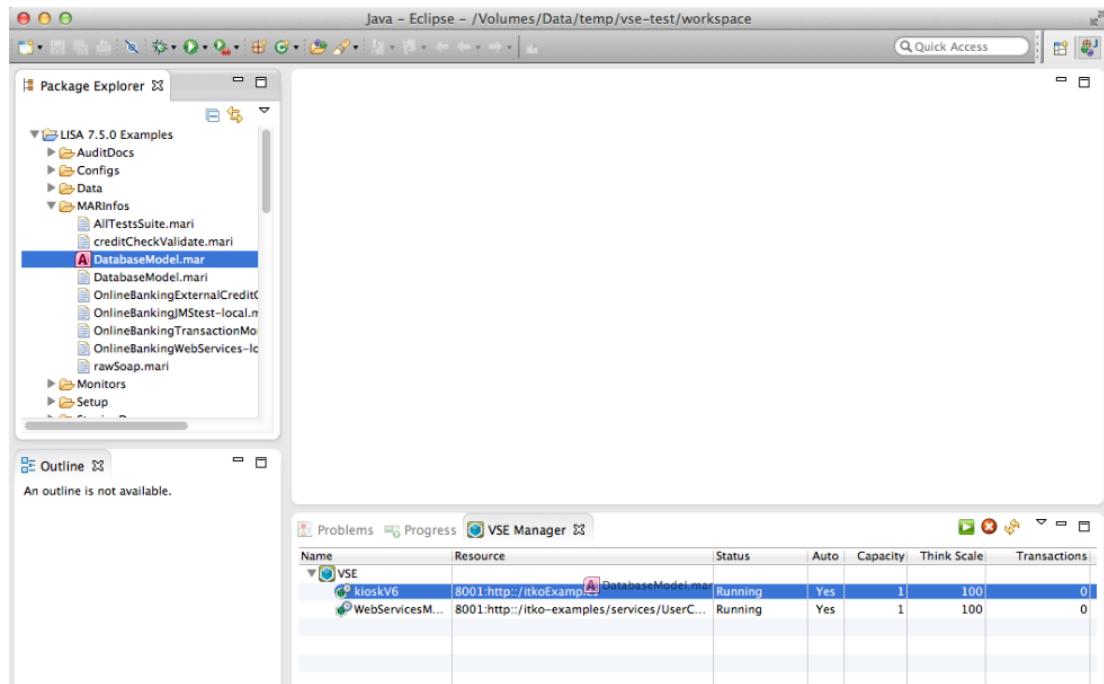


VSE Manager properties - integration view

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.



More Information:

- Video about [Integrating Eclipse with CA LISA Service Virtualization](https://www.youtube.com/watch?v=ZpwxoOdlh24) (<https://www.youtube.com/watch?v=ZpwxoOdlh24>)

VSE Commands

This section contains the following pages:

- [VSE Manager Command - Manage Virtual Service Environments](#) (see page 999)
- [ServiceManager Command - Manage Services](#) (see page 1001)
- [ServiceImageManager Command - Manage Service Images](#) (see page 1001)
- [VirtualServiceEnvironment Commands](#) (see page 1005)

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](#) (see page 1001).

VSE Manager is included as a tool under the bin directory in <lsrv>.

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]] [group tag 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](#) (see page 661) 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]] [--group tag *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.

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 \(see page 1452\)](#).

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***
Specifies the recording session file that contains all the configuration information for the recording or import operation.
- **--go -G**
Specifies that for interactive or disconnected styles, the wait for a start signal is bypassed.
- **--start=*time-spec* -S *time-spec***
Indicates that the recorder starts recording after the specified interval.
- **--stop=*time-spec* -E *time-spec***
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**

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.

- **--signal=start|stop -g start|stop**

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**

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>

- DRDA

- IMS Connect

- SAP RFC via JCo

- JCo IDoc Protocol

- Opaque Data Processing

- **--request-data=protocol -r protocol**

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**
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**
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**
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**
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**
Specifies the name of the virtual service model file to create during a recording.
- **--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**
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**
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

VirtualServiceEnvironment Commands

The VirtualServiceEnvironment command is used 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=port-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 VSE.

You can configure these properties from the [Agents window \(see page 1013\)](#) of the 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 CA Continuous Application Insight

CA Continuous Application Insight 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 CA Continuous Application Insight \(see page 1008\)](#)

- [Configuring Agents \(see page 1013\)](#)

- [Analyzing Business Transactions \(see page 1025\)](#)
- [Using the Shelf \(see page 1061\)](#)
- [Creating and Managing Tickets \(see page 1066\)](#)
- [Creating Virtual Services \(see page 1077\)](#)
- [Creating Baselines \(see page 1110\)](#)
- [Creating Data Sets \(see page 1153\)](#)
- [Creating Request and Response Pairs \(see page 1154\)](#)
- [Documenting Transactions for Testing \(see page 1156\)](#)
- [Native MQ CAI \(see page 1165\)](#)
- [CAI Command-Line Tool \(see page 1174\)](#)
- [VS Traffic to CA Application Insight Companion \(see page 1182\)](#)
- [CA Continuous Application Insight Agent Light \(see page 1183\)](#)
- [CAI Extension for Google Chrome \(see page 1196\)](#)

Getting Started with CA Continuous Application Insight

CA Continuous Application Insight (CAI) is designed to enhance the process of delivering applications to market.

The following roles are the intended users of CAI:

- Architects
- Developers
- Quality assurance engineers
- Data architects
- Release engineers

CAI captures the data that flows through an application. CAI uses this data to generate paths that can be viewed in the DevTest Portal. From the 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 the DevTest Portal.

To view a tutorial, see [CA Continuous Application Insight Tutorial \(see page 306\)](#).

The following video provides a brief overview and introduction to CAI.

CA Continuous Application Insight Prerequisites

CAI does not have specific system requirements. Review the DevTest Server requirements in [System Requirements \(see page 50\)](#).

You can use the following methods to add transactions to the CA Continuous Application Insight database:

- Install and configure the [DevTest Java Agent \(see page 1253\)](#) for a Java-based application
- Run the CAI Agent Light
- Run Native MQ CAI

In the LISA Bank demo application, the DevTest Java Agent is installed by default.

Once you add transactions to the database, you can view and analyze them in the DevTest Portal.

The DevTest Portal can monitor agents from the same release or from an older release. The 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 the 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 \(see page 1378\)](#).

The **Maximum buffer memory** property is a broker property that you can configure from the **Agents** window of the 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.

CA Continuous Application Insight Supported Platforms

CA Continuous Application Insight (CAI) 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
- 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 CAI and when they are supported, see [CAI Support Matrix \(see page 60\)](#).

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 \(see page 1017\)](#) 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**.

Open the DevTest Portal

You open the DevTest Portal from a web browser.



Note: For information about the server components that must be running, see [Start the DevTest Processes or Services \(see page 115\)](#).

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**.

Business Transactions and Transaction Frames

Business transactions and transaction frames are key concepts in CA Continuous 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 or a CAI Agent Light intercepted. The data includes such information as:

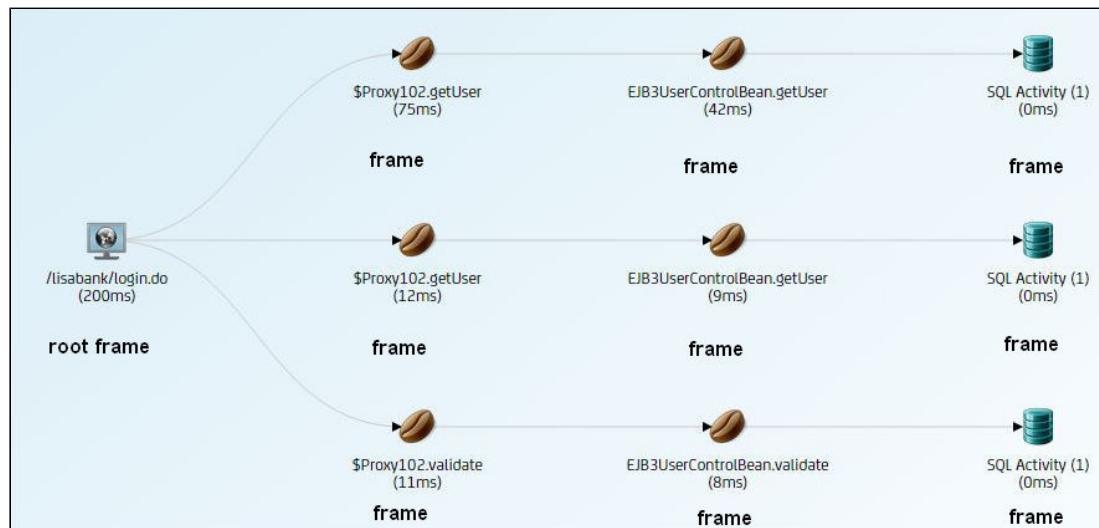
- The name of the method
- The name of the class to which the method belongs
- The arguments that were passed to the method
- The value that the method returned
- Log4j messages that are written
- Exceptions that are thrown by the Java classes

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 the DevTest Portal.



Screen capture of transaction path

SAP NetWeaver Notes

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.

Configuring Agents

You configure the agents from DevTest Portal from the Agents window. The **Agents** window lets you perform the following tasks:

- [View the agent status \(see page 1014\)](#)
- [Filter agents \(see page 1015\)](#)
- [View performance data \(see page 1016\)](#)
- [Configure the capture levels for an agent \(see page 1017\)](#)
- [View agent and broker information \(see page 1019\)](#)
- [Stop and start agent capture data \(see page 1020\)](#)
- [View and update configuration properties \(see page 1020\)](#)
- [Upgrade an agent \(see page 1021\)](#)
- [Delete an offline agent \(see page 1022\)](#)
- [Create and manage agent groups \(see page 1023\)](#)

You view the **Agents** window by selecting **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. When you select an agent or broker, the right pane of the **Agents** window displays detailed information. You can delete agents and upgrade agents to the latest version from this pane.

Agent groups can be added, configured, and managed. See [Create and Manage Agent Groups \(see page 1023\)](#) for more information about working with agent groups.

- **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.

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 change the state, click the down arrow next to **VSE Mode** and select an option from the menu.

Filter Agents

The **Agents** window lets you find an agent by narrowing down the list of agents that display in the left pane. You can sort and filter agents and can 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.
The default type is **Name**.
5. To filter agents, select an option from the **Filter by** list.
The default type is **Name**.
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**  .

View Performance Data

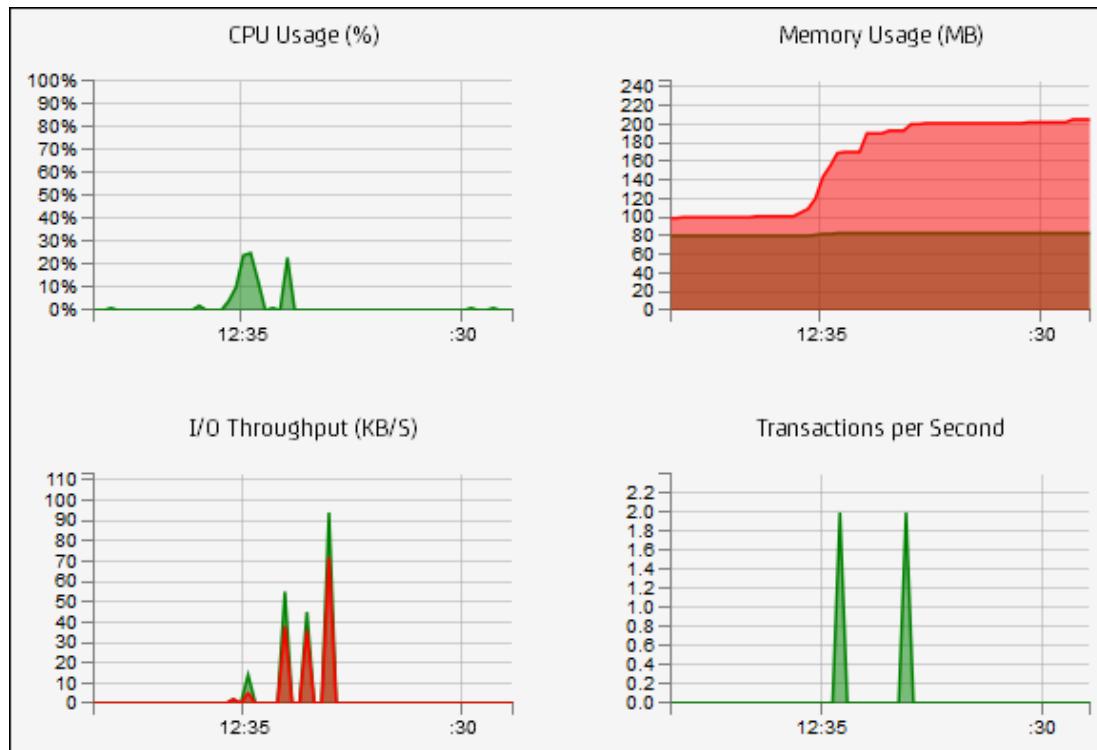
You can view a set of performance graphs for an agent or broker.

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.
5. To collapse the performance pane, click **Collapse panel** .
6. To expand the performance pane, click **Expand panel** .

Configure Capture Levels

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.

Adjust Captures for Protocol				Auto Refresh <input type="checkbox"/>
Sort by: Sort by Count		Protocol Capture: Select a capture level.		
580	HTTP Client	<input type="checkbox"/>	195	JMS
	Full Data	<input type="checkbox"/>		Full Data
32	JDBC	<input type="checkbox"/>	2	Logging
	Full Data	<input type="checkbox"/>		Full Data
150	HTTP Server	<input type="checkbox"/>	150	EJB
	Full Data	<input type="checkbox"/>		Full Data
150	Exception	<input type="checkbox"/>	150	SAP
	Full Data	<input type="checkbox"/>		Full Data
150	TIBCO	<input type="checkbox"/>	150	UI Framework
	Full Data	<input type="checkbox"/>		Full Data
150	webMethods	<input type="checkbox"/>	150	Java
	Full Data	<input type="checkbox"/>		Full Data

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](#) (see page 1066), ensure that the capture level for the **HTTP Server** protocol is set to **Full Data**.

If you want to create [baselines](#) (see page 1110) or [virtual services](#) (see page 1077), ensure that the capture levels for the appropriate protocols are set to **Full Data**.

A red badge displays the number of times that the agent has observed the protocol since the agent was started.

If the count is 0, the red badge does not appear.

If the count is more than one thousand, the red badge displays the text **999+**. 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 **Protocol Capture** 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, click **Show Protocols with counts**  .
6. To reset all of 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

You can view the following categories of information about an agent or broker:

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.



Note: The VM properties tab cannot display information 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.

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.

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 and Update Properties

You can view and update the configuration properties for an agent or broker.

The properties are grouped into categories for example, **General** and **Capture**.

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**.

De-identifying Data in CAI

CAI 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 the 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 \(see page 969\)](#).



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, 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, VSE might return a response that is different from the original response.

Upgrade an Agent in the DevTest Portal

From the **Settings, Agents** window, you can upgrade agents to the latest version in the DevTest Portal. The button appears next to the agent when a new version is available. You hover over to see the version information.



Note: Not all agents can be upgraded in the DevTest Portal. Native agents and agents using JRE 1.5 and **LisaAgent2.jar** must be [manually upgraded \(see page 1022\)](#).

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 displays.
3. From the **Agents** pane, click .
 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 the DevTest Portal \(see page 1021\)](#), you can 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 can delete an offline agent from the **Agents** pane of the **Agents** window. When an agent is deleted, all transactions and frames of the agent are removed.



Note: The **Delete**  button is enabled when an agent is offline.

Follow these steps:

1. Select **Settings, Agents** from the left navigation menu.
The **Agents** window displays.
2. From the **Agents** pane, locate the agent that you want to delete.
3. (Optional) Click **View filters**  and click **Turn on filters**  to use the filter criteria to search for an agent.
4. Click **Show more** .

5. Click **Delete** .

Create and Manage Agent Groups

The Agents window lets you create agent groups and apply settings to configure multiple agents simultaneously instead of one at a time. Agent groups can be added, configured, and managed in the agents pane. Each group displays a count of all agents that are in the group.

You can perform the following agent group configuration tasks:

- [Create agent group \(see page 1023\)](#)
- [Rename agent group \(see page 1023\)](#)
- [Delete agent group \(see page 1023\)](#)
- [Move agent to a group \(see page 1024\)](#)
- [Copy agent and group settings \(see page 1024\)](#)

Create Agent Group

From the agents pane, you can add an agent group. A group can have multiple agents or no agents.

To add a group, click the Add Group button, 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

You can change the name of an agent group 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. When deleting a group, the agents that are defined within the group are moved to the default agent group automatically.

Follow these steps:

1. Select the checkbox for the group or multiple groups to delete.
The Actions pane displays.

2. Select **Delete Groups and/or Agents** and click **OK**.

You can select multiple groups to delete.

3. 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, click  , hold, and drag into a group.
The following graphic displays the drag-and-drop handle is at the left of the agent status flag:


2. To move multiple agents, perform the following steps:

- a. Select the checkbox 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.

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

CAI displays the [business transactions \(see page 1011\)](#) that have been captured. These transactions are also referred to as *paths*.

CAI exposes information about the transactions and their frames. This information includes execution times, thread utilization, log messages, SQL statements, request and response data, and stack traces.

You view transactions from the **Analyze Transactions** window. Transactions display in the graphical or list view. By default, transactions are displayed in the list view.



Note: Frames for each transaction follow a specific execution order and are not arranged by a particular agent.

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 in this view:

- Search for paths by full text or by a duration of time
- View the transaction details by clicking the frame. The transaction details display a visual path of transactions and tabs that contain the details of the frame. For example, frame information, tags, request, response, thread name, log messages, XML, and exception when applicable.
- View paths for transactions that have identical transaction paths
- Filter paths using refinement options
- Generate artifacts for an agent
- Shelve and unshelve a transaction frame
- Pin and unpin transactions
- Share a link
- Share Point of Interest
- Export a path diagram to a PDF file
- Show and hide an outline of a path

- Disable capture

In list view, transactions are listed with the name, start time, wall time, CPU time, and agent information. You can do the following operations in this view:

- Search for transactions by full text or by a duration of time
- View information of a path. For example, the path and agent name, start time, wall time, and CPU time
- Generate artifacts for an agent
- Import and export transaction capability
- View the transaction details by clicking **Open transaction details**  in the **Actions** column. A visual path of transactions displays in the path graph and tabs contain the details of the frame. For example, frame information, tags, request, response, thread name, log messages, XML, and exception when applicable.
- Import and export transaction capability
- View paths for transactions that have identical paths

Follow these steps:

1. Select **Application Insight, Analyze Transactions** from the left navigation menu.
The **Analyze Transactions** window displays. By default transactions display in the list view.
2. To open the graphical view, click **Graphical** .
The transactions display in graphical view.
3. (Optional) Click **Show Outline** to view a portion of a large transaction at a time.



More Information:

- [View Transaction Details \(see page 1027\)](#)
- [Generate Artifacts for an Agent \(see page 1036\)](#)
- [Export Path Diagram to PDF \(see page 1037\)](#)
- [Merge Repeated Paths \(see page 1038\)](#)
- [Pin and Unpin Transactions \(see page 1042\)](#)
- [Exclude Data from Agent Capture \(see page 1042\)](#)
- [Search and Filter Transactions \(see page 1045\)](#)
- [Shelve Transactions \(see page 1054\)](#)

- [Exporting and Importing Paths \(see page 1056\)](#)
- [Share Link \(see page 1044\)](#)
- [Share Point of Interest \(see page 1044\)](#)
- [Working with Defects \(see page 1059\)](#)

View Transaction Details

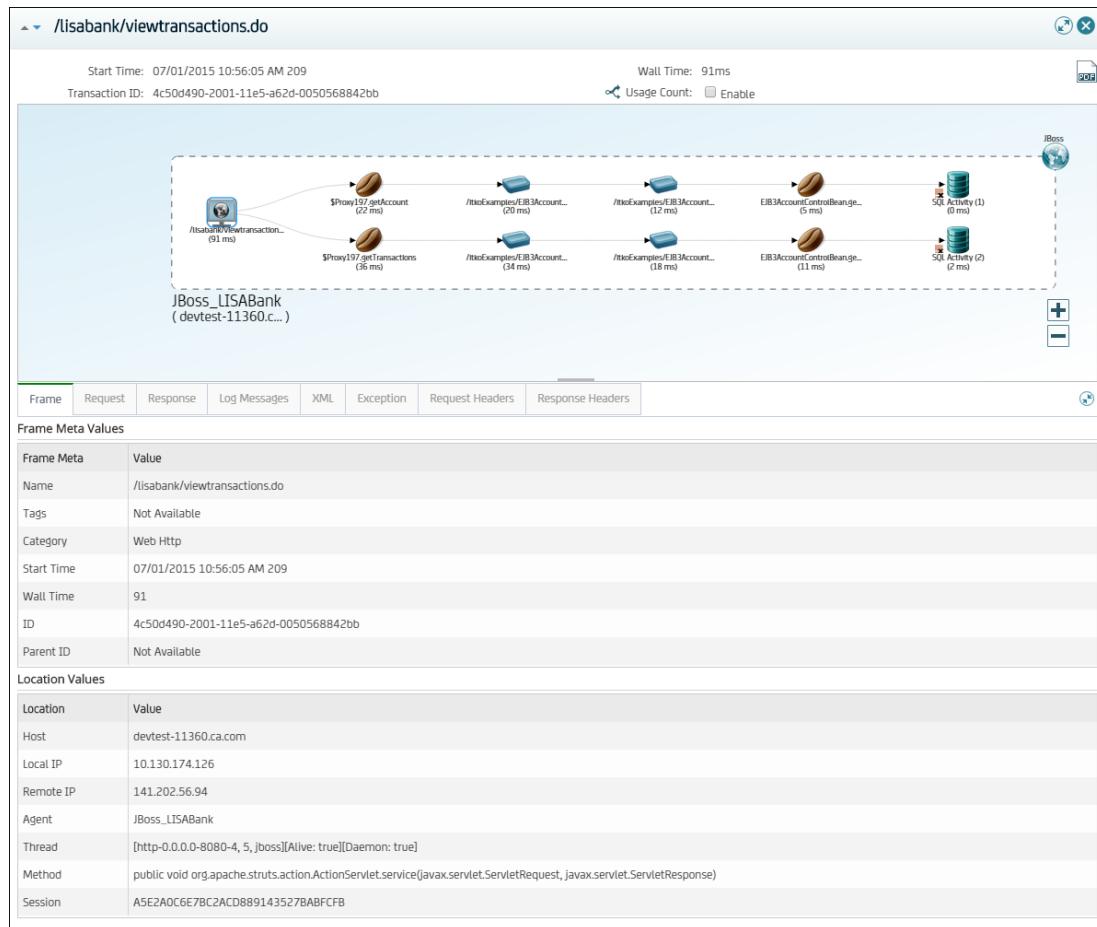
The transaction detail dialog contains information about each transaction in a path. The dialog is divided into the following areas:

- The upper area contains the [path graph \(see page 1030\)](#).
- The lower area contains the [frame information \(see page 1033\)](#).



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 Detail dialog

In the path graph, a visual path of the transaction displays an icon representing the container type. For example, container types can be JBoss, Websphere, and webMethod. Container types that are not recognized, display a Java icon by default. From the graphic, the path displays a JBoss icon as the container type.

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. From the graphic of the path graph, the SQL calls are not eligible for baseline creation.

You can view the usage counts for identical and partial transaction paths by selecting the **Enable** checkbox for the **Usage Count** field at the top of the path graph. See [View Usage Counts for Identical and Partial Transaction Paths \(see page 1039\)](#) for more information about this capability.

You can do the following operations by right-clicking a frame:

- Shelve, shelve all occurred transactions, or unshelve transactions
- Pin transactions
- Disable capture or disable URL capture
- Share link

- Share Point of Interest

Path Graph Information

The path graph displays the path and its frames in a graphical representation.

Agent name

The name of the agent that generated the path.

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 Information

The frame information displays the details of the selected frame in the path graph.

Frame

The frame information which includes meta and location values.

Tags

Any frame tags that the frame has.

Request

The actual request that the selected frame sent. This information does not appear for JDBC frames.

Response

The actual response that the selected frame received. This information does not appear for JDBC frames.

SQL

Information about the SQL call. This information includes the SQL statement, the results, and the output. This information appears only for JDBC frames.

Thread Name

The name of the thread on which the frame ran.

Log Messages

Any log messages. 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 \(see page 1013\)](#) of the DevTest Portal. The property appears in the **Settings** tab.

XML

The raw XML of the instrumentation response in the frame. This information does not appear for JDBC frames.

Exception

Displays detailed information when a transaction contains an exception.

Request Headers

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

Information about the components of the header section of response. This tab displays for only SOAP, REST, HTTP, HTTPS, and Web Service frames.

Follow these steps:

1. Select **Application Insight, Analyze Transactions** from the left navigation menu.
The **Analyze Transactions** window displays.

2. (Default) In list view, click **Open transaction details**  in the **Actions** column.

3. In graphical view, double-click the parent frame.
The transaction detail dialog opens. The title bar contains the full name of the frame.

4. Review the information in the path graph and the frame information.

5. Click the arrows at the left of the title bar to navigate to the previous and next transaction.

6. Click **X** to close the dialog.

Path Graph

A *path graph* contains a graphical representation of a path and its frames. The path graph displays in the upper area of the transaction detail dialog.

When you select a [root transaction frame \(see page 1011\)](#) 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 **graphical** view.

- clicking **Open transaction details** from the **Actions** column in **list** view.

The icons indicate the type of frame. Examples of the frames 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 model.

The text below the icon provides the following information:

- Frame name
- Execution time

Use Zoom In and Zoom Out to size the path display.

Frame Name

The format of the frame name depends on the type of frame.

The frame name for HTTP is the path portion of the URL. The name also contains the method. Paths that are created using the **VS Traffic to CAI Companion** are indicated with the virtualized label in the component name.

An example is **POST/itkoExamples/EJB3AccountControlBean(virtualized)**.

See [VS Traffic to CAI Companion \(see page 1182\)](#) 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

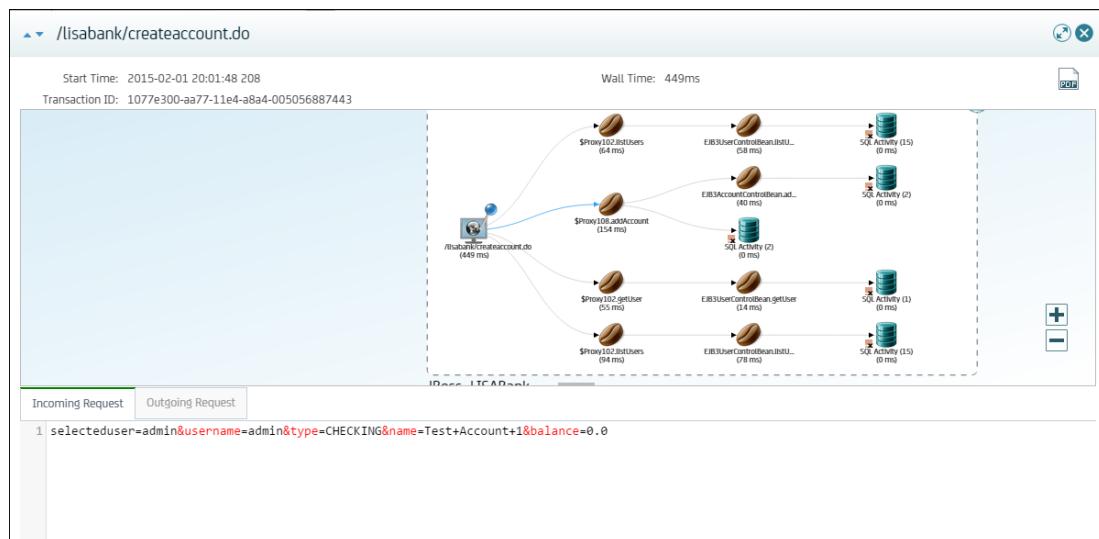
In certain scenarios, the execution time below the leftmost component is the time for the entire transaction.

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** tab and **Outgoing Request** tab displays 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 and response tab

Follow these steps:

1. Select **Application Insight, Analyze Transactions** from the left navigation menu.
The **Analyze Transactions** window displays in the list view.

2. Click **Open transaction details** in the **Actions** column.
The transaction detail dialog displays.

3. From the path graph, select a line between a frame.

The **Incoming** and **Outgoing** tabs display in the frame information portion of the transaction detail dialog.

Frame Information

The transaction detail dialog displays information about a frame that is selected in the path graph. The frame information displays in the lower portion of the transaction detail dialog.

The frame information contains the following tabs:

- **Frame**
- **Request**
- **Response**
- **Log Messages**
- **XML**
- **Exception**
- **Payload**
- **SQL Summary**
- **Request Headers**
- **Response Headers**
- **Incoming Request**
- **Outgoing Request**

The **Frame** tab and the **Log Messages** tab appear for all frame types.

Frame

The **Frame** tab shows the metadata values and the location values of the selected frame. The following details are listed:

Frame Meta Data

- 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 name: 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.

[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.

[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 \(see page 1013\)](#) 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 \(see page 1017\)](#) 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. See [Incoming and Outgoing Request Information \(see page 1032\)](#) for more information on how to view the request information.

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. See [Incoming and Outgoing Request Information \(see page 1032\)](#) for more information on how to view the request information.

Generate Artifacts for an Agent

You can generate virtual services and baselines directly from an agent without shelving any transactions. From **Analyze Transactions**, 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 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.

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 artifact options and click **Generate**.
5. Select a project, edit the prefix (optional), and click **Create**.
A confirmation dialog displays a link to the artifacts that were successfully created.
6. (Optional) Click a link and review the information or perform additional tasks related to the artifact type created.
A new tab opens with the project name of the artifact.
7. Click **OK** to close the Confirmation dialog.
8. Close the Agents dialog.

**More Information:**

- [Business Transactions and Transaction Frames \(see page 1011\)](#)
- [Search for Transactions by Time \(see page 1049\)](#)

Export Path Diagram to PDF

You export a path diagram into a PDF document to view, print, and save 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 \(see page 1045\)](#) for more information about refining your search results to find transaction paths.



Note: 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.

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**.

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.

You can merge repeated paths in the list or graphical view.

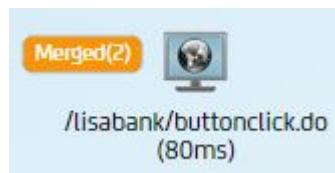
In the graphical and list view, repeated paths are indicated with an orange badge. The number in the badge is the total number of paths that are repeated for the transaction.

The following graphic displays the list view for a transaction that has two merged, repeated paths:

Name	Start Time	Wall Time	CPU Time	Agent	Actions
/lisabank/buttonclick.do Merged(2)	2014-10-30 10:07:43 0...	80 ms	46 ms	JBoss_LISABank	
/lisabank/addaddress.do	2014-10-30 10:07:40 5...	244 ms	140 ms	JBoss_LISABank	
/lisabank/buttonclick.do	2014-10-30 10:06:49 2...	397 ms	359 ms	JBoss_LISABank	

Screen capture of merge repeated paths in list view

The following graphic displays a transaction with two merged, repeated paths in the graphical view:



Screen capture of a merged repeated 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.
The **Analyze Transactions** window opens.
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.



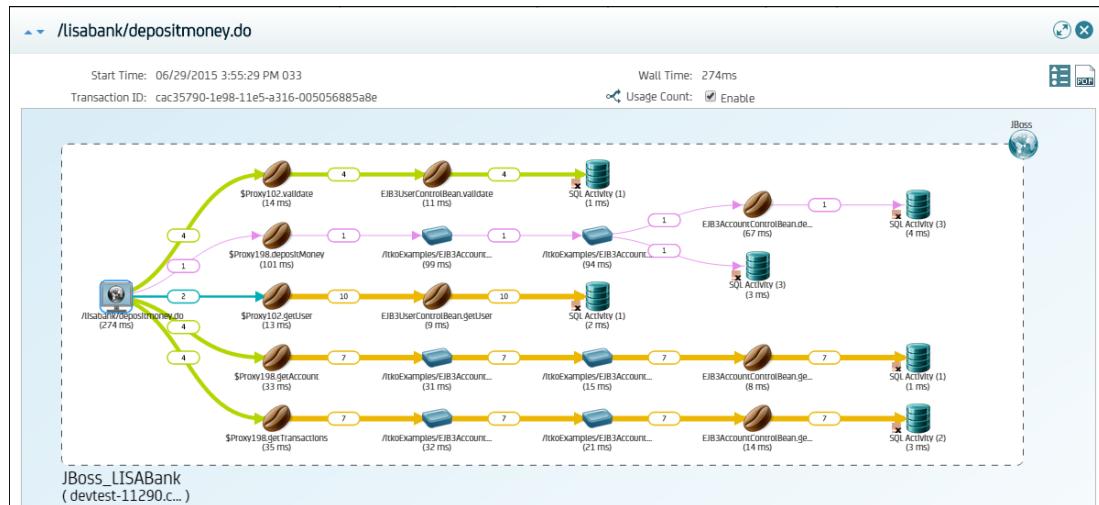
3. To unmerge the transactions, click

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 details 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:



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 **Open transaction details**  in the **Actions** column.

- In the graphical view, double-click the parent frame.

The transaction detail dialog opens.

4. Select the **Enable** checkbox 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.

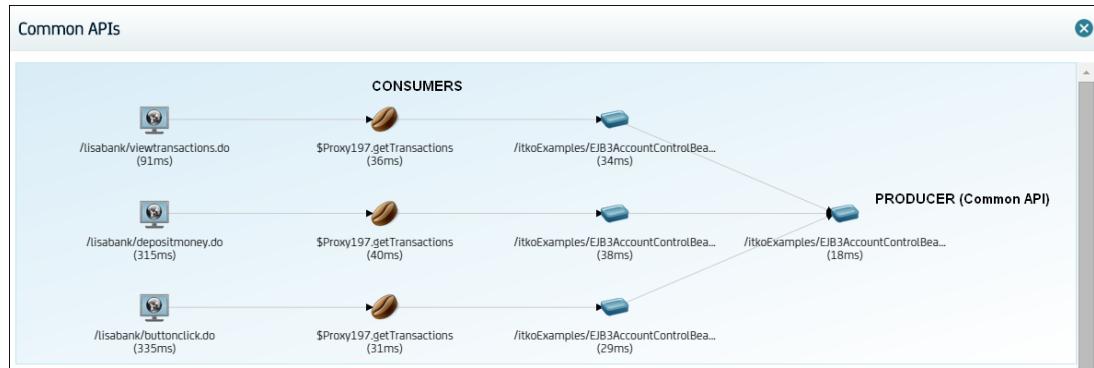
View Producer and Consumer Relationship



Note: The common API feature is in a preview phase of this release. See [Enable and Disable Preview Features \(see page 204\)](#) for more information about enabling and disabling the preview features.

From the Analyze Transactions window, you can identify and visualize the transactions of producers and consumers for an agent. The producer is also known as the common API. 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 allows a developer to understand how changes to the API can affect multiple consumers.

The following graphic displays the **Common APIs** dialog with a visual representation of the consumer and producer relationship:



Screen capture of the Common API dialog

Transaction frames from the following protocols support the common API feature:

- EJB
- HTTP
- HTTPS
- JMS
- MQ
- REST
- RMI
- Web Service

You can shelf the transaction frames in the **Common APIs** dialog by right-clicking a frame and selecting one of the following shelving methods:

- **Shelf Frame**
This option is available for consumer and producer transactions.
- **Shelf All Occurred Transactions**
This option is only available for consumer transactions.

The shelved producer or consumer transactions are labeled in the shelf as **Common API Frame**. See [Shelve Transactions \(see page 1054\)](#) and [Using the Shelf \(see page 1061\)](#) for more information about shelving transactions.

Follow these steps:

1. Open the Analyze Transactions window by selecting **Application Insight**, **Analyze Transactions** from the left navigation menu.
The Analyze Transactions window displays.
2. Click **Common APIs**  and select an agent from the drop-down list.
The common APIs for the agent display within the pane.

3. Select the checkbox for the common APIs that you want to display the producer and consumer and click **Show Common**.
The Common APIs dialog displays the producer and consumer for the common APIs for the agent.

Pin and Unpin Transactions

You can pin transactions to identify them as points of interest (POI). 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. Open the **Analyze Transactions** window.
2. In list view (default), click **Open transaction details**  (see page 1042), right-click a transaction, and select **Pin Transaction**.
3. In graphical view, right-click the transaction 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 click **Refresh** in the **Points of Interest** portlet.
The pinned transaction displays in the **Points of Interest** list.

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 \(see page 1282\)](#).

Some frames and Java classes do not support being excluded.

The following graphic shows an excluded URL.

The screenshot shows a window titled "Exclude from capture" with the sub-instruction "Select item to make available for capture". A tree view displays a single node under "Rule": "JBoss_LISABank (1)". Underneath it, a specific URL is listed: "http://localhost:8080/lisabank/buttonclick.do". In the bottom right corner of the window, there is a button labeled "Include".

Screen capture of excluded URL.



Note: 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.

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 **Open transaction details** from the **Actions** 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**.

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 CA Continuous 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 **Open transaction details**  in the **Actions** 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.

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 CA Continuous 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 \(see page 1042\)](#).

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 **Open transaction details**  in the **Actions** column or click **Graphical** .

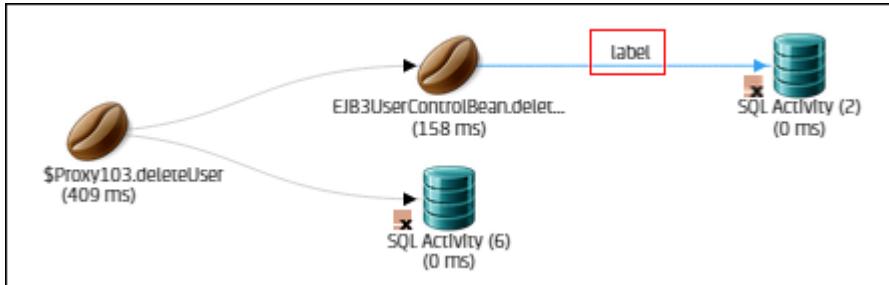
3. Right-click the transaction and select **Share, Share Point of Interest** to view the URL.

4. Click **X** to close the **Share Point of Interest** dialog.

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 \(see page 1027\)](#).

The following graphic displays a label in the path graph:



Screen capture of path graph with labeled link

Follow these steps:

1. Open the **Analyze Transactions** window.
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**.

Search and Filter Transactions

CAI 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

You enter text strings and can select a time period to limit the number of transactions frames that display. The default time period is **1 Day**.

You can search transactions using the following procedures:

- [Intelligent Search for Transactions \(see page 1048\)](#)
- [Search for Transactions by Time \(see page 1049\)](#)



Click to display the **Refine By** pane to use the filtering options. The following graphic displays the **Refine By** pane:

Refine By

Agent
 JBoss_LISABank

Category
 Web Http
 Web Service Http
 EJB
 JDBC
 Framework

Class Name
Auto Complete Search...
 com.itko.examples.ejb3.EJB3AccountControlBean
 com.itko.examples.ejb3.EJB3UserControlBean
 com.sun.proxy.\$Proxy102
 com.sun.proxy.\$Proxy108
 com.sun.proxy.\$Proxy197

Exec Time (ms)
Enter Search Number Go

Local IP
 10.130.175.141

Operation
Auto Complete Search...
 _jspService
 addAccount
 commit
 createQuery
 depositMoney

Point of Interest
Enter Search Text Go

Remote IP
 10.130.175.141

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.

The following refinement icon displays when a filter is applied:



Screen capture of refinement filter with filter applied

You can filter transactions using the following procedures:

- [Filter transactions \(see page 1049\)](#)
- [Filter Frames Within Transactions \(see page 1050\)](#)
- [Filter by Starting and Ending Transactions \(see page 1054\)](#)

Intelligent Search for Transactions

CAI enables you to find transactions using natural language processing (NLP) keywords. You enter a payload text search from the **Enter Search Text** field. You can enter a payload search string for example, **response has lisa_simpson and category has EJB**.

You can also use keywords in the search. For example, you enter a transaction ID, a category like EJB, and a local or remote IP address.

The following graphic displays the search bar in **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,	is, has, have, include,	and, or	n = JBoss_LISA_Bank
exec time, local IP, remote IP, response,			n + v + n = category is Java
request, transaction id, transaction time,			n + v + n + c + n= point of interest has EJB and Java
point of interest, operation, tags			

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 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.

You specify the following time-range by:

- Minutes (1 through 30)
- Hours (1 through 12)
- Days (1 through 15)
- All Time

You specify the following date-range criteria in the **Advanced** time option:

- Start date and time
- End date and time

The default date range is **1 Day**.

Filter Transactions

You use the filter options in the **Refine By** pane to narrow down the number of transactions that appear in the **Analyze Transactions** window. You enter a filter option from the search text fields or select the check boxes. For example, if you know the class name you want to filter, select the corresponding **Class Name** check box. You can select multiple filter options. 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 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

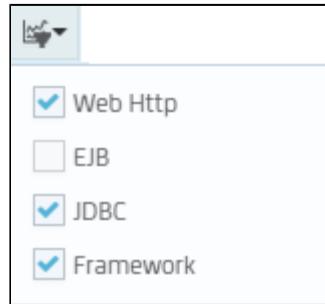
In the graphical view, you can filter the frames that occur within transactions. For example, the following graphic displays paths that contain EJB frames:



Screen capture of filter frames within transactions

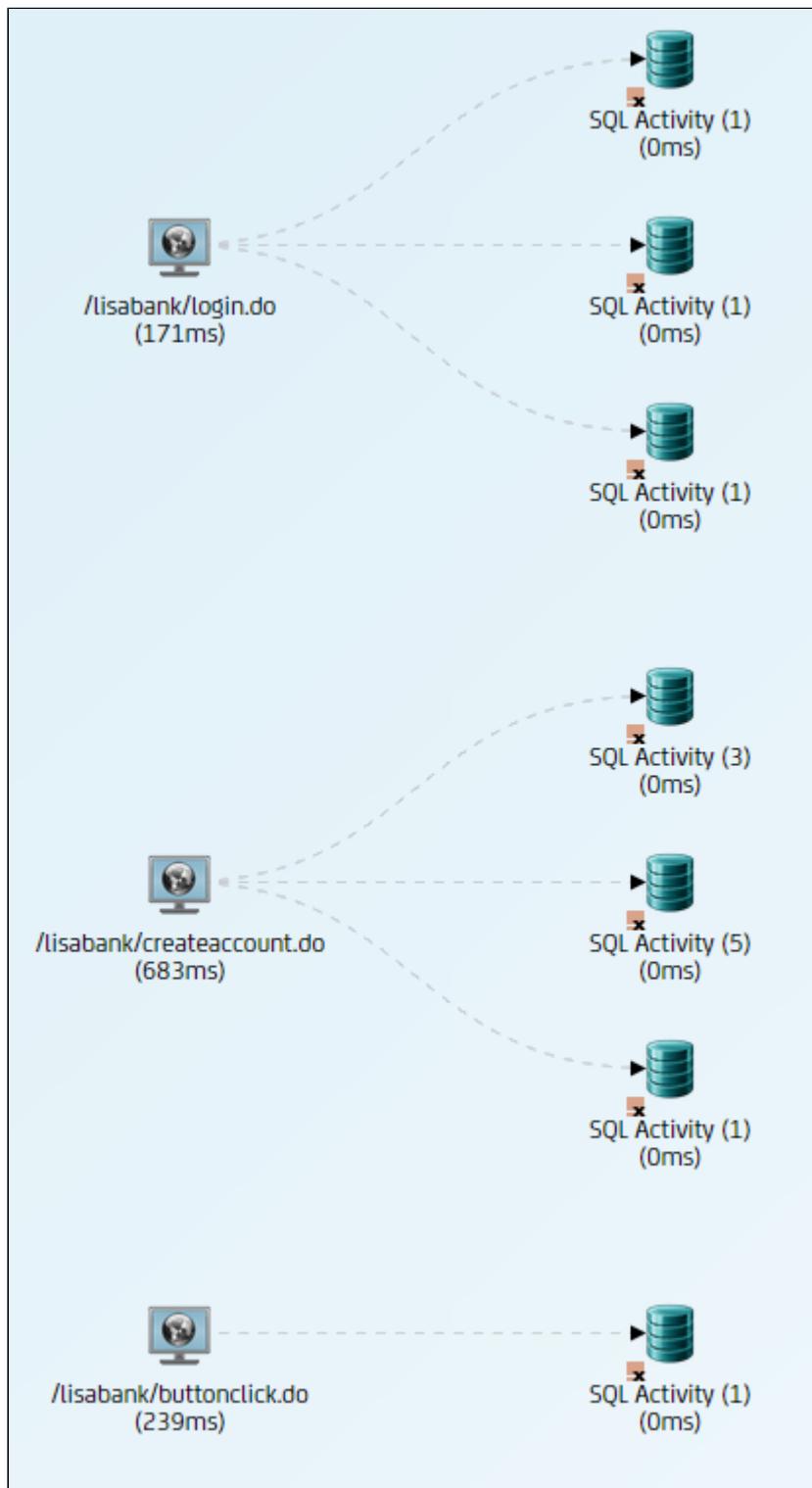
You can remove the EJB calls from the path by clicking **the Filtering options**  and clearing the **EJB** check box.

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** in the left navigation menu.
 2. Click **Graphical**
 3. Click **Filtering Options**
- The transactions display in the graphical view.
- The transaction paths display the frames according to the selected or cleared category.

Filter by Starting and Ending Transactions

In Analyze Transactions, transactions that display can be further refined by setting a start and an end transaction filter. 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** in the left navigation menu.
The Analyze Transactions window displays in the list view by default.
2. Perform one of the following operations:
 - 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.

Shelve Transactions

From the **Analyze Transactions** window, you can 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 R/R 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 \(see page 1061\)](#)



Note: When the login session expires or 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 \(see page 1027\)](#). Right-click a transaction and select one of the following shelving options:

- **Shelve Frame**
You can add individual frames into the shelf.

- **Shelve Frames (Advanced)**

Shelving with the advanced options lets you shelve the same types of transaction frames from selected transactions or from transactions based on a search result.

- **Same category of frames from this transaction**

Allows you to shelve transactions with the same category by selecting frames from the list and clicking **Shelve Frames**.

- **Same category of frames from the search result**

Allows you to shelve transactions with the same category from a search result by selecting the frames in the list and clicking **Shelve Frames**.

- **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. An example of all occurred transactions is 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:

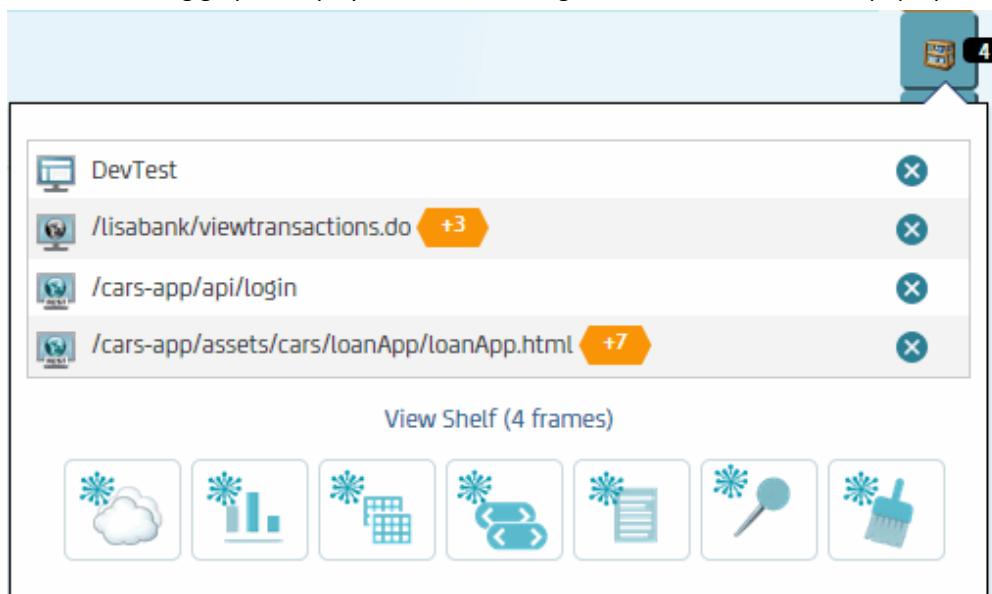


Screen capture of a shelved all occurred transactions in graphical view

- **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 increase by one. See [Merge Repeated Paths \(see page 1038\)](#) for more information about merging repeated transactions.

The following graphic displays shelved and merged transactions in the shelf pop-up:



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 list view by default.

2. Click  in the **Actions** column or click .

3. Right-click a transaction and select a shelving option.
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 shelve transactions with the same frame type or from a search result:
 - a. right-click a transaction and select **Shelf Frames (Advanced)**.
 - b. select an option to shelve frame from the selected transaction or from the search result.
 - c. select frames from the list to shelve and click **Shelf Frames**.

5. To unshelve a transaction, right-click a transaction, and select **Unshelve**.

Exporting and Importing Paths

You can export paths from the CAI database using the **Analyze Transactions** window. The paths can be imported into other CAI installations or can be imported from Splunk. You can export paths for CA Support to debug.

More Information:

- [Export Paths \(see page 1056\)](#)
- [Import Paths \(see page 1058\)](#)
- [Import Paths from Splunk \(see page 1058\)](#)

Export Paths

CAI 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. Hold down the Shift key to select multiple transactions.
- **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 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** in the left navigation menu.
2. Use the search refinements to find the transactions you want to export.



3. Click 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 \(see page 1048\)](#)
- [Search for Transactions by Time \(see page 1049\)](#)

- [Filter Transactions \(see page 1049\)](#)
- [Import Paths \(see page 1058\)](#)

Import Paths

You can import a zip file that contains transaction paths from various sources into the DevTest database.

Follow these steps:

1. Select **Application Insight, Analyze Transactions** in the left navigation menu.



2. Click and select **Import**.
The **Import Path** dialog displays.

3. Click **Choose File**, 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**.
The imported paths display in the list.

Import Paths from Splunk

From the **Analyze Transactions** window in list or graphical view, you can import paths that were captured from Splunk. See [Agent Light \(see page 1183\)](#) for more information about using Splunk and the CAI Agent Light. The **Search** field in the **Import from Splunk** dialog is the agent light built-in Splunk search string. You can enter different search strings that generate the Splunk HTTP transactions. See [Agent Light for HTTP \(see page 1187\)](#) for more information about the HTTP search query language syntax.

Follow these steps:

1. Select **Application Insight, Analyze Transactions** in the left navigation menu.
The **Analyze Transactions** window displays.



2. Click and select **Import from Splunk**.
The **Import from Splunk** dialog opens.

3. Enter the fields and click **Import**.

A confirmation dialog indicates that the import was successful.

4. Click **OK** to close the dialog.

5. Click **Refresh**  to display the Splunk paths.

The imported paths from Splunk display at the top of the list.

Working with Defects

The **Analyze Transactions** window enables you to 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 \(see page 1059\)](#)
- [Pin annotated transactions \(see page 1042\)](#)

From the shelf, you generate virtual services, baselines, and produce defect documentation for transactions with defects. For more information, see [Using the Shelf \(see page 1061\)](#).

Annotate a Transaction to View Defects

CAI 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**

Identify and view log messages for errors and warning.

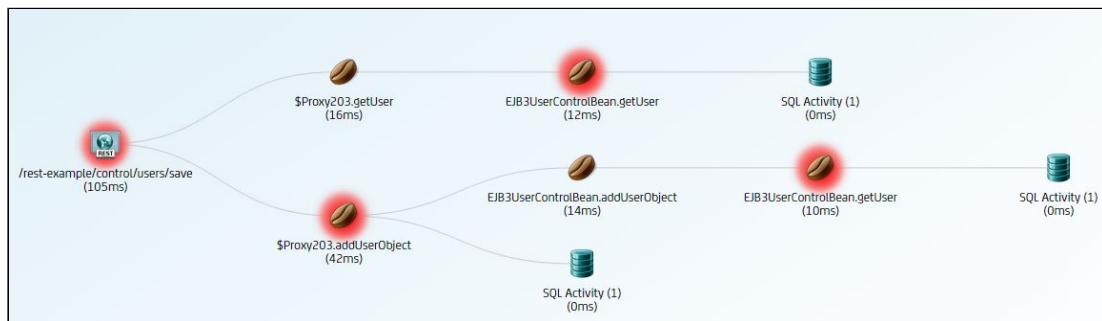
- **Response times percentile**

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, **Point 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) shelf the transaction and generate artifacts. For more information about shelving, see [Shelving Transactions \(see page 1054\)](#).

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

From the shelf, you can generate artifacts and can set POIs for multiple shelved transaction frames. For information about how to add transactions to the shelf, see [Shelve Transactions \(see page 1054\)](#).

The shelf is accessible from the **Analyze Transactions** window in the list and graphical view. The number that is located to the right of the icon indicates the total number of shelved frames.

The following graphic displays the shelf icon:



Note: Transactions in the shelf are removed when the login session expires or the browser running DevTest Portal is closed or refreshed.

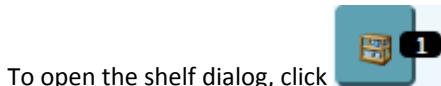
You can generate the following artifacts:

- Virtual services
- Baselines
- Data sets
- Request/response pairs
- Documentation

The shelf has two methods of generating artifacts:

- Shelf dialog
This method allows you to search and manage shelved transactions and create artifacts.
- Shelf popup
This method provides a quick way to create artifacts without opening the shelf dialog.

Shelf Dialog

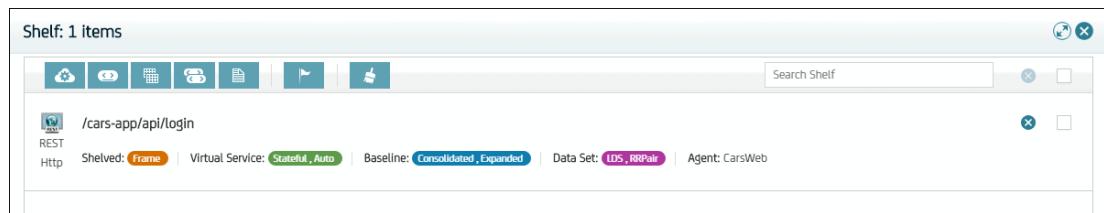


To open the shelf dialog, click or click **View Shelf (X frames)** from the popup.

From the shelf dialog, you can perform the following operations on the transactions listed:

- Search for transactions
- Delete a transaction frame
- Clear the entire contents of the shelf
- Create artifacts - virtual service, baseline, Large Data data sets, RR pairs, and documentation. The shelf dialog identifies which artifacts you can generate.
- Set Point of Interest

The following graphic displays the shelf dialog with labels of artifact types that can be generated:



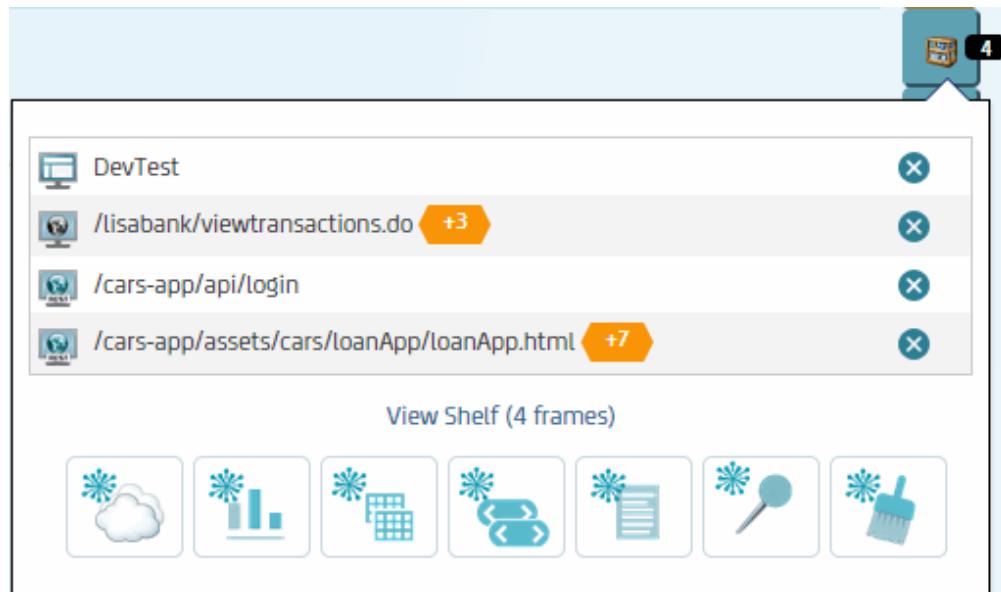
Screen capture of the shelf dialog with artifact labels

The types of artifacts that can be generated for each transaction are indicated with colored labels. You can create a stateful virtual service, consolidated and expanded baseline, Large Data data set, and R/R pairs.

Shelf Popup

The shelf provides a quick method of generating artifacts. Point to the shelf icon to view the shelf popup. From the popup, you can manage the shelved transactions without opening the shelf dialog.

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:

- [How to Work with Transactions in the Shelf \(see page 1063\)](#)
- [Creating Virtual Services \(see page 1077\)](#)
- [Creating Baselines \(see page 1110\)](#)
- [Creating Data Sets \(see page 1153\)](#)
- [Creating Request and Response Pairs \(see page 1154\)](#)

How to 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. See [Shelve Transactions \(see page 1054\)](#) for information about how to shelve a transaction frame.

1. [Search for Paths in the Shelf \(see page 1064\).](#)
2. [Delete Frames in the Shelf \(see page 1064\).](#)
3. [Set Points of Interest Transactions in the Shelf \(see page 1064\).](#)

4. [Generate Artifacts \(see page 1064\).](#)
5. [Create Documentation from the Shelf. \(see page 1065\)](#)
6. [Clear All Transactions from the Shelf \(see page 1066\).](#)

Search for Paths in the Shelf

The search capability lets you search for paths 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 pop-up.

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 identify transactions as a point of interest (POI). The points of interest transactions that are set in the shelf display in the **Points of Interest** portlet from the **Home** page dashboard.

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 generate the following business artifacts for transaction frames in the shelf:

- Virtual services
- Baselines
- Data sets
- RR Pairs

The types of artifacts that can be generated are indicated with colored labels in the list of transactions.

The following procedure assumes that you have shelved a transaction frame. See [Shelve Transactions \(see page 1054\)](#) for information about how to shelve a transaction frame.

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**.



When creating baselines, virtual services, and data sets, there may 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 \(see page 1077\)](#)
- [Creating Baselines \(see page 1110\)](#)
- [Creating Data Sets \(see page 1153\)](#)
- [Creating R/R Pairs \(see page 1154\)](#)

Create Documentation from the Shelf

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:

- Agent name
- Machine name
- IP address
- Agent version
- Properties
- Summary information such as the number of folds and response times

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.

Creating and Managing Tickets

CAI lets you capture detailed information about application behavior, including defects. This feature can help speed up the resolution of defects by enhancing collaboration between testers and developers.

The application must be running on an agent-enabled computer. A ticket is created in the application. The ticket is viewed in the DevTest Portal.

Tickets include such information as a title, severity level, and description. Tickets also include the corresponding [paths \(see page 1025\)](#), which expose the actions of the underlying components.



More Information:

- [Create Tickets in an Application \(see page 1067\)](#)
- [Email Settings for New Tickets \(see page 1069\)](#)

- [Send Tickets to HP ALM - Quality Center \(see page 1069\)](#)
- [Managing Tickets \(see page 1070\)](#)

Disabling the Tickets Feature

To make this functionality work, CAI injects JavaScript into the user interface code of the application.

If you want to prevent CAI 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 the 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 an agent-enabled computer.

The following graphic shows the ticket submission dialog.

The screenshot shows a modal dialog box titled "CA Continuous Application Insight". It contains fields for "Title", "Email", "External Defect Tracking Number", "Severity" (set to "Low"), and a large "Description" area. At the bottom, there is a checked checkbox labeled "Take screenshot" and a blue "Submit" button.

Screen capture of ticket submission dialog.

For the email functionality to work, the [email settings \(see page 1069\)](#) 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 \(see page 1069\)](#).

Before you begin, open the **Settings, Agents** window in the DevTest Portal and ensure that the [capture level \(see page 1017\)](#) 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.

For the take screenshot functionality to work, you must configure browsers. See [Configure Browsers to Support Screenshots \(see page 1076\)](#) 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, CAI 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 \(see page 1020\)](#) 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 \(see page 1067\)](#), 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

CAI 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 broker.
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. Save the property changes.
7. Restart the DevTest Java Agent.

Managing Tickets

The **Manage Tickets** window lets you view all the tickets that were captured and stored in the CAI database.

You can perform the following operations:

- [Search Tickets \(see page 1071\)](#)
- [View the Transactions for a Ticket \(see page 1073\)](#)
- [Edit Ticket Information \(see page 1073\)](#)
- [Obtain the Direct URL of a Ticket \(see page 1072\)](#)
- [View a Screen Shot for a Ticket \(see page 1073\)](#)
- [Generate Artifacts for a Ticket \(see page 1074\)](#)
- [Generate Documentation for a Ticket \(see page 1075\)](#)

You search for tickets by a specific reporter, status, and duration of time. Tickets appear in descending order by date. New and updated tickets also appear in the **New Tickets Alert** portlet in the **Home** page.

Tickets can be one of following statuses:

- **New**
- **Identified**
- **Closed**

How to Manage a Ticket

This page takes you through the steps of managing a ticket in CAI.

1. [Search Tickets \(see page 1071\)](#)
2. [Identify Transaction Defects \(see page 1072\)](#)
3. [Obtain the Direct URL of a Ticket \(see page 1072\)](#)
4. [View the Transactions for a Ticket \(see page 1073\)](#)
5. [View Screen Shot for a Ticket \(see page 1073\)](#)
6. [Edit Ticket Information \(see page 1073\)](#)
7. [Generate Artifacts for a Ticket \(see page 1074\)](#)
8. [Generate Documentation for a Ticket \(see page 1075\)](#)

Search Tickets

You search for tickets to display the defects that are found during testing from the **Manage Tickets** window. The **Manage Tickets** window contains search criteria for finding tickets in the CAI 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 updated display in the **New Ticket Alerts** portlet in the **Home** page.

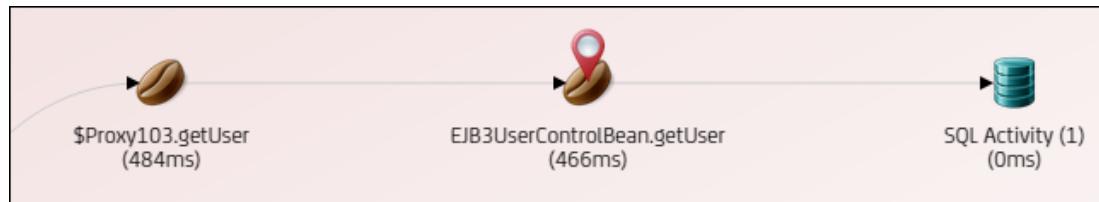
Identify Transaction Defects

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

CAI lets you 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](#)

CAI lets you 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.

Ticket Editor

Title:	ticketName	Date:	2014-07-08 08:57:05 645
Reporter:	Rob Banks	Status:	<input checked="" type="radio"/> New <input type="radio"/> Identified <input type="radio"/> Closed
Defect:	RTC-123	Severity:	Low ▾
Description:	This is a test of the emergency broadcasting system.		

Cancel **Save**

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.



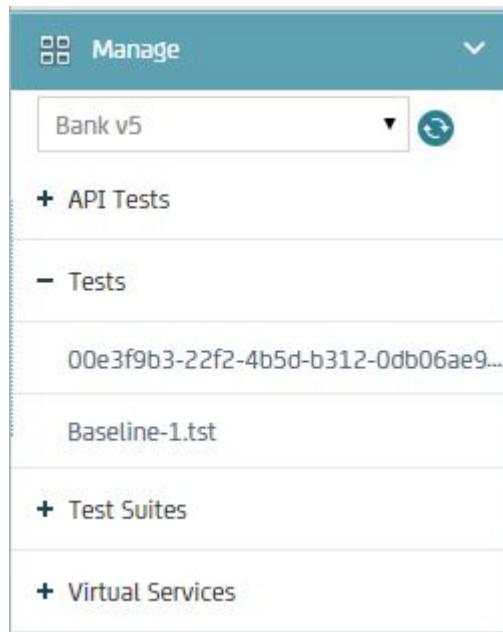
Note: If you change the **Status** from **New** to **Identified** or **Closed**, the ticket is removed from the **New Tickets Alert** portlet in the **Home** page dashboard.

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.

Follow these steps:

1. Select **Application Insight, Manage Tickets** in the left navigation menu.
2. Click **Graphical**  , right-click a transaction, and select **Generate Document**.
The **Title for Report** dialog displays.
3. Enter a title and click **OK**.
The report opens in a separate browser.
4. To save, click **Save** from the menu bar at the bottom, right of the browser, enter a title and click **OK**.
5. 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> (<chrome://plugins/>).
 - b. For Java(TM), select **Enable** and **Always allowed to run**.

Creating Virtual Services

CAI lets you create virtual services from transactions in the CAI database.



Notes:

- This feature requires a separate license.
- For detailed information about service virtualization, see [Using CA Service Virtualization \(see page 661\)](#).

When you generate the transactions, ensure that the [capture levels \(see page 1017\)](#) for the appropriate protocols are set to **Full Data**.

You can create virtual services by using the following approaches:

- **Analyze Transactions** window in the DevTest Portal
- **Document Transactions** window in the DevTest Portal
- CAI 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, CAI 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: JMS and SAP protocols are not supported.

See [Shelve Transactions \(see page 1054\)](#) for more information about all occurred transactions. For more information about merged transactions, see [Merge Repeated Paths \(see page 1038\)](#).

This section contains the following pages:

- [Create Virtual Services from EJB Transactions \(see page 1079\)](#)
- [Create Virtual Services from Java Transactions \(see page 1081\)](#)
- [Create Virtual Services from JCA Transactions \(see page 1083\)](#)
- [Create Virtual Services from JDBC Transactions \(see page 1085\)](#)
- [Create Virtual Services from JMS Transactions \(see page 1086\)](#)
- [Create Virtual Services from REST Transactions \(see page 1088\)](#)
- [Create Virtual Services from SAP ERPConnect Transactions \(see page 1089\)](#)
- [Create Virtual Services from SAP IDoc Transactions \(see page 1092\)](#)
- [Create Virtual Services from SAP JCo Transactions \(see page 1095\)](#)
- [Create Virtual Services from TIBCO ActiveMatrix BusinessWorks Transactions \(see page 1099\)](#)
- [Create Virtual Services from Web HTTP Transactions \(see page 1102\)](#)
- [Create Virtual Services from Web Service Transactions \(see page 1103\)](#)
- [Create Virtual Services from webMethods Transactions \(see page 1105\)](#)
- [Create Virtual Services from WebSphere MQ Transactions \(see page 1107\)](#)
- [VRS Files in CAI \(see page 1109\)](#)

Create Virtual Services from EJB Transactions

Use the following procedure to create a virtual service from a set of EJB transactions in the CAI 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



save button

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) To use the settings in a [VRS file \(see page 1109\)](#), 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 **Create**.

9. Select the project where the virtual service will be created.
10. (Optional) Edit name prefix. The default prefix is the user name.
11. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Create**.
12. To navigate to the artifact that was created, perform the following step that applies:
 - For a single artifact created, click the artifact name link at the top of the shelf.
 - For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.



Note: For more information about consolidated transactions, see [Creating Virtual Services \(see page 1077\)](#).

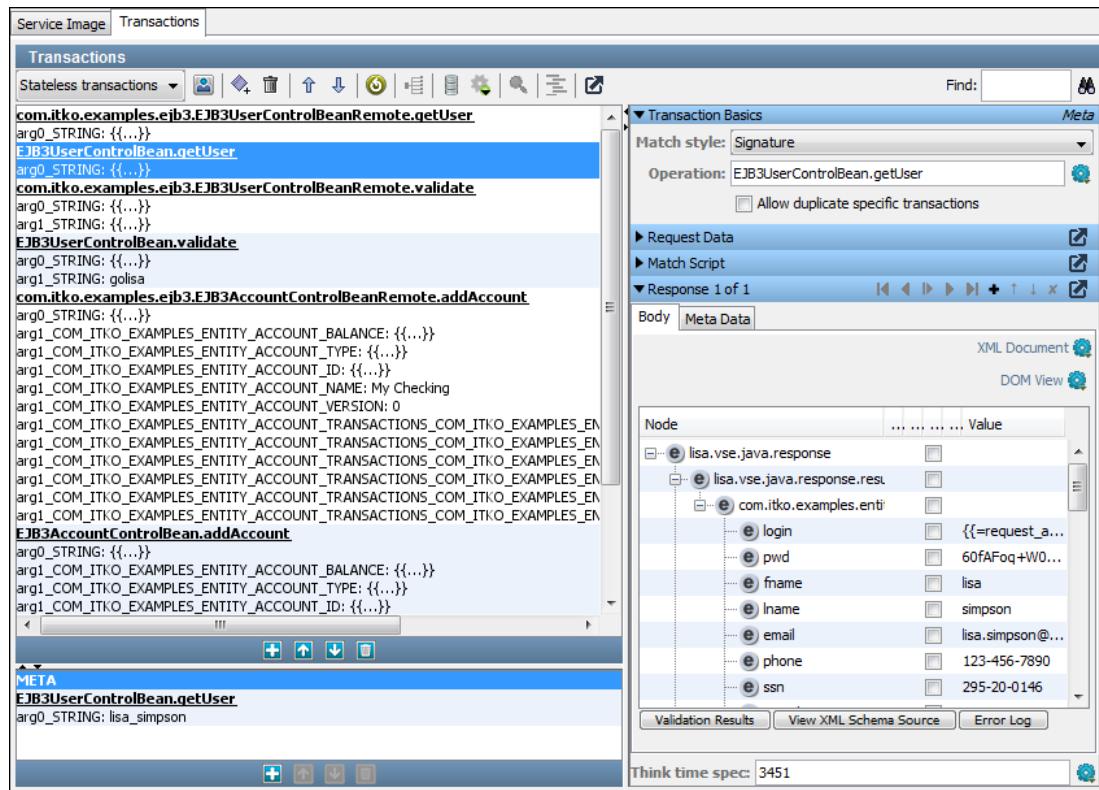
Example: EJB User Control Bean

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](#) (see page 1077) from Java transactions, ensure that the capture level for the Java protocol is set to **Full Data**. See [Configure Capture Levels](#) (see page 1017) 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) To use the settings in a [VRS file \(see page 1109\)](#), 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 **Create**.
9. Select the project where the virtual service will be created.
10. (Optional) Edit name prefix. The default prefix is the user name.
11. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Create**.
12. To navigate to the artifact that was created, perform the following step that applies:
 - For a single artifact created, click the artifact name link at the top of the shelf.
 - For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.

[Managing Java Intercepts](#)

The **Intercepts** tab in the **Agents** window lets you manage Java intercepts.

The following graphic shows the **Intercepts** tab:

Type filter text	Action	Type	Return Type	Method	Signature
EDU			void	<init>	(int)
bsh			java.lang.Object	eval	(bsh.CallStack, bsh.Interpreter)
classpath			bsh.Name	getName	(bsh.NameSpace)
BSHAmbiguousName			java.lang.Class	toClass	(bsh.CallStack, bsh.Interpreter)
BSHArguments			bsh.LHS	toLHS	(bsh.CallStack, bsh.Interpreter)
BSHImportDeclaration			java.lang.Object	toObject	(bsh.CallStack, bsh.Interpreter)
BSHMethodInvocation			java.lang.String	toString	0
BSHPrimaryExpression					
BSHReturnStatement					
BshClassManager					
BshClassManager\$Listener					
BshClassManager\$SignatureKey					
CallStack					
Capabilities					
ClassIdentifier					
ConsoleInterface					
Interpreter					
JTTParserState					
JavaCharStream					
LHS					
Name					
NameSource					
NameSpace					
Node					
Parser					

Screen capture of the Intercepts tab

You can use the search box in the left pane to filter the list of interfaces and classes.



When entering the filter criteria in the search box, you must type a minimum of three letters.

When you select an interface or class in the left pane, the right pane displays the methods.

To determine whether the agent is intercepting a method, place the mouse pointer over the icon in the **Action** column.

To enable or disable the interception of a method, click the icon in the **Action** column.

Create Virtual Services from JCA Transactions

Use the following procedure to create a virtual service from a set of JCA transactions in the CAI 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  save button

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) To use the settings in a [VRS file \(see page 1109\)](#), 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 **Create**.

9. Select the project where the virtual service will be created.

10. (Optional) Edit name prefix. The default prefix is the user name.

11. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Create**.

12. To navigate to the artifact that was created, perform the following step that applies:

- For a single artifact created, click the artifact name link at the top of the shelf.
- For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.



Note: For more information about consolidated transactions, see [Creating Virtual Services \(see page 1077\)](#).

Create Virtual Services from JDBC Transactions

Use the following procedure to create a virtual service from a set of JDBC transactions in the CAI 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) To use the settings in a [VRS file \(see page 1109\)](#), 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 **Create**.

9. Select the project where the virtual service will be created.

10. (Optional) Edit name prefix. The default prefix is the user name.

11. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Create**.

12. To navigate to the artifact that was created, perform the following step that applies:

- For a single artifact created, click the artifact name link at the top of the shelf.
- For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.



Note: For more information about consolidated transactions, see [Creating Virtual Services \(see page 1077\)](#).

Create Virtual Services from JMS Transactions

The result of the first procedure is a raw traffic file that can be imported into the Virtual Service Image Recorder. The benefit of this approach is that it eliminates the need to do proxy recording.

In addition to the request and response bodies, the raw traffic file contains all the connection and queue information in metadata and attributes.

In the second procedure, you use the raw traffic file to create a service image and a virtual service model.



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.



Note: For detailed information about the Virtual Service Image Recorder, data protocols, magic strings, and deploying virtual services, see [Using CA Service Virtualization \(see page 661\)](#).

To create the raw traffic file from JMS transactions:

1. Add a JMS 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



save button

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) To use the settings in a [VRS file \(see page 1109\)](#), click **Choose** and select the file.
6. Click **Create**.
7. Select the project where the raw traffic file will be added.
8. (Optional) Edit name prefix. The default prefix is the user name.
9. Select an option to keep or delete the transactions in the shelf once the raw traffic file is created and click **Create**.
10. To navigate to the artifact that was created, perform the following step that applies:
 - For a single artifact created, click the artifact name link at the top of the shelf.
 - For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.

To create the service image and virtual service model:

1. From the main menu of DevTest Workstation, select **File, New, VS Image, By recording**.
The Virtual Service Image Recorder appears.
2. Do the following steps:
 - a. In the **Write image to** field, enter the fully qualified name of the service image to be created.
 - b. In the **Import traffic** field, browse to and select the raw traffic file in the **Data** folder.
 - c. In the **Transport protocol** field, select **Standard JMS**.
 - d. In the **Model file** field, enter the fully qualified name of the virtual service model to be created.
 - e. Click **Next**. The next step prompts you to select the message recording style.
3. If you want to review the request and response queue information, do the following steps:
 - a. Select the **Review the destinations and transaction tracking mode** check box.
 - b. If the correlation scheme in the **Correlation** drop-down list is incorrect, change the value.
 - c. Click **Next**. The next step contains a **Destination Info** tab and a **Connection Setup** tab.

- d. The values in the **Destination Info** and **Connection Setup** tabs are automatically populated. The **Proxy Destination** field is set to **N/A** because you are not doing proxy recording. You should not need to update either tab, but you can do so if CAI does not set a correct value. Click **Next**. The next step contains response information.
 - e. The values in this step are automatically populated. The **Response Destinations** area contains one or more response queues. You should not need to make changes, but you can do so if CAI does not set a correct value. Click **Next**. The data protocols step appears.
4. Do the following steps:
- a. In the **Request Side Data Protocols** area, click the plus sign.
 - b. Click the left column of the newly added row and select the appropriate data protocol. For XML-based applications, **Generic XML Payload Parser** is a good generic choice.
 - c. If the application responses are not in an XML or text format, a response-side data protocol might be required for VSE to perform magic string substitutions on the response.
 - d. Click **Next**.
5. The next step or steps that appear (if any) depend on the data protocol that you selected. For example, if you selected the **Generic XML Payload Parser** data protocol, the next step prompts you to create XPaths to form VSE requests. See *Using CA Service Virtualization* as needed to complete the step or steps. The last step indicates that the recorder is performing post-processing on what has been recorded.
6. Click **Finish**.
The transactions are saved to a service image, and the virtual service model is created.

To run the virtual service:

1. Shut down the original service.
2. Go to DevTest Workstation and deploy the virtual service model that you created.
3. Run a client application with the virtual service as the service.

Create Virtual Services from REST Transactions

Use the following procedure to create a virtual service from a set of REST transactions in the CAI 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



save button

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) To use the settings in a [VRS file \(see page 1109\)](#), 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 **Create**.

8. Select the project where the virtual service will be created.

9. (Optional) Edit the name prefix. The default prefix is the user name.

10. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Create**.

11. To navigate to the artifact that was created, perform the following step that applies:

- For a single artifact created, click the artifact name link at the top of the shelf.
- For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.



Note: For more information about consolidated transactions, see [Creating Virtual Services \(see page 1077\)](#).

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 that CAI captured.

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 \(see page 661\)](#).

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 CAI 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"**

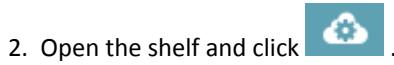
Causes an exception to be raised in the virtualized application.

- **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.



3. To change the default name, select the name and make your edits, then click



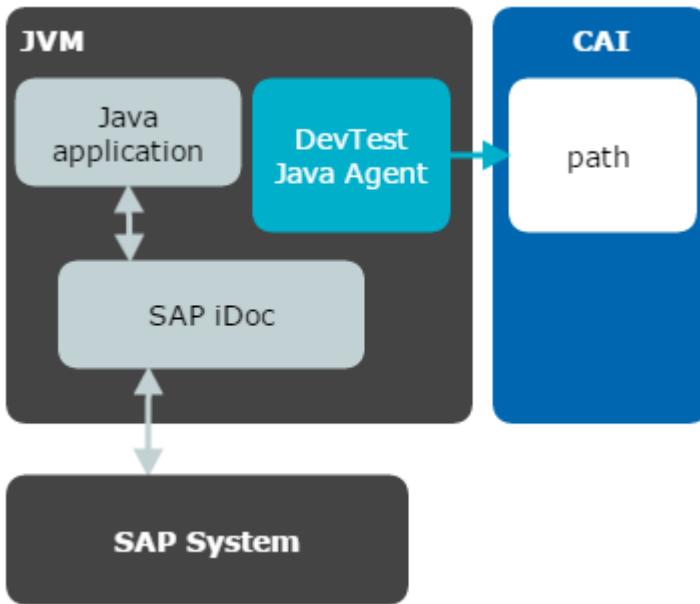
save button

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) To use the settings in a [VRS file \(see page 1109\)](#), click **Choose** and select the file.
7. Click **Create**.
8. Select the project where the virtual service will be created.
9. (Optional) Edit the name prefix. The default prefix is the user name.
10. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Create**.
11. To navigate to the artifact that was created, perform the following step that applies:
 - For a single artifact created, click the artifact name link at the top of the shelf.
 - For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.

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 the 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 \(see page 1093\)](#) and deploy the virtual service.



Note: For information about installing and configuring the agent, see [Agents \(see page 1253\)](#). For information about deploying the virtual service, see [Using CA Service Virtualization \(see page 661\)](#).

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.

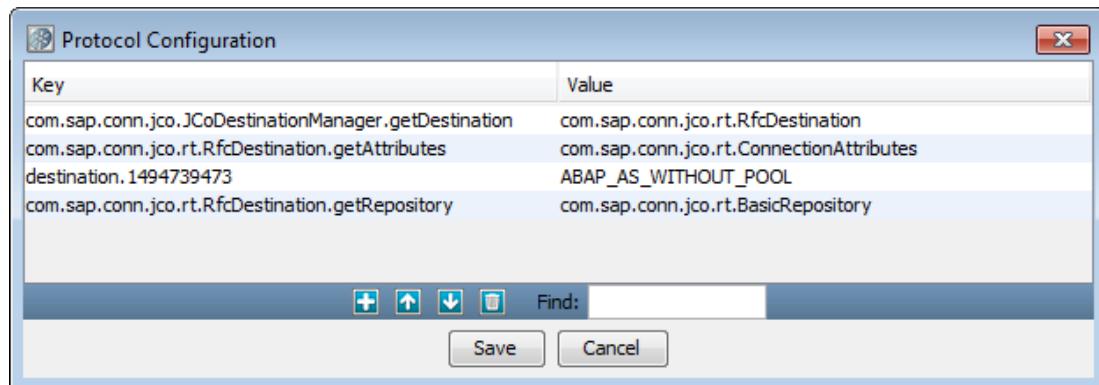
2. Open the shelf and click  .
3. To change the default name, select the name and make your edits, then click  save button to save.
4. If the **Report "No Match"** and **Bypass Virtual Service** options are available, configure the response for unknown requests.
5. If the **Treat all transactions as 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) To use the settings in a [VRS file \(see page 1109\)](#), click **Choose** and select the file.
7. Click **Create**.
8. Select the project where the virtual service will be created.
9. (Optional) Edit name prefix. The default prefix is the user name.
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 **Create**.
12. To navigate to the artifact that was created, perform the following step that applies:
- For a single artifact created, click the artifact name link at the top of the shelf.
 - For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog. A tab for the artifact opens in the DevTest Portal.

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.

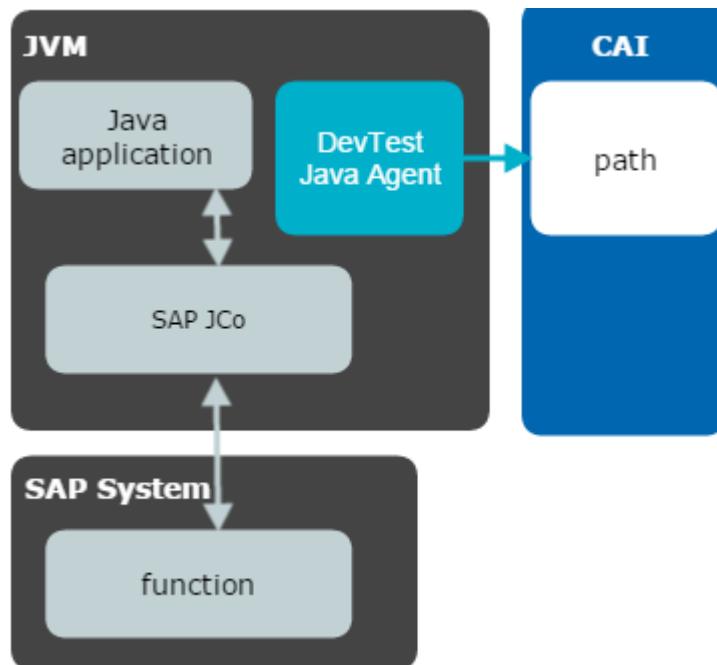
To virtualize multiple destinations, add a destination key for each one. You can use any number, as long as the number is unique in the **Protocol Configuration** dialog.

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

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 the 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 \(see page 1096\)](#) and deploy the virtual service.



Note: For information about installing and configuring the agent, see [Agents \(see page 1253\)](#). For information about deploying the virtual service, see [Using CA Service Virtualization \(see page 661\)](#).

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:

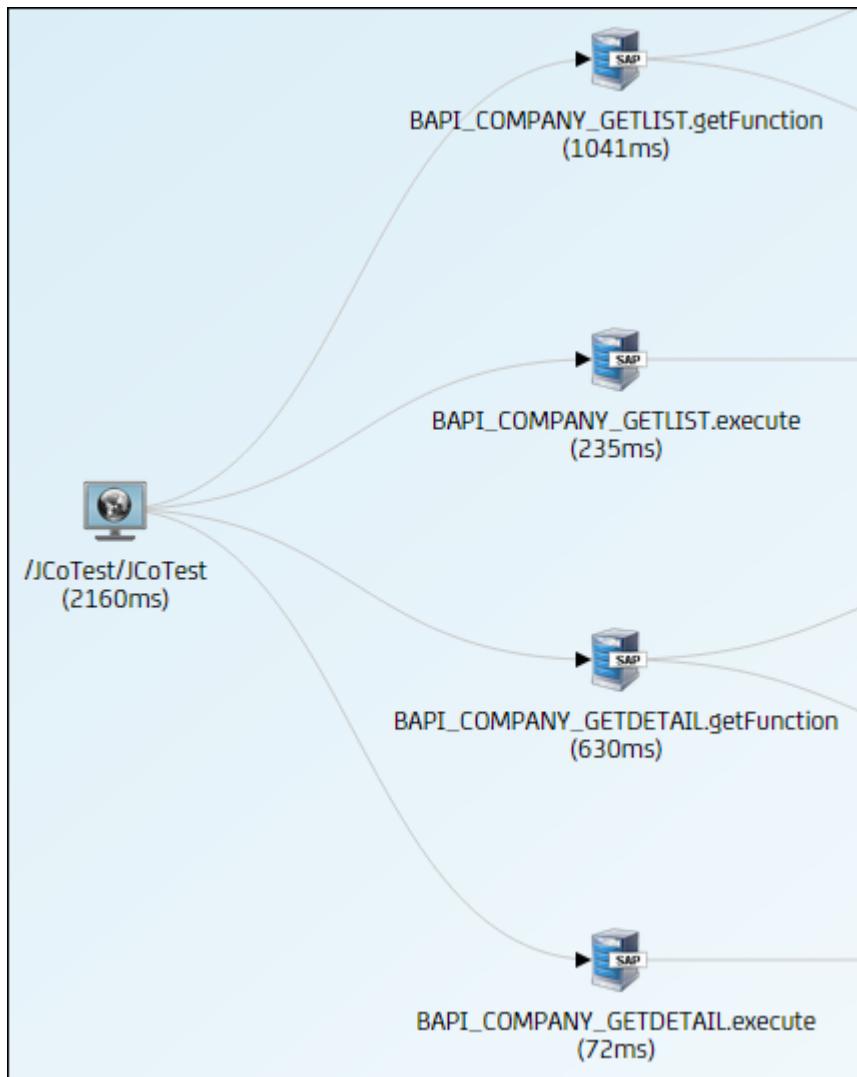
- **Report "No Match"**

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, CAI checks the destination name of the selected SAP component. CAI 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



save button

to save.

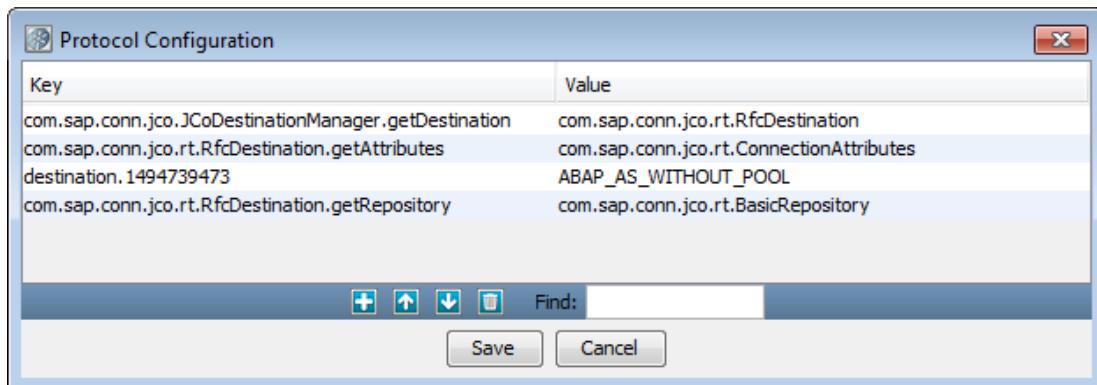
4. If the **Report "No Match"** and **Bypass Virtual Service** options are available, configure the response for unknown requests.
5. If the **Treat all transactions as stateless** check box is editable and you want the virtual service to contain only stateless transactions, ensure that the check box is selected.
6. (Optional) To use the settings in a [VRS file \(see page 1109\)](#), click **Choose** and select the file.
7. Click **Create**.
8. Select the project where the virtual service will be created.
9. (Optional) Edit name prefix. The default prefix is the user name.
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 **Create**.
12. To navigate to the artifact that was created, perform the following step that applies:
 - For a single artifact created, click the artifact name link at the top of the shelf.
 - For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.

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 CA Continuous 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 CAI.
- Virtualize from the DevTest Portal. You generate transactions that appear in the DevTest Portal. At any time afterward, you can use the transactions to create the virtual service. CAI 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 the 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 \(see page 1253\)](#). For detailed information about using the Virtual Service Image Recorder and deploying virtual services, see [Using CA Service Virtualization \(see page 661\)](#).

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 the 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



Screen capture of save button

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) To use the settings in a [VRS file \(see page 1109\)](#), 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 **Create**.
10. Select the project where the virtual service 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 virtual service is created and click **Create**.
13. To navigate to the artifact that was created, perform the following step that applies:
 - For a single artifact created, click the artifact name link at the top of the shelf.
 - For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.



Note: For more information about consolidated transactions, see [Creating Virtual Services \(see page 1077\)](#).

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.

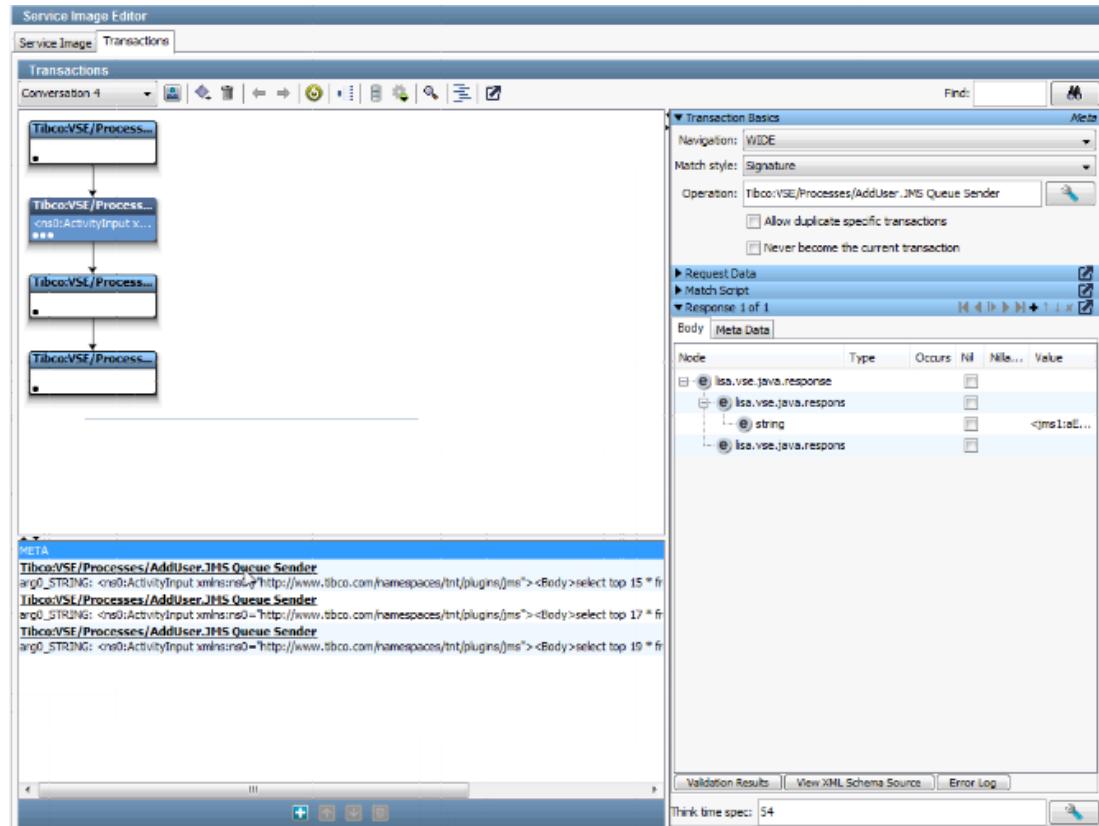


Screen capture of process definition example

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 CAI 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



Screen capture of save button

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) To use the settings in a [VRS file \(see page 1109\)](#), 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 **Create**.

8. Select the project where the virtual service will be created.

9. (Optional) Edit name prefix. The default prefix is the user name.

10. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Create**.

11. To navigate to the artifact that was created, perform the following step that applies:

- For a single artifact created, click the artifact name link at the top of the shelf.
- For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.



Note: For more information about consolidated transactions, see [Creating Virtual Services \(see page 1077\)](#).

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 CAI 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) To use the settings in a [VRS file \(see page 1109\)](#), 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 **Create**.

8. Select the project where the virtual service will be created.

9. (Optional) Edit name prefix. The default prefix is the user name.

10. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Create**.

11. To navigate to the artifact that was created, perform the following step that applies:

- For a single artifact created, click the artifact name link at the top of the shelf.
- For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.



Note: For more information about consolidated transactions, see [Creating Virtual Services \(see page 1077\)](#).

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 CAI 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) To use the settings in a [VRS file \(see page 1109\)](#), 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.

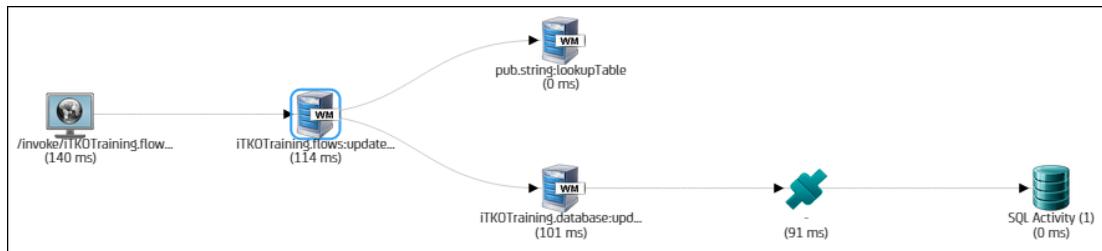
- c. Click **X** and click **Continue** to remove a frame from the listing.
8. Click **Create**.
9. Select the project where the virtual service is created.
10. (Optional) Edit the name prefix. The default prefix is the user name.
11. Select an option to keep or delete the transactions in the shelf once the virtual service is created and click **Create**.
12. To navigate to the artifact that was created, perform the following step that applies:
 - For a single artifact created, click the artifact name link at the top of the shelf.
 - For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.



Note: For more information about consolidated transactions, see [Creating Virtual Services \(see page 1077\)](#).

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

The result of the first procedure is a raw traffic file that can be imported into the Virtual Service Image Recorder. The benefit of this approach is that it eliminates the need to do proxy recording.

In addition to the request and response bodies, the raw traffic file contains all the connection and queue information in metadata and attributes.

If you are working with binary payloads, use an extension data protocol handler to convert the binary requests into an XML form that VSE can handle. You select this data protocol handler in the Virtual Service Image Recorder.

If the response is also binary, you have two options:

- Use an extension data protocol handler to parse the binary responses into a text format for the service image. At runtime, the original binary format is reconstructed for each response.
- Do not use an extension data protocol handler for the response. At runtime, the VSE engine returns the original binary response and is not able to perform magic string substitutions.

In the second procedure, you use the raw traffic file to create a service image and a virtual service model.



Note: For detailed information about the Virtual Service Image Recorder, data protocols, magic strings, and deploying virtual services, see [Using CA Service Virtualization \(see page 661\)](#).

To create the raw traffic file from WebSphere MQ transactions:

1. Add a WebSphere MQ 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



Screen capture of save button

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) To use the settings in a [VRS file \(see page 1109\)](#), click **Choose** and select the file.

6. (Optional) To view and manage the consolidated frames, perform the following steps:

- 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.

- Drag-and-drop the frames up and down the list to change the order.

- Click **X** and click **Continue** to remove a frame from the listing.

7. Click **Create**.

8. Select the project where the raw traffic file will be added.

9. (Optional) Edit the name prefix. The default prefix is the user name.

10. Select an option to keep or delete the transactions in the shelf once the raw traffic file is created and click **Create**.

11. To navigate to the artifact that was created, perform the following step that applies:

- For a single artifact created, click the artifact name link at the top of the shelf.
- For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.



Note: For more information about consolidated transactions, see [Creating Virtual Services \(see page 1077\)](#).

To create the service image and virtual service model:

1. From the main menu of DevTest Workstation, select **File, New, VS Image, By recording**. The Virtual Service Image Recorder appears.

2. Do the following steps:

- In the **Write image** to field, enter the fully qualified name of the service image to be created.
- In the **Import traffic** field, browse to and select the raw traffic file in the **Data** folder.
- In the **Transport protocol** field, select IBM MQ Series.
- In the **Model file** field, enter the fully qualified name of the virtual service model to be created.
- Click **Next**. The next step prompts you to select the message recording style.

3. If you want to review the request and response queue information, do the following steps:
 - a. Select the **Review the queues and transaction tracking mode** check box.
 - b. If the correlation scheme in the **Correlation** drop-down list is incorrect, change the value.
 - c. Click **Next**. The next step contains a **Destination Info** tab and a **Connection Setup** tab.
 - d. The values in the **Destination Info** and **Connection Setup** tabs are automatically populated. The **Proxy Queue** field is set to N/A because you are not doing proxy recording. You should not need to update either tab, but you can do so if CAI does not set a correct value. Click **Next**. The next step contains response information.
 - e. The values in this step are automatically populated. The **Response Destinations** area contains one or more response queues. You should not need to make changes, but you can do so if CAI does not set a correct value. Click **Next**. The data protocols step appears.
4. Do the following steps:
 - a. In the **Request Side Data Protocols** area, click the plus sign.
 - b. Click the left column of the newly added row and select the appropriate data protocol. For XML-based applications, **Generic XML Payload Parser** is a good generic choice.
 - c. If the application responses are not in an XML or text format, a response-side data protocol might be required for the VSE engine to perform magic string substitutions on the response.
 - d. Click **Next**.
5. The next step or steps that appear (if any) depend on the data protocol that you selected. For example, if you selected the **Generic XML Payload Parser** data protocol, the next step prompts you to create XPaths to form VSE requests. See *Using CA Service Virtualization* as needed to complete the step or steps. The last step indicates that the recorder is performing post-processing on what has been recorded.
6. Click **Finish**.
The transactions are saved to a service image, and the virtual service model is created.

To run the virtual service:

1. Shut down the original service.
2. Go to DevTest Workstation and deploy the virtual service model that you created.
3. Run a client application with the virtual service as the service.

VRS Files in CAI

The final step of the [Virtual Service Image Recorder \(see page 775\)](#) 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 the DevTest Portal or the CAI 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

CAI lets you create baseline test cases and suites from transactions in the CAI database.



Note: This feature requires a separate license.

When you generate the transactions, ensure that the [capture levels \(see page 1017\)](#) for the appropriate protocols are set to **Full Data**.

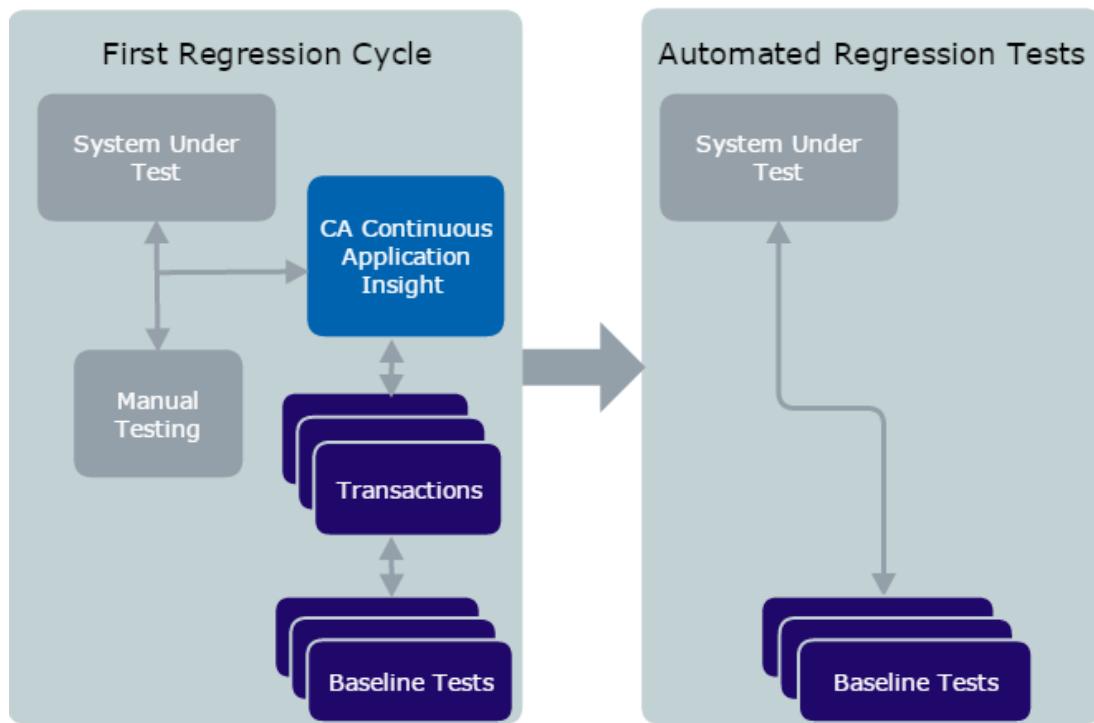
You can create baselines by using the following approaches:

- **Analyze Transactions** window in the DevTest Portal
- **Document Transactions** window in the DevTest Portal
- CAI command-line tool

The procedures in this section describe how to create baselines by using the shelf in the **Analyze Transactions** window.

You cannot create baselines for transaction frames that have a category of **GUI**.

The following graphic provides a high-level view of how most baselines work.



The left portion shows the initial regression cycle. Manual testing is performed on a system. CAI captures the activity and creates transactions that are 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.

Baseline Types

You can create the following types of baselines:

- Stateful
- Data-driven
- Expanded

If you add a [merged \(see page 1038\)](#) transaction frame to the shelf, you cannot create a stateful baseline. You can create data-driven and expanded baselines.

In some cases, CAI automatically selects a baseline type for you. For example, if you [generate all artifacts for an agent \(see page 1036\)](#) in the **Analyze Transactions** window, CAI selects the appropriate baseline type.

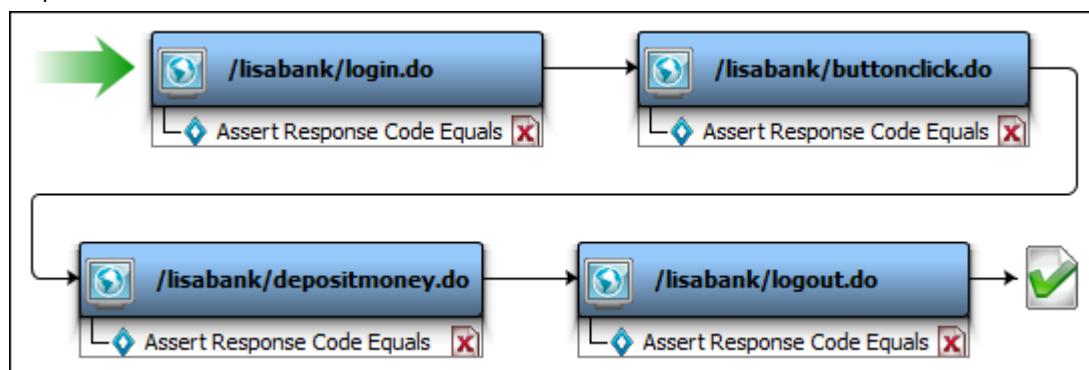
Stateful Baselines

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. The baseline contains four **HTTP/HTML Request** steps.

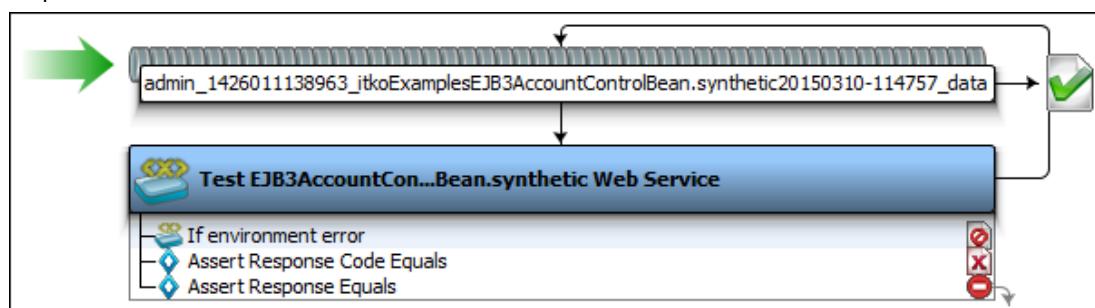


Screen capture of a stateful baseline.

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. The test step is a **Web Service Execution (XML)** step.



Screen capture of a data-driven baseline.

Data-driven baselines are useful when you need to modify a baseline after it is generated.

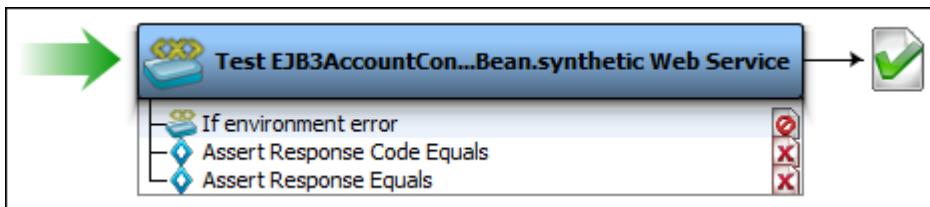
For example, assume that you shelve a merged transaction frame that has a large number of repeated paths. If you create an expanded baseline, you might need to make the same change to all the test cases in the baseline. If you create a data-driven baseline, you can make the change just once.

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. The test step is a **Web Service Execution (XML)** step.



Screen capture of a test case in an expanded baseline.

EJB Baselines

This section contains the following pages:

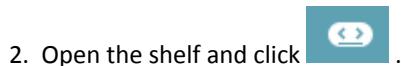
- [Generate an EJB Baseline \(see page 1113\)](#)
- [EJB Baseline Contents \(see page 1115\)](#)
- [Workaround for EJB Baseline Over SSL on WebSphere App Server 8.5 \(see page 1116\)](#)

Generate an EJB Baseline

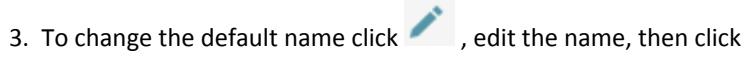
Use the following procedure to generate an EJB baseline from a set of transactions in the CAI database.

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
save button

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 \(see page 1111\)](#).

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) To configure the baseline to validate any database inserts, updates, and deletes, select the [Database Validation \(see page 1144\)](#) check box.

9. (Optional) Click **Choose File** and select a [test template \(see page 1147\)](#) 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 **Create**.
 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 **Create**.
 16. To navigate to the artifact that was created, perform the following step that applies:
 - For a single artifact created, click the artifact name link at the top of the shelf.
 - For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.

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/IOP 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/IOP security, CSIV2 inbound communications.
3. Set the value of the **Transport** field to **SSL-supported**.
4. Go to Security, Global Security, RMI/IOP 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\ProfileName\config\cells\CellName\nodes\NodeName\key.p12
- C:
 \Websphere\AppServer\profiles\ProfileName\config\cells\CellName\nodes\NodeName\trust.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}CD09Hgw=`
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 CA Continuous Application Insight is undefined.

This section contains the following pages:

- [Generate a JMS Baseline \(see page 1119\)](#)

- [JMS Baseline Transaction Frames \(see page 1120\)](#)
- [JMS Baseline Contents \(see page 1121\)](#)

Generate a JMS Baseline

Use the following procedure to generate a JMS baseline from a set of transactions in the CAI database.

Follow these steps:

1. Add a [JMS transaction frame \(see page 1120\)](#) 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 \(see page 1111\)](#).

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) To configure the baseline to validate any database inserts, updates, and deletes, select the [Database Validation \(see page 1144\)](#) check box.

9. (Optional) Click **Choose File** and select a [test template \(see page 1147\)](#) for the baseline.

10. (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.

11. Click **Create**.

12. Select the project where the baseline will be created.

13. (Optional) Edit name prefix. The default prefix is the user name.

14. Select an option to keep or delete the transactions in the shelf once the baseline is created and click **Create**.

15. To navigate to the artifact that was created, perform the following step that applies:

- For a single artifact created, click the artifact name link at the top of the shelf.
- For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.

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 shelf 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.

CAI 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:

- [Generate a REST Baseline \(see page 1122\)](#)
- [REST Baseline Contents \(see page 1125\)](#)
- [HTTP Client Calls \(see page 1125\)](#)

Generate a REST Baseline

Use the following procedure to generate a REST baseline from a set of transactions in the CAI database.

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 \(see page 1111\)](#).

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. CAI encrypts the value and adds the result to the Authorization header field in the baseline.

- **Http Password**

Specifies the web application user password. CAI 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 \(see page 1144\)](#) check box.

10. (Optional) Click **Choose File** and select a [test template \(see page 1147\)](#) 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 **Create**.

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 **Create**.

16. To navigate to the artifact that was created, perform the following step that applies:

- For a single artifact created, click the artifact name link at the top of the shelf.
- For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.

Stateful Baselines and Parameterization

You can add *parameterization* to REST stateful baselines. When CAI detects common items in the request and response, it does one or more of the following actions:

- Adds a property to the request string
- Adds the JSON Path filter to the baseline
- Adds the XML XPath filter to the baseline
- Configures the JSON Ignored Nodes companion to ignore an XPath node.
- Configures the Graphical XML Side-by-Side Comparison assertion to ignore an XPath node

The request body can be in JSON or XML format.

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 body:

```
GET http://myserver:8080/datamanagement/a/templates/4?user=adminRequest: {"name": "demo_02", "applicationId": "3", "new": true}
```

If you set the parameter level to value only, CAI examines the following items: **a**, **templates**, **4**, **admin**, **demo_02**, and **3**.

If you set the parameter level to key and value, CAI 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 body:

```
<addUser xmlns="http://ejb3.examples.itko.com/" userID=123>
<username>webapp-1381707309</username>
<password>example-pwd</password>
</addUser>
```

If you set the parameter level to value only, CAI examines the following items: **http://ejb3.examples.itko.com/**, **123**, **webapp-1381707309**, and **example-pwd**.

If you set the parameter level to key and value, CAI examines the following items: **userID/123**, **username/webapp-1381707309**, and **password/example-pwd**.

The following properties in the [lisa.properties file](#) (see page 1638) let you configure the minimum length and exclusion strings:

- lisa.pathfinder.stateful.baseline.parameter.string.min.length

- lisa.pathfinder.stateful.baseline.parameter.string.exclusion

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.

Consolidated 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 consolidated 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, CAI assigns the REST category 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:

- [Generate a TIBCO BusinessWorks Baseline \(see page 1126\)](#)
- [TIBCO BusinessWorks Baseline Contents \(see page 1127\)](#)

Generate a TIBCO BusinessWorks Baseline

Use the following procedure to generate a TIBCO ActiveMatrix BusinessWorks baseline from a set of transactions in the CAI 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 \(see page 1111\)](#).

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) To configure the baseline to validate any database inserts, updates, and deletes, select the [Database Validation \(see page 1144\)](#) check box.

8. (Optional) Click **Choose File** and select a [test template \(see page 1147\)](#) 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 and click **Create**.

14. To navigate to the artifact that was created, perform the following step that applies:

- For a single artifact created, click the artifact name link at the top of the shelf.
- For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.

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:

- [Generate a Web HTTP Baseline \(see page 1128\)](#)
- [Web HTTP Baseline Contents \(see page 1130\)](#)
- [HTTP Client Call \(see page 1130\)](#)

Generate a Web HTTP Baseline

Use the following procedure to generate a Web HTTP baseline from a set of transactions in the CAI database.

Signed Web HTTP transactions are supported. However, CAI 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 \(see page 1702\)](#).

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 \(see page 1111\)](#).
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. CAI encrypts the value and adds the result to the Authorization header field in the baseline.

- **Http Password**

Specifies the web application user password. CAI 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 \(see page 1144\)](#) check box.

10. (Optional) Click **Choose File** and select a [test template \(see page 1147\)](#) 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 **Create**.

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 **Create**.
17. To navigate to the artifact that was created, perform the following step that applies:
 - For a single artifact created, click the artifact name link at the top of the shelf.
 - For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.

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.

Consolidated 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 consolidated 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, CAI assigns the REST category 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:

- [Generate a Web Service Baseline \(see page 1131\)](#)

- [Web Services with Attachments \(see page 1134\)](#)
- [Web Service Baseline Contents \(see page 1134\)](#)

Generate a Web Service Baseline

Use the following procedure to generate a web service baseline from a set of transactions in the CAI database.

Web services with [attachments \(see page 1134\)](#) are supported.

Signed SOAP messages are supported on all platforms except IBM WebSphere Application Server 8.5. However, CAI 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 \(see page 1711\)](#).

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 \(see page 1111\)](#).

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. CAI encrypts the value and adds the result to the Authorization header field in the baseline.

- **Http Password**

Specifies the web application user password. CAI 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 \(see page 1144\)](#) check box.

10. (Optional) Click **Choose File** and select a [test template \(see page 1147\)](#) 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 **Create**.
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 **Create**.
16. To navigate to the artifact that was created, perform the following step that applies:
 - For a single artifact created, click the artifact name link at the top of the shelf.
 - For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.

Stateful Baselines and Parameterization

You can add *parameterization* to web service stateful baselines. When CAI detects common items in the request and response, it does one or more of the following actions:

- Adds a property to the request string
- Adds the XML XPath filter to the baseline
- Configures the Graphical XML Side-by-Side Comparison assertion to ignore an XPath node

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 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, CAI examines the following items: **http://ejb3.examples.itko.com/**, **123**, **webapp-1381707309**, and **example-pwd**.

If you set the parameter level to key and value, CAI examines the following items: **userID/123**, **username/webapp-1381707309**, and **password/example-pwd**.

The following properties in the [lisa.properties file \(see page 1638\)](#) let you configure the minimum length and exclusion strings:

- lisa.pathfinder.stateful.baseline.parameter.string.min.length
- lisa.pathfinder.stateful.baseline.parameter.string.exclusion

Web Services with Attachments

You can generate a baseline for web services that have the following attachment types:

- MIME
- DIME
- XOP
- MTOM

CAI 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 \(see page 1013\)](#). 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:

- [Generate a webMethods Baseline \(see page 1135\)](#)
- [webMethods Baseline Contents \(see page 1137\)](#)

Generate a webMethods Baseline

Use the following procedure to generate a webMethods Integration Server baseline from a set of transactions in the CAI 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 \(see page 1380\)](#).

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 \(see page 1111\)](#).

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) To configure the baseline to validate any database inserts, updates, and deletes, select the [Database Validation \(see page 1144\)](#) check box.

9. (Optional) Click **Choose File** and select a [test template \(see page 1147\)](#) for the baseline.

10. (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.

11. Click **Create**.

12. Select the project where the baseline will be created.

13. (Optional) Edit name prefix. The default prefix is the user name.

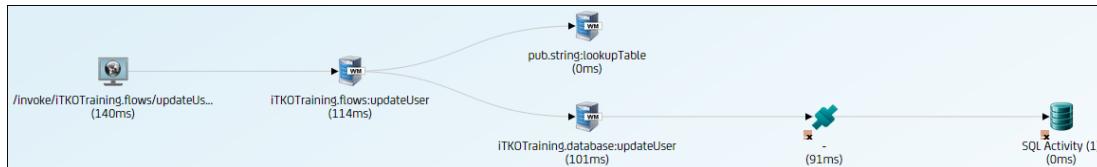
14. Select an option to keep or delete the transactions in the shelf once the baseline is created and click **Create**.

15. To navigate to the artifact that was created, perform the following step that applies:

- For a single artifact created, click the artifact name link at the top of the shelf.
- For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.

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:

- [Generate a WebSphere MQ Baseline \(see page 1138\)](#)
- [WebSphere MQ Baseline Scenarios \(see page 1139\)](#)

- [WebSphere MQ Baseline Properties \(see page 1140\)](#)
- [WebSphere MQ Baseline Contents \(see page 1141\)](#)

Generate a WebSphere MQ Baseline

Use the following procedure to generate a WebSphere MQ baseline from a set of transactions in the CAI 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 \(see page 1380\)](#).

Follow these steps:

1. Determine which WebSphere MQ scenario [\(see page 1139\)](#) you want to use, and enable the DevTest Java Agent as appropriate.
2. (Optional) Configure the WebSphere MQ properties [\(see page 1140\)](#) 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 shelf 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 \(see page 1111\)](#).
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) To configure the baseline to validate any database inserts, updates, and deletes, select the [Database Validation \(see page 1144\)](#) check box.
11. (Optional) Click **Choose File** and select a [test template \(see page 1147\)](#) for the baseline.
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 **Create**.
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 **Create**.
17. To navigate to the artifact that was created, perform the following step that applies:
 - For a single artifact created, click the artifact name link at the top of the shelf.
 - For multiple artifacts created, click the **x pending (x success)** link and click the name of the artifact from the list in the **Alerts** dialog.
A tab for the artifact opens in the DevTest Portal.

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 \(see page 1253\)](#).

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.

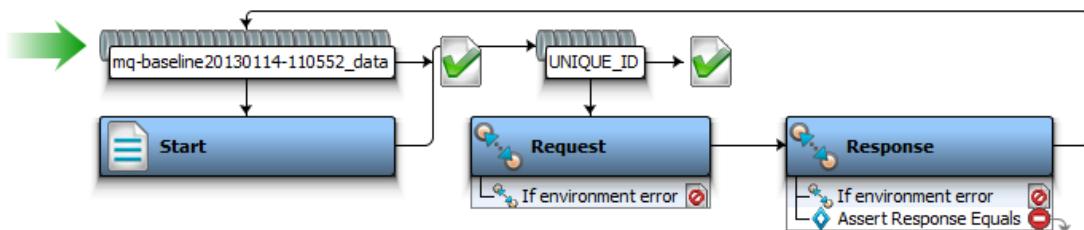
In the WebSphere MQ steps, the client mode is set to native client.

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 \(see page 1143\)](#)
- [Generic Baseline Contents \(see page 1144\)](#)

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 \(see page 1142\)](#).



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 \(see page 1111\)](#).

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 \(see page 1147\)](#) 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 baselines from the DevTest Portal, 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 not compatible with the database validation process.

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

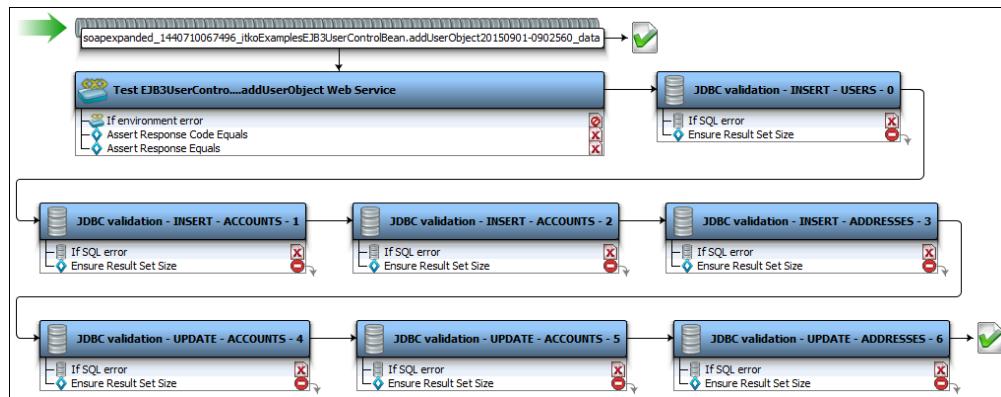
The following SQL formats are supported:

- 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 \(JDBC\) \(see page 1768\)](#) steps.

The following graphic shows a baseline that has database validation steps.



Screen capture of 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 - 0
- JDBC validation - UPDATE - ACCOUNTS - 4

The **Documentation** area for each step displays the complete SQL statement.

Each step contains a SELECT statement that is used to perform the validation. For example, the following statement validates an update of the ACCOUNTS table:

```
SELECT * FROM ACCOUNTS WHERE (USER_ID = '{{cai_login_1}}') AND (account_id = '{{cai_resp.Filter_id_6}}')
```

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: CAI might not have enough information to create database validation steps with an appropriate degree of certainty. The steps are generated, but the names begin with the word **Suggested**. If you want to, you can modify or remove these steps.

Test Templates

While creating a baseline from the DevTest Portal or the CAI command-line tool, you can specify an existing test case whose steps will be included in the baseline. The existing test case is referred to as a *test template*.

The following windows in the DevTest Portal let you select the test template:

- Analyze Transactions
- Document Transactions

The following graphic shows the template chooser in the Document Transactions window.

Step Type	Step Name
REST	http POST /devtest/api/login
Do-Nothing Step	CAI Template Place Holder
Output Log Message	Output Log Message

Screen capture of template chooser.

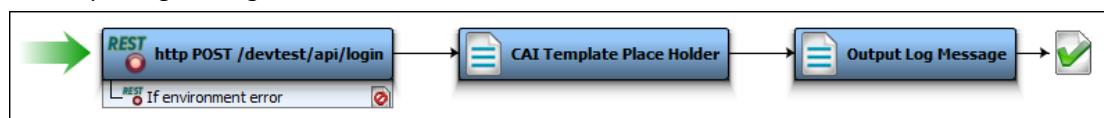
To specify the template from the command-line tool, use the **--cai-template** option.

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.

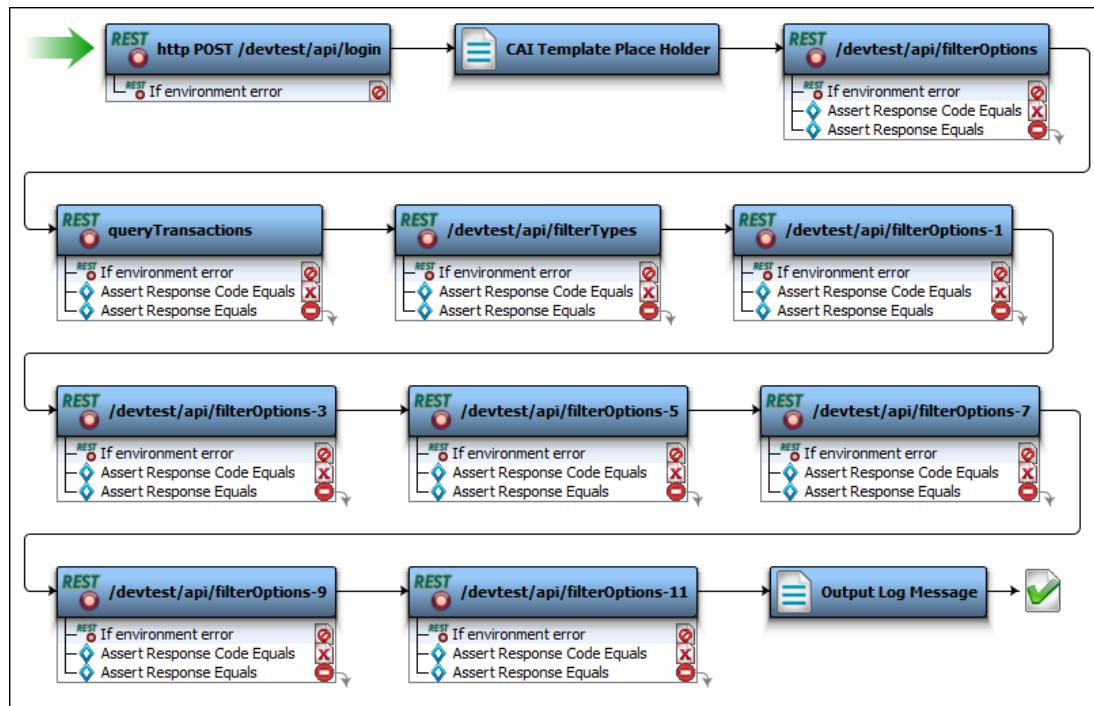
The following graphic shows a template that contains the following steps:

- REST
- Do-Nothing
- Output Log Message



Screen capture of template.

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.

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.

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 \(see page 533\)](#).

You can monitor the results in the [Consolidated Baseline tab \(see page 1150\)](#).

For information about how to run an expanded baseline, see [Run a Test Suite \(see page 557\)](#).

You can monitor the results in the [Baseline Results panel \(see page 1151\)](#).

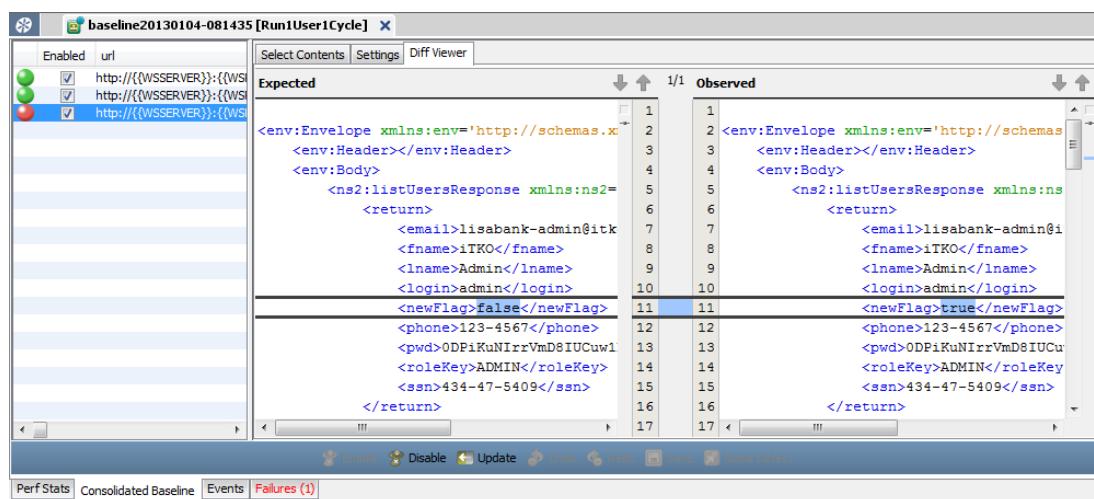
Consolidated Baseline Tab

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 \(see page 1482\)](#).

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**

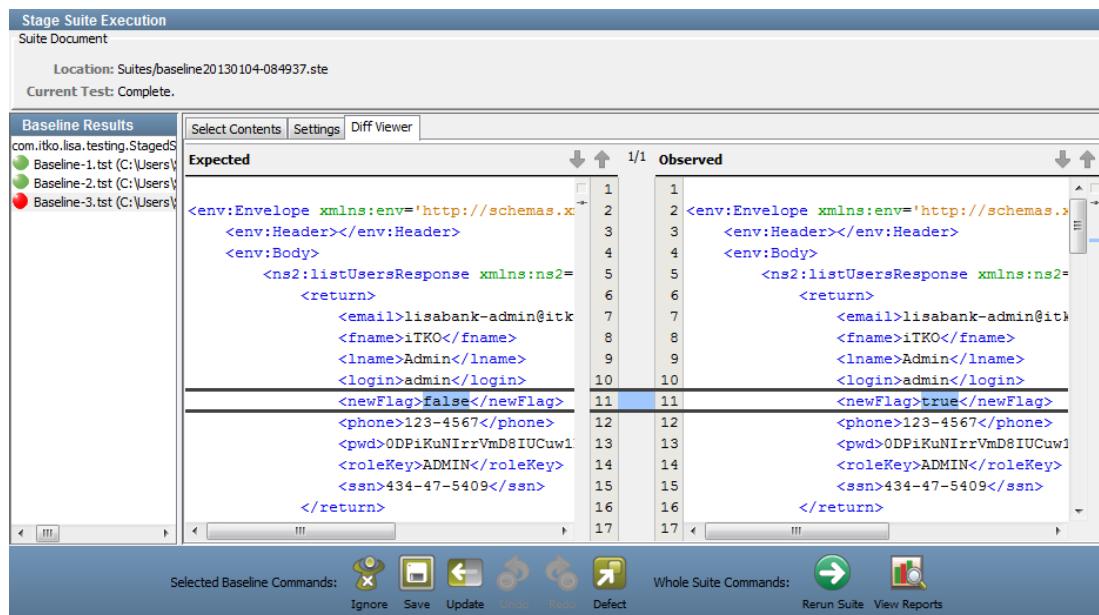
Saves any changes that were made.

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 Reporting Console.

Dynamic Responses and Transaction Frame Steps

Assume that a baseline has an [Execute Transaction Frame \(see page 1866\)](#) step.

If the response changes on every request, running the Execute Transaction Frame step in the ITR 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.

Creating 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 the 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



Save button

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 **Create**.

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 **Create**.

The LDS file is created and available in the DevTest Workstation from the data folder of the specified project.

Creating Request and Response Pairs

You can create a data set that contains request and response (R/R) pairs for shelved transaction frames. The R/R pairs can be uploaded from CAI and imported into a test from DevTest Portal or from the DevTest Workstation.



Note: To create data sets, the Extract Data permission must be set for the Runtime User in the Pathfinder Administration Permission. See [Standard Permissions \(see page 1422\)](#) for more information about the Pathfinder Administration permission.

See [Create a Test by Importing R/R Pairs from a File \(see page 327\)](#) for more information about importing R/R pairs into the **Test Editor**.

You can create R/R pairs for the following transaction types:

- JMS
- MQ Series
- SOAP
- REST

The following conditions apply to shelved transaction frames when creating R/R pairs:

- Meta data files for JMS and MQ Series transaction frames are created.
- Single frames generate one R/R pair per frame.
- Merged frames generate R/R pairs for each frame in the merged path.
- All occurred frames generate R/R pairs for each frame matching the frame name.
- Each transaction creates one R/R pair zip file.

The RR pair zip file uses a naming convention of (prefix)_(transactionframename)_rrpairs_(UTCTimeStamp).zip. For an example,
admin_1426740114537_itkoExamplesEJB3AccountControlBean_getTransactions_rrpairs_1426751389.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)-nrsp.txt

The files in the zip file for JMS and MQ Series use the following naming convention:

- (prefix)_(transactionframename)_rrpairs_(frametype)-n-req.txt
- (prefix)_(transactionframename)_rrpairs_(frametype)-n-req-meta.properties
- (prefix)_(transactionframename)_rrpairs_(frametype)-nrsp.txt
- (prefix)_(transactionframename)_rrpairs_(frametype)-nrsp-meta.properties

n presents 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 an R/R pair created.
5. To download the R/R pairs from individual transactions frames, click  . A zip file is created.
6. To create R/R pairs for all the transaction frames in the shelf, click **Create**.
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 **Create**.
The R/R pairs are created and available in the 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 R/R pairs manually.

Documenting Transactions for Testing

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.

You document a test by recording test transactions and analyzing the generated backend calls. 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 \(see page 1157\)](#)
- [Search the CAI database for saved transaction recordings \(see page 1158\)](#)
- [View and analyze recordings \(see page 1158\) in list or graphical view](#)
- [Create documentation from recorded transactions \(see page 1161\)](#)
- [Compare recordings \(see page 1159\)](#)
- [Show recordings in difference view \(see page 1160\)](#)
- [Export recorded transactions \(see page 1162\)](#)
- [Import recorded transactions \(see page 1162\)](#)

- [Create baseline for recorded transactions \(see page 1163\)](#)

**More Information:**

- [How to Document a Transaction for a Manual Test \(see page 1157\)](#)

How to Document a Transaction for a Manual Test

This page takes you through the steps of documenting a transaction for a manual test.

1. [Record Your Test Transactions \(see page 1157\)](#)
2. [Search for Recorded Test Transactions \(see page 1158\)](#)
3. [View and Analyze Recorded Test Transactions \(see page 1158\)](#)
4. [Compare Recordings \(see page 1159\)](#)
5. [Show Recording Difference View \(see page 1160\)](#)
6. [Create Documentation from Recorded Transactions \(see page 1161\)](#)
7. [Export Recorded Transactions \(see page 1162\)](#)
8. [Import Recorded Transactions \(see page 1162\)](#)
9. [Create a Functional Test for Recorded Transactions \(see page 1163\)](#)

Record Your Test Transactions

You can document the exceptions that occur in business transactions for a specific IP address using the recording feature.

Follow these steps:

1. Select **Application Insight, Document Transactions** from the left navigation menu.
The **Document Transactions** window opens.
2. Click **Create a Recording** to start a new case recording.
The **Record Case** dialog opens. 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 you are running the browser that connects to the agent application from a remote system.
3. (Optional) Select **Auto Refresh** to display the captured transactions automatically.
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. Click **Start**.

The **Start** button changes to an animated **Recording...** icon to indicate that the recording is in progress.

6. Run the agent application in another browser window, either on the same system or on a remote system.

7. Perform the required work on the agent system that is being recorded.

The frames that are recorded 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**.



8. (Optional) To view the recorded transactions in a graphical view, click .



9. (Optional) To view the recorded transactions in a list view, click .

10. (Optional) Click **Show Outline** and move the map to view portions of large transactions at a time.

11. Click **Stop**.

12. To save the recording, click **Save**.

The recordings that are saved are listed in the **Recording** pane of the **Document Transactions** window.

13. To create documentation for your recorded transactions, click **Create Document**.

Search for Recorded Test Transactions

You can 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 or delete a recording.

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 displays.

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 **Recording** pane.

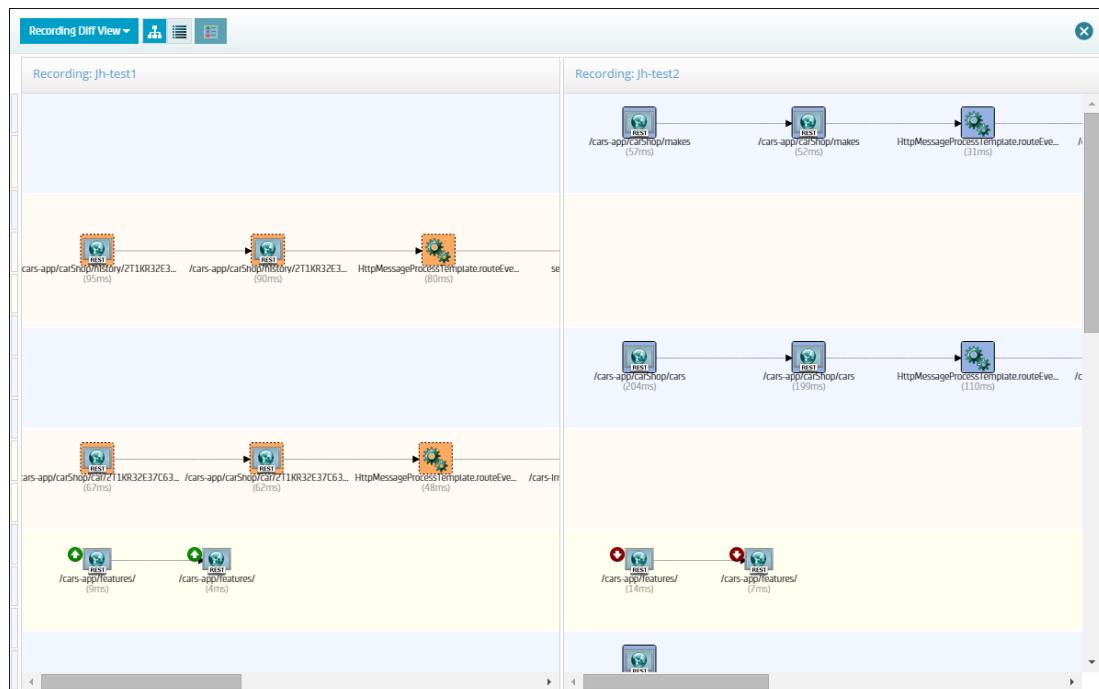
7. To delete a recording, click **Delete Recording** in the **Recording** pane.

Compare Recordings

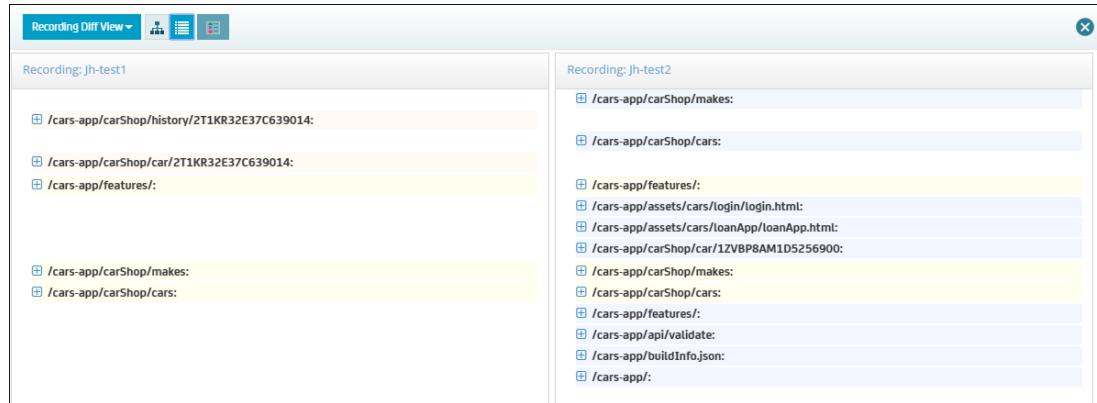
You can compare two recordings side-by-side in the **Document Transactions** window. In the following graphic, recording **Jh-test1** displays in the left pane and recording **Jh-test2** displays in the right pane. The comparison of the two recordings displays in **Show graphical differences** view by default. You can 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, you can 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 with the specified response time difference display.

- 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 do not display any transactions.

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

You can 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. You can save and print the report.

Follow these steps:

1. From the **Document Transaction** 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 task bar, click **Save**, and enter the fields.
5. To print, click **Print** and enter the fields.

Export Recorded Transactions

From the Document Transactions window, you can 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 CAI installations or can be imported from Splunk. 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 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** from the **Actions** column.
2. (Optional) To select multiple recordings, select the checkboxes next to the list of recordings or select the **Recording** checkbox to select all recordings.
The **Export Recording** dialog displays.
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 can import a zip file that contains transaction paths from various sources. The imported transactions display in the **Saved Recordings** pane with a POI. The imported, recorded transactions can be searched by name of the recording name.



Note: Duplicate recordings cannot be imported.

Follow these steps:

1. From the **Document Transactions** window, click **Import Recording**.
The **Import Recording** dialog displays.
2. Click **Choose File** 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 displays.
5. Click **OK**.
The imported recording displays in the **Saved Recordings** pane.

Create a Functional Test for Recorded Transactions

Testing a business functionality may 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

You have an option to add assertions when at least one of the supported protocols is a root frame in the recorded transactions.

Follow these steps:

1. From the **Document Transactions** window, select a recording, and click **Create Functional Test** from the graphical or list view.
2. Select a project, enter a prefix name (optional, default is admin).
3. 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. CAI encrypts the value and adds the result to the Authorization header field in the baseline.

- **Http Password**

Specifies the web application user password. CAI encrypts the value and adds the result to the Authorization header field in the baseline.

4. (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.

5. (Optional) To configure the baseline to validate the database, select [Database Validation \(see page 1144\)](#).

6. If the assertion fields are available, configure them as needed:

- **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 Content**.

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.

7. (Optional) Click **Choose File** and select a [test template \(see page 1147\)](#) for the baseline.
8. Click **Create**.
9. To navigate to the artifact that was created, click the artifact name link at the top of the Document Transaction window.

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:

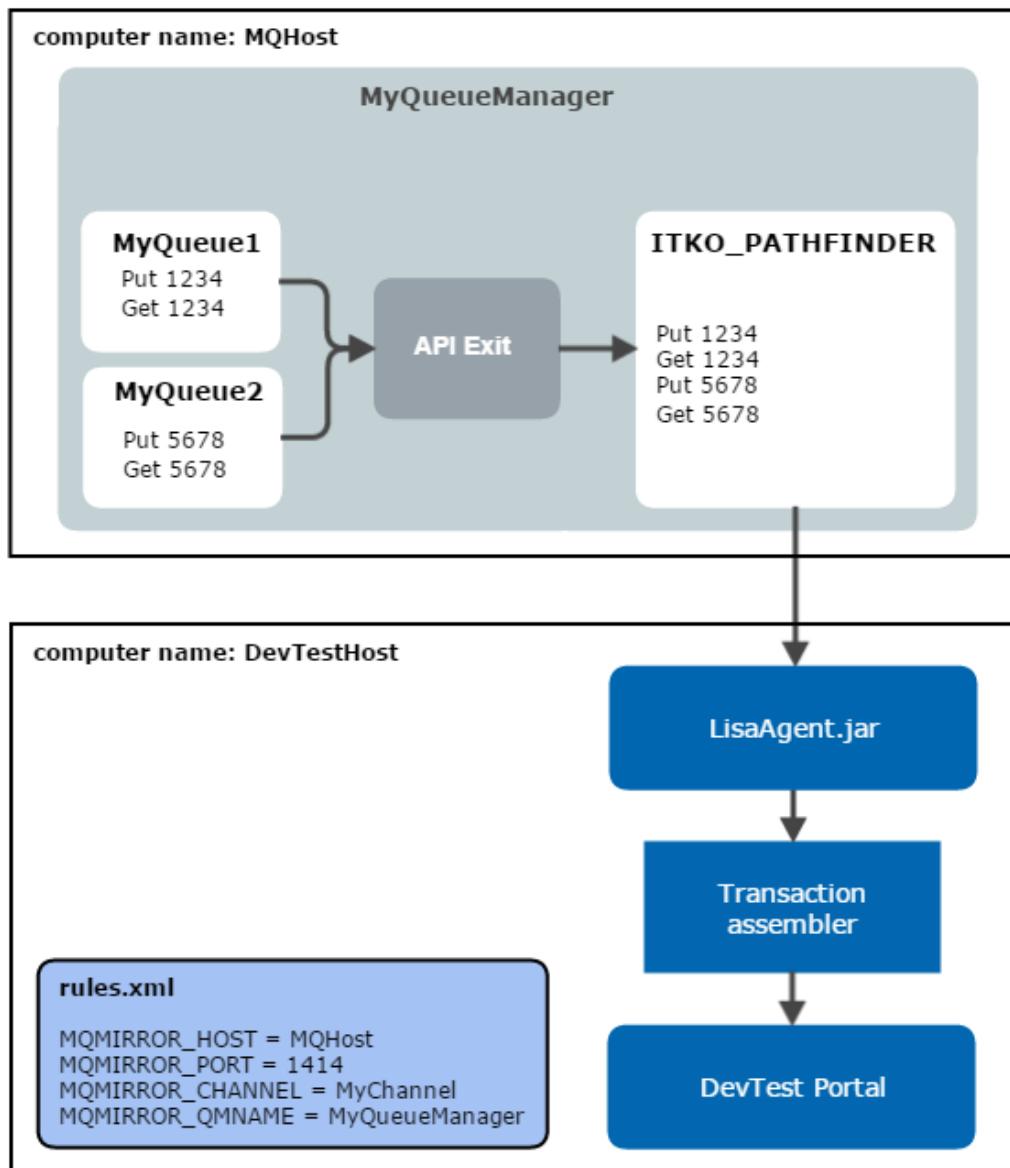
1. Go to the **Analyze Transactions** window.
2. Open the transaction detail dialog and select the **XML** tab.
3. Locate the **rip** attribute. The value specifies the remote IP address.
4. Use that IP address in the **Document Transactions** window.

Native MQ CAI

Native MQ CAI is a WebSphere MQ API exit that monitors the get and put calls for the queues of a queue manager. Native MQ CAI 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 the 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 CAI 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 the DevTest Portal.

Native MQ CAI 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.



More information:

- [Create the ITKO_PATHFINDER Queue \(see page 1167\)](#)
- [Install the Native MQ CAI API Exit \(see page 1168\)](#)
- [Configure the Native MQ CAI API Exit \(see page 1169\)](#)
- [Configure the Agent Properties for Native MQ CAI \(see page 1170\)](#)
- [Generate Transactions from the ITKO_PATHFINDER Queue \(see page 1172\)](#)
- [Native MQ CAI Troubleshooting \(see page 1173\)](#)

WebSphere MQ User Privileges for Native MQ CAI

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 CAI 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 Native MQ CAI API Exit

The LISA_HOME\agent\native_mq directory contains the API exit files for various operating systems.

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+r).
 - **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+r).
 - **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+r).
 - **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+r).
 - **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+r).
 - **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+r).
 - **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+r).
4. Restart WebSphere MQ.

Configure the Native MQ CAI API Exit

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 CAI 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 CAI

The **rules.xml** file for the DevTest Java Agent includes a set of Native MQ CAI 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_PASSTWD** 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 CAI 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** (see page 1170) 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.

Native MQ CAI Troubleshooting

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 \(see page 1168\)](#) 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)
```

CAI Command-Line Tool

The **PFCmdLineTool** command lets you perform various CA Continuous 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 \(see page 1175\)](#)

- [Querying and Display \(see page 1176\)](#)
- [Export and Import \(see page 1177\)](#)
- [Baselines \(see page 1177\)](#)
- [Virtualization Artifacts \(see page 1179\)](#)
- [Agent Condition \(see page 1179\)](#)
- [Capture Levels \(see page 1180\)](#)
- [Miscellaneous \(see page 1180\)](#)
- [Example: Display Root Transaction Frames \(see page 1181\)](#)
- [Example: Display Transaction Frame Hierarchy \(see page 1181\)](#)
- [Example: Generate a Baseline Suite \(see page 1181\)](#)
- [Example: Generate a Stateful Baseline \(see page 1181\)](#)
- [Example: Generate a Baseline with a Template \(see page 1181\)](#)
- [Example: Generate Virtualization Artifacts \(see page 1182\)](#)
- [Example: Check the Agent Condition \(see page 1182\)](#)
- [Example: Configure the Capture Level \(see page 1182\)](#)
- [Example: Search Transaction Frame \(see page 1182\)](#)
- [Example: Generate Large Data Set File \(see page 1182\)](#)

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 CAI records.
- **--remoteIP=IP address**
Specifies the server-side IP address that CAI 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.

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 into the database. Any search criteria that you specify is ignored.
- **--no-persist**
Specifies that the import operation does not persist the data, causing the import file to be validated only.



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 [CAI test template \(see page 1147\)](#) to use when generating baseline tests.

▪ --http-user=http-user

Specifies the web application user name. CAI 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. CAI 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 \(see page 1122\)](#) and [web service \(see page 1131\)](#) 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.

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 \(see page 1109\)](#) 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.

- **--help -h**
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 CAI 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 CAI 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
```

VS Traffic to CA Application Insight Companion

The VS Traffic to CA Application Insight Companion pushes data to the CAI 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 CAI 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 VSE server.
6. Invoke the virtual service.
The transaction frame is created.



Note: The transaction frames that are created by this companion display in the 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 \(see page 1030\)](#) for more information about component names.

7. Go to the DevTest Portal and create the baseline test. See [Creating Baselines \(see page 1110\)](#) 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.

CA Continuous Application Insight Agent Light

The CAI Agent Light is a lightweight agent that sits on any external system and pushes data into CAI. The data is integrated directly into an application and captures transactions using log file format without needing to inject an agent. The light agent is a Java executable JAR file that is named **LisaAgentLight.jar**. The JAR file is executed from the command line. See [Command-Line \(see page 1186\)](#) for more information about using the command line with the agent lights.

The data that is captured by the log file is added to the CAI database and displays in the DevTest Console.

CAI has the following agent lights:

- [Agent Light for webMethods HTTP \(see page 1187\)](#)
- [Agent Light for JMS \(see page 1189\)](#)
- [Agent Light for WebSphere MQ \(see page 1193\)](#)

This section contains the following pages:

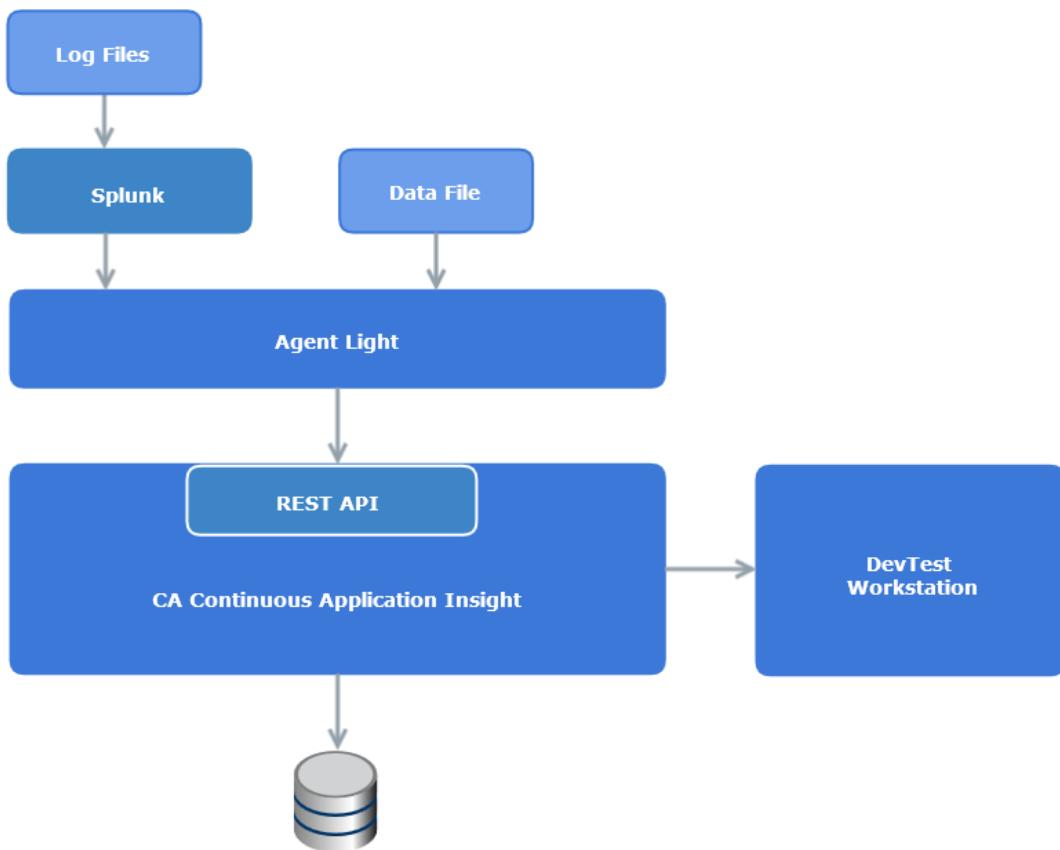
- [Agent Light Architecture \(see page 1184\)](#)
- [Agent Light Prerequisites \(see page 1185\)](#)
- [Install the Agent Light \(see page 1185\)](#)
- [REST API \(see page 1185\)](#)
- [Command-Line \(see page 1186\)](#)
- [Agent Light for webMethods HTTP \(see page 1187\)](#)
- [Agent Light for JMS \(see page 1189\)](#)
- [Agent Light for WebSphere MQ \(see page 1193\)](#)

Agent Light Architecture

The CAI Agent Light architecture has the following components:

- Data from a third-party log file or a data transaction
- Agent Light
- CAI REST API
- DevTest Workstation
- Database

The following graphic shows the components and how they interact:



The agent light receives the data from data transaction files or from third-party log files using a Splunk transaction query.

The agent light sends the data to CAI to create the path through a REST API call.

Agent Light Prerequisites

The following list is requirements to use the agent light:

- Java 1.7 or later
- **LisaAgentLight.jar**

Install the Agent Light

The **LisaAgentLight.jar** file is installed in the same location as the other CAI agent modules in the **LISA_HOME\agent** directory. The **LisaAgentLight.jar** file can be copied anywhere on any system.

REST API

The CAI REST API takes the transaction data and passes it to the CAI to create the transaction path.

- **URL**

Defines the URL, `http://LisaRegistryServer:1505/api/Pathfinder/convertTransaction?Protocol={protocolvalue}`.

- **Method**

Defines the POST method.

- **Parameters**

Defines the protocol. HTTP, HTTPS, JMS, and MQ are supported.

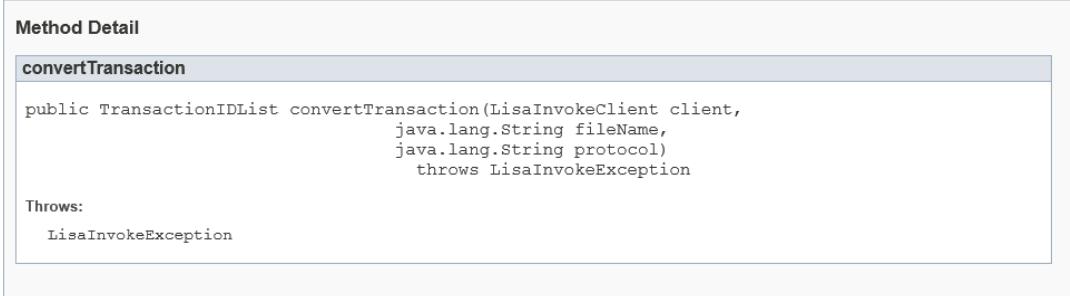
- **Request Body**

(Required) Defines the entire content of the transaction data in Multipart format.

- **Return**

Defines the transaction IDs.

The CAI REST API uses the Java SDK to wrap the REST API, **lisa-invoke2-client.jar**. In the Java SDK, the class **com.itko.lisa.invoke.client.PathfinderServer** wraps the CAI REST APIs. The following graphic displays the method within this class that wraps the REST call:



Method Detail

convertTransaction

```
public TransactionIDList convertTransaction(LisaInvokeClient client,
                                             java.lang.String fileName,
                                             java.lang.String protocol)
                                             throws LisaInvokeException
```

Throws:

`LisaInvokeException`

Screen capture of a REST API method detail.

Command-Line

You use the command line to import the transaction data into CA Continuous Application Insight for the agent lights. Information is parsed and inserted into the CAI database. The command-line provides a method of inserting nodes into CAI without needing to customize or implement a CAI agent. This method includes the ability to specify a source for data (file base and other data) that can read and parse the data. The data is then formatted and inserted into the CAI database.

From the command line, run the **LisaAgentLight.jar** file and use the **-f** and **-t** parameters to specify the location and type of data file. The paths are created and displayed in the CAI Console.

- **-f, -file**

Defines the transaction data file to be imported into CAI.

- **-t, -type**

Defines the data protocol type. HTTP, JMS, and MQ are supported.

Default: HTTP

Agent Light for webMethods HTTP

The Agent Light for webMethods HTTP is an agent that communicates with CA Continuous Application Insight using a REST API. The purpose of the light agent is to take the input from a webMethods HTTP request/response data file and send the data to CAI through the REST API call.

The agent light can receive the webMethods HTTP transactions from the following sources:

- **HTTP transaction data file**

The light agent uses the **-f** parameter to take a data file as an input. The data in this file must be in the standard HTTP request/response format. See [Input Transaction Data Format for HTTP \(see page 1188\)](#) for more information about the standard data format.

The transaction parameters from the data file are:

- **-f, -file**

Defines the transaction data file to be imported into CAI.

- **-t, -type**

Indicates the data protocol type.

Default: http

Create a path from data file using the following example code:

```
Java - jar lisaagentlight.jar - url http://<LisaServer>:1505 - f webmethods.log - t http
```

- **webMethods log file using a Splunk query**

When the webMethods integration server is set to **trace**, the web service call details for the HTTP request/response are logged to the log file. Splunk indexes the log file and produces the HTTP transaction data. The agent light queries Splunk to retrieve the webMethods integration server HTTP transactions and creates the CAI paths for them.

The transaction parameters from Splunk are:

- ▪ **-splunk**

Indicates transactions acquired from Splunk.

- **-h, -hostname**

Defines the Splunk server hostname or IP address.

- **-port**

Defines the Splunk port number.

- **-u, -username**

Defines the Splunk server user name.

- **-p, -password**

Defines the Splunk server password.

- **-s, -search**

(Optional) Defines the Splunk transaction search statement.

- **-t, -type**

Defines the data protocol type. HTTP, JMS, and MQ are supported.

Default: HTTP

- **-m, -maxtrans**

Defines the maximum transactions to be imported.

Default: 500

Create a path from Splunk using the following example code:

```
Java - jar lisaagentlight.jar - url http://<LisaServer>:1505 - splunk - hostname 10.130.151.105 - port 8089 - username admin - password admin
```

When you query the Splunk server for webMethods HTTP transactions, a Splunk query statement must be provided. The agent has an integrated default query statement. If you want a different query, pass in the new query statement using the **-search** parameter.

The following example displays a default query statement:

```
"search index=main ((CASE(GET) OR CASE(POST) OR CASE(PUT) OR CASE(DELETE))  
/*) OR ((Accept OR User-Agent OR Accept-Encoding OR \"  
-- Host\" OR Authorization OR Cookie OR Accept-Language OR \"  
-- Connection\" OR lisaFrameRoot OR lisaFrameRemoteIP OR lisaFrameID OR Authorization  
OR Set-Cookie OR SOAPAction OR Content-Type OR Content-  
Length: *) OR (\\" SOAP Request:\") OR (\\" SOAP Response:\") OR (HTTP/1.  
*) | transaction startsWith=(CASE(GET) OR CASE(POST) OR CASE(PUT) OR CASE  
(DELETE)) endsWith=(\\"--> Content-Length: \") | where !searchmatch(\\"-- User-  
Agent: webMethods\")"
```



Note: When the Agent Light for webMethods HTTP is executed from a remote system, the DevTest Server REST URL must be specified or the agent assumes the DevTest Server is local. The URL format is `http://<lisa-server>:1505`.

The agent light has the following arguments:

- **-url**

Defines the CAI REST base URL.

Default: `http://localhost:1505`

- **-wsuser**

(Optional) Defines the CAI REST username.

- **-wspassword**

(Optional) Defines the CAI REST password.

Input Transaction Data Format for webMethods HTTP

The input transaction data for webMethods HTTP must be in the standard HTTP request/response data format. The agent light takes the input data and converts it to the standard HTTP form.

```
HTTP request method URL {version} [line break]  
HTTP request header [line break]
```

```

...
HTTP request header [line break]
[line break - a blank line]
{HTTP request body - optional}[line break]
HTTP response status line [line break]
HTTP response header [line break]
...
HTTP response header [line break]
[line break - a blank line]
{HTTP response body - optional}[line break]

```

Agent Light for webMethods HTTP Baseline Support

The paths that are created by the agent light support only the DevTest test steps. The **use transaction frame step** option is disabled in the web interface. If the web service has the authorization mechanism through the HTTP header, the HTTP authorization header must be modified for the generated baseline. See [Generate a Web HTTP Baseline \(see page 1128\)](#) for information about generating a Web HTTP baseline.

Agent Light for webMethods HTTP Virtualization Support

The paths that are created by the agent light can be virtualized. The playback goes through the VSE and not the CAI VSE. See [Create Virtual Services from Web HTTP Transactions \(see page 1102\)](#) for information about creating virtual services from web HTTP transactions.

Agent Light for JMS

The Agent Light for JMS is an agent that communicates with CA Continuous Application Insight using a REST API. The purpose of the light agent is to take the input from a JMS transaction data XML file and send the data to CAI through the REST API call. The agent light can receive the JMS transactions from the JMS transaction data file.

The light agent uses the **-f** parameter to take a data file as an input. The data in this file must be in the XML format that is defined by CAI. See [Input Transaction Data Format for JMS \(see page 1189\)](#) for more information about the standard data format.

The transaction parameters from the data file are:

- **-f, -file**
Defines the transaction data file to be imported into CAI.
- **-t, -type**
Indicates the data protocol type.
Default: http

Create a path from a data file using the following example code:

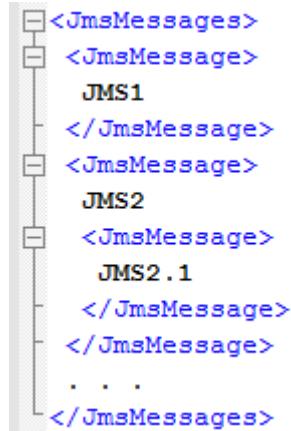
```
Java - jar lisaagentlight.jar - url http://<LisaServer>:1505 - f sampleJmsDataFile.xml - t jms
```

Input Transaction Data Format for JMS

The input transaction data for JMS must be in the XML format specified by CA Continuous Application Insight.

JmsMessages Tag

JMS message begins with a <JmsMessage> tag and ends with a matching </JmsMessage> tag. JMS messages can contain child JMS messages.

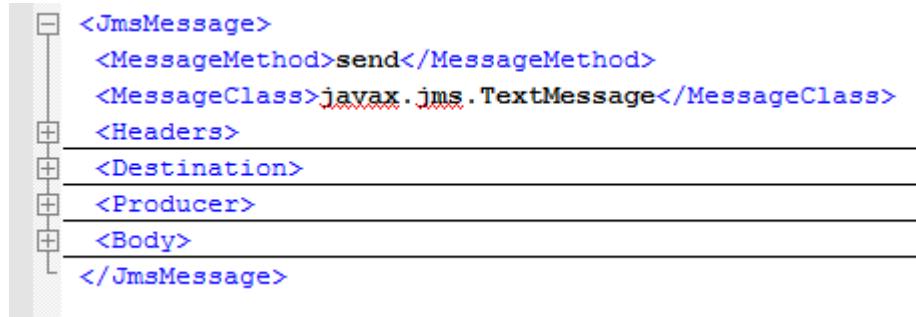


JMS Message tag

MessageMethod and MessageClass Tags

The <MessageMethod> tag is used to specify the message method. The valid message methods are send, receive, and onMessage. When the tag is missing, the default value is send.

The <MessageClass> tag specifies the message class. The only valid message class is javax.jms.TextMessage. When the tag is missing, the default value is javax.jms.TextMessage.



Message Method and Message Class tag

Headers Tags

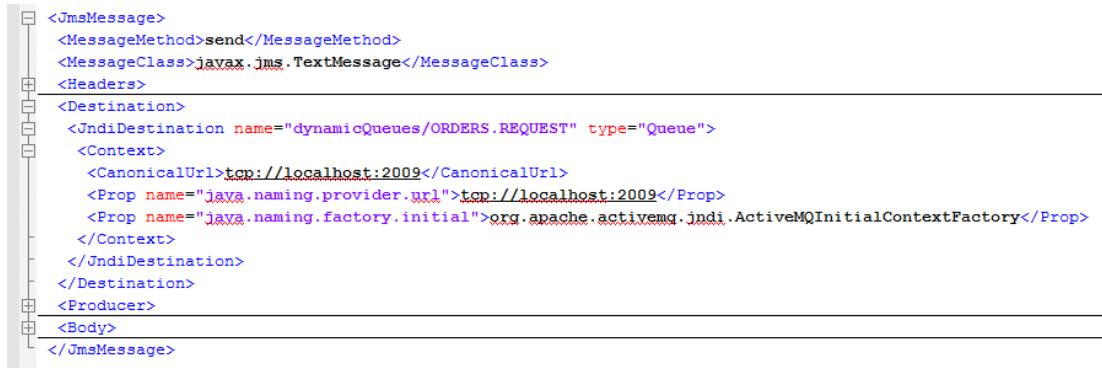
The <Headers> tag specifies the message headers. The valid headers are JMSCorrelationID, JMSDeliveryMode, JMSExpiration, JMSMessageID, JMSPriority, JMSRedelivered, JMSTimestamp, and JMSType.



Header tag

Destination Tags

The <Destination> tag specifies the message destination. The valid destination is a JNDI destination (a JNDI-registered Queue or Topic). JMS message with the send method will specify the message destination.



Destination tag

Producer Tags

The <Producer> tag is used to specify the message producer. The Producer can contain a Destination (JNDI) and a Session. The session contains the session information (Transacted, AcknowledgeMode) and a Connection Factory (JNDI). JMS message with the **send** method will specify the message producer.

```

<JmsMessage>
  <MessageMethod>send</MessageMethod>
  <MessageClass>javax.jms.TextMessage</MessageClass>
  <Headers>
  <Destination>
  <Producer>
  <Session>
    <Transacted>false</Transacted>
    <AcknowledgeMode>1</AcknowledgeMode>
    <ConnectionFactory>
      <JndiConnectionFactory name="ConnectionFactory">
        <Context>
          <CanonicalUrl>tcp://localhost:2009</CanonicalUrl>
          <Prop name="java.naming.provider.url">tcp://localhost:2009</Prop>
          <Prop name="java.naming.factory.initial">org.apache.activemq.jndi.ActiveMQInitialContextFactory</Prop>
        </Context>
      </JndiConnectionFactory>
    </ConnectionFactory>
  </Session>
  </Producer>
  <Body>
  </JmsMessage>

```

Producer tag

Consumer Tags

The <Consumer> tag is used to specify the message consumer. The Consumer has the same structures as the Producer. JMS message with the receive/onMessage method will specify the message consumer.

Body Tags

The <Body> tag is used to specify the message payload (text). If the text payload itself contains the XML tags, the text payload must be inside a CDATA section. The payload is ignored by the parser.

```

<JmsMessage>
  <MessageMethod>send</MessageMethod>
  <MessageClass>javax.jms.TextMessage</MessageClass>
  <Headers>
  <Destination>
  <Producer>
  <Body>
    <![CDATA[
      <order>
        <id>2-22</id>
        <name>ouid-2</name>
        <product>prod200</product>
      </order>
    ]]>
  </Body>
</JmsMessage>

```

Body tag

Agent Light for JMS Baseline Support

The paths that are created by the agent light support only the DevTest test steps. Consolidated and expanded baselines are only supported. See [Generate a JMS Baseline \(see page 1119\)](#) for information about generating a JMS baseline.

Agent Light for JMS Virtualization Support

The paths that are created by the agent light can be virtualized. See the [Create Virtual Services from JMS Transactions \(see page 1086\)](#) for more information about creating a virtual service for MQ transactions.

Agent Light for WebSphere MQ

The Agent Light for WebSphere MQ is an agent that communicates with CA Continuous Application Insight using a REST API. The purpose of the light agent is to take the input from an MQ request /response data file and send the data to CAI through the REST API call.

The agent light can receive the MQ transactions from the following sources:

- MQ transaction data file

The light agent uses the **-f** parameter to take a data file as an input. The transaction parameters from the data file are:

- **-f, -file**

Defines the transaction data file to be imported into CAI.

- **-t, -type**

Defines the data protocol type. HTTP, JMS, and MQ are supported.

Default: HTTP

Create a path from data file using the following example code:

```
Java - jar lisaagentlight.jar - url http://<LisaServer>:1505 - f mqSample.xml - t mq
```

- Third-party log file from a Splunk query

Third-party log files are supported using a Splunk query. Splunk indexes the log file and produces the MQ transaction data. The agent light queries Splunk to get the MQ transactions and creates a CAI path for them. The third-party log file may not contain all the information required by the MQ transaction. An extended XML data file is defined to allow users to input the missing data. The data in the extended file must be in the XML format defined by DevTest Solutions.

The transaction parameters from Splunk are:

- ▪ **-splunk**

Indicates transactions acquired from Splunk.

- **-h, -hostname**

Defines the Splunk server hostname or IP address.

- **-port**

Defines the Splunk port number.

- **-u, -username**
Defines the Splunk server user name.
- **-p, -password**
Defines the Splunk server password.
- **-s, -search**
(Optional) Defines the Splunk transaction search statement.
- **-t, -type**
Defines the data protocol type. HTTP, JMS, and MQ are supported.
Default: HTTP
- **-m, -maxtrans**
Defines the maximum transactions to be imported.
Default: 500
- **-source**
Defines the Splunk data source, either webMethods, swamq or JMS.
- **-extf, -extfile**
Defines the extended transaction data file.

Create a path from Splunk using the following example code:

```
Java - jar lisaagentlight.jar - url http://<LisaServer>:  
1505 - splunk - hostname 10.130.151.105 - port 8089 - username admin - passwo  
rd admin -source swamq -type mq -extfile extendedDataFile.xml
```

When you query Splunk for third-party transactions, a Splunk query statement must be provided to query for the transactions. The agent has a default query statement built-in. If a different query is needed, you can pass a new query statement using the **-search** parameter.

The following statement is the built-in default query statement:

```
"search index=main CASE(\"- REQUEST: <?xml\\\" OR CASE(\"- RESPONSE: <?  
xml\\\") | transaction startsWith=CASE(REQUEST:) endsWith=CASE(RESPONSE:)\")";
```

If the Agent Light for WebSphere MQ is executed from a remote system, the DevTest Server REST URL must be specified. Otherwise the agent assumes the DevTest Server is local. The URL format is <http://<lisa-server>:1505>.

The agent light has the following optional parameters:

- **-url**
Defines the CAI REST base URL.
Default: http://localhost:1505
- **-wsuser**
(Optional) Defines the CAI REST username.
- **-wspassword**
(Optional) Defines the CAI REST password.



Note: The data in the transaction data file and in the extended XML data file must be in the XML format that is defined by DevTest Solutions. See [Input Transaction Data Format for MQ \(see page 1195\)](#) for more information about the standard data format.

Input Transaction Data Format for MQ

The input transaction data for MQ must be in the format specified by DevTest Solutions. Every MQ transaction must look like the following example:

```
<MQTransactions>
  <MQTransaction>
    <MQMessage>
      ...
      <QueueConnection>
      ...
      </QueueConnection>
      <request>request content</request>
    </MQMessage>
    <MQMessage>
      ...
      </MQMessage>
      <MQMessage>
        ...
        </MQMessage>
        ...
      </MQMessage>
      ...
    </MQMessage>
  </MQTransaction>
  ...
  <MQTransaction/>
</MQTransactions>
```

The data in the extended data file needs to be in the XML format defined by DevTest Solutions.

```
<Platforms>
  <SWA>
    <!-- The log is from mq client or not, default value is true -->
    <IsClient>true</IsClient>
    <QueueConnection>
      <!-- required -->
      <RequestQueue>ORDERS.REQUEST</RequestQueue>
      <ResponseQueue>ORDERS.RESPONSE</ResponseQueue>
      <QueueManager>QueueManager</QueueManager>
      <Host>HostName</Host>
      <Port>1414</Port>
      <Channel>SERVERCON</Channel>
      <!-- often appear -->
      <Username>administrator</Username>
      <Password>dcam09@CTC</Password>
    </QueueConnection>
  </SWA>
</Platforms>
```

Agent Light for WebSphere MQ Baseline Support

The path created by the agent light supports only the DevTest test steps. Consolidated and expanded baselines are only supported. See [Generate a WebSphere MQ Baseline \(see page 1138\)](#) for information about generating a WebSphere MQ baseline.

Agent Light for WebSphere MQ Virtualization Support

The paths created by the agent light can be virtualized. See [Create Virtual Service from WebSphere MQ Transactions \(see page 1107\)](#) for more information about creating a virtual service for MQ transactions.

CAI Extension for Google Chrome

This release includes a preview of the CAI extension for the Google Chrome browser. You can use the extension to perform the following tasks:

- Capture and analyze transactions
- Create baselines
- Create tickets

The following graphic shows the main portion of the user interface.

The screenshot displays the CAI Paths v2 interface within a browser window. The top navigation bar includes links for Elements, Network, Sources, Timeline, Profiles, Resources, Audits, Console, and CAI Paths v2. Below the navigation bar are several buttons: Server Setting, Url Setting, SSL Setting, Create Defect, cleanAll, shelveREST, shelveAll, and clearShelf. The main content area is a table with columns: Tag, Name, Time, Status, and U. It lists three transactions:

- /cars-app/assets/cars/check... at 10:31:25 with a red error status and a tooltip showing /cars-app/assets/cars/checkInventor... (355ms).
- REST /cars-app/carShop/makes at 10:31:33 with a red error status.
- REST /cars-app/carShop/cars at 10:31:33 with a red error status.

 A small icon of a computer monitor is visible on the right side of the table area.

Screen capture of main portion of user interface.

Installing and Configuring the CAI Extension

The procedures in this page are based on version 44 of the Chrome browser.

Install the Extension

The CAI extension is available from the Sandbox.



Note: As of version 44.0.2403, Google Chrome disables any extension that is not installed from the Chrome Web Store. After the following procedure, you can use the CAI extension. However, when you restart the browser, the extension is no longer available because of this restriction. To use the extension again, you must remove and then reinstall it. We do plan to add the extension to the Chrome Web Store.

Follow these steps:

1. Open the DevTest Portal.
2. Click the down arrow to the right of your user name and select **The Sandbox**.
3. Click the **Download Extension** link.
The **pf-chromeextension.crx** file is downloaded.
4. Open the **chrome://extensions** page.

5. Select the **Developer mode** check box.
6. Drag the **pf-chromeextension.crx** file to the page.
7. When prompted to confirm the new extension, click **Add**.

Configure the Server Settings

To communicate with the DevTest Server, the extension uses a group of server settings. The following procedure shows how to change the values.

Follow these steps:

1. Click the Chrome menu icon and select **More tools, Developer tools**.
2. Click **CAI Paths v2**.
3. Click **Server Setting**.
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.
 - b. **Console Port**
The port number of the embedded web server. The default value is 1505.
 - c. **Portal Port**
The port number of the DevTest Portal. The default value is 1507.
 - d. **User Name**
The user name that will be used to access DevTest Solutions. The default value is admin.
 - e. **Password**
The password that will be used to access DevTest Solutions. The default value is admin.
5. Click **Save**.

Include and Exclude URLs

The extension gives you control over the types of URLs to include or exclude.



Note: The include and exclude settings are mutually exclusive.

By default, the extension is configured to exclude certain types of requests and responses. For example:

- Request URLs that end in .js
- Request URLs that end in .png or jpg
- Responses that have a content-type header of **image/png** or **image/jpeg**
- Responses that have a content-type header of **application/javascript**

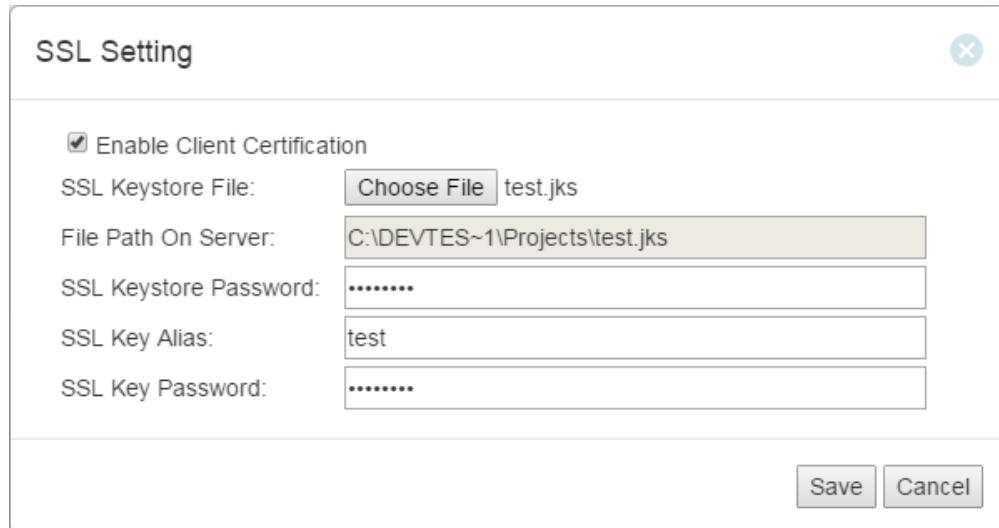
Follow these steps:

1. Click the Chrome menu icon and select **More tools, Developer tools**.
2. Click **CAI Paths v2**.
3. Click **Url Setting**.
4. To configure the include settings:
 - a. Select the **Include Url** option.
 - b. Enter the URL to include as a string or regular expression, and click **Add**.
5. To configure the exclude settings:
 - a. Select the **Exclude Url** option.
 - b. Add, change, and remove the request filters as needed. You can also disable all of the request filters.
 - c. Add, change, and remove the response filters as needed. You can also disable all of the response filters.
 - d. Click **Apply**.

Configure the SSL Settings

If necessary, you can configure the client certificate information for two-way SSL.

The following graphic shows the **SSL Setting** dialog after the configuration settings are saved.



Screen capture of SSL Setting dialog.

Follow these steps:

1. Click the Chrome menu icon and select **More tools, Developer tools**.
2. Click **CAI Paths v2**.
3. Click **SSL Setting**.
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 is uploaded to the DevTest Server. The **File Path On Server** field displays the location.

Using the CAI Extension

The procedures in this page are based on version 44 of the Chrome browser.

Capture and Analyze Transactions

You can capture transactions from any web page.

The details for a node in the path graph include the transaction frame data in JSON format.

Follow these steps:

1. Open the web page.
2. Click the Chrome menu icon and select **More tools, Developer tools**.

3. Click **CAI Paths V2**.
4. Perform operations in the web page.
One or more paths appear in the left pane.
5. Click a path.
The path graph appears in the middle pane.
6. To view the details for a node in the path graph, right-click the node and select **Detail**.
7. To remove the transactions in the left pane, click **cleanAll**.

Shelve Transactions

The CAI extension includes a shelf that is similar to the [shelf \(see page 1061\)](#) in the **Analyze Transactions** window of the DevTest Portal.

The shelf is located in the right pane.

Follow these steps:

1. To shelve a single frame, right-click a node in the path graph and select **Shelf**.
2. To shelve only the REST frames, click **shelveREST**.
3. To shelve all frames, click **shelveAll**.
4. To remove all frames from the shelf, click **clearShelf**.

Create Baselines

You can generate baselines from transactions that have been captured. You can then view the baselines in DevTest Workstation.

Follow these steps:

1. Add one or more frames to the shelf.
2. Click **Generate Baseline**.
3. (Optional) Change the default name.
4. Click **Create**.
5. Select the project where the baseline will be created.
6. Select an option to keep or delete the transactions in the shelf once the baseline is created.
7. Click **Create**.

Create Tickets

You can use the extension to create [tickets \(see page 1066\)](#) from a web page. You can then view the tickets in the DevTest Portal.

Follow these steps:

1. Click **Create Defect**.
2. Enter the title, external defect tracking number, severity, and description.
3. To include a screen shot of the application at the time of capture, ensure that the **Take screenshot** check box is selected.
4. Click **Submit**.

Using CA Application Trace Kit

CA Application Trace Kit provides visibility into Java-based applications.



Note: CA Application Trace Kit is a preview feature in this release.

Contents

- [Start the Application Trace Kit \(see page 1201\)](#)
- [Live Paths \(see page 1202\)](#)
- [Saved Paths \(see page 1203\)](#)
- [Configure the Amount of Data Captured \(see page 1204\)](#)
- [Configure Agent Properties \(see page 1205\)](#)
- [View and Configure Logging \(see page 1205\)](#)
- [View and Manage Classes \(see page 1205\)](#)
- [Manage Files \(see page 1206\)](#)
- [Execute Remote Commands from the Terminal \(see page 1207\)](#)

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:\DevTest\bin>ATK.exe -u tcp://localhost:2009
```

You can add the **-console** option to display console output.

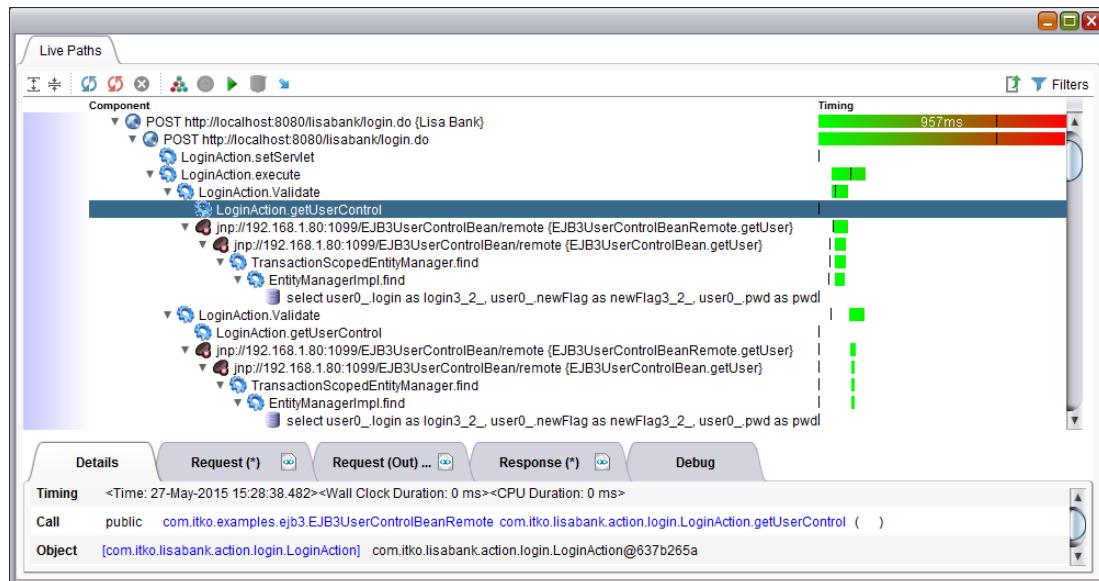
```
C:\DevTest\bin>ATK.exe -u tcp://localhost:2009 -console
```

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.

Live Paths

When you exercise the Java-based application, the **Live Paths** tab displays the transactions that one or more DevTest Java Agents captured.

The following graphic shows a transaction in the **Live Paths** tab.



Screen capture of transaction in Live Paths tab.

Each node in the **Component** column represents a transaction frame.

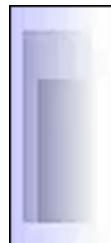
The horizontal bars in the **Timing** column provide a visual indicator of the duration of each frame. For frames other than the root frame, the horizontal bars also indicate when the frame started and ended in relation to the root frame. A horizontal bar can include a vertical divider, which indicates 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 three agents.



Screen capture of shading in first column.

To view more information about a frame, click the frame in the **Component** column. One or more of the following tabs appear:

- **Details**
Displays basic information about the frame.
- **Request**
Displays the request payload.
- **Request (Out)**
Displays the value of the arguments at the end of method execution.
- **Response**
Displays the response payload.
- **Screenshot**
Displays the image that is associated with a GUI event.
- **Debug**
Displays information that can be used for debugging.

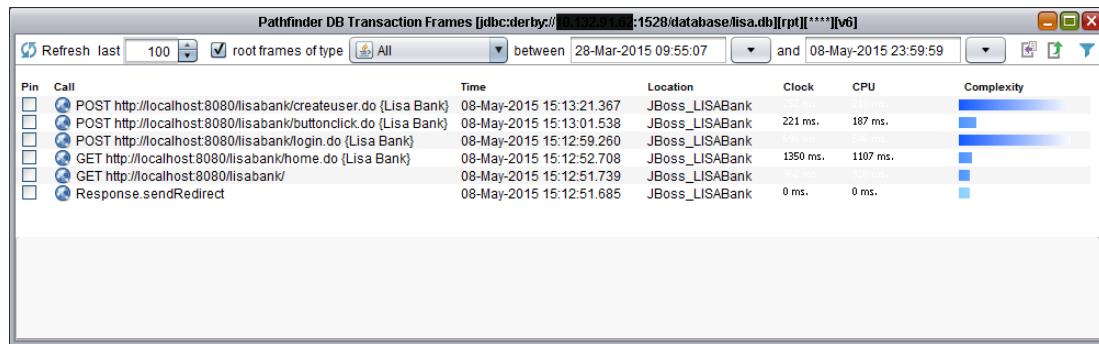
If you enable profiling and then record some transactions, you can display the full call tree for any frame. Right-click the frame and select **Show call tree**.

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 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]
```

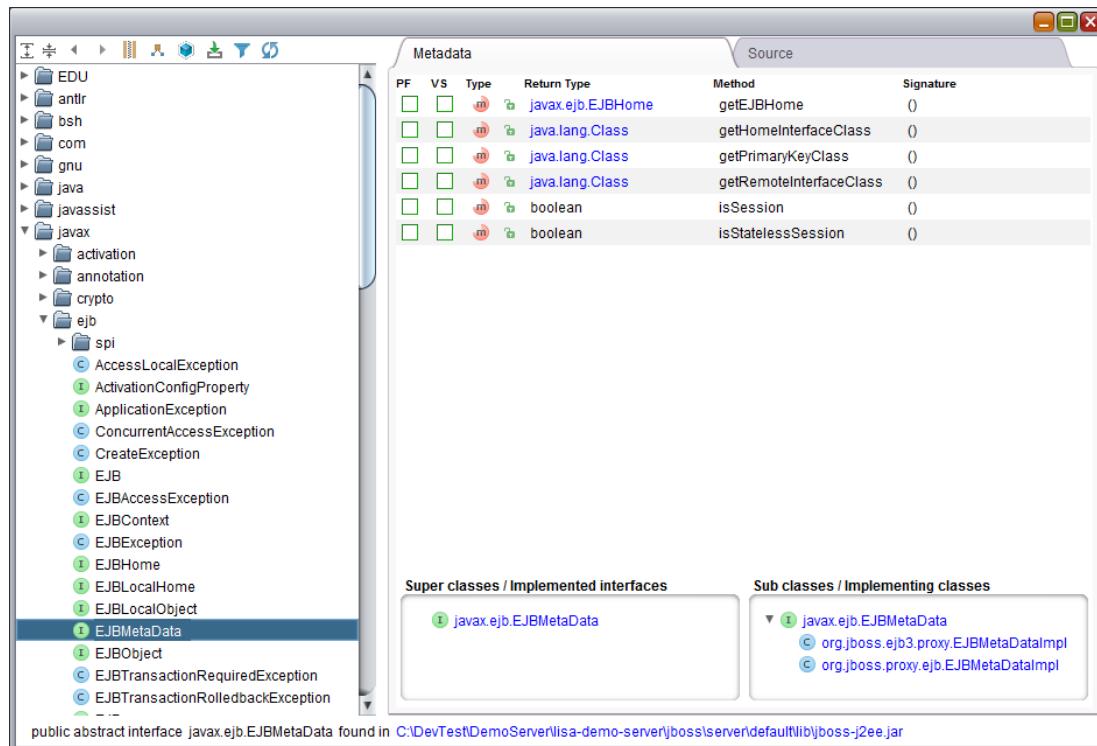
Follow these steps:

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 \(see page 1210\)](#) 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**.

CA Application Trace Kit Database Tools

CA Application Trace Kit includes tools for various types of databases.



Note: CA Application Trace Kit is a preview feature in this release.

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 arbitrary queries

Follow these steps:

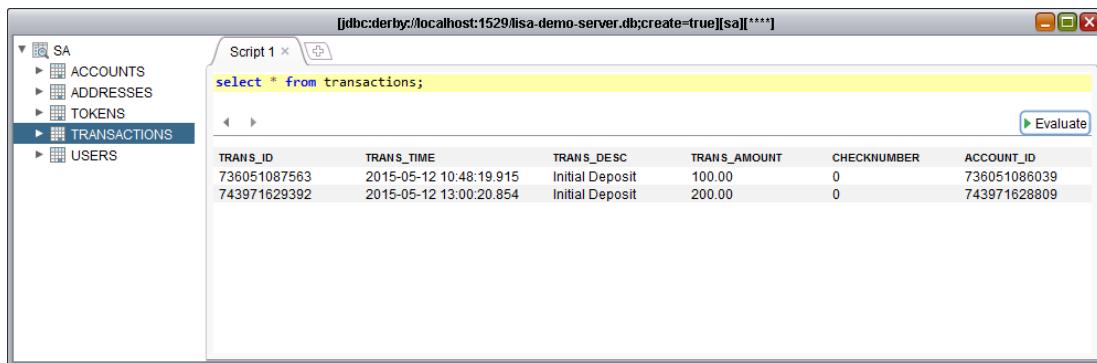
1. Click **Database**.
2. View the database schema in the left pane.
3. To display the DDL statements, click the database icon in the upper left corner.
4. To run a SQL query, enter the query in the script tab and click **Evaluate**.

Explore Databases Connected to Agents

You can perform the following tasks on a database that is connected to a Java agent:

- 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 lets you create the following types of extensions for the DevTest Java Agent:

- CAI
- 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 \(see page 1300\)](#).

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 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.

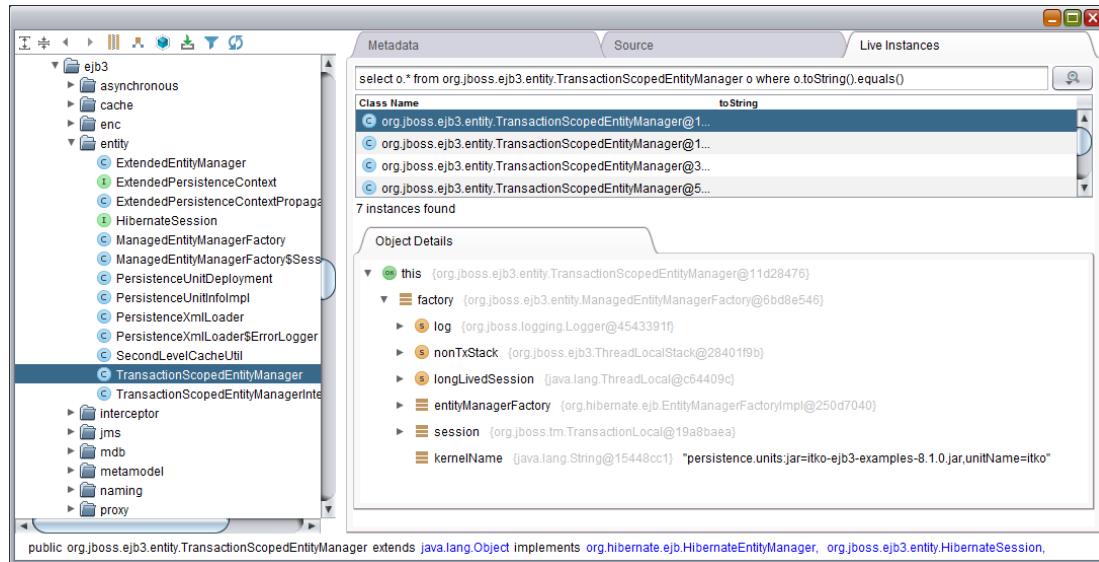
Live Instances

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

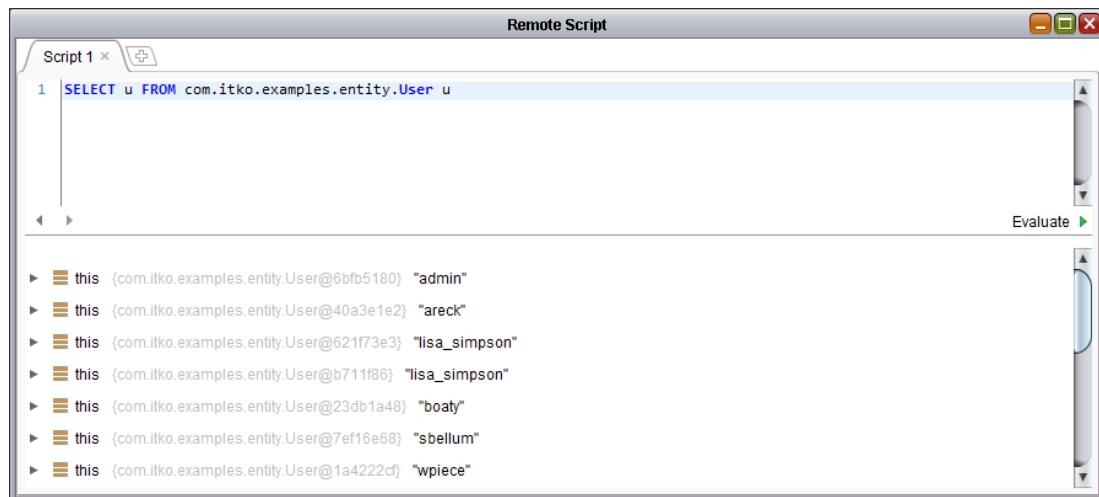
You can use OQL to query for object instances or values in the Java heap using arbitrary filters.

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 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 \(see page 1214\)](#).
- **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 \(see page 1214\)](#).
- **HasLisalIntegrator 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 HasLisalIntegrator interface, see [Constructing a Response Object \(see page 1218\)](#).

- **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 \(see page 1219\)](#).
- **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 \(see page 1219\)](#).

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 \(see page 1214\)](#).
3. Handle the output from the integrated method.
For more information about handling integrated output, see [Handle Integrated Output \(see page 1218\)](#).
4. Use integration filters in the test case that tests the server-side component.
For more information about integration filters, see [Integration Filters \(see page 1219\)](#).
5. Use integration assertions in the test case that tests the server-side component.
For more information about integration Assertions, see [Integration Assertions \(see page 1219\)](#).

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 \(see page 1214\)](#)
- [Collect Transaction Information \(see page 1216\)](#)
- [Integrators \(see page 1217\)](#)
- [Handle Integrated Output \(see page 1218\)](#)

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 \(see page 1216\)](#).

- 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 \(see page 1216\)](#).

- 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 \(see page 1218\)](#).

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.ComplInfo** 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 ComplInfo 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 **ComplInfo** 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 \(see page 1216\)](#).

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 to DevTest 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 HasLisalIntegrator 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 ComlInfo 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 \(see page 318\)](#).

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 ComplInfo 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 \(see page 318\)](#).

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 \(see page 318\)](#).

Extending DevTest Solutions

This section explains the main concepts that are involved in extending the 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 \(see page 1223\)](#)

Extension Concepts

The following concepts are common to all customization scenarios:

- [lisaextensions File \(see page 1223\)](#)
- [wizards.xml File \(see page 1224\)](#)
- [The NamedType Interface \(see page 1225\)](#)
- [Parameters and Parameter Lists \(see page 1226\)](#)
- [The Test Exec Class \(see page 1227\)](#)
- [Test Exceptions \(see page 1228\)](#)

[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:

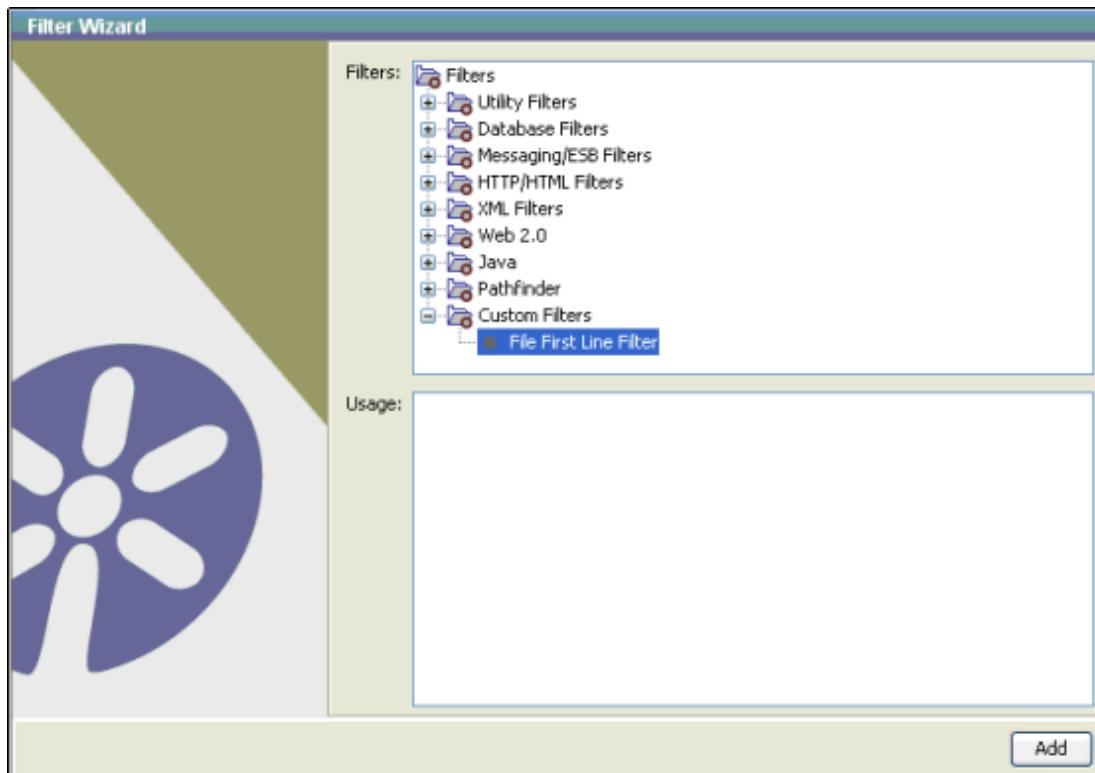
```
<Entry>
  <Name>Make Diff-like Comparisons on the HTML Result</Name>
  <Help>This is the right assertion to use when you have a HTML result and you wish to perform diff-like operations on it. It lets you define simply what portions of the text may change, what may not change, and what portions must 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 get TypeName()
{
    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**.

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 \(see page 1229\)](#)
- [Native Test Steps \(see page 1233\)](#)

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 is provided in the **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 the 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 the 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
(http://ftp.suse.com)&path=/pub&file=INDEX" );
    return pl;
}
```

The previous example uses ftp.suse.com/pub/INDEX (<http://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 \(see page 1238\)](#).

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.

To deploy a custom Java test step:

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 \(see page 1212\)](#).

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/ (<http://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.

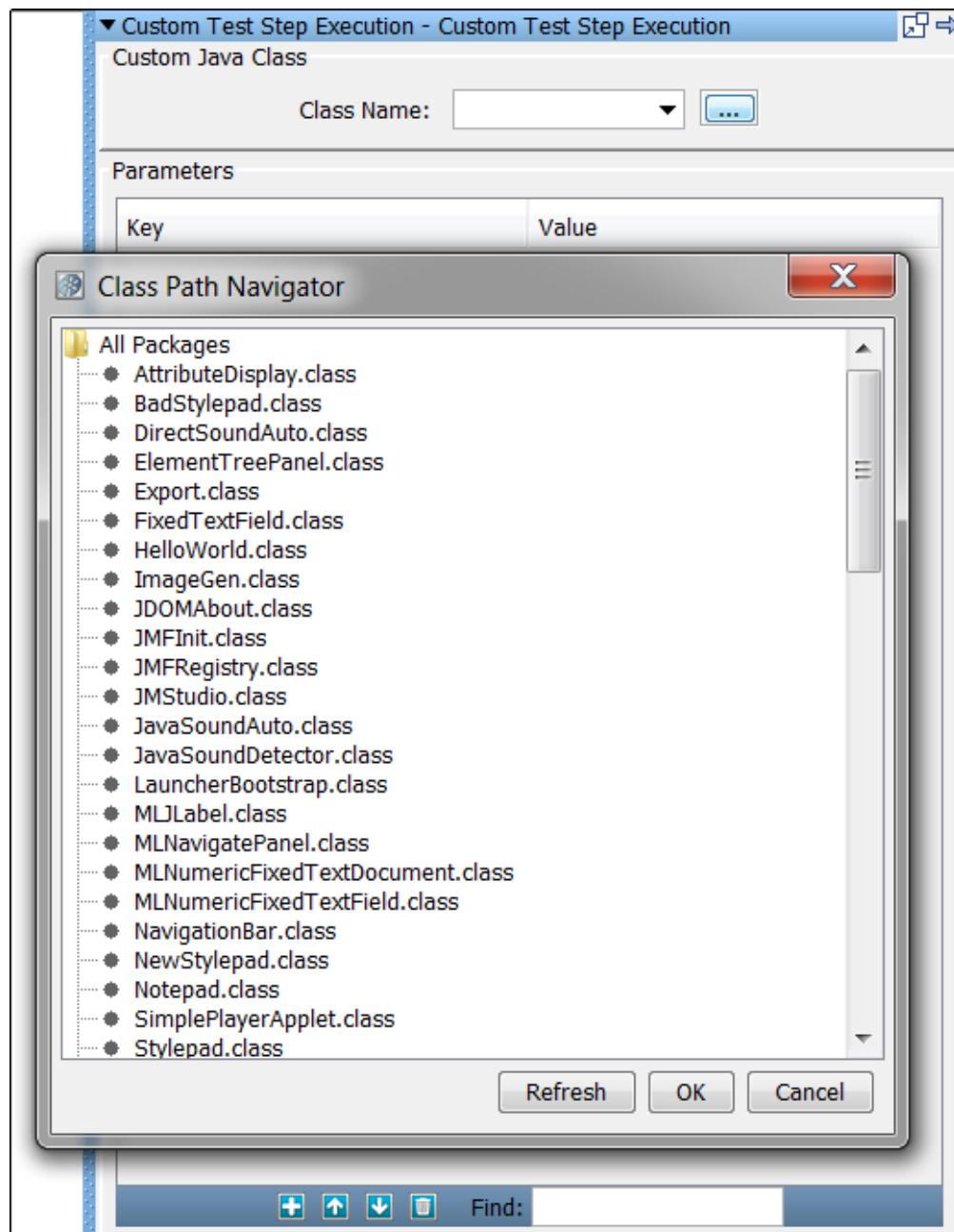
Follow these steps:

1. Change the **Type** of the node to **Custom Test Step Execution**.

2. Specify the custom test step to test.

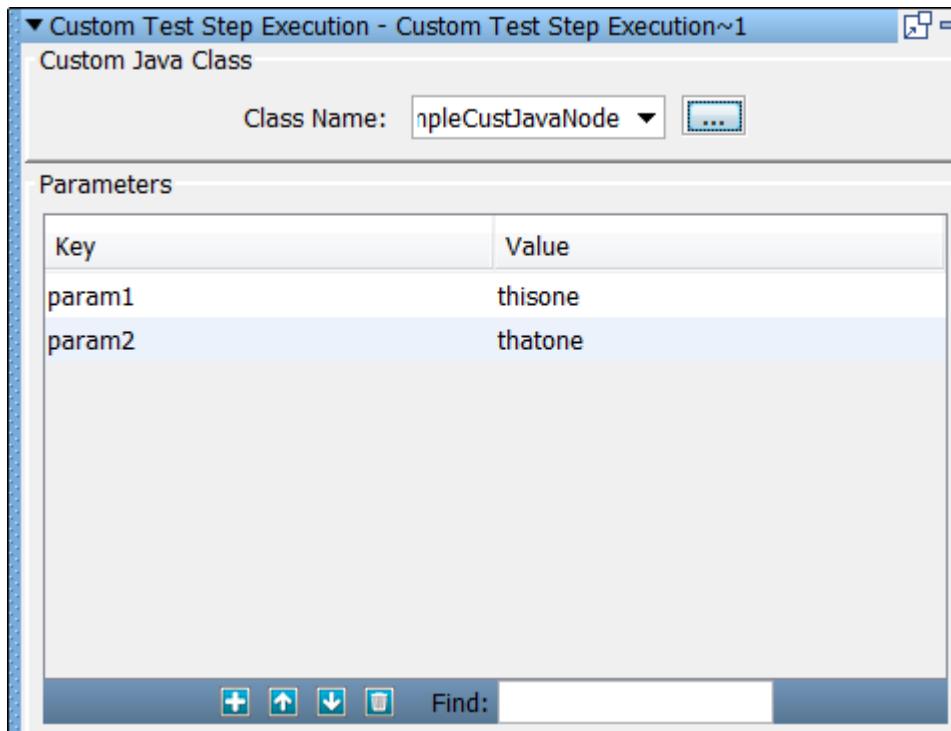
Enter or select the fully qualified name of the Java class in the **Class Name** field.

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 \(see page 1214\)](#).

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 \(see page 1222\)](#).

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 \(see page 318\)](#) 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 **get TypeName** 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 **get TypeName** 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 specific
d.", 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
```

You can also add this assertion to the wizards in the **wizards.xml** file.

2. 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 \(see page 1212\)](#). This custom assertion 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 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 **get TypeName** 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 **get TypeName** 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 \(see page 1212\)](#). 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 **get TypeName** 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 Reports

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 \(see page 496\)](#).

```
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 \(see page 496\)](#).

```
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 (h  
ttp://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 \(see page 1222\)](#).

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 **get TypeName** method.

This method provides the name that is used to identify the companion in the Test Case Editor.

```
public String get TypeName()
{
    return "Nifty Data Set";
}
```

The string that the **get TypeName** method returns is the default name of a new data set.

3. Implement the **get Parameters** 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 get Parameters() throws RemoteException
{
    ParameterList pl = super.get Parameters();
    // ...
    return pl;
}
```

For more information about Parameters and ParameterLists, see [Extending DevTest Solutions \(see page 1222\)](#).

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 \(see page 1222\)](#) 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.

Java .NET Bridge

On Windows, DevTest embeds a library that enables in-process, bi-directional communication between the Java VM and the CLR (.NET). This library is named jdbridge (as in **java dotnet bridge**) and consists of these components:

- lisa.jdbridge-x.x.x.jar, where x.x.x is the product release (the Java-side stubs)
- jdbridge.dll (the .NET-side stubs)
- jdglue.dll and jdglue64.dll (the glue between the two)

These components can be found in the usual DevTest locations (the bin and lib\core directories).

The easiest way to take advantage of this bridge is by using the custom JavaScript step, but extensions are also possible. This section covers only the Java -> .NET API because it is the natural usage from DevTest. The following classes are the three central classes.

com.itko.lisa.jdbridge.JDInvoker

```
/** Loads the .NET CLR in the Java process */
public static native void startCLR();

/** Stops and unloads the .NET CLR from the Java process */
public static native void stopCLR();

/**
 * Invokes a methods in the specified .NET assembly (.dll or .exe).
 * @param assembly the full path to the assembly the type resides in
 * @param type      the fully qualified name of the type on which to invoke
 * @param method    the name of the method to invoke
 * @param args      an array of arguments expected by the method
 * @return the return value of the .NET method
 */
public static Object invoke
(String assembly, String type, String method, Object ... args)
```

com.itko.lisa.jdbridge.JDProxy

```
/**
 * Returns a proxy to a .
NET instance that exists in the CLR after invoking its constructor.
 * @param assembly the full path to the assembly the type resides in
 * @param type      the type to instantiate
 * @param args      an array of arguments expected by the constructor
 * @return a proxy to the .NET instantiated type
 */
public static JDProxy newInstance(String assembly, String type, Object ... args)
```

```

/**
 * Invokes the specified method on the object represented by this proxy
 * @param method the name of the method to invoke
 * @param args an array of arguments expected by the method
 * @return the return value of the method
 */
public Object invoke(String method, Object ... args)

/**
 * If the object represented by this proxy exposes .
NET event delegates, this method enables the Java
 * program to register event listeners in Java code.
 * @param event the name of the event to listen for
 * @param l the listener interface whose onEvent method gets invoked when the even
t is fired.
 */
public void addListener(String event, JDProxyEventListener l)

/**
 * Removes an event listener previously added via addListener.
 * @param event the name of the event to listen for
 * @param l the listener interface whose onEvent method gets invoked when the even
t is fired.
 */
public void removeListener(String event, JDProxyEventListener l)

/**
 * Method to invoke to release resources when done with the proxy.
 */
public void destroy()

```

com.itko.lisa.jdbridge.JDProxyEventListener

```

/**
 * This method gets invoked (from .
NET) on all listeners registered via JDProxy's addListener method.
 * @param source the proxy to the object raising the event (on which addListener was c
alled)
 * @param evt the name of the event being raised
 * @param arg a string representation of event data
 */
public void onEvent(JDProxy source, String evt, Object arg)

```

As usual when two technologies communicate with each other, it is important to understand the marshaling mechanism for arguments and return values. The approach that jdbridge takes is similar to RMI and .NET remoting in that there is marshaling by value or by reference.

All primitive types (Boolean, byte, short, char, int, long, float, double) and Strings are marshaled by value and map one-to-one between Java and .NET. No special handling is required.

Similarly, framework collections classes are mapped one-to-one (Java **Lists** to .NET **Lists** and Java **Maps** to .NET **Dictionaries**).

For general objects, if the .NET object implements the **IXmlSerializable** interface, it is marshaled back to Java by value. This means that a Java class with the exact same format (package, name, methods, and so on) must exist in the classpath. Otherwise it is marshaled by reference as a **JDProxy**. This lets you chain **JDProxy** calls and pass a **JDProxy** as an argument to another **JDProxy** call.

Exceptions that are raised in .NET are also propagated to Java and thrown as **RuntimeExceptions** with a stack trace spanning both Java and .NET code.

Example

Assume there is a .NET dll named **AcmeUtils.dll** in the **bin** directory. This dll contains the type **com.acme.Calculator** that has all the usual arithmetic functions: **Add**, **Subtract**, and so on. You want to invoke those functions from a JavaScript step. Here is a script that invokes them:

```
import com.itko.lisa.jdbridge.JDInvoker;
import com.itko.lisa.jdbridge.JDProxy;

JDInvoker.startCLR();
JDProxy calc = JDProxy.newInstance(Environment.LISA_HOME + "bin\\AcmeUtils.dll", "com.acme.Calculator", new Object[0]);

Integer sum = (Integer) calc.invoke("Add", new Object[] { 3, 4 });
Double ratio = (Double) calc.invoke("Divide", new Object[] { 3.0, 4.0 });
Integer square = (Integer) calc.invoke("Square", new Object[] { 5 });
...
```

The arguments are passed explicitly into an array of Objects because the DevTest BeanShell does not support the varargs (...) notation. If you were to code this in an extension, the syntax becomes less cumbersome.

Suppose the **Calculator** object exposes the **OnCalculationStart** and **OnCalculationEnd** events and you want to subscribe to those events to measure the duration of the calculation:

```
...
JDProxy calc = JDProxy.newInstance(Environment.LISA_HOME + "bin\\AcmeUtils.dll", "com.acme.Calculator", new Object[0]);

calc.addListener("OnCalculationStart", new JDProxyEventListener() {
    public void onEvent(JDProxy source, String evt, Object arg) {
        //capture timestamp here
    }
});

calc.addListener("OnCalculationEnd", new JDProxyEventListener() {
    public void onEvent(JDProxy source, String evt, Object arg) {
        //capture timestamp here
    }
});

Long fact = (Long) calc.invoke("Factorial", new Object[] { 30 });
...
// diff the timestamps here
...
```

If your project involves extensive use of .NET assemblies, it is a good idea to wrap all interactions with .NET code inside compiled extensions. In this case, the standard pattern would look like the following:

```
package com.acme;

import com.itko.lisa.jdbridge.JDInvoker;
import com.itko.lisa.jdbridge.JDProxy;
import com.itko.lisa.jdbridge.JDProxyEventListener;

public class Calculator {
    static {
        JDInvoker.startCLR();
    }

    private JDProxy m_proxy = JDProxy.newInstance(Environment.LISA_HOME + "bin\\AcmeUtils.dll", "com.acme.Calculator");

    public int add(int x, int y) {
        return ((Integer) m_proxy.invoke("Add", new Object[] { x, y })).intValue();
    }
}
```

```
    }    ...
```

This extension can be invoked from a custom JavaScript step or even the [Complex Object Editor \(COE\)](#) ([see page 443](#)).

Agents

This section describes how to install and configure each DevTest agent.

- [DevTest Java Agent \(see page 1253\)](#)
- [Mainframe Bridge \(see page 1316\)](#)
- [DevTest CICS Agent \(see page 1318\)](#)
- [DevTest LPAR Agent \(see page 1355\)](#)

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 JVMs 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 CA Continuous 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:

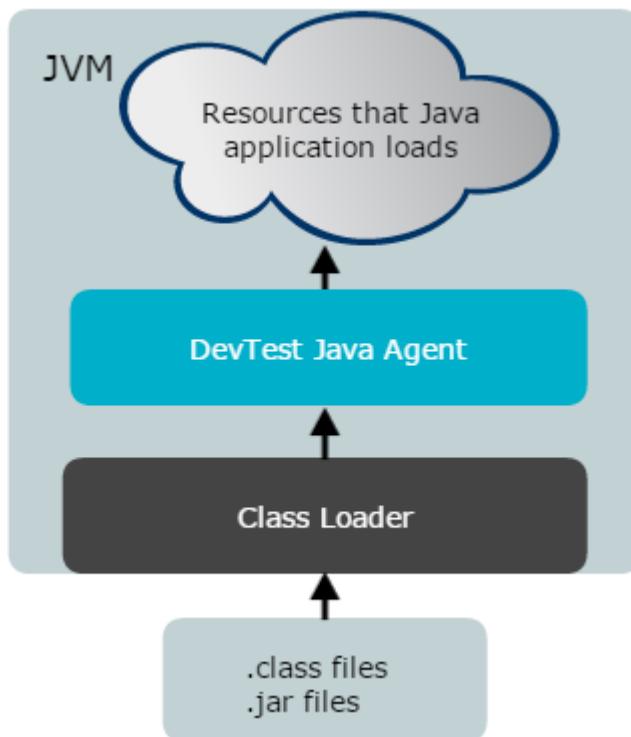
- [Java Agent Architecture \(see page 1254\)](#)
- [Java Agent Installation \(see page 1258\)](#)
- [Start the Broker \(see page 1272\)](#)
- [rules.xml File \(see page 1272\)](#)
- [Adding a Method to Intercept \(see page 1281\)](#)
- [Excluding from Interception and Virtualization \(see page 1282\)](#)
- [Java Agent Auto-Response Generation \(see page 1284\)](#)
- [Java Agent REST APIs \(see page 1285\)](#)
- [Developing Against the Java Agent \(see page 1287\)](#)
- [Java Agent Extensions \(see page 1300\)](#)
- [Load Balancers and Native Web Servers \(see page 1306\)](#)
- [Java Agent Security \(see page 1307\)](#)
- [Java Agent Log Files \(see page 1309\)](#)
- [Java Agent Troubleshooting \(see page 1310\)](#)

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, which runs embedded in a Java process
- The broker
- The consoles (or clients)

- The database

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 full transactions for the benefit of consoles.

After the data is assembled, the data is sent to the consoles (short term) and persisted to the database for long-term storage.

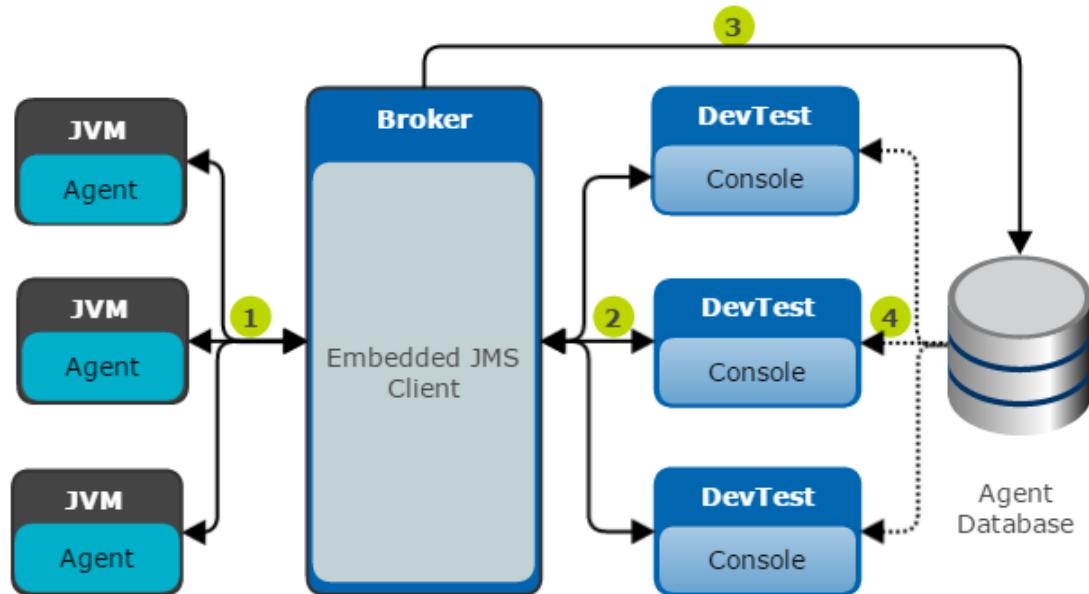
This documentation ignores the distinction and refers to the embedded client as the broker.

The **consoles** 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 DevTest Workstation, VSE, and simulators.

If the agents and the broker are on different computers, ensure that the system times are synchronized.

Java Agent Data Flow

The following graphic shows the Java agent components and how they interact.



The following 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.

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 by running 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.

You can also obtain the DDL statements from the following files in the **LISA_HOME\database** directory:

- **db2_cai.ddl**
- **derby_cai.ddl**
- **mysql_cai.ddl**
- **oracle_cai.ddl**
- **sqlserver_cai.ddl**

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.



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 \(see page 1259\)](#)
- [Install the Native Agent \(see page 1259\)](#)
- [Options for Agent Parameters String \(see page 1260\)](#)
- [Java Agent Install Assistant \(see page 1261\)](#)
- [Platform-Specific Agent Installation \(see page 1265\)](#)
- [Wily Introscope Agent \(see page 1271\)](#)

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 [Java Agent Troubleshooting \(see page 1310\)](#) before contacting Support.

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 agent installation \(see page 1265\)](#) 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 [Java Agent Troubleshooting \(see page 1310\)](#) 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.

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
Mac 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 agent installation \(see page 1265\)](#) 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 \(see page 1307\)](#).

- **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

Contents

- [Automatic Configuration \(see page 1262\)](#)
- [Run the Java Agent Install Assistant \(see page 1262\)](#)

The Java agent install assistant is a command-line utility that helps you determine the value for the agent parameters string.

The Java agent install assistant also verifies that the JVM runs properly with the agent installed.



Notes:

- The Java agent install assistant is supported on all platforms except IBM WebSphere Application Server 7.0 on AIX 6.1.
- 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

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 XML 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.

The following prerequisite steps must be performed to use the option to configure the Java agent automatically:

Follow these steps:

1. Search for the platform that is based on the platform name and operating system type that is listed in the **JavaAgentInstaller.xml** file.

2. Define the environment variable, if applicable.

The environment variable is embedded in \${}, for example \${JBOSS_HOME}.

3. 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>
```

4. 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 \(see page 1263\)](#)
- [Example 2: Pure Java Agent Using Automatic Configuration \(see page 1264\)](#)
- [Example 3: Native Agent \(see page 1264\)](#)

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 command-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 configuration of JBoss 4.2 pure Java agent on Windows using the automatic configuration option.



Note: Configuring WildFly is same as configuring JBoss EAP 6.2 or JBoss AS 7. To configure WildFly, select the first platform (WildFly, JBoss) and then select the first version (WildFly or JBoss 6.2).

```
java -jar LisaAgent.jar -ia -java -broker tcp://localhost:2009 -name TestAgent66
Enter the path of the Java executable [C:\Program Files\Java\jdk1.7.0
_40\jre\bin\java]:
=====
System verification complete. You can manually add these options to the java command-
line to start the DevTest agent:
    -javaagent:C:\Project\DevTest\main\remote\dist\agent\InsightAgent.
jar=url=tcp://localhost:2009,name=TestAgent66
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
[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] 4.2 [X] Manually Configure
2
Enter path for Java agent configuration file [C:\DevTest\jboss\bin\run.bat]:
Automatic configuration of the Java Agent is complete. Please start your service.
The configuration info has been updated. The previous configuration was:
SET JAVA_TOOL_OPTIONS=-javaagent:C:
\Project\DevTest\main\remote\dist\agent\InsightAgent.jar=url=tcp://localhost:2009,
name=TestAgent7 %JAVA_TOOL_OPTIONS%
The previous configuration has been saved to file 'C:\DevTest\jboss\bin\run.bat.2014-
54-25_09-54-22'
The current agent configuration is:
SET JAVA_TOOL_OPTIONS=-javaagent:C:
\Project\DevTest\main\remote\dist\agent\InsightAgent.jar=url=tcp://localhost:2009,
name=TestAgent66 %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 Installation

This section contains platform-specific information for installing the DevTest Java Agent.

- [IBM WebSphere App Server \(see page 1265\)](#)
- [JBoss \(see page 1266\)](#)
- [Jetty \(see page 1267\)](#)
- [Oracle WebLogic Server \(see page 1268\)](#)
- [SAP NetWeaver \(see page 1269\)](#)
- [webMethods Integration Server \(see page 1270\)](#)

IBM WebSphere App Server

This page applies to 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 \(see page 1261\)](#) 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
```

JBOSS

This page applies to the following versions:

- JBoss Application Server 4.2
- JBoss Application Server 7.3

- JBoss Enterprise Application Platform 6.2
- WildFly 8.2



Note: JBoss EAP 6.2 is built from JBoss AS 7.

You can use the [agent install assistant \(see page 1261\)](#) 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.

Jetty

This page applies to 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 \(see page 1261\)](#) 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 \(\[https://bugs.eclipse.org/bugs/show_bug.cgi?id=425736\]\(https://bugs.eclipse.org/bugs/show_bug.cgi?id=425736\)\)](#).

The workaround is to install Java in a folder that does not contain a space.

This error was fixed in Jetty 9.2.

Oracle WebLogic Server

This page applies to 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 \(see page 1261\)](#) 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.

SAP NetWeaver

This topic applies to 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 \(see page 1261\)](#) 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.

webMethods Integration Server

This page applies to 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 \(see page 1261\)](#) 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##%
```

Wily Introscope Agent

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.

To view agents in the Enterprise Dashboard and the 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"/>
```

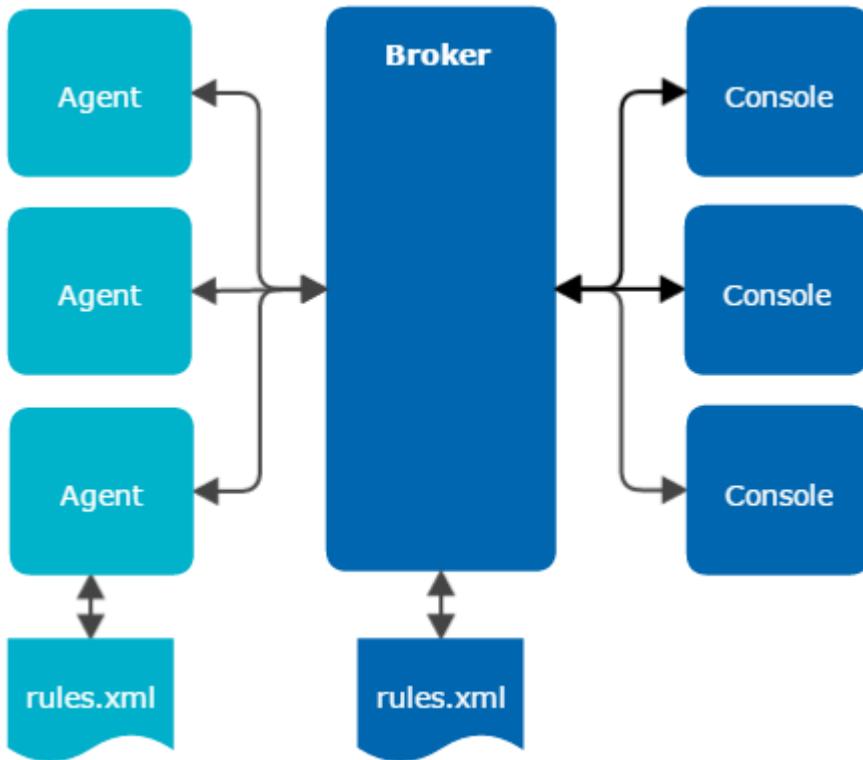
The following XML shows the general format of the **rules.xml** file. The **agent** element contains the properties for an agent. The **broker** element contains the properties for the broker. The **console** element contains the properties for the consoles.

```
<?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.



Note: 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

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 \(see page 1279\)](#)
- **database** directive: [Configure a Database Sink \(see page 1279\)](#)
- **exclude** directive: [Excluding from Interception and Virtualization \(see page 1282\)](#)
- **feature** directive: [Protocol Weight Configuration \(see page 1275\)](#)
- **intercept** directive: [Adding a Method to Intercept \(see page 1281\)](#)
- **loadbalancer** directive: [Load Balancers and Native Web Servers \(see page 1306\)](#)
- **response** directive: [Java Agent Auto-Response Generation \(see page 1284\)](#)
- **track** directive: [Add a Class for Tracking \(see page 1281\)](#)
- **virtualize** directive: [Instrumentation Rules for VSE \(see page 1277\)](#)

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 the DevTest Portal, the setting is saved to the stand-alone agent section of the **rules.xml** file.



Note: For detailed information about the DevTest Portal, see [Using CA Continuous Application Insight \(see page 1007\)](#).

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 \(see page 1017\)](#) 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 \(see page 1276\)](#).

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 CAI 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 allows instances to be easily retrieved from the heap later.

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 \(see page 1276\)](#).

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, CA Continuous 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"/>
```

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

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 \(see page 1276\)](#).

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, CA Continuous 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.**" />
```

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"/>
```

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 \(see page 1276\)](#).
- **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:

<

If a value contains a greater than sign, replace the greater than sign with the following text:

>

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="<init>" args="1099"/>
<respond class="javax.naming.Context" method="lookup" args="EJB3UserControlBean
/remote"/>
<respond class="com.itko.examples.ejb3.EJB3UserControlBeanRemote"/>
```

Java Agent REST APIs

The DevTest Java Agent includes a set of REST APIs that you can use to manage the agent.

When you invoke an API, the response is in JSON format.

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.

The screenshot displays a web-based REST API documentation tool. On the left, under 'API INFO', the base path is listed as `http://[REDACTED]:1505/lisa-pathfinder-invoke` and the version as 'Version: 0.1.1'. Below this are two sections: 'APIS' and 'OBJECTS'. The 'APIS' section lists 'Agents', 'Artifacts', 'CaseRecording', 'NoiseFilter', 'Tickets', and 'Transaction Service'. The 'OBJECTS' section lists 'Agent', 'AgentExclusion', 'AgentInfo', 'AgentJndiInfo', and 'AgentPerf'. The central area, titled 'AGENTS', shows 'Methods for Agents' with four entries: `/api/v1/agents` (GET), `/api/v1/agents/{agentId}` (DELETE), `/api/v1/agents/{agentId}/protocol` (GET), and `/api/v1/agents/{agentId}/performance` (GET). Each entry includes a 'Path' field with the URL, a 'Description' field with a brief description, a 'Method' field with the HTTP method, and a 'Produces' field with the media type. To the right, the 'PLAYGROUND' section shows the URL `/api/v1/agents/{agentId}/performance`. It includes an 'Accept' dropdown set to 'application/vnd.ca.v1+json', a 'Path parameters' section with a 'Submit' button, and a text input field for 'agentId' containing 'agentId'.

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 middle area displays the APIs for the category.

If you select an object, the middle 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 middle 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 \(see page 1288\)](#)
- [Agent Discovery APIs \(see page 1289\)](#)
- [Agent Transaction APIs \(see page 1293\)](#)
- [Agent VSE APIs \(see page 1296\)](#)
- [Agent API Examples \(see page 1299\)](#)

Agent General APIs



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.

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.

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

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

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

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

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

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

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

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

```

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

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

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

```

We can now look at some of the APIs directly off **com.itko.lisa.remote.client.AgentClient**. This list is not exhaustive but 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 in al
l other API calls
 * @param tokens a map of agent guids or names to tokens, null if no agent has token-
enabled security
 * @return a set of IAgentInfo objects representing agents that are currently online
 */
public Set<IAgentInfo> getRemoteAgentInfos(Map<String, String> 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 insta
nce.
 * @return the value returned by the code
 * @throws JMSInvocationException thrown 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 from an
 * d the to dates.
 * @param agentInfo the agent for which to retrieve statistics
 * @param from      how far back to filter
 * @param to        how recently to filter
 * @return          the desired StatsFrames list ordered by decreasing time (starting
 * at the to date)
 */
public List getStatistics(IAgentInfo agentInfo, Date from, Date to);

/**
 * Returns a list of StatsFrames recorded for the specified agent between the from an
 * d 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 res-
 * ults of every 10 secs, etc...
 * @param limit     the maximum number of results
 * @return          the desired StatsFrames list ordered by decreasing time (starting
 * at the to date)
 */
public List getStatistics
(IAgentInfo agentInfo, Date from, Date to, int interval, int limit);

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

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

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

/**
 * Returns the web applications deployed in the specified J2EE container
 * @param info
 * @return
 */

```

```

public WebApplication[] getWebApps(IAgentInfo info) throws JMSInvocationException;

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

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

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

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

/**
 * Returns the class hierarchy under the specified path
 * @param info
 * @param fromPath
 * @return
 */
public ClassNode getClassNodes
(IAgentInfo info, String fromPath) throws JMSInvocationException;

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

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

/**
 * Decompile and return the source to a class
 * @param info
 * @param clazz
 * @param loc decompile on the client or in the agent
 * @return
 * @throws JMSInvocationException
 */
public String getClassSrc

```

```

(IAgentInfo info, String clazz, boolean loc) throws JMSInvocationException, IOException;
}

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

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

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

/**
 * A crude graph representation of an object (recursively computed fields)
 * @param info
 * @param clazz
 * @param hashCode
 * @return
 * @throws JMSInvocationException
 */
public ClassNode getObjectGraph
(IAgentInfo info, String clazz, int hashCode) throws JMSInvocationException;

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

/**
 * Gets a file on the agent filesystem, downloads it to the client in a temp location and return a handle to it
 * @param info
 * @param file
 * @return
 * @throws JMSInvocationException
 */
public File getFile
(IAgentInfo info, String file) throws JMSInvocationException, IOException;

```

Agent Transaction APIs



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.

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*.

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

/**
 * Stop tracking socket usage on the client to use in reconciling client and server t
 * ransactions.
 * This should be called after the call we're adding in addClientTransaction.
 * @param global uninstall on all sockets or only on this thread
 */
public void uninstallSocketTracker(boolean global);

/**
 * As a client, you can invoke this method to root a transaction tree at a new client
 * transaction created using
 * the specified parameters.
 * Note: you must call installSocketTracker before you initiate the client side trans
 * action you're adding here
 * and it is recommended you call uninstallSocketTracker after you're done with the n
 * etwork call.
 * @param stamp      a unique string you can later use to identify the root transactio
 * n

```

```

        *      (in calls to getTransactions for ex.)
        * @param stepInfo a nice human-
readable string that tells what the transaction is doing (can be null)
        * @param args      the parameters the client passes to the transaction (can be empty
)
        * @param result    the result of the transaction (can be null)
        * @param duration  the client's view of the transaction duration
        */
public void addClientTransaction(String  stamp, String stepInfo, Object
[] args, Object result, long duration);

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

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

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

```

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 talking from */
public String getLocalIP();
public void setLocalIP(String ip);

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

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

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

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

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

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

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

/** Signature of the method this frame was recorded in */
public String getSignature();
public void setSignature(String signature);

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

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

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

/** Number of times this frame was duplicated within its parent */
```

```

public long getHits();
public void setHits(long hits);

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

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

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

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

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

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

/** Bitwise or'ed combination of various internal pieces of information - see TransactionFrame.FLAG_XXX */
public long getFlags();
public void setFlags(long flags);

```

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 syntax
    :
    * class regex@annotation regex (e.g. ".*.Remote@.*.Stateless").
    * @param agentInfo
    * @param regex
    * @param impl
    * @return
    */
    public String[] getMatchingClasses
    (IAgentInfo info, String regex, boolean impl) throws JMSInvocationException

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

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

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

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

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

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

```

The interface for the callback APIs is defined by **com.itko.lisa.remote.vse.IVSECallback** and defines the following methods:

```

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

```

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

Finally, the **com.itko.lisa.remote.vse.VSEFrame** object is a POJO with getters and setters for the following properties:

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

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

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

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

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

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

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

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

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

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

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

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

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

Agent API Examples

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 (JMSSInvocationException e) {}  

});

```

Java Agent Extensions

The following pages describe various aspects of creating extensions for the DevTest Java Agent:

- [Extending the Agent \(see page 1300\)](#)
- [Adding Tags to Transaction Frames \(see page 1303\)](#)
- [Extending the Broker \(see page 1304\)](#)
- [Extending Java VSE \(see page 1305\)](#)

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 signature
     , Object[] args, Object ret);

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

    /**
     * Called before method exit to possibly modify the current frame based on intercep
     tor logic
     */
    public boolean postProcess
    (TransactionFrame frame, Object src, String spec, String clazz, String method, String
     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 it and package it in a JAR file with a manifest containing the following entry:

Agent-Extension: extension class name

When you drop this JAR in the 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 as 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 signature, 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, String 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, String signature, Object[] args, Object ret) {
        if (ServiceManager.class.getName().equals(clazz) && "invoke".equals(method)) {
            doCustomLogic((BaseService) args[0], (IData) args[1], false);
        }
        return super.preprocess(frame, src, spec, clazz, method, signature, args, ret);
    }

    public boolean postProcess
    (TransactionFrame frame, Object src, String spec, String clazz, String method, String signature, Object[] args, Object ret) {
        if (ServiceManager.class.getName().equals(clazz) && "invoke".equals(method)) {
            doCustomLogic((BaseService) args[0], (IData) ret, true);
        }
        return super.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 filter by tags in the CAI Console. For more information, see [Using CA Continuous Application Insight \(see page 1007\)](#).

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);
}
```

To deploy a broker extension, compile the extension and package it in a JAR file with a manifest that contains the following entry:

```
Broker-Extension: extension class name
```

When you drop this JAR in the broker (registry) directory, the broker automatically picks it up. If you add the file after the broker (registry) has started, a hot load mechanism helps to ensure that the file is applied. If you update the extension JAR as the broker (registry) is running, the JAR classes are reloaded dynamically. Dynamic reloading makes it easy and fast to test your extension code without restarting the broker (registry).

Example

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 that case, the address and port fields of frames are either left blank or incorrect and we must assemble the frames in an extension:

```
import com.itko.lisa.remote.plumbing.IAssemblyExtension;
import com.itko.lisa.remote.transactions.TransactionFrame;
import com.itko.lisa.remote.utils.Log;
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;
}

private void stitch(TransactionFrame parent, TransactionFrame child) {
    String newFrameId = UUID.randomUUID();
    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);
    }
}

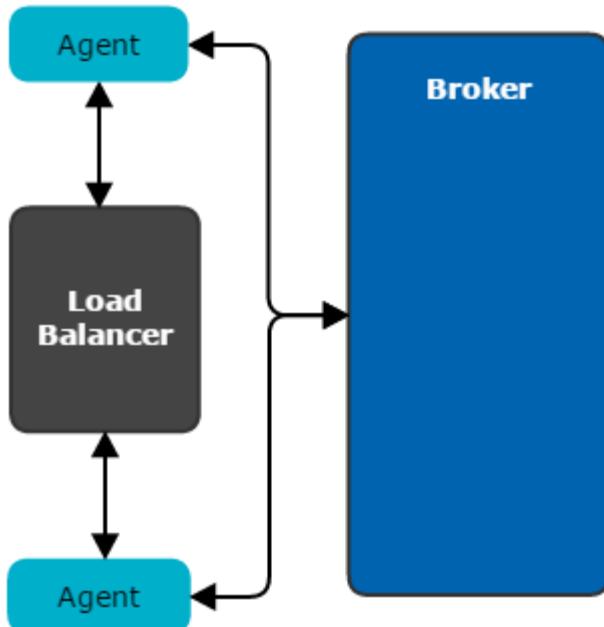
```

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 CAI 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 \(see page 1013\)](#) of the 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

Agent extensions for CA Continuous 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 CA Continuous Application Insight \(see page 1007\)](#).

Follow these steps:

1. Configure a custom keystore by following the instructions in [Using SSL to Secure Communication \(see page 1409\)](#).
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

You can use the following log files to help [troubleshoot \(see page 1310\)](#) 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**. DevTest Workstation has a log file, DevTest Portal has a log file, each simulator has a log file, and VSE has a log file. 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 [Administering \(see page 1362\)](#).

The following configuration properties let you control the logging behavior:

- **Java logging level**
Sets the log level at startup.
- **Maximum log size**
Sets the maximum size of an agent log before it gets rolled over.
- **Maximum log archives**
Sets the maximum number of rolled-over archived logs to keep.
- **Agent log**
Overrides the default location.

You can configure these properties from the [Agents window \(see page 1013\)](#) of the DevTest Portal. The properties appear in the **Settings** tab.

If you need to configure these properties from the **rules.xml** file, the corresponding property names are as follows:

- **lisa.agent.java.logging.level**
- **lisa.agent.log.max.size**
- **lisa.agent.log.max.archives**
- **lisa.agent.agent.log**

Java Agent Troubleshooting



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 [Using the Agent Install Assistant \(see page 1261\)](#).

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 **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.

The 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**.

The 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.

The agent exits with the message "UTF ERROR" ["../../src/solaris/instrument /EncodingSupport_md.c":66]: ..."

The message can vary depending on the exact 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 due to 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 \(see page 1013\)](#) of the 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 \(see page 1307\)](#).

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 CAI 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 \(see page 1013\)](#) of the DevTest Portal. The property appears in the **Settings** tab.

I can see the agent started, but CAI 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 CAI 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 CAI on the agent side.

You can turn off CAI by disabling the **Auto-start** property. Then restart the JVM.

You can configure this property from the [Agents window \(see page 1013\)](#) of the DevTest Portal. The property appears in the **Settings** tab.

Do not turn off CAI on the broker side or Java VSE stops working altogether.

Case 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 CAI 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 the 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.TransactionSizeExceeded`
`Exception: 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 the DevTest Portal and increase the value of the **Maximum graph size** setting. The default value is 100000.

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 the DevTest Portal and increase the value of the **Reply timeout** setting. The default value is 5000.

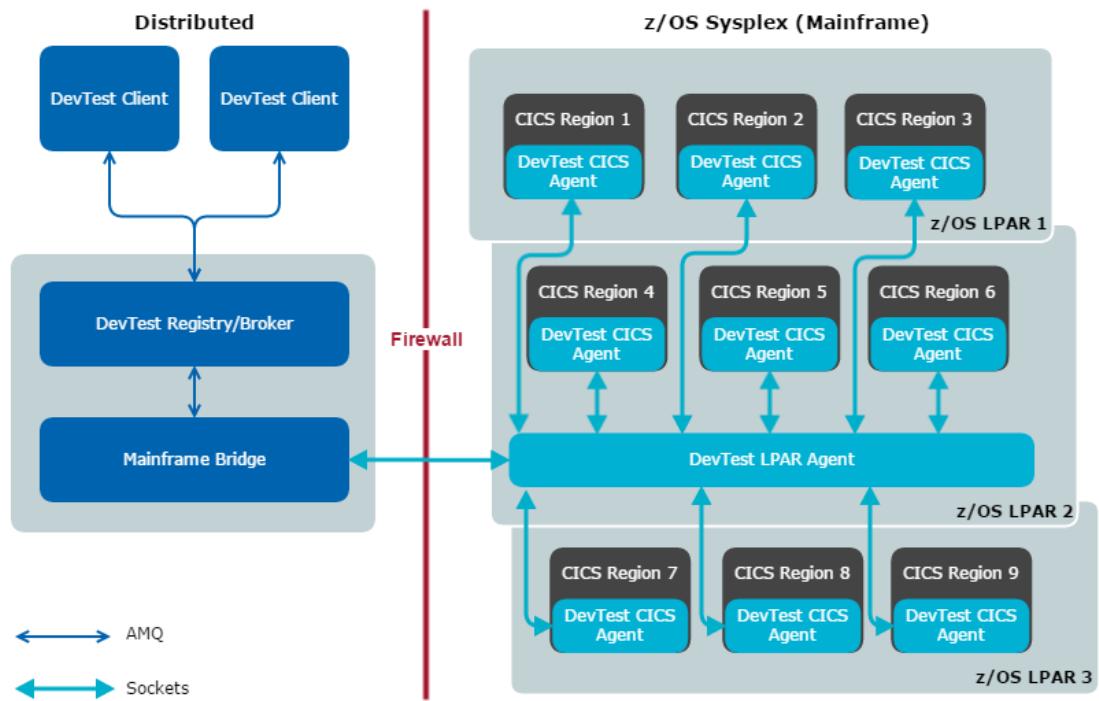
You can also configure this setting in the **rules.xml** file. The corresponding property name is **lisa.agent.virtualize.reply.timeout**. For example:

```
<property key="lisa.agent.virtualize.reply.timeout" value="60000"/>
```

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.



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 \(see page 1013\)](#) of the DevTest Portal. The properties appear in the **Settings** tab.

Mainframe Bridge Properties

The following 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: localhost

- **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

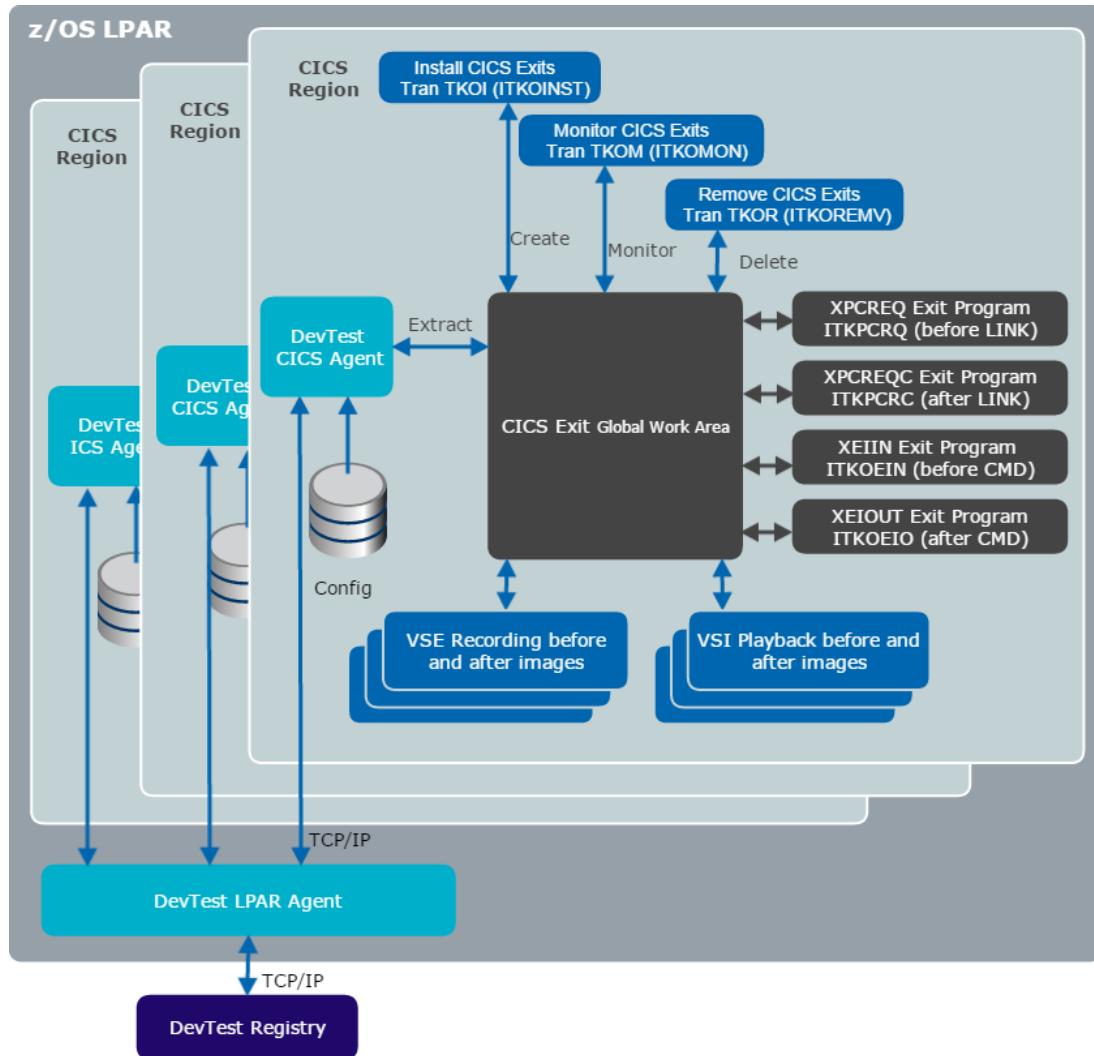
- **lisa.mainframe.bridge.connid**

This property is reserved for future use.

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.



How to Install the CICS Agent



Note: 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
- iTKOCICSAgentLoadCICS32
- iTKOCICSAgentLoadCICS41
- iTKOCICSAgentLoadCICS42

- iTKOCICSAgentLoadCICS51
- iTKOCICSAgentLoadCICS52
- iTKOCICSAgentCntl

The steps for the installation are:

1. FTP the files to the target z/OS system.

Use the load library that matches your CICS version:

- For CICS Version 3.2, use iTKOCICSAgentLoadCICS32
- For CICS Version 4.1, use iTKOCICSAgentLoadCICS41
- For CICS Version 4.2, use iTKOCICSAgentLoadCICS42
- For CICS Version 5.1, use iTKOCICSAgentLoadCICS51
- For CICS Version 5.2, use iTKOCICSAgentLoadCICS52

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 /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. This file is accessed as a Transient Data (TD) Queue during agent startup. This data set has attributes: DSORG=PS, RECFM=FB, LRECL=80.

Lines in the file with an asterisk ('*') in column 1 are ignored. The first record without an asterisk in column 1 contains the IP address and port number of the LPAR Agent. This record must be in the following format:

Bytes 1 to 15	Byte 16	Bytes 17 to 21	Bytes 22 to 80
IP address of LPAR Agent	blank	Port number of LPAR Agent	unused

The member CONFIG of the control PDS contains the following example:

```

* ITKO CICS Agent Configuration
* Columns 1 - 15 contain LPAR Agent IP address (15 chars)
* Columns 17 - 21 contain LPAR Agent port number (5 chars)
192.168.0.100    2998

```

5. Define CICS Agent resources to CICS.

The following table lists the resources that must be defined to CICS:

Nam	Type	Description
e		
TKOC	TDQUE	Configuration Transient Data Queue
	UE	
TKOI	TRANS	Transaction to install CICS exits and start agent transaction (TKOA). Typically ACTIO run during PLTPI processing.
	N	
TKOR	TRANS	Transaction to remove CICS exits and stop agent transaction (TKOA). Typically ACTIO run during PLTSD processing.
	N	
TKOA		Agent transaction. Typically started by TKOI and stopped by TKOR, although it may be started and stopped by the monitor transaction (TKOM).

	TRANS
	ACTIO
	N
<hr/>	
TKO	TRANS Monitor transaction. Used to manually start and stop exits and agent and
M	ACTIO monitor agent processing.
N	
<hr/>	
ITKO	PROGR Agent program. Started by TKOA.
AGN	AM
T	
<hr/>	
ITKOI	PROGR Exit installation program. Started by TKOI.
NST	AM
<hr/>	
ITKO	PROGR Exit removal program. Started by TKOR.
REM	AM
V	
<hr/>	
ITKPC	PROGR CICS XPCREQ (before CICS LINK exit). Enabled by TKOI, disabled by TKOR.
RQ	AM
<hr/>	
ITKPC	PROGR CICS XPCREQC exit (after CICS LINK exit). Enabled by TKOI, disabled by TKOR.
RC	AM
<hr/>	
ITKO	PROGR CICS XEIIN exit (before CICS Command exit) Enabled by TKOI, disabled by TKOR.
XEIN	AM
<hr/>	
ITKO	PROGR CICS XEIOUT exit (after CICS Command exit) Enabled by TKOI, disabled by TKOR.
XEIO	AM
<hr/>	
ITKO	PROGR CICS XPCABND exit (Task Abend exit) Enabled by TKOI, disabled by TKOR.
TABN	AM
<hr/>	
ITKO	PROGR Monitor program.
MON	AM
<hr/>	

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 CA Continuous Application Insight.

XPCREQ and XPCREQC are required. If you do not require DTP virtualization or CAI, you can remove XEIIN and XEIOUT. To remove them, delete ITKOEIN and ITKOEIO from the load library (or rename them).

If you do not require CAI, 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 XEIN) 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

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:

Parameter	Length in Bytes	Input or Output	Data Type	Description
Token	8	Input /Output	Unknown	Eight bytes that may be used by the user exits. Maintained across exit calls.
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	Character	CICS SYSID from CICS ASSIGN SYSID().
Applid	8	Input	Character	CICS APPLID from CICS ASSIGN APPLID().
Jobname	8	Input	Character	CICS job name from CICS INQUIRE SYSTEM JOBNAME().
LPAR	8	Input	Character	LPAR name from CVTSNAME in which this CICS resides.
Sysplex	8	Input	Character	SYSPLEX name from ECVTSPLX in which this CICS resides.

ITKO.CICSAGNT.CNTL(ITKOUEX1) 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	Unknow	Eight bytes that may be used by the user exits.
	n	Maintained across exit calls.

ITKO.CICSAGNT.CNTL(ITKOUEX2) 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 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 named ITKOUEX3 in the DFHRPL concatenation.
- Disable the exit in the PPT.

Parameters are passed in COMMAREA.

The following table describes the parameters:

Paramet er	Description
---------------	-------------

	Leng	Input	Dat	
	th in or a			
	Byte Outp Ty			
	s ut pe			
Token	8	Input Un	Eight bytes that may be used by the user exits. Maintained across exit calls.	
	/Out kn			
	put ow			
	n			
Call Type	1	Input Bin	12 indicates Playback ary 16 indicates Termination	
Image Type	1	Input Bin	4 indicates a Before Image ary 8 indicates an After Image	
COMMA REA Address	4	Input Ad	Address of COMMAREA passed on the application CICS LINK. dre ss	
COMMA REA Length	2	Input Bin	Length of COMMAREA passed on the application CICS LINK. ary	
Calling Program	8	Input Ch	Name of program issuing the CICS LINK command. ara cte r	
Target Program	8	Input Ch	Name of program being LINKed to. ara cte r	
New COMMA REA Address	4	Input Ad	For recording: The address of the new consolidated COMMAREA is placed /Out dre here by this exit. This exit typically GETMAINS the area. put ss	
			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 COMMA REA Length	2	Input Bin	For recording: The length of the new consolidated COMMAREA is placed /Out ary here by this exit. put	
			For Playback: The length of the new consolidated COMMAREA from the VSE response is placed here for this exit to use.	
Return Code	1	Outp ut	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.

EIBRCO	6 DE	Outp ut	Bin ary	The EIBRCODE value to be used when the Return Code is 8 or 12.
EIBRESP	4	Outp ut	Bin ary	The EIBRRESP value to be used when the Return Code is 8 or 12.
EIBRESP	4 2	Outp ut	Bin ary	The EIBRRESP2 value to be used when the Return Code is 8 or 12.

Assembler language sample ITKOUEX3 is in ITKO.CICSAGNT.CNTL(ITKOUEX3).

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 set to 1 in the address indicates the last parameter. The third fullword is the address of a list of parameter lengths.

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 XEIN 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 hexadecimal
- 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 040100
XPCREQ Started XPCREQC Started XEINN Started XEIOUT Started
GWA:00135534 Length:00D8 EYE:ITKOEXITGWA V0800000C040100 Agent ECB:807BF2D0
0000 C9E3D2D6 C5E7C9E3 C7E6C140 E5F0F8F0 F0F0F0C3 F0F4F0F1 F0F00000 00000008
0020 03050000 001F0100 E4E2D9E3 D6D2C5D5 807BF2D0 00000000 00000000 00000000
0040 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0060 00000000 00000000 00000000 1B912000 00000000 00000000 00000000 00000000
0080 00000000 00000000 00000000 00000000 00000000 00000000 1B5F7000 1B91C810
00A0 1B5F8F50 E2F6F6F0 C3C9C3E2 E3E2F4F1 C3C9C3E2 C1404040 E2F0E6F1 40404040
00C0 E2E5E2C3 D7D3C5E7 C4C5E5D7 D3C5E7F1 C7D9D6E4 D7F14040
STATUS 1: EXITS INITIALIZED,RUNNING
STATUS 2: AGENT RUNNING,CONNECTED,PF OFF
RecQ Seq: 00000000 Head: 00000000 Tail: 00000000 Count: 00000000 Max: 00000000
PLbQ Seq: 00000000 Head: 00000000 Tail: 00000000 Count: 00000000 Max: 00000000
LPAQ Head: 00000000 Tail: 00000000 Count: 00000000 Max: 00000000
CTBL:01F4 Token: 00000000 Used:0000 XTBL:01F4 Used:0000 FMHT:01F4 Used:0000
Prog List @ 1B912000 U Use: 00000000 Rec Ct: 00000000 Plb Ct: 00000000
Record Count: 00000000 Playback Count: 00000000 Event Count: 00000000
Agent ENQ held by: Transaction TKOA Task Number 00000041

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

```

Monitor Transaction Display

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=xXXXXXXXXXXXXX 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 9999
- ITKO0108 - iTKO Agent GWA error- Address is xxxxxxxx Length is 9999
- ITKO0109 - Socket func xxxxxxx failed. RETCODE=xXXXXXXXXX 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 nnnnn recorded images

- ITKO0118 - iTKO Agent canceled nnnnn playback images
- ITKO0119 - iTKO Agent canceled nnnnn 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

The CICS XEIN (before CICS command) Exit (module ITKOXEIN) or the CICS XEOUT (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



More Information:

- [DevTest CICS Agent Message Descriptions \(see page 1334\)](#)

DevTest CICS Agent Message Descriptions

Contents

- ITKO0001 - iTKO exit init failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999 (see page 1336)
- ITKO0002 - iTKO exit [XPCREQ | XPCREQC] at xxxxxxxx GWA at xxxxxxxx Len 9999 is enabled and started (see page 1337)
- ITKO0003 - iTKO exit [XPCREQ | XPCREQC] was enabled but not started (see page 1337)
- ITKO0004 - iTKO exit GWA error - Address is xxxxxxxx Length is 9999 (see page 1337)
- ITKO0005 - iTKO exit remove failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999 (see page 1338)
- ITKO0006 - iTKO exit [XPCREQ | XPCREQC] disabled (see page 1338)
- ITKO0007 - GWA missing or too small for [ITKPCRQ | ITKPCRC] Exit (see page 1338)
- ITKO0008 - Program program-name Error EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999 (see page 1339)
- ITKO0009 - Recording Abandoned of calling-program LINK to called-program Tran xxxx Task 9999999 Term xxxx (see page 1339)
- ITKO0010 - [Recorded Image | Playback Image | LPAR Agent list] PLO failed in [ITKPCRQ | ITKPCRC] Exit (see page 1340)
- ITKO0011 - BEFORE/AFTER image creation failed with reason code xxx (see page 1340)
- ITKO0012 - Image update failed with reason code 999 (see page 1341)
- ITKO0013 - PDU Parsing Error in ITKPCRC Exit (see page 1342)
- ITKO0014 - ERROR - Corrupt token in ITKPCRC Exit (see page 1342)
- ITKO0101 - AGENT waiting for exits (see page 1342)
- ITKO0102 - AGENT initialization complete (see page 1342)
- ITKO0103 - AGENT shutting down (see page 1343)
- ITKO0104 - iTKO [Recorded Image | Playback Image | LPAR Agent Img] PLO failed in Agent (see page 1343)
- ITKO0105 - iTKO Agent started while another agent is running (see page 1343)
- ITKO0106 - iTKO Agent connected to LPAR Agent (see page 1344)
- ITKO0107 - Agent Config LPAR Agent IP: nnn.nnn.nnn.nnn Port 99999 (see page 1344)
- ITKO0108 - iTKO Agent GWA error- Address is xxxxxxxx Length is 9999 (see page 1344)
- ITKO0109 - Socket func [INITAPI | SELECTEX | SEND | RECV | CONNECT | CLOSE | PTON | TERMAPI] failed. RETCODE=xxxxxxxxx ERRNO=999999 (see page 1345)
- ITKO0110 - Agent CICS cmd failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999 (see page 1345)
- ITKO0111 - iTKO Agent disconnected from LPAR Agent (see page 1346)
- ITKO0112 - iTKO Agent error sending recorded image to LPAR Agent (see page 1346)
- ITKO0113 - iTKO Agent error sending playback image to LPAR Agent (see page 1346)
- ITKO0114 - iTKO Agent error sending response to LPAR Agent (see page 1347)
- ITKO0115 - iTKO Agent error receiving request from LPAR Agent (see page 1347)
- ITKO0116 - iTKO Agent error - invalid LPAR Agent IP address (see page 1347)
- ITKO0117 - iTKO Agent released nnnnnn recorded images (see page 1348)
- ITKO0118 - iTKO Agent canceled nnnnnn playback images (see page 1348)
- ITKO0119 - iTKO Agent canceled nnnnnn pending playback images (see page 1348)
- ITKO0120 - iTKO Agent released playback image with token xxxxxxxxxxxxxxxx (see page 1349)
- ITKO0121 - iTKO Agent error reading configuration (see page 1350)
- ITKO0122 - Agent TCP/IP API Timeout (see page 1350)

- ITKO0123 - Agent waiting for TCP/IP API (see page 1350)
- ITKO0124 - ITKO Agent response timeout for Token xxxxxxxxxxxxxxxx (see page 1351)
- ITKO0125 - Call to ITKOUEX1/ITKOUEX2/ITKOUEX3 failed (see page 1351)
- ITKO0126 - Handshake error encountered. Shutting down Agent. (see page 1351)
- ITKO200 - ITKOXEIN XPI INQ_APPLICATION_DATA Failed (see page 1351)
- ITKO201 - ITKOXEIN/ITKOXEIO XPI Getmain Dynamic Stg Failed (see page 1352)
- ITKO202 - ITKOXEIN/ITKOXEIO XPI Freemain Dynamic Stg Failed (see page 1352)
- ITKO203 - ITKOXEIN/ITKOXEIO XPI Getmain Image Stg Failed (see page 1352)
- ITKO204 - ITKOXEIN/ITKOXEIO XPI Freemain Image Stg Failed (see page 1353)
- ITKO205 - ITKOXEIN Task table full - discarding image (see page 1353)
- ITKO206 - ITKOXEIN XPI WAIT Failed (see page 1353)
- ITKO207 - ITKOXEIN XPI Freemain plbk req/rsp failed (see page 1353)
- ITKO208 - ITKOXEIO Add to recording queue failed (see page 1354)
- ITKO209 - ITKOXEIN/ITKOXEIO XPI Freemain SET() Stg Failed (see page 1354)
- ITKO210 - ITKOXEIN XPI Getmain SET() Stg Failed (see page 1354)

ITKO0001 - iTKO exit init failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXXX EIBRESP=9999
EIBRESP2=9999

Message Type:

Fatal Error

Description:

A CICS command failed and ITKOINST was aborted. The hexadecimal 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 xxxxxxx 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.

ITK00005 - 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 hexadecimal 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).

ITK00006 - iTKO exit [XPCREQ | XPCREQC] disabled

Message Type:

Informational

Description:

The indicated exit (XPCREQ or XPCREQC) was disabled.

Action:

None

ITK00007 - 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.

ITK00008 - Program program-name Error EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXXX
EIBRESP=9999 EIBRESP2=9999

Message Type:

Fatal Error

Description:

A CICS command failed in the indicated exit program. The hexadecimal 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.

ITK00009 - 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 be processed by the agent. 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.

[ITK00012 - 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.

ITK00013 - 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.

ITK00014 - 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.

ITK00101 - 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.

ITK00102 - AGENT initialization complete

Message Type:

Informational

Description:

The DevTest CICS Agent has completed initialization and starts connecting to the DevTest LPAR Agent.

Action:

None.

[ITK00103 - AGENT shutting down](#)

Message Type:

Informational

Description:

The DevTest CICS Agent is shutting down.

Action:

None.

[ITK00104 - 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'E000000000000', EIBRESP=16 and EIBRESP2=500).

Action:

Report the error to CA support.

[ITK00105 - 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.

[ITK00106 - 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.

[ITK00107 - 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

[ITK00108 - 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.

ITK00109 - Socket func [INITAPI | SELECTEX | SEND | RECV | CONNECT | CLOSE | PTON | TERMAPI] failed. RETCODE=xXXXXXXXXX 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 ITK00111 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.

ITK00110 - Agent CICS cmd failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999

Message Type:

Fatal Error

Description:

A CICS command failed in the DevTest CICS Agent. The hexadecimal 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).

[ITK00111 - iTKO Agent disconnected from LPAR Agent](#)

Message Type:

Informational

Description:

The DevTest CICS Agent has disconnected from the DevTest LPAR Agent.

Action:

None.

[ITK00112 - 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.

[ITK00113 - 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.

[ITK00114 - 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.

[ITK00115 - 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.

[ITK00116 - 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.

ITK00117 - iTKO Agent released nnnnn 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.

ITK00118 - iTKO Agent canceled nnnnn 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.

ITK00119 - iTKO Agent canceled nnnnn 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.

ITK00120 - iTKO Agent released playback image with token xxxxxxxxxxxxxxxxx**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 ITK00124 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.

[ITK00121 - 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.

[ITK00122 - 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.

[ITK00123 - 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.

[ITK00124 - 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.

[ITK00125 - 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.

[ITK00126 - 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.

[ITKO200 - ITK0XEIN XPI INQ_APPLICATION_DATA Failed](#)

Message Type:

Error

Description:

The XEIN exit (module ITK0XEIN) received an error in the XPI INQ_APPLICATION_DATA command.

Action:

Report the error to CA support.

ITK0201 - ITK0XEIN/ITK0XEIO XPI Getmain Dynamic Stg Failed

Message Type:

Error

Description:

The XEIN exit (module ITK0XEIN) or XEOUT exit (module ITK0XEIO) received an error in the XPI GETMAIN command while allocating dynamic storage.

Action:

Investigate the CICS short on storage condition.

ITK0202 - ITK0XEIN/ITK0XEIO XPI Freemain Dynamic Stg Failed

Message Type:

Error

Description:

The XEIN exit (module ITK0XEIN) or XEOUT exit (module ITK0XEIO) received an error in the XPI FREEMAIN command while releasing dynamic storage.

Action:

Report the error to CA support.

ITK0203 - ITK0XEIN/ITK0XEIO XPI Getmain Image Stg Failed

Message Type:

Error

Description:

The XEIN exit (module ITK0XEIN) or XEOUT exit (module ITK0XEIO) received an error in the XPI GETMAIN command while allocating storage for a recorded image.

Action:

Investigate the CICS short on storage condition.

ITK0204 - ITKOXEIN/ITKOXEIO XPI Freemain Image Stg Failed

Message Type:

Error

Description:

The XEIN exit (module ITKOXEIN) or XEOUT exit (module ITKOXEIO) received an error in the XPI FREEMAIN command while releasing image storage.

Action:

Report the error to CA support.

ITK0205 - ITKOXEIN Task table full - discarding image

Message Type:

Error

Description:

The XEIN exit (module ITKOXEIN) tried to allocate an entry in the task table and the table was full.

Action:

Report the error to CA support.

ITK0206 - ITKOXEIN XPI WAIT Failed

Message Type:

Error

Description:

The XEIN exit (module ITKOXEIN) received an error in the XPI WAIT command while waiting for a playback response.

Action:

Report the error to CA support.

ITK0207 - ITKOXEIN XPI Freemain plbk req/rsp failed

Message Type:

Error

Description:

The XEIN exit (module ITK0XEIN) received an error in the XPI FREEMAIN command while releasing a playback request or response.

Action:

Investigate the CICS short on storage condition.

ITK0208 - ITK0XEIO Add to recording queue failed

Message Type:

Error

Description:

The XEOUT exit (module ITK0XEIO) tried to add a recorded image to the recording queue and the attempt failed.

Action:

Report the error to CA support.

ITK0209 - ITK0XEIN/ITK0XEIO XPI Freemain SET() Stg Failed

Message Type:

Error

Description:

The XEIN exit (module ITK0XEIN) or XEOUT exit (module ITK0XEIO) 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.

ITK0210 - ITK0XEIN XPI Getmain SET() Stg Failed

Message Type:

Error

Description:

The XEIN exit (module ITK0XEIN) received an error in the XPI GETMAIN command while allocating storage for the SET parameter in a RECEIVE or CONVERSE command.

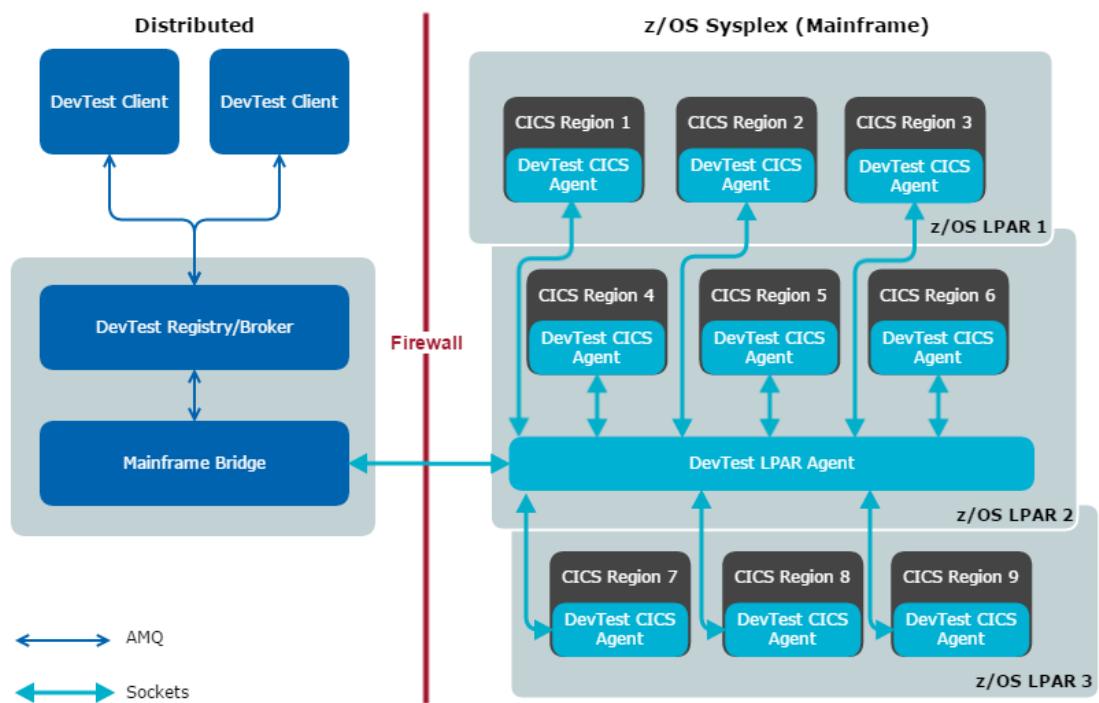
Action:

Report the error to CA support.

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 also 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.



How to Install the LPAR Agent



Note: 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:

- iTKOLPARGentDoc.pdf
- iTKOLPARGentLoad
- iTKOLPARGentCtl

The steps for the installation are as follows:

1. FTP the files to the target system.

The files iTKOLPARGentLoad and iTKOLPARGentCtl 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. The z/OS file names can be changed 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 iTKOLPARGentLoad '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 iTKOLPARGentCtl '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
+ITK09003 - 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
+ITK09003 - Command accepted
+Tracing for ID 3 is ON
F LPARAGNT,T 3 OFF
+ITK09003 - 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):

```
2014:10:09 19:05:38 - Starting via operator command.
2014:10:09 19:05:38 - LPAR Agent started. Version: V800 Compiled: Feb 7 2014 13:08:00
2014:10:09 19:05:38 - Using configuration file: //DD:CONFIG
2014:10:09 19:05:38 - ---: Name = CA DevTest LPAR Agent
2014:10:09 19:05:38 - ---: InitialTrace = False
2014:10:09 19:05:38 - ---: Mode = Server
2014:10:09 19:05:38 - ---: PortClient = 4997
2014:10:09 19:05:38 - ---: PortAgent = 3998
2014:10:09 19:05:38 - ---: BufferSize = 65536
2014:10:09 19:05:38 - ---: MaxBufferSize = 65536
2014:10:09 19:05:38 - ---: AppKASeconds = 60
2014:10:09 19:05:38 - ---: AppKALimit = 5
2014:10:09 19:05:38 - Configuration file processing successful.
2014:10:09 19:05:38 - LPAR Agent named CA DevTest LPAR Agent
2014:10:09 19:05:38 - Agent socket ready.
2014:10:09 19:05:38 - Client socket ready.
2014:10:09 19:05:38 - Memory for message buffers allocated.
2014:10:09 19:05:38 - Known interfaces: 192.86.32.105
2014:10:09 19:05:38 - Initial configuration complete.
2014:10:09 19:06:45 - ID: 2 - Accepted agent connection from 192.168.0.100.
2014:10:09 19:06:48 - ID: 3 - Accepted agent connection from 192.168.0.100.
2014:10:09 19:06:48 - ID: 4 - Accepted client connection from 192.168.0.101.
```

Administering

This section provides administrators with procedures to help them perform regular administrative duties in DevTest Solutions.

- [General Administration \(see page 1362\)](#)
- [Database Administration \(see page 1385\)](#)
- [License Administration \(see page 1396\)](#)
- [Security \(see page 1408\)](#)
- [Logging \(see page 1448\)](#)
- [Monitoring \(see page 1452\)](#)

General Administration

This section contains the following pages:

- [Default Port Numbers \(see page 1362\)](#)
- [Shared Installation Type \(see page 1365\)](#)
- [Directory Structure \(see page 1366\)](#)
- [Running Server Components as Services \(see page 1370\)](#)
- [Running DevTest Solutions with Ant and JUnit \(see page 1371\)](#)
- [Memory Settings \(see page 1378\)](#)
- [Third-Party File Requirements \(see page 1380\)](#)
- [Enabling Additional Scripting Languages \(see page 1385\)](#)

Default Port Numbers

This page defines each default port number that the following DevTest components use:

- [DevTest Server \(see page \)](#)
- [DevTest Workstation \(see page \)](#)
- [Demo Server \(see page 1364\)](#)

This page also describes how to change the port numbers for the DevTest Portal and the DevTest Console.



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
1528	Derby database	lisadb.internal.port, lisadb.pool.common.url
1529	Demo server	Not applicable
1530	Enterprise Dashboard Derby db 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
8443	Embedded web server (if SSL is enabled)	lisa.webserver.ssl.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** needs to 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 **lisa.enterprisedashboard.service.url**, 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	<code>lisa.webserver.port</code>
2004	VSE Manager	<code>lisa.net.9.port</code>
2005	Test Runner	<code>lisa.net.5.port</code>
2006	ServiceManager	<code>lisa.net.11.port</code>
2009	Broker	<code>lisa.pathfinder.broker.port</code>
2999	JDBC simulation driver	Not applicable
3128	HTTP proxy	<code>laf.httpproxy.port</code>
8443	Embedded web server (if SSL is enabled)	<code>lisa.webserver.ssl.port</code>

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 Numbers for the DevTest Portal and DevTest Console

To change the port number of the DevTest Portal or the DevTest Console, 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. To change the port number of the DevTest Portal, add the following property and set the value to the new port number.

```
# DevTest UI
devtest.port=port-number
```
5. To change the port number of the DevTest Console, add the following property and set the value to the new port number.

```
# Our embedded web server
lisa.webserver.port=port-number
```
6. Save the **local.properties** file.
7. If the **LISA_HOME** directory does not contain a **phoenix.properties** file, copy **_phoenix.properties** to **phoenix.properties**.
8. Open the **phoenix.properties** file.
9. To change the port number of the DevTest Portal, uncomment the **phoenix.port** property and set the value to the new port number.
10. To change the port number of the DevTest Console, uncomment the **registry.portal.port** property and set the value to the new port number.
11. Save the **phoenix.properties** file.

Shared Installation Type

The shared installation type is designed for environments that have the following characteristics:

- Multiple users share an installation of DevTest from multiple computers.
- Each user has separate data.

In a shared installation, all data and temporary files are stored in user-specified directories. Each user has their own data, but they share a common DevTest installation. With a shared installation, users only need read access to the DevTest programs directory.



Note: If you want to install Component Services and you are an Administrator, use the local install.

If you select the shared installation type in the installer, the installer prompts you to specify the data directory and the temporary files directory. The default location of each directory is the **USER_HOME** directory.

In a shared installation, the **lisa.user.properties** file is added to the **LISA_HOME** directory and the **USER_HOME** directory for the user that is running the installation. This file contains the **lisa.data.dir** and **lisa.tmpdir** properties. You can specify the location of the file by setting the **lisa.user.properties** system property or the **LISA_USER_PROPERTIES** environment variable. If either of the properties have been defined as a system property, the system property definition takes precedence.



Note: Although **lisa.tmpdir** allows you to change the location of where you can store your temporary files, CA does 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 may instruct you to go back to using a local disk for temp file storage for continued support of your environment.

New users on different computers must manually set up their own **lisa.user.properties** file in their **USER_HOME** directory before they attempt to access a shared installation. You can copy the file from the **LISA_HOME** directory, or you can create it manually if you want to use different directories.

If you attempt to start any DevTest component before setting up this property file, the components will not start. An error message appears on the screen and in the log files indicating an error reading the **lisa.user.properties** file.

Directory Structure

Contents

- [DevTest Workstation Directories \(see page 1367\)](#)
- [DevTest Server Directories \(see page 1368\)](#)

This page describes the directory structure of DevTest Workstation and DevTest Server.

This page assumes that the local installation type was selected during the installation of DevTest Solutions. If the [shared installation type \(see page 1365\)](#) was selected, the following directories can be in a location other than **LISA_HOME**:

- cvsMonitors
- database
- hotDeploy
- locks
- tmp

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.
- **database**
Contains the DDL files for various databases.
- **defaults**
Contains the audit document and staging documents common across all projects. Project-specific staging documents are located in the Projects directory.
- **doc**
Contains the license agreement.
- **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 in-container 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 inter-process concurrency.
- **Projects**
Contains three 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.

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 CAI upgrade scripts for various databases.

- **defaults**

Contains the audit document and staging documents 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 in-container 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 inter-process concurrency.

- **Projects**

Contains three 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:

```
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
```

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 Solutions 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 leverage 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/>.

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> (<http://junit.org>/).

- The [Execute JUnit Test Case/Suite step \(see page 1789\)](#) can execute a JUnit test. [DevTest Ant Tasks \(see page 1372\)](#) 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. Make sure both Ant and JUnit are available on your PC and ANT_HOME\bin is set in your PATH.
2. Copy **junit.jar** from your JUnit installation into ANT_HOME\lib.
3. Define the system property **LISA_HOME** and set its value to the 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

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**
Enables you to write out the name and description of each test case, without executing the test cases.
Example: preview="true"
- **user**
Specifies the user name for ACL.
Example: user="admin"
- **password**
Specifies 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. It 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 in most cases, although it defaults to the current working directory.

Ant and JUnit Usage Examples

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 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

Contents

- [JUnit3 Test Cases \(see page 1376\)](#)
- [JUnit3 Test Suites \(see page 1376\)](#)
- [JUnit 4 Test Cases \(see page 1376\)](#)

- [JUnit4 Test Suites \(see page 1377\)](#)

JUnit3 Test Cases

For JUnit3 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);
    }
}
```

JUnit3 Test Suites

For JUnit3 test suites, your suite is not required to extend the `junit.framework.TestSuite`. However, it must implement the `suite()` method, with test cases wrapped by the `JUnit4TestAdapter`.

For example:

```
import junit.framework.JUnit4TestAdapter;
import junit.framework.TestSuite;

public class JUnit3VanillaTestSuite {

    public static TestSuite suite() {
        TestSuite suite = new TestSuite();
        suite.addTest ( new JUnit4TestAdapter ( MyJUnit3TestCase.class ) );
        return suite;
    }
}
```

JUnit 4 Test Cases

For JUnit4 test cases, the test methods must have the "`@org.junit.Test`" annotation on the methods, as required by JUnit4.

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); }
}
```

```
public void twoIsThree() { assertEquals (2, 3); }
}
```

JUnit4 Test Suites

To implement a JUnit4 test suite, add the @RunWith and @Suite.SuiteClasses annotations to flag the class as a test suite.

For example:

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses ( { 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 \(see page 1375\)](#), `${testReportDir}` is defined as `${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 \(see page 1375\)](#), 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

Contents

- [Change the Java heap size on Windows and UNIX \(see page 1378\)](#)
- [Change the Java heap size on Mac OS X \(see page 1379\)](#)
- [Adjust the memory for individual processes \(see page 1379\)](#)

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 leverage more memory when needed.

Change Memory Allocation

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 are used to customize the memory allocation settings for each of the DevTest processes used in the server. When you install the DevTest Server, the LISA_HOME\bin folder is populated with a **.vmoptions** file with the same name as each executable file.

A full list of JVM parameters are detailed in the Oracle website under Configuring the Default JVM and Java Arguments.

In CAI, 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

1. Open the target file with a **.vmoptions** extension.

For example, open **RegistryService.vmoptions**, with the following content:

```
# Enter one VM parameter per line
# For example, to adjust the maximum memory usage to 512 MB, uncomment the following 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 Mac 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 only apply to DevTest Workstation.

Adjust the memory for individual processes

You can adjust memory for the following programs:

- Broker
- BrokerService
- CoordinatorServer
- CoordinatorService
- Registry
- RegistryService
- Simulator
- SimulatorService
- VirtualServiceEnvironment
- VirtualServiceEnvironmentService
- Workstation

To tweak the memory for every process, you can edit the individual script files. For example, you can allow the registry to have 128M but the simulator to have 1024M.

If you reinstall DevTest Solutions, these edits are overwritten by the installer.

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. 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.commonservices.jar;C:\Program Files\IBM\MQSeries\Java\lib\com.ibm.mq.headers.jar;C:\Program Files\IBM\MQSeries\Java\lib\com.ibm.mq.jar;C:\Program Files\IBM\MQSeries\Java\lib\com.ibm.mq.jmqi.jar;C:\Program Files\IBM\MQSeries\Java\lib\com.ibm.mq.pcf.jar;C:\Program Files\IBM\MQSeries\Java\lib\com.ibm.mqjms.jar;C:\Program Files\IBM\MQSeries\Java\lib\connector.jar;C:\Program Files\IBM\MQSeries\Java\lib\dhbcore.jar"
```

CA does not provide the third-party files listed in this page.



Note: This page assumes that the local installation type was selected during the installation of DevTest Solutions. If the [shared installation type \(see page 1365\)](#) was selected, the **hotDeploy** directory can be in a location other than **LISA_HOME**.

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's 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.

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 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 TIBCO 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:

- 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 need to 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 WebSphere MQ 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 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

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

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

The JAR files are available in the **MQ_HOME\java\lib** directory.

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

This section contains the following pages:

- [Internal Database Configuration \(see page 1385\)](#)
- [External Registry Database Configuration \(see page 1386\)](#)
- [External Enterprise Dashboard Database Configuration \(see page 1393\)](#)
- [Database Maintenance \(see page 1394\)](#)

Internal Database Configuration

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. This out of the box 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
- VSE
- Enterprise Dashboard

User your site-specific procedures to back up the enterprise databases you use for DevTest Solutions.



Note: For information about database system requirements, see [Installing \(see page 50\)](#).

External Registry Database Configuration

You can configure DevTest 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 \(see page 50\)](#).

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 general steps are as follows:

1. Locate the `_site.properties` file in the **LISA_HOME** directory and change the name to `site.properties`.
2. Modify the new `site.properties` file, uncommenting the properties to match the target database.
3. When configuring to a DevTest supported external database, such as Oracle, DB2, MySQL, or SQL Server:

- a. Uncomment the following properties for that external database in the appropriate section.
For example:
lisadb.pool.common.driverClass=oracle.jdbc.driver.OracleDriver
lisadb.pool.common.url
lisadb.pool.common.url=jdbc:oracle:thin:@HOST:1521:SID
- b. Update the **lisadb.pool.common.url** property with the appropriate hostname, port, and SID values.
- c. Update the following user and password properties with the correct values for accessing the database:
lisadb.pool.common.user
lisadb.pool.common.password
- d. Comment out the following two properties for the Derby database:
#lisadb.pool.common.driverClass=org.apache.derby.jdbc.ClientDriver
#lisadb.pool.common.url=jdbc://localhost:1528/database/lisa.db;create=true
- e. Set the following property to false:
lisadb.internal.enabled=false

4. Start the registry or DevTest Workstation.

Configure DevTest to Use DB2

This page describes how to configure DevTest to use an IBM DB2 database.

In the following procedure, you configure the DB2 version of the **site.properties** file. The following properties in this file are relevant to the database configuration:

```

lisadb.reporting.poolName=common
lisadb.acl.poolName=common
lisadb.broker.poolName=common

lisadb.pool.common.driverClass=com.ibm.db2.jcc.DB2Driver
lisadb.pool.common.url=jdbc:db2://HOSTNAME:PORT/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
  
```

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. Set the **lisadb.internal.enabled** property to false.
9. Start the registry.

Configure DevTest to Use MySQL

This page describes how to configure DevTest to use a MySQL database.

In the following procedure, you configure the MySQL version of the **site.properties** file. The following properties in this file are relevant to the database configuration:

```
lisadb.reporting.poolName=common
lisadb.acl.poolName=common
lisadb.broker.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.

In addition, you must set the **lower_case_table_names** system variable to the value 1.

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. Set the **lisadb.internal.enabled** property to false.
7. Add the MySQL JDBC driver to the **LISA_HOME\lib\shared** directory. The minimum version of the MySQL JDBC driver is 5.1.25.
8. Start the registry.

Configure DevTest to Use Oracle

This page describes how to configure DevTest to use an Oracle database.

In the following procedure, you configure the Oracle version of the **site.properties** file. The following properties in this file are relevant to the database configuration:

```
lisadb.reporting.poolName=common
lisadb.acl.poolName=common
lisadb.broker.poolName=common

lisadb.pool.common.driverClass=oracle.jdbc.driver.OracleDriver
lisadb.pool.common.user=oracle_username
lisadb.pool.common.password=oracle_password
## Select one of the two connection URLs depending on usage of SID or SERVICE:
lisadb.pool.dradis.url=jdbc:oracle:thin:@[HOST]:1521:[SID]
lisadb.pool.dradis.url=jdbc:oracle:thin:@/[HOST][:PORT]/SERVICE

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.
7. Set the **lisadb.internal.enabled** property to false.
8. 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**.
9. Start the registry.

Configure DevTest to Use SQL Server

Contents

This page describes how to configure DevTest to use a Microsoft SQL Server database.

In the following procedure, you configure the SQL Server version of the **site.properties** file. The following properties in this file are relevant to the database configuration:

```
lisadb.reporting.poolName=common
lisadb.acl.poolName=common
lisadb.broker.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
```

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 additional 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. Set the **lisadb.internal.enabled** property to false.
7. Start the registry.

External Enterprise Dashboard Database Configuration

The Enterprise Dashboard uses an internal Derby database by default. We recommend that you use an external database instead.

For an Oracle database, the following prerequisites apply:

- Create a tablespace.
- Create a new user.
- Associate the new user with the tablespace.
- Grant dba privileges to the user.

For a MySQL database, you must set the **lower_case_table_names** system variable to the value 1.

For an IBM DB2 database, you must set the page size to 16 KB.

Follow these steps:

1. Log on to the server where the Enterprise Dashboard is installed.
2. Navigate to **LISA_HOME** and locate the **_local.properties** file.
3. Change the file name to **local.properties**.
4. Open the **local.properties** file.
5. Locate the following property in the Enterprise Dashboard Options section. This property specifies whether to use the internal Derby database in the Enterprise Dashboard.

```
dradisdb.internal.enabled=true
```
6. Change the value to **false**.

```
dradisdb.internal.enabled=false
```
7. Perform one of the following actions:

- To use an **Oracle** database, edit the following properties to set values to an external database similar to the following example:

```
lisadb.pool.dradis.driverClass=oracle.jdbc.driver.OracleDriver
dradis.user=oracle_username
lisadb.pool.dradis.password=oracle_password
## Select one of the two connection URLs depending on usage of SID or SERVICE:
lisadb.pool.dradis.url=jdbc:oracle:thin:@[HOST]:1521:[SID]
lisadb.pool.dradis.url=jdbc:oracle:thin:@//[HOST][:PORT]/SERVICE
```

- To use a **SQL Server** database, edit the following properties to set values to an external database similar to the following example:

```
lisadb.pool.dradis.driverClass=com.microsoft.sqlserver.jdbc.SQLServerDriver
lisadb.pool.dradis.url=jdbc:sqlserver://[_HOST];databaseName=[DATABASENAME]
```

```
lisadb.pool.dradis.user=sqlserver_username
lisadb.pool.dradis.password=sqlserver_password
```

- To use a **MySQL** database, edit the following properties to set values to an external database similar to the following example:

```
lisadb.pool.dradis.driverClass=com.mysql.jdbc.Driver
lisadb.pool.dradis.url=jdbc:mysql://DBHOST:DBPORT/DBNAME
lisadb.pool.dradis.user=mysql_username
lisadb.pool.dradis.password=mysql_password
```

- Save the **local.properties** file.

Database Maintenance

Automatic Reporting Maintenance

Two processes that run in the background 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.what rpt.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 \(see page 1438\)](#) in the DevTest 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. If you want to change the default value, add the property to the **local.properties** file.

Automatic Deletion of Transactions

You can configure CA Continuous Application Insight to delete old transactions automatically from the DevTest 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 \(see page 1013\)](#) of the DevTest Portal. The properties appear in the **Settings** tab.

Automatic Deletion of Tickets

You can configure CA Continuous Application Insight to delete old [tickets \(see page 1066\)](#) automatically from the DevTest 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 \(see page 1013\)](#) of the DevTest Portal. The properties appear in the **Settings** tab.

License Administration

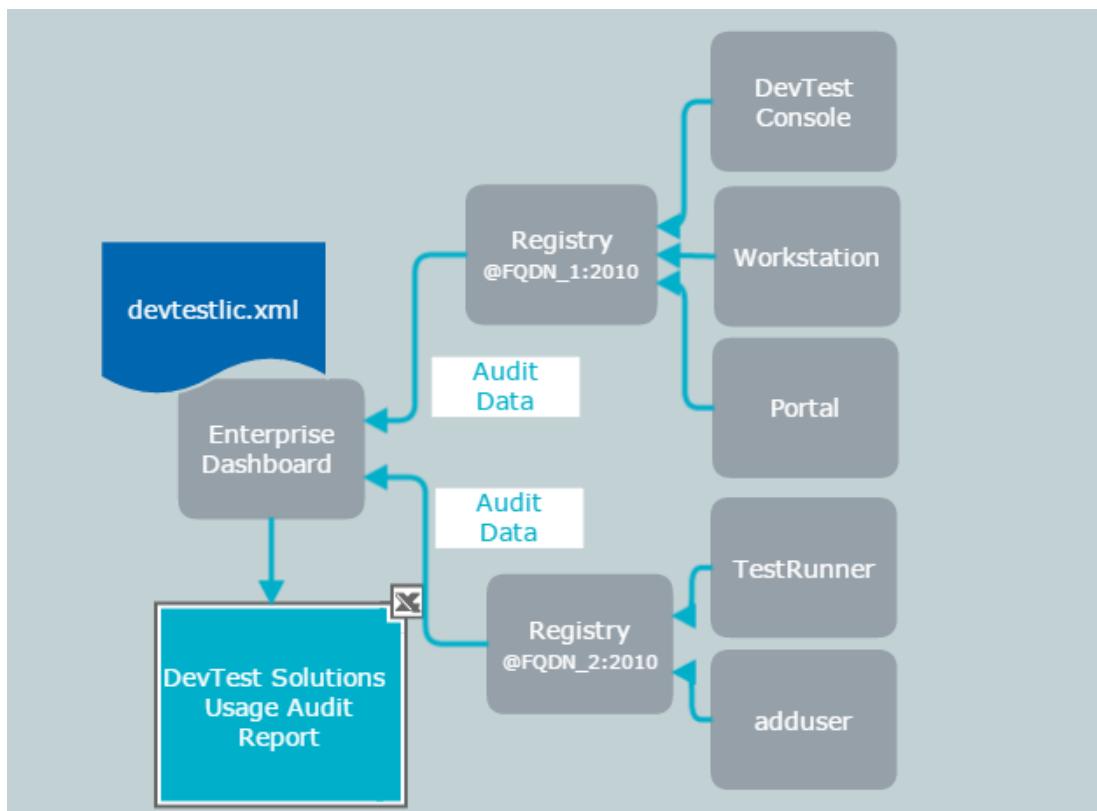
Licensing overview

- The DevTest 8.0 and later license agreement is based on the maximum allowed concurrent user sessions by user type. Contact your account team if you have any questions about your specific licensing agreement.
- The license is file-based, where there is one file for the enterprise. This file activates DevTest Solutions at the initial startup following installation.

- The Local License Server (LLS) and Internet Based License Server are no longer supported for DevTest Solutions 8.0.
- Concurrent usage data by user type is automatically collected.
- The Enterprise Dashboard produces licensing reporting.
- A Usage Audit Report, which reports maximum concurrent usage by user type, is a tool that helps you assess compliance with the license agreement. Users with administrative privileges can access this report.

For more information, see [How Licensing, ACLs, and Audit Reports Work Together \(see page 1399\)](#).

The following diagram shows the activation of DevTest Solutions by a license file that is stored with the Enterprise Dashboard. The registries collect audit data from the UIs and CLIs that users log in to, including the DevTest Console, the Workstation, the Portal, and command-line utilities such as TestRunner and adduser. The registries forward the audit data to the Enterprise Dashboard. Administrators can generate a DevTest Solutions Usage Audit Report to verify compliance with the license agreement.



License Server to Client Data Center

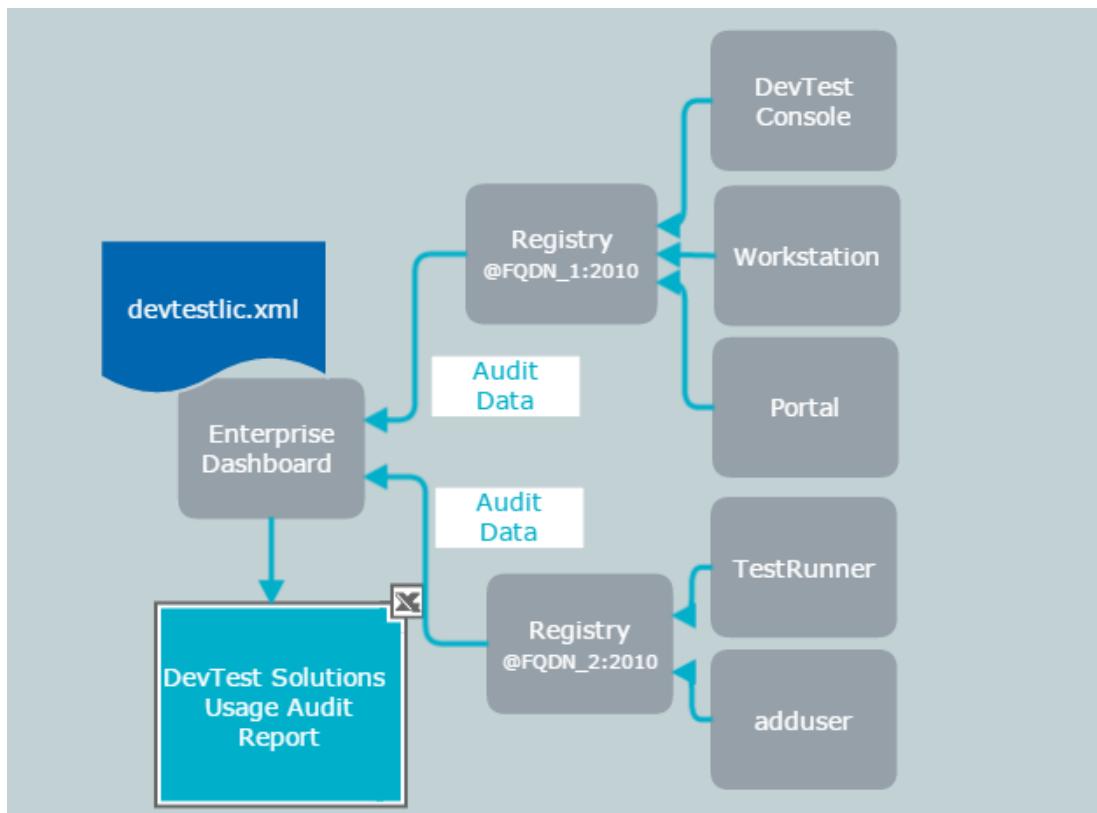
Honor-Based Licensing

Honor-based licensing overview:

- The DevTest 8.0 and later license agreement is based on the maximum allowed concurrent user sessions by user type. Contact your account team if you have any questions about your specific licensing agreement.
- The license is file-based, where there is one file for the enterprise. This file activates DevTest Solutions at the initial startup following installation.
- The Local License Server (LLS) and Internet Based License Server are no longer supported for DevTest Solutions 8.0.
- Concurrent usage data by user type is automatically collected.
- The Enterprise Dashboard produces licensing reporting.
- A Usage Audit Report, which reports maximum concurrent usage by user type, is a tool that helps you assess compliance with the license agreement. Users with administrative privileges can access this report.

For more information, see [How Licensing, ACLs, and Audit Reports Work Together \(see page 1399\)](#).

The following diagram shows the activation of DevTest Solutions by a license file that is stored with the Enterprise Dashboard. The registries collect audit data from the UIs and CLIs that users log in to, including the DevTest Console, the Workstation, the Portal, and command-line utilities such as TestRunner and adduser. The registries forward the audit data to the Enterprise Dashboard. Administrators can generate a DevTest Solutions Usage Audit Report to verify compliance with the license agreement.



How Licensing, ACLs, and Audit Reports Work Together

The page includes:

- Usage-based license agreement
- File-based license activation
- ACL activation
- Honor-based compliance
- Continuous collection of usage data by user type
- Usage Audit Report

Usage-Based License Agreement

Your license agreement with CA Technologies is based on the maximum number of concurrent user sessions by user type. User types are:

- Runtime User
- Test Power User
- SV Power User
- PF Power User

Each activity that a user can perform with DevTest Solutions has a corresponding permission. Each permission is associated with a role. Each role is associated with one of the user types. Administrators assign permissions to users.

License Activation

License activation for DevTest Solutions is file-based. CA Technologies provides you with a license file (devtestlic.xml). During installation or upgrade of DevTest Solutions, the Setup wizard puts the file in the LISA_HOME directory on the host where the Enterprise Dashboard is installed. Subsequent installations of the Server component (DevTest Server) reference the URL of the Enterprise Dashboard. License activation occurs the first time you start the Enterprise Dashboard and the registries. No other component requires license activation.

If you are upgrading, request a new devtestlic.xml license from CA Technologies. The current license key generation process is leveraged to generate a product key delivered via the devtestlic.xml file. This key is used to activate or unlock the Enterprise Dashboard.

ACL Activation

Access Control (ACL) is enabled by default. Before users can access DevTest Solutions, they must be authenticated with valid credentials and then authorized to perform the tasks associated with their role. Using the Administration tab on the Server Console, you define for each user a user name, a password, and a role that is composed of a set of permissions. Roles are tied to user types.

Users must log in with valid credentials to access any UI (User Interface) or CLI (Command Line Interface). When a user opens the Workstation, browses to the Portal, or browses to the DevTest Console, a logon dialog is presented. The user logs in with the credentials defined in DevTest or in LDAP, if you have configured DevTest to use LDAP credentials. If the entered credentials match that of an authorized user, the UI opens. The features that are presented to the user are based on the permissions you grant to that user.

For backward compatibility, one exception exists. Although ACL is enabled, authorized users can run tests and start virtual services without specifying a valid login user name and password.



Note: We recommend that you [change the passwords for the standard users](#) (see page 1430) to ensure that only authenticated users gain access to DevTest Solutions.

Honor-based Compliance

The customer is responsible for license compliance with the terms of contract. The license issued does not enforce the maximum concurrency by user type. As outlined in the Usage Audit Report section, DevTest Solutions Usage Audit Reports are sent to CA Technologies upon request. The CA account manager may contact you to assess your interest in expanding your purchase agreement if the Audit Reports indicate concurrent usage of a user type that exceeds the agreement.

Customers who wish to evaluate functionality that has not been purchased should contact their Account Team to discuss a Proof of Concept trial, or to address questions about the new functionality. Any Support cases raised for product functionality that has not been purchased are treated as "Out of Scope".

Continuous Collection of Usage Data by User Type

When users log in to a DevTest Solutions UI or CLI, each registry captures session data by user type and forwards it to the Enterprise Dashboard at the top of each hour. If connectivity between the registries and the Enterprise Dashboard is lost, the usage audit data accumulates on the registries until connectivity is re-established. The Enterprise Dashboard compiles usage data from across the enterprise to count concurrent sessions by user type.

Usage Audit Report

Periodically, an administrator logs onto the Enterprise Dashboard and generates the Usage Audit Report for a specified time period. Data from the specific UI or CLI users log into is reported. The concurrent sessions with the maximum number of users of a given user type are used in the report calculations. The report details concurrent usage counts by user type across registries. For each user type with data, the report generator creates a tab with drill-down details. The access detailed in the Audit Reports can be compared to the license agreement to determine the level of compliance.

The Usage Audit Report also reports instances of VSEs with the VSE Performance mode activated.

You can evaluate this report in light of your license agreement to verify compliance or to see if you want to upgrade your contract.

DevTest Solutions Usage Audit Report

A DevTest Solutions Usage Audit Report is generated as an Excel workbook with the following tabs:

- First tab: Overview
- User Type tabs: Admin User, PF Power User, SV Power User, Test User, Runtime User
There is a tab for each user type with any activity over the time period, either concurrent or single usage. While Admin User is included in the report, you do not need to consider it when evaluating your compliance to your licensing agreement.
- Last tab: Component by User, with a column for each UI or CLI with access during this time period. Entries on this tab do not necessarily represent concurrent access.

Overview

The Overview tab provides details on the extent an enterprise is using the number of current users of each user type that their license permits. A given report provides data from all registries for a specified date range. The Count data is shown by user type.

- **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.
- Key ID: The license key identifier
- Expiration: The date the license expires
- Reporting Period Start: The starting date for this report.
- Reporting Period End: The ending date for this report.

▪ **User Type and Count**

Reported user types include PF Power User, SV Power User, Test Power User, and Runtime User. Admin User is also reported, although it is not counted as a user type for licensing. 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. PF Power User or SV Power User (equivalent user types)
- b. Test Power User
- c. Runtime User
- d. Admin User

See [User Types \(see page 1404\)](#).

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.

The Count calculations for the report are based on group counts where the maximum concurrency count is reached. The Min, Max, and standard deviation convey the distribution across registries. See the [Count Calculation Example \(see page 1403\)](#).

▪ **Maximum Concurrent Instances**

Counts concurrent VSEs that have the [VSE Performance \(see page 661\)](#) option enabled. If the VSE Performance option is not enabled, the count is 0.

User Type tabs (Admin User, PF Power User, SV Power User, Test User, Runtime User)

Each User Type tab reports metrics and statistics on concurrency groups that reached the maximum concurrency count for this reporting period.

- Average:
 - The average is calculated for each registry that is part of the concurrent group.
 - Total is for the user type. This value also appears on the Overview page.
- STDEV: Standard deviation that is calculated for each registry sample set.
- Minimum: The lowest value for each registry sample set.
- Maximum: The highest value for each registry sample set.

Component By User

The Component by User tab reports user session data from each registry that is connected to the Enterprise Dashboard.

▪ **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. User types are displayed as:

- ADMIN_USER
- PF_POWER_USER
- SV_POWER_USER
- TEST_POWER_USER
- SVT_RUNTIME_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 Console, DevTest Workstation, and Portal. Examples of CLIs include AddUser and Test Runner. Columns may vary by report, depending on where users log in. Some examples follow:

- Add User: The number of times this user initiated the adduser command-line utility.
- Console: The number of times this user logged in to the DevTest Console for this registry.
- Workstation: The number of times this user opened a DevTest Workstation, connected to the associated registry, and logged in.
- Portal: The number of times this user browsed to `http://hostname:1507/devtest` for this registry and logged in.



Note: For details on generating this report, see [Export Usage Audit Data \(see page 1467\)](#).

Count Calculation Example

The Count calculation for the [DevTest Solutions Usage Audit Report \(see page 1401\)](#) is a three-step process.

1. Each time a concurrency is detected across the enterprise, a group count is recorded for all involved registries. Consider the following example for the Test Power User user group.
2. For reporting purposes, the group counts with maximum concurrency are used. In this sample example, the shaded rows (1, 3, and 4) are used.

Test Power User				
Timestamp	Registry 1	Registry 2	Registry 3	Group Count
1	1	0	3	4
2	0	0	2	2
3	1	2	1	4
4	0	0	4	4
5	1	1	1	3

3. After identifying the maximum group count data by registry, the average of the counts by registry is calculated. The minimum and maximum count values are stored. These three values (average, minimum, and maximum) appear on the report.

Test Power User						
Registry	Count 1	Count 2	Count 3	Avg	Min	Max
Registry 1	1	1	0	0.67	0	1
Registry 2	0	2	0	0.67	0	2
Registry 3	3	1	4	2.67	1	4
TOTAL				4	1	7

This data is summarized on the Overview page of the report as User Type and Count. The Count data that is displayed is combined for each user type. The data is calculated by taking the average of the total Min and Max across the registries in the sample for each user type. The count represents the maximum concurrency for that user type because the sample is composed of only group counts with maximum concurrency.

User Type	Count
Test Power User	4

User Types

ACL has the following user types:

- PF Power User
- SV Power User
- Test Power User
- Runtime User



Note: The Admin User user type, which is displayed on the Roles page as a combination PF Power User and SV Power User, is not used in licensing. The Admin User is, however, included in the DevTest Solutions Usage Audit Report for your information.

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:

- PF Power User and SV Power User are equal in the hierarchy and both are the highest user type.
- Test Power User is lower than PF 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

Add User

User Information

User ID:	fakeID (* required)
Password:	*****
Re-type Password:	*****
Name:	fakeName

Roles for the User

<input checked="" type="checkbox"/> Test Administrator
<input type="checkbox"/> System Administration
<input type="checkbox"/> PF Power
<input type="checkbox"/> SV Power
<input type="checkbox"/> Test Power
<input checked="" type="checkbox"/> Runtime
<input type="checkbox"/> Test Runner
<input type="checkbox"/> Test Observer

A user can be assigned one or more roles.

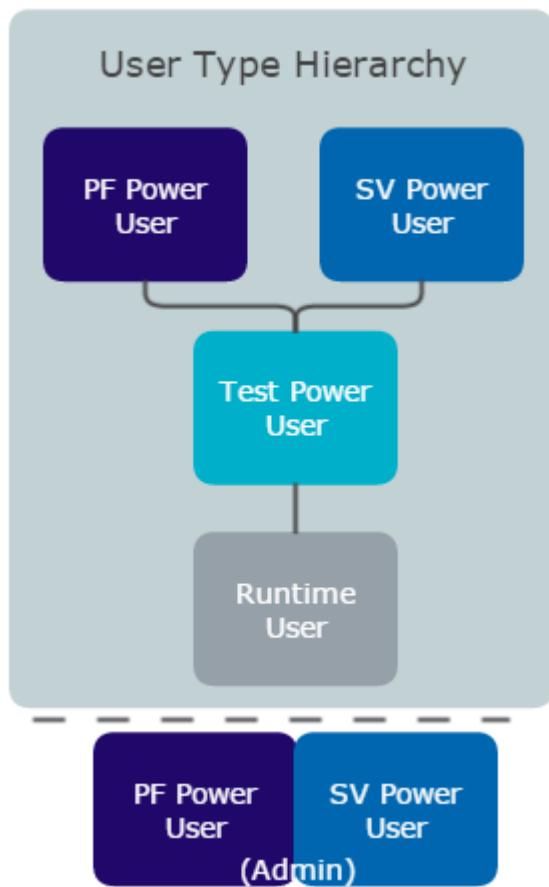
Different user types are associated with the assigned roles:

- The Test Administrator role is associated with the SV Power User type.
- The Runtime role is associated with the Runtime User type.



Note: See [Standard User Types and Standard Roles \(see page 1421\)](#) for details.

User types are hierarchical.

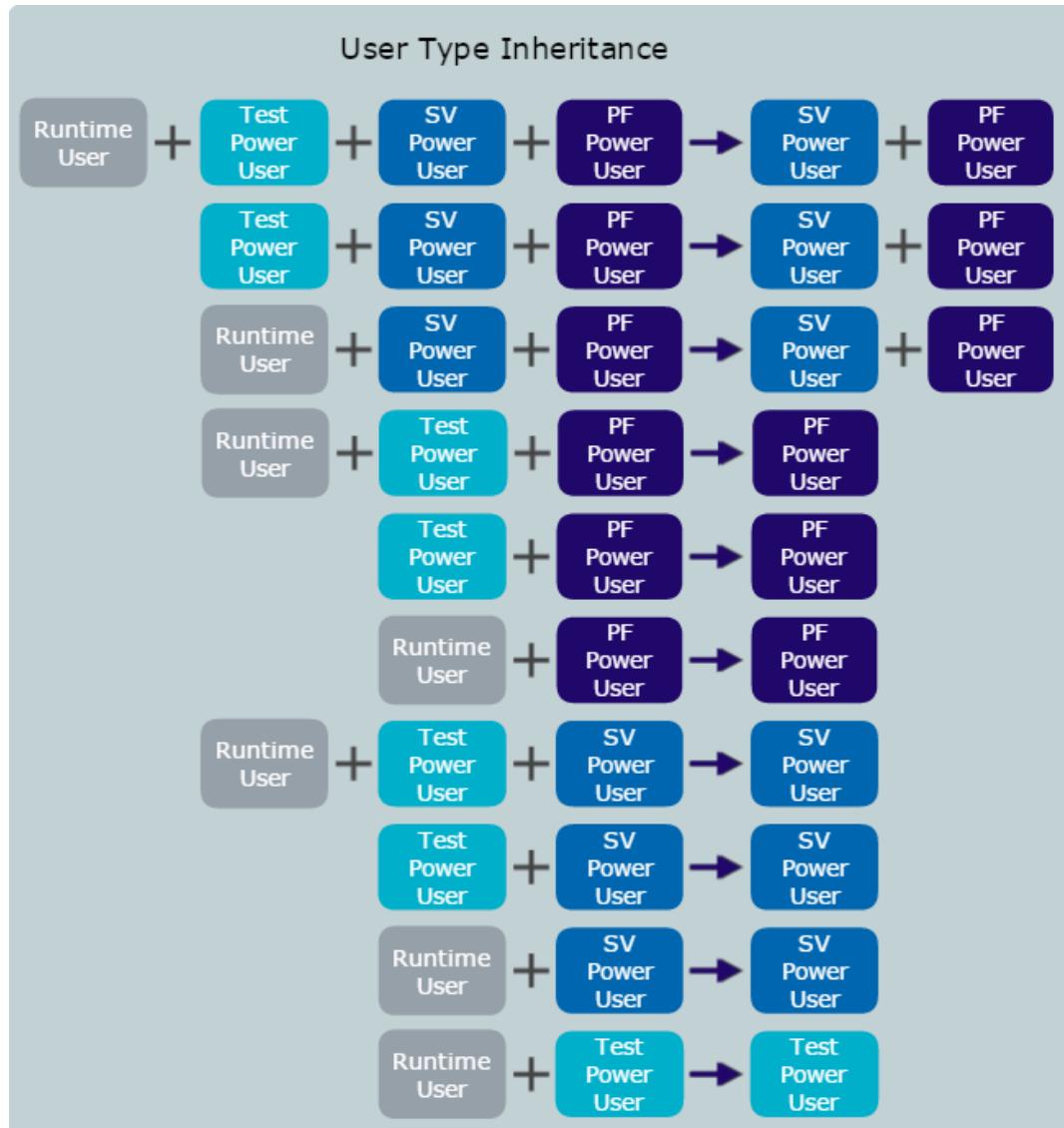


For purposes of the DevTest Solutions Usage Audit Report, the user type under which a user is counted is the highest user type that is associated with an assigned role. In this example, when the *fakeName* user logs in to a DevTest UI or CLI, the user session is audited as belonging to the SV Power User 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.



Security

This section contains the following pages :

- [Using SSL to Secure Communication \(see page 1409\)](#)
- [Using HTTPS Communication with the DevTest Console \(see page 1413\)](#)
- [Using Kerberos Authentication \(see page 1416\)](#)
- [Access Control \(ACL\) \(see page 1417\)](#)

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 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. There is some overhead using SSL encryption, 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 needs to be configured to have the value of **ssl**. Set this in the **local.properties** file.

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\webreckey.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, but it 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
```

Create 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. Enter the appropriate responses to the prompts.

```
prompt>keytool -genkey -alias serverA -keyalg RSA -validity 365 -
keystore keystore.ks
Enter keystore password: MyNewSecretPassword <the actual plaintext won't be sh
own>
Re-enter new password: MyNewSecretPassword
What is your first and last name?
[Unknown]: serverA
What is the name of your organizational unit?
[Unknown]: dev
What is the name of your organization?
[Unknown]: ITKO
What is the name of your City or Locality?
[Unknown]: Dallas
What is the name of your State or Province?
[Unknown]: TX
What is the two-letter country code for this unit?
[Unknown]: US
Is CN=serverA, OU=dev, O=ITKO, L=Dallas, ST=TX, C=US correct?
[no]: yes
Enter key password for <serverA>
      (RETURN if same as keystore password) <just hit return>
```

The utility creates a file containing a certificate that is valid for 365 days.

2. Copy the 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-----
MIICEzCCAXygAwIBAgIEThZnYzANBgkqhkiG9w0BAQUFADB0MQswCQYDVQQGEwJDQjELMAkGA1UE
CBM420IxCzAJBgNVBAcTAKNCMQswCQYDVQKewJDQjELMAkGA1UECxMCQ0IxCzAJBgNVBAMTAkNC
MB4XDTExDw0DAyMTE0N1oXDTEyMDcwNzAyMTE0N1owTJELMAkGA1UEBhMCQ0IxCzAJBgNVBAgT
AkNCMQswCQYDVQQHEwJDQjELMAkGA1UECDMCQ0IxCzAJBgNVBAAsTAKNCMQswCQYDVQQDEwJDQjCB
nzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAhYfaN+dCrK0wYZ+KeaaPU18DeXNiq0/mS+KGnXnh
Pz08vdX/7HDLW4pzFhntjmxx0i9dMwI02thTD1c0xI571PotenMENo4nyiUAEnMK9MTiWEYr2cQ
b6/TUueBCjRJ9I0GPCI0WPS+0Na2Q/wq8gPCHmDRpw1Xgo4uZ1v6C/ECAwEAATANBgkqhkiG9w0B
AQUFAAOBgQByCsX9EoBFIGhcSwoRwEvapIrv8wTaqpOKKeIevSmrnERRu6+oi+cJftbdEf w6GG
CBddJH+dGZ9VeqLU8zBGasbU+JPzG5E10g0XcUGeQQEaM1YMv6XWrIwNSljQk/MPZSt3R0tJ0lae
JPKJXSQ610xof9+yLHH0ebUGHUjd1Q==
-----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 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.

Using HTTPS Communication with the DevTest Console

Complete the following tasks to enable HTTPS communication with the DevTest Console.

1. [Generate a New Key Pair and Certificate \(see page 1413\)](#)
2. [Copy the New Keystore to LISA_HOME \(see page 1415\)](#)
3. [Update lisa.webserver Properties \(see page 1415\)](#)

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://wiki.eclipse.org/Jetty/Howto/Configure_SSL.

Follow these steps:

1. Open a command prompt window.
2. Type the following command:

```
cd JAVA_HOME\bin
```
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. Normally, this is the unqualified host name of the server. For example, for a machine named jetty.eclipse.org, you would enter **jetty.eclipse.org**.

However, it is possible to 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.

Copy the New Keystore to LISA_HOME

Follow these steps:

1. Copy the new keystore file to your **LISA_HOME** directory.
2. Rename the keystore file to **webserver.ks**.



Note: webserver.ks is the default file specified in the lisa.properties file. If you want to use a different file name, open lisa.properties and modify the **lisa.webserver.ssl.keystore.location** property to reflect the correct path and file name. For more information, see [Update Webserver Properties \(see page 1415\)](#).

Update Webserver Properties

Follow these steps:

1. Open the local.properties file in your LISA_HOME directory.

2. Add the following properties to this file.

```
# enable https and setup the webserver ssl keystore
lisa.webserver.https.enabled=true
lisa.webserver.ssl.keystore.location={{LISA_HOME}}webserver.ks
lisa.webserver.ssl.keystore.password=yourpassword
lisa.webserver.ssl.keymanager.password=yourpassword
lisa.webserver.port=8443
# should lisa workstation use https when launching the portals?
lisa.portal.use_https=true
lisa.portal.url.prefix=http://
```

3. Modify each property to specify the correct value.

- **lisa.webserver.https.enabled**

Set this property to **true** to use HTTPS with the DevTest Console.

- **lisa.webserver.ssl.keystore.location**

The default value for this property is **{{LISA_HOME}}webserver.ks**. Modify this value if you want to use a keystore file with a different name or in a different directory.

- **lisa.webserver.ssl.keystore.password**

Set this property to the password you defined when generating your keystore file.

- **lisa.webserver.ssl.keymanager.password**

Set this property to the key manager password you defined when generating your keystore file. Unless you specified a different password, this password is the same as your keystore password.

- **lisa.webserver.port**

Setting this property is optional, but the default port for HTTPS is 8443.

- **lisa.portal.url.prefix**

Change the value for this property from **http://** to **https://**.



Note: The first time the system reads the passwords in this local.properties, it converts the password to an encrypted property.

4. Save your changes and close local.properties.

5. 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](#) (see page 1671) 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;
};
```

Access Control (ACL)

Access control (ACL) is the mechanism that DevTest uses to authenticate users and enforce roles. ACL grants users access to only the application functionality they require to perform their role-based activities. ACL is required for DevTest Solutions so that you can monitor and maintain compliance to the license agreement. License agreements are based on the maximum number of concurrent user sessions.

Planning Deployment of ACLs

Starting with DevTest Solutions 8.0, controlling access through the Access Control Lists (ACL) system is required. Access to DevTest Workstation or the DevTest Portal is not possible without authenticating against the ACL system.

Before configuring ACL, carefully consider how each user will use DevTest Solutions and the corresponding access that is required for each type of user.

By default, only the Super User and the System Administrator have access to the Server Console that provides administrative access to the ACL system.

ACL Administration

The ACL Administrator is responsible for the following activities:

- Creating users in the ACL system.
- Assigning roles to users, based on the responsibilities and required access of each user.
- Restricting access to various parts of DevTest Solutions by assigning components to Resource Groups.
- Assigning user access to defined resource groups.

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 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.

Using the Lightweight Directory Access Protocol (LDAP) with DevTest Solutions

You can also choose to manage the passwords for DevTest Solutions 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 ACL administrator is no longer able to perform password changes.

Important! The implementation of the ACL system, 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 ACL is the responsibility of ACL 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 needs to ensure that the ACL/LDAP administrator is available when engaging CA support. ACL Administrators must take these factors into account when assigning roles to their users and groups.

Authentication

You can determine how DevTest authenticates users. You can [manually add users \(see page 1435\)](#) to the ACL database and specify credentials. The credentials are the user ID and password with which the user can log in to the DevTest Solutions user interface or command-line interface. Or, if your users are already defined with credentials in an LDAP database, you can use the LDAP server for authentication. In this case, you perform the steps in [Configure ACL to Use LDAP Authentication \(see page 1439\)](#).

Authorization

DevTest limits what DevTest features individual users can access based on their business role. DevTest Solutions is installed with over 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 \(see page 1421\)](#).

Important! To ensure security, the ACL Administrator should change the default password for (admin, guest) as soon as possible to a password they will not forget.

When you manually add users to the ACL 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, the ACL is automatically populated with a row for each user. In this case, you assign only roles as described in [authorize users authenticated by LDAP \(see page 1443\)](#).

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. See [ACL and User Sessions](#).

ACL and Backward Compatibility for CLIs and APIs that Ran Without Credentials

To access any DevTest user interface or command-line interface, users must log in with a valid user name and password. The requirement for credentials also applies to running tests and starting virtual services. If test cases that you automated in a previous release execute without credentials, you can temporarily override ACL so that they can continue to execute as scheduled. See [ACL and Command-Line Tools or APIs \(see page 1420\)](#).

ACL Database

The ACL 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 \(see page 1385\)](#).

ACL and Command Line Tools or APIs

LISA releases 7.5.2 and before did not enforce ACL. DevTest Solutions enforces ACL by default. CA recognizes the need for transition time to update existing command-line tools and APIs that have been running without a specified user name and password. While you are updating your automated tests, virtual services, and programs like TestRunner with login credentials, you can set a parameter that tells your program to run with an internally derived Runtime user name. In this case, the identity of the user that is logged in to the OS is stored in the session object.

To override ACL temporarily, follow these steps:

1. Log on to the DevTest Server containing the in-use registry.
2. Navigate to the installation directory.
3. Open the local.properties file for edit.
4. Add the lisa.acl.use.runtime parameter and set it to true, that is:

```
lisa.acl.use.runtime=true
```
5. Save local.properties.
 Your command-line program runs as '_runtime_<username>' where '<username>' comes from the 'user.name' system property.

To enforce ACL without exceptions, follow these steps:

1. Log on to the DevTest Server containing the in-use registry.
2. Navigate to the installation directory.
3. Open the local.properties file for edit.
4. Comment out the following parameter or set it to false.

```
lisa.acl.use.runtime=
```
5. Save local.properties
 All UIs and CLIs will run with valid login credentials.

Permission Types

Permissions can be divided into the following types:

- [Boolean \(see page 1421\)](#)
- [Numeric Limit \(see page 1421\)](#)

- [List \(see page 1421\)](#)
- [Custom \(see page 1421\)](#)

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 allows a user to 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.

List

A permission can be associated with a list of strings. The strings could be regular expressions.

Custom

Cloud-based provisioning of development and test environments use custom properties. DevTest creates custom properties for specifying the user name and password that is required to access the Virtual Lab Manager (VLM) provider.

For more information, see [Configuring DCM Properties \(see page 588\)](#).

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 \(see page 1422\)](#). You can [change \(see page 1436\)](#) the permissions that DevTest assigns to a standard role. In addition, you can [delete \(see page 1436\)](#) a standard role.

- **PF Power User and SV Power User**

The PF Power User / SV Power User combination user types includes the following roles:

- Super User

- DevTest Administrator

- **PF Power User**

The PF Power User user type includes the following role:

- PF 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:

- System Administrator
- Runtime
- Guest

Standard Permissions

Contents

- [DevTest Console Administration Permission \(see page 1423\)](#)
- [User and Role Administration Permission \(see page 1423\)](#)
- [Resource Administration Permission \(see page 1423\)](#)
- [Test/Suite Administration Permission \(see page 1423\)](#)
- [Virtual Services Administration Permission \(see page 1423\)](#)
- [DevTest Server Administration Permission \(see page 1425\)](#)
- [CVS Administration Permission \(see page 1426\)](#)
- [Report Administration Permission \(see page 1426\)](#)
- [Metric/Event Administration \(see page 1427\)](#)
- [Pathfinder Administration Permission \(see page 1427\)](#)
- [DevTest Workstation Permission \(see page 1428\)](#)
- [Cloud Lab Integration Permission \(see page 1429\)](#)
- [Can access any target Permission \(see page 1430\)](#)
- [Can only access a given list of targets Permission \(see page 1430\)](#)

A permission controls whether a user can perform a specific activity.

If a parent permission is allowed, then all of its child permissions are allowed.

The following top-level permissions are automatically created.

DevTest Console Administration Permission

The DevTest Console Administration permission has the following child permissions.

Child Permission	Type	Description
Access DevTest Console	Boolean	Allows a user to log in to the DevTest Console.
Access Server Console	Boolean	Allows a user to log in to the Server Console.
Access Pathfinder Console	Boolean	Allows a user to log in to the Pathfinder Console.
View VSEasy Console	Boolean	Allows a user to view the VSEasy Console.

User and Role Administration Permission

The User and Role Administration permission is a Boolean permission that allows a user to manage access control (ACL) from the **Server Console**.

Resource Administration Permission

The Resource Administration permission is a Boolean permission that allows a user to manage [resource groups](#) (see page 1445).

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 in the Registry Monitor .
Kill Test	Boolean	Allows a user to kill a test in the Registry Monitor .
Optimize Test	Boolean	Allows a user to optimize a test in the Registry Monitor .
View Test	Boolean	Allows a user to view a test in the Registry Monitor .
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			Bo	Allows a user to view the VSE Dashboard in the VSE Server Console.
VSE Server Administration			Bo	
	Configure VSE Tracking Data Cleanup		Bo	Allows a user to configure the process that deletes tracking data older than a specified amount of time.
	Stop VSE Server		Bo	Allows a user to shut down a Virtual Service Environment.
	Reset VSE Server		Bo	Allows a user to reset a Virtual Service Environment.
	Monitor VSE Server		Bo	Allows a user to monitor a Virtual Service Environment.
VSE Service Administration			Bo	
	VSE Service Deployment		Bo	Allows a user to deploy a virtual service.
	VSE Service Archive Retrieval		Bo	Allows a user to download the Model Archive (MAR) associated with a virtual service.
	VSE Service Execution		Bo	
	Start VSE Service		Bo	Allows a user to start a virtual service.
	Stop VSE Service		Bo	Allows a user to stop a virtual service.
	Update Deployed VSE Service		Bo	
	Update Virtual Service Capacity		Bo	Allows a user to update the concurrent capacity for a deployed virtual service.
				Allows a user to update the think time scale for a deployed virtual service.

	Update Virtual Service Think Scale	Bool ole an
	Update Virtual Service Auto-Restart	Bool Allows a user to update the auto-restart option ole for a deployed virtual service. an
	Update Virtual Service Group Tag	Bool Allows a user to update the group tag on a ole deployed virtual service an
	Set Virtual Service Execution Mode	Bool Allows a user to select a new execution mode ole for a deployed virtual service. an
	Reset Virtual Service Transaction Counts	Bool Allows a user to reset the transaction and error ole counts for a deployed virtual service. an
	Heal Virtual Service Image	Bool Allows a user to perform model healing. ole an

DevTest Server Administration Permission

The DevTest Server Administration permission has the following child permissions.

Child Permission	Type	Description
Debug DevTest Server	Bool ean	Allows a user to create heap dumps, create thread dumps, and perform garbage collection for a server component.
Monitor Registry	Bool ean	Allows a user to access the Registry Monitor .
Reset Registry	Bool ean	Allows a user to reset the registry.
Stop Registry	Bool ean	Allows a user to stop the registry.
Monitor Coordinator	Bool ean	Allows a user to view a coordinator status message in the Server Console or the Registry Monitor .
Reset Coordinator	Bool ean	Allows a user to reset a coordinator in the Server Console or the Registry Monitor .
Stop Coordinator	Bool ean	Allows a user to stop a coordinator in the Server Console or the Registry Monitor .
Monitor Simulator	Bool ean	Allows a user to view a simulator status message in the Server Console or the Registry Monitor .
Reset Simulator	Bool ean	Allows a user to reset a simulator in the Server Console or the Registry Monitor .

Stop Simulator	Bool	Allows a user to stop a simulator in the Server Console or the Registry Monitor .
----------------	------	---

CVS Administration Permission

The CVS Administration permission has the following child permissions.

Child Permission	Type	Description
View CVS Dashboard	Boole	Allows a user to access the CVS Dashboard .
Deploy or re-deploy monitor to CVS	Boole	Allows a user to deploy and redeploy CVS monitors.
Delete monitor from CVS	Boole	Allows a user to delete CVS monitors.
Run monitor immediately on CVS	Boole	Allows a user to run CVS monitors immediately, regardless of when they are scheduled to run.
Activate/Deactivate monitor on CVS	Boole	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	Bool	Allows a user to access the Reporting Console .
View report data	Bool	Allows a user to view his or her own report data in the Reporting Console .
View report data of other users	Bool	Allows a user to view other users' report data in the Reporting Console .
View PDF report data	Bool	Allows a user to view his or her own report data as a PDF file in the Reporting Console .
View PDF report data of other users	Bool	Allows a user to view other users' report data as a PDF file in the Reporting Console .
Import XML report data	Bool	Allows a user to import XML data in the Reporting Console .
Export XML report data	Bool	Allows a user to export his or her own report data as an XML file from the Reporting Console .
Export XML report data of other users	Bool	Allows a user to export others users' report data as an XML file from the Reporting Console .
Export Excel report data	Bool	Allows a user to export his or her own report data as an Excel file from the Reporting Console .
Export Excel report data of other users	Bool	Allows a user to export other users' report data as an Excel file from the Reporting Console .

Delete report data	Bool	Allows a user to delete his or her own report data from the Reporting Console .
Delete report data of other users	Bool	Allows a user to delete other users' report data from the Reporting Console .
Create report filter	Bool	Allows a user to create a filter in the Reporting Console .
Delete report filter	Bool	Allows a user to delete his or her own filters from the Reporting Console .
Delete report filter of other users	Bool	Allows a user to delete other users' filters from the Reporting Console .
View all filters	Bool	Allows a user to view all users' filters in the Reporting Console .

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.

Pathfinder Administration Permission

The Pathfinder Administration permission has the following child permissions.

Child Permission	Type	Description
View Paths	Boolean	Allows a user to view paths in the DevTest Portal.
Create Baselines	Boolean	Allows a user to create baselines in the DevTest Portal.
Create Virtual Service Models	Boolean	Allows a user to create virtual service models and raw traffic files in the DevTest Portal.
Extract Data	Boolean	Allows a user to create data sets in the DevTest Portal.
View Cases	Boolean	Allows a user to view tickets in the DevTest Portal.
Edit Cases	Boolean	Allows a user to edit tickets in the DevTest Portal.

View Agents	Boole	Allows a user to view agents in the DevTest Portal. an
Agent Administration	Boole	Allows a user to stop and start dispatching for an agent in the an DevTest Portal.
Import Paths	Boole	Allows a user to import one or more paths in the DevTest Portal. an
Export Paths	Boole	Allows a user to export one or more paths from the DevTest Portal. an
Use Dev Console	Boole	This permission is no longer valid. an
Create Document	Boole	Allows a user to create a document from transactions in the DevTest an Portal.
View Documented Transactions	Boole	Allows a user to view documented transactions in the DevTest an Portal.
Create Documented Transactions	Boole	Allows a user to document transactions for testing in the DevTest an Portal.
Edit Point of Interest	Boole	Allows a user to edit Point of Interest transactions in the DevTest an Portal.
View Point of Interest	Boole	Allows a user to view Point of Interest transactions in the DevTest an Portal.
View Defects	Boole	Allows a user to view transactions with defects in the DevTest Portal. an

DevTest Workstation Permission

The DevTest Workstation permission has the following child permissions.

Child Permission	Type	Description
Start DevTest Workstation	Boole	Allows a user to start DevTest Workstation. an
View a Configuration	Boole	Allows a user to view configurations. an
Edit a Configuration	Boole	Allows a user to edit configurations. an
Create a New Configuration	Boole	Allows a user to create configurations. an
View a Staging Document	Boole	Allows a user to view staging documents. an
Edit a Staging Document	Boole	Allows a user to edit staging documents. an
Create a New Staging Document	Boole	Allows a user to create staging documents. an
View a Suite	Boole	Allows a user to view suites. an

Edit a Suite	Boole Allows a user to edit suites. an
Create a New Suite	Boole Allows a user to create suites. an
View a Test Case	Boole Allows a user to view test cases. an
Edit a Test Case	Boole Allows a user to edit test cases. an
Create a New Test Case	Boole Allows a user to create test cases. an
View an Audit Document	Boole Allows a user to view audit documents. an
Edit an Audit Document	Boole Allows a user to edit audit documents. an
Create a New Audit Document	Boole Allows a user to create audit documents. an
View Virtual Service Model	Boole Allows a user to view virtual service models. an
Edit Virtual Service Model	Boole Allows a user to edit virtual service models. an
Create a New Virtual Service Model	Boole Allows a user to create virtual service models. an
View Virtual Service Image	Boole Allows a user to view service images. an
Edit Virtual Service Image	Boole Allows a user to edit service images. an
Create a New Virtual Service Image	Boole Allows a user to create service images. an
Execute ITR	Boole Allows a user to start the Interactive Test Run utility. an
Execute Instant Replay	Boole Allows a user to replay a test case to a specific point in the Interactive Test Run utility. an
View ITR Properties	Boole Allows a user to view the Properties tab in the Interactive Test Run utility.
View ITR Test Events	Boole Allows a user to view the Test Events tab in the Interactive Test Run utility.

Cloud Lab Integration Permission

The Cloud Lab Integration permission has the following child permissions.

Child Permission	Type	Description
List Available Labs	Boolean	Allows a user to view the list of available labs.

Start / Stop Labs	Boolean	Allows a user to start and stop labs.
Expand Labs	Boolean	Allows a user to dynamically expand a test lab.
Kill Labs of other users	Boolean	Allows a user to kill a lab that another user started.

Can access any target Permission

This permission is reserved for future use.

Can only access a given list of targets Permission

This permission is reserved for future use.

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 \(see page 1421\)](#) 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 PF Power User and SV Power User combination user type. The default password is **admin**.

- **pfpower**

The pfpower user has the PF Power role, a PF Power User user type. The default password is **pfpower**.

- **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**.

Experience the Roles of Standard Users

To prepare for assigning roles to new users, use the standard users to get hands-on familiarity with roles. You can log in with the credentials of each standard user to experience what users will experience as they use the permissions that are associated with the various roles.

Follow these steps:

1. Browse to the **Server Console** and log in as a user with a Super User role.

`http://hostname:1505`

2. Expand the Administration pane in the left navigation bar. In the Security area, the Super User provides credentials to authenticate users and grants permissions to access features through role assignments.

3. Click **Users**.

Standard users are displayed.

User Id	Super User	System Administration	PF Power	SV Power	Test Power	Runtime	Guest
admin	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
sysadmin	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
pfpower	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
svpower	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
tpower	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
devtest	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
guest	<input type="checkbox"/>	<input checked="" type="checkbox"/>					

Default standard users

The default password is the same as the User Id for each standard user.

- Super User: admin, admin
- PF Power: pfpower, pfpower
- SV Power: svpower, svpower
- Test Power: tpower, tpower
- Runtime: devtest, devtest
- System Administration: sysadmin, sysadmin
- Guest: guest, guest

4. Log in to each UI and CLI with the credentials of a Super User.

5. Examine the functionality that the users with this role can access.

6. Repeat the last two steps with other standard users.

This process prepares you to set up access control (ACL). You set up access control by adding users and assigning roles that grant only those permissions that are necessary for the tasks they perform.

Important! As part of access control, we recommend that you change the passwords for standard users. Changing passwords is one way to prevent unauthorized users from accessing your system.

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. Ensure that DevTest Solutions is running. See [Start the Server Components \(see page 115\)](#).
2. Browse to the DevTest Console.
<http://localhost:1505>
3. Log in to the DevTest Console. If you have no credentials, take one of the following approaches:
 - If you plan to use LDAP for authentication, [log in as the standard Super User \(see page 118\)](#).
 - If you plan to define the credentials for users and you have not yet defined yourself as a user, [create a user with the Super User role \(see page 119\)](#).
4. Click **Server Console**.
5. Click the **Administration** navigation tab.
6. Click **Users**.
The standard users are displayed.
7. For each standard user, perform the following steps:
 - a. Click **Show User Details** for one of the standard users.

DevTest Users	
User Id	Name
admin	
sysadmin	 Show User Details
pfpower	
svpower	
tpower	
devtest	
guest	

Show User Details

- b. Type a new password in the **Password** field.
- c. Type the new password in the **Re-type Password** field.
- d. Click **Save**.

8. Click **Logout**.

View User Information from DevTest Workstation

When you are logged in to DevTest Workstation, you can open a dialog that provides the following 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 Users and Roles

Administrators with Super User access manage users and roles from the Administration panel in the **Server Console**. Consider the following approach:

1. Review roles and user types.
 - a. Select Roles.
 - b. On the Permissions tab, notice that the available roles are listed with the associated user types. For example, the Test Administrator role is associated with the SV Power User user type.
 - c. Your license agreement specifies the maximum number of concurrent users allowed for each user type. Keep this figure in mind as you assign roles to users. Notice that many roles map to the same user type.
 - d. Examine permissions associated with each role and, optionally, customize permissions.
 - e. Optionally, add a new role based on a standard role. The new role inherits the user type of the role on which it is based.
2. Add all DevTest users.
 - a. Select Users.
 - b. DevTest uses the user ID and password you specify to authenticate the named user.
 - c. DevTest uses the role you specify to authorize the user to perform various activities based on the granted permissions.
3. Verify that you have satisfactorily configured all of your users with the appropriate roles. If you customized existing roles or added new ones, verify the configuration.
4. **Back up your database** strategically per your in-house maintenance policies. This is a precaution that enables you to quickly recover your configuration of users and roles in case of database corruption. The resolution to a corrupted ACL database is to perform a Restore that should be managed by your database administrator.



More Information:

- [Change the Priority Order of Roles \(see page 1438\)](#)
- [View the Audit Log \(see page 1438\)](#)
- [adduser Command-Line Utility \(see page 1439\)](#)
- [Add and Update Roles \(see page 1436\)](#)
- [Add and Update Users \(see page 1435\)](#)

Add and Update Users

You use the [Server Console \(see page 1455\)](#) to add users, change the details for a user, and delete users. The details that you can change include the password.

User IDs and passwords differ regarding case-sensitivity.

- User IDs are not case-sensitive. For example, the user IDs **AAAA1** and **aaaa1** are treated as the same value.
- Passwords are case-sensitive.

You cannot delete the user that you are currently logged in as.

You can show or hide any of the columns on the **Users** window. Click the drop-down arrow in a column. Select **Show/Hide Columns**. Select or clear the appropriate check boxes.

To add a user:

1. In the **Server Console**, display the **Administration** panel.
2. Click the **Users** node.
3. At the bottom of the right panel, click **Add User**.
The **Add User** dialog appears.
4. In the **User ID** field, enter a unique ID for the user.
You can enter any combination of alphanumeric, hyphen (-), underscore (_), period (.), and ampersand (@) characters. The maximum number of characters is 100.
5. In the **Password** field, enter a password for the user.
You can enter any combination of alphanumeric, hyphen (-), underscore (_), and ampersand (@) characters.
6. In the **Re-type Password** field, enter the password again.
7. In the **Name** field, enter the user name.
You can enter any combination of alphanumeric, hyphen (-), underscore (_), and space characters. The maximum number of characters is 100.
8. In the **Misc Info** field, enter any additional information.
The maximum number of characters is 600.
9. In the **Roles for the User** area, select one or more roles to assign to the user.
10. Click **Add User**.
The **User Added** message appears.
11. Click **OK**.

To update a user:

1. Click **Show User Details** to the right of the user ID.
The **User Details** dialog appears.
2. Make the appropriate changes.
You can view the permissions for a role by clicking the role name.
3. Click **Save**.

To delete a user:

1. Select the check box to the left of the user ID.
2. Click **Delete User(s)**.
3. Click **Yes**.

Add and Update Roles

You use the [Server Console \(see page 1455\)](#) to add roles, change the details for a role, and delete roles.

The **Server Console** displays the roles and permissions in a grid format.

- The permissions appear on the left as rows.
- The roles appear on the top as columns. The role on the left has the highest priority. The role on the right has the lowest priority. The order becomes important when a user has more than one role. The **Server Console** lets you [change the priority order \(see page 1438\)](#).

Effective User Type	Roles													
	PF Power User SV Power User	PF Power User SV Power User	SV Power User	Runtime User	PF Power User	SV Power User	Test Power User	Runtime User	SV Power User	Test Power User	Test Power User	Test Power User	User	Guest
Permissions +	Super User	LISA Administrator	Test Administrator	System Adminis...	PF Power	SV Power	Test Power	Runtime	Test Runner	Test Observer	Load Tester			
<input checked="" type="checkbox"/> CVS Administration	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
Can access any target (TP)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
<input checked="" type="checkbox"/> Cloud Lab Integration	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Expand Labs (R)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Kill Other User's Labs (TP)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
List Available Labs (R)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Start/Stop Labs (TP)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> LISA Console Administration	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> LISA Server Administration	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> LISA Workstation	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> MetricEvent Administration	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> Pathfinder Administration	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> Report Administration	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Resource Administration (A)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> TestSuite Administration	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
User and Role Administration (A)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> Virtual Services Administration	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

If you select a parent permission, the child permissions are automatically selected. If you clear a parent permission, the child permissions are *not* automatically cleared. If you clear a child permission, the parent permissions are automatically cleared.

You can show or hide any of the columns on the **Roles** window. Click the drop-down arrow in the **Permissions** column. The drop-down arrow in the **Permissions** column appears when you place the mouse pointer over the column. Select **Show/Hide Columns**. Select or clear the appropriate check boxes.

To add a role:

1. In the **Server Console**, display the **Administration** panel.

2. Click the **Roles** node.
3. At the bottom of the right panel, click **Add Role**.
The **Add Role** dialog appears.
4. In the **Name** field, enter the role name.
You can enter any combination of alphanumeric, hyphen (-), underscore (_), and space characters. The maximum number of characters is 50.
5. In the **Description** field, enter the role description.
The maximum number of characters is 200.
6. In the **Permissions for new Role** area, assign permissions to the role.
You can use the drop-down box to select an existing role on which to base the new role.
7. Click **Add Role**.
A column is added for the new role. The column appears to the right of the existing roles.

To update a role:

1. Locate the role column in the roles and permissions grid.
2. To update a [Boolean \(see page 1420\)](#) permission, select or clear the check box.
3. To update a [numeric limit \(see page 1420\)](#) permission:
 - a. Click **Numeric Limits**.
 - b. For the **Maximum number of ...** field, select **Unlimited**, **None**, or enter a value in the numeric limit.
 - c. Click **Save**.
4. To update a [list item \(see page 1420\)](#) permission:
 - a. Click **List Items**.
 - b. Add, edit, and delete the list items as necessary.
 - c. Click **Save**.
5. Click **Save**.

To delete a role:

1. In the roles and permissions grid, select the role name and click the drop-down arrow.
2. Click **Delete Role**.
3. Click **Yes**.

Change the Priority Order of Roles

The **Server Console** displays the roles in order of their priority. The order becomes important when a user has more than one role.

Follow these steps:

1. In the **Server Console**, display the **Administration** panel.
2. Do one of the following actions:
 - Click the **Roles** node.
 - Click **Display Roles**.
3. Click **Reorder Priority**.
The **Reset Role Priority Order** dialog appears.
4. To change the role order, select roles and drag them to the target destination.
5. Click **Save Order**.

View the Audit Log

The audit log contains a series of entries for activities that ACL controls. Each entry includes the following information:

- **Timestamp**
When the entry was created.
- **User ID**
The unique ID of the user.
- **Role**
The role that is assigned to the user.
- **Permission ID**
A numeric identifier for the permission.
- **Permission Name**
The permission that controls the activity.
- **Status**
Whether the activity was allowed or denied.
- **Acted On**
The name of the server component, test case, or virtual service model (if applicable).
- **Other Details**
More information (if applicable).

Follow these steps:

1. In the **Server Console**, display the **Administration** panel.
2. Do one of the following actions:
 - Click the **Audit Log** node.
 - Click **Display Audit Log**.
3. Click **Refresh**.

adduser Command-Line Utility

You can use the **adduser** command-line utility to add a user. The utility is located in the **LISA_HOME\bin** directory of a DevTest Server installation. To use the utility, you must have the User and Role Administration permission.

This utility has the following format:

```
adduser -d userid -w password [-r role] -u userid -p password [-m registry_url]
```

The following options let you assign basic information to the new user:

- Use the **-d** or **--adduser** option to assign a user ID to the new user.
- Use the **-w** or **--addpassword** option to assign a password to the new user.
- (Optional) Use the **-r** or **--rolename** option to assign a role to the new user. If you do not assign a role, the user has no permissions until a role is assigned from the Server Console.

The following options let you specify your credentials:

- Use the **-u** or **--username** option to specify your user ID.
- Use the **-p** or **--password** option to specify your password.

If the registry is on a remote computer, use the **-m** or **--registry** option to specify the registry URL.

Example

This example adds a user who is named **user1**. Because the role name has more than one word, quotation marks are used.

```
adduser -d user1 -w password1 -r "Load Tester" -u admin -p myadminpassword
```

Configure Authentication Providers for ACL

You can configure access control (ACL) so that user authentication is based on the information in an LDAP server, multiple LDAP servers, the database, or LDAP servers and the database. The ACL administrator should consult with your LDAP administrator for configuration and implementation that is based on the following properties.

If LDAP successfully authenticates the user and the user does not exist in the database, the user can be automatically added to the database.

Follow these steps:

1. Edit the **authentication-providers.xml** file in the home directory,



Note: The **authentication-providers.xml** file must exist in the home directory.

2. Authentication providers that are listed in this file are tried in the order they appear in the file. Enter as many authentication providers as you need, in the order they should be used.

3. Complete the following fields for each authentication provider:

- **name**

The label for this authentication provider.

Required

- **type**

The type of authentication provider.

Required

Values: LDAP, ActiveDirectory, Embedded, Custom, or Legacy. These values are case-sensitive and **Legacy** is deprecated and will be removed in a future release.

The `<authentication-provider>` element has two required attributes of **name** and **type**. The name attribute provides a user-defined name that is associated with the respective authentication provider that is used in the UI and in error reporting. The **type** attribute determines the category of the authentication provider and how it is constructed

ActiveDirectory or LDAP

When the type is defined with **ActiveDirectory** or **LDAP** then an internal `AuthenticationProviderFactory` is used to create an authentication provider that will be used for looking up users in the Microsoft Active Directory server or a normal LDAP server.
All attributes are valid except for **factory-class**.

Example:

```

<authentication-provider
    name="Corp. Active Directory Server"
    autoAddUsers="false"
    authenticateOnly="false"
    enabled="true"
    type="ActiveDirectory"
    defaultRole="SV Power"
    rejectUnmappedUsers="true">
    <url>ldaps://server.example.com:3269</url>
    <user-dn>cn=readOnlyUser,ou=users,dc=example,dc=com</user-dn>
    <user-password>drowssap</user-password>
    <user-dn-pattern>cn={0},ou=users,dc=example,dc=com</user-dn-pattern>
    <user-search-base>dc=example,dc=com</user-search-base>
    <user-search-filter>(objectClass=user)</user-search-filter>
    <group-search-base>ou=groups</group-search-base>
    <group-search-filter>(member={0})</group-search-filter>
</authentication-provider>

```

Embedded

The **Embedded** type creates an authentication provider for the internal DevTest database. Only **name**, **type**, and **enabled** are valid attributes.

Example:

```
<authentication-provider
    name="DevTest ACL Database"
    type="Embedded"
    enabled="true"/>
```

Legacy

A type attribute value of **Legacy** creates an authentication provider that is based off the older and now *deprecated* com.itko.lisa.acl.IAuthenticationModule interface. Users of the IAuthenticationModule who are using it to access the internal DevTest database or an external LDAP/Active Directory server can switch to using the direct type replacements of **Embedded** and **LDAP/ActiveDirectory**. Any other implementations should be re-implemented as **Custom** authentication providers.

Only **name**, **type**, **enabled**, and **defaultRole** are valid attributes.

```
<authentication-provider
    name="ITKO Authentication Module"
    type="Legacy"
    enabled="true"
    defaultRole="Guest"/>
```

Custom

A value of **Custom** for the type attribute allows you to create your own DevTestAuthenticationProvider instance. When the value of the type attribute is **Custom**, then you must also define the **factory-class** attribute and the value of the **factory-class** attribute must be the fully qualified name of a Java class that implements the [com.ca \(http://com.ca\)](http://com.ca).dts.security.authentication.AuthenticationProviderFactory interface.

Only **name**, **type**, **enabled**, **defaultRole**, and **factory-class** are valid attributes. Any subelements like <property1>value1</property1> are passed to the instantiated factory-class as properties with the element name (property1) being the key and the text of the element (value1) as the value.

Example:

```
<authentication-provider
    name="My Authentication Provider"
    type="Custom"
    enabled="true"
    defaultRole="Guest"
    factory-class="com.example.FooAuthenticationProviderFactory">
    <property1>value1</property1>
    <property2>value2</property2>
</authentication-provider>
```

- **enabled**

Controls whether this authentication provider is available for authenticating users.

Values: true, false

Default: true

- **autoAddUsers**

Controls whether successfully authenticated users are automatically added to the database.

When this field has the value of *false*, you must explicitly create accounts in for your LDAP users.

If you enable this field, you must manually remove users from the DevTest database as users are removed from the system.

Values: true, false

Default: true

- **defaultRole**

The role to assign to new users upon successful login if they have no other roles assigned.

Default: Guest

- **rejectUnmappedUsers**

Prevent users with no LDAP groups mapped to roles from logging in. If this value is set to *false*, and a user has no entry in the **ldap-mapping.xml** file, the user is given the role associated with the **defaultRole** parameter.

Values: true, false

Default: true

- **authenticateOnly**

Controls whether LDAP/AD is only used to authenticate a user. When this field has the value of *true*, you must explicitly create accounts in the ACL database or configure **autoAddUsers** to *true*.

Values: true, false

Default: false

If the value of the **authenticateOnly** attribute is **true**, then the **rejectUnmappedUsers** attribute must be **false**

- **url**

The URL of the server.

- **user-dn**

The distinguished name of the LDAP user that is used to connect to the server.

- **user-password**

The password of the LDAP user that is used to connect to the server. Unencrypted passwords will automatically be encrypted shortly after you save the file when the registry is running.

- **user-dn-pattern**

A pattern that is used to generate a distinguished name for the LDAP/AD user who is used to bind to the server. This element may be specified 0 or more times and the patterns are tried in the order in which they occur in the **<authentication-provider>** element. The pattern argument {0} will contain the username. An example is: "cn={0},ou=users,dc=example,dc=com".

- **user-search-base**

The relative name to use in searching for users.

- **user-search-filter**

The LDAP filter that is used to find user entries.

- **group-search-base**

The relative name to start searching for groups.

- **group-search-filter**

The LDAP filter that is used to find group entries.

- **factory-class**

The fully qualified name of a Java class that implements the **com.ca.dts.security.authentication.AuthenticationProviderFactory** interface.



Note: If you use Microsoft Active Directory as your LDAP server, use the Global Catalog port number in the LDAP configuration. The Global Catalog port number is 3268 by default, or 3269 by default if using SSL. You may need to contact your system administrator to determine the correct port number to use.

Using a Certificate with LDAP

Follow these steps:

1. Import the trusted public certificate for the LDAP server into the Java keystore for the JRE that uses.
2. For a normal install, the jre keystore is the keystore at **LISA_HOME\jre\lib\security\cacerts**.
3. Execute the following command in the LISA_HOME directory. <aliasname> can be anything that you choose, and <public_cert_file> is the path and filename for the public certificate from the LDAP server.

```
jre\bin\keytool -import -alias <aliasname> -keystore jre\lib\security\cacerts -trustcacerts -file "<public_cert_file>" -storepass changeit -noprompt
```

More Information:

For information about adding LDAP users to the database, see [Authorize Users Authenticated by LDAP \(see page 1443\)](#).

Authorize Users Authenticated by LDAP

Use the **Idap-mapping.xml** file in the DevTest home directory to assign roles to groups of LDAP users.

Follow these steps:

1. Edit the **Idap-mapping.xml** file.
2. For each role, enter the LDAP distinguished name for each LDAP group to associate with that role.
3. A sample **Idap-mapping.xml** file appears below.

```
<?xml version="1.0" encoding="UTF-8" ?>
<mappings>
    <mapping role="Super User">
        <groupDN>cn=Administrators,ou=groups,dc=example,dc=com</groupDN>
        <groupDN>CN=Team - Forward Cars - QA2 - Team Only,OU=Groups,OU=North America,DC=ca,DC=com</groupDN>
    </mapping>
    <mapping role="LISA Administrator">
```

```

</mapping>
<mapping role="Test Administrator">
</mapping>
<mapping role="System Administration">
    <groupDN>cn=administrators,ou=groups,dc=example,dc=com</groupDN>
</mapping>
<mapping role="PF Power">
</mapping>
<mapping role="SV Power">
</mapping>
<mapping role="Test Power">
</mapping>
<mapping role="Runtime">
</mapping>
<mapping role="Test Runner">
</mapping>
<mapping role="Test Observer">
</mapping>
<mapping role="Load Tester">
</mapping>
<mapping role="User">
    <groupDN>cn=users,ou=groups,dc=example,dc=com</groupDN>
    <groupDN>CN=Team - Forward Cars - All,OU=Groups,OU=North America,DC=ca,DC=com</groupDN>
    </mapping>
    <mapping role="Guest">
    </mapping>
</mappings>

```

In the example, members of the **Team - Forward Cars - QA2 - Team Only** distinguished name group are given the role of Super User, and members of **Team - Forward Cars - All** are given the role of User. If a user was a member of both groups, both roles and the permissions for each role are assigned.

To update the DevTest Users table by making individual role assignments

1. Ask all DevTest users to log in to DevTest Solutions and then log out.
2. Browse to the DevTest Console and log in.
<http://hostname:1505>
3. Click **Server Console** and then click the **Administration** tab in the left navigation pane.
4. Click **Users**.
The DevTest Users table opens.
5. For each user, clear the default role and select the appropriate role.
6. (Optional) To view permissions for a role in the right pane of the User Details dialog, click **Show User Details** and click the role name.
7. Click **Save**.

ACL Configuration Scenarios

Authenticate with LDAP/AD and manage user roles in DevTest

```

<authentication-provider
    name="Corp. Active Directory Server"
    autoAddUsers="false"
    authenticateOnly="true"

```

```

enabled="true"
type="ActiveDirectory"
defaultRole="SV Power"
rejectUnmappedUsers="false">
<url>ldaps://server.example.com:9999</url>
<user-dn>cn=readOnlyUser,ou=users,dc=example,dc=com</user-dn>
<user-password>drowssap</user-password>
<user-dn-pattern>cn={0},ou=users,dc=example,dc=com</user-dn-pattern>
<user-search-base>dc=example,dc=com</user-search-base>
<user-search-filter>(objectClass=user)</user-search-filter>
<group-search-base>ou=groups</group-search-base>
<group-search-filter>(member={0})</group-search-filter>
</authentication-provider>

```

[Authenticate with LDAP/AD and manage user roles with mapped LDAP groups and reject unmapped users](#)

```

<authentication-provider
  name="Corp. Active Directory Server"
  autoAddUsers="false"
  authenticateOnly="false"
  enabled="true"
  type="ActiveDirectory"
  defaultRole="SV Power"
  rejectUnmappedUsers="true">
<url>ldaps://server.example.com:3269</url>
<user-dn>cn=readOnlyUser,ou=users,dc=example,dc=com</user-dn>
<user-password>drowssap</user-password>
<user-dn-pattern>cn={0},ou=users,dc=example,dc=com</user-dn-pattern>
<user-search-base>dc=example,dc=com</user-search-base>
<user-search-filter>(objectClass=user)</user-search-filter>
<group-search-base>ou=groups</group-search-base>
<group-search-filter>(member={0})</group-search-filter>
</authentication-provider>

```

[Authenticate with LDAP/AD and manage user roles with mapped LDAP groups and allow unmapped users to be assigned a default role](#)

```

<authentication-provider
  name="Corp. Active Directory Server"
  autoAddUsers="false"
  authenticateOnly="false"
  enabled="true"
  type="ActiveDirectory"
  defaultRole="SV Power"
  rejectUnmappedUsers="false">
<url>ldaps://server.example.com:3269</url>
<user-dn>cn=readOnlyUser,ou=users,dc=example,dc=com</user-dn>
<user-password>drowssap</user-password>
<user-dn-pattern>cn={0},ou=users,dc=example,dc=com</user-dn-pattern>
<user-search-base>dc=example,dc=com</user-search-base>
<user-search-filter>(objectClass=user)</user-search-filter>
<group-search-base>ou=groups</group-search-base>
<group-search-filter>(member={0})</group-search-filter>
</authentication-provider>

```

Resource Groups

Resource groups are one or more DevTest Servers or VSEs. Define resource groups to determine the resources that a user or a project can access.

When using ACL, 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 do need to 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 **Server Console** 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** window. Click the drop-down arrow in a column. Select **Show/Hide Columns**. Select or clear the appropriate check boxes.

To add a resource group:

1. In the **Server Console**, display the **Administration** panel.
2. Click the **Resource Groups** node.
3. At the bottom of the right panel, click **Add Resource Group**. The **Add Resource Group** dialog appears.
4. In the **Name** field, enter a unique name for the resource group. The name must contain only alphanumeric characters, spaces, hyphens, or underscores.
5. In the **Description** field, enter a description of the resource group.
6. Select a resource group or groups by selecting the check box beside the group.
7. Click **Add**.
The Resource Groups panel 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 their labels using "strikethrough" text.

To delete a resource group:

1. Select the black triangle next to the name of the resource group.
2. From the drop-down list, select **Delete Resource Group**.
3. Click **Yes** in the **Deleting Resource Group** dialog.

To remove a resource from a resource group:

Clear the check box next to the resource group and click **Save**.

Note: You cannot remove a resource from a resource group when the resource is active.

Grant Roles to Resource Groups

To grant roles to resource groups, use the **Server Console**.

Follow these steps:

1. Select **Administration, Security, Resource Groups**.
2. Select the check boxes for the roles to associate with each resource group.
3. 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.

Logging

This section contains the following pages :

- [Logging Properties File \(see page 1449\)](#)
- [Status Messages for Server Components \(see page 1450\)](#)
- [Automatic Thread Dumps \(see page 1451\)](#)
- [Test Step Logger \(see page 1451\)](#)

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 the DevTest Portal.
- **registry.log**: the logging output for the registry.
- **simulator.log**: the logging output for the simulator.
- **svcimgr.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.

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 \(see page 1370\)](#), 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, CA does 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 in the **LISA_HOME** directory lets you configure the logging behavior.

To get more logging information from DevTest Workstation, 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
log4j.logger.org.apache=ERROR
log4j.logger.com.smardec=ERROR
log4j.logger.org.apache.http=ERROR
log4j.logger.org.apache.http.header=ERROR
log4j.logger.org.apache.http.wire=ERROR
log4j.logger.com.mchange.v2=ERROR
log4j.logger.org.hibernate=WARN
```

```
log4j.logger.org.jfree=ERROR
log4j.logger.com.jniwrapper=ERROR
log4j.logger.sun.rmi=INFO
```

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.appenders.A1=com.itko.util.log4j.TimedRollingFileAppender
log4j.appenders.A1.File=${lisa.tmpdir}/${LISA_LOG}
log4j.appenders.A1.MaxFileSize=10MB
log4j.appenders.A1.MaxBackupIndex=5
log4j.appenders.A1.layout=org.apache.log4j.EnhancedPatternLayout
log4j.appenders.A1.layout.ConversionPattern=%d{ISO8601}{UTC}Z (%d{HH:mm}) [%t] %-5p % -30c - %m%n
```

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 \(see page 1451\)](#).

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.

```
2015-01-15 17:40:12,228Z (09:40) [Event Sink Thread Pool Thread 5] INFO com.itko.lisa.coordinator.TestRegistryImpl - Coordinator Servers: 0 Simulator Servers: 0 VSEs: 0 Running vusers: 0 Labs: 1
```

```
Memory used 356mb, allocated 538mb, max 569mb (62%) labSims: 0 labVSEs: 0 labCoords: 0
Our cpu usage 0%, system cpu used 6% GC time 0% db ping no connection
```

The default interval of the status messages is 30 seconds. You can change the interval by editing the following properties in the **lisa.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_DUMP
```

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.

Test Step Logger

As described in [Elements of a Test Step \(see page 378\)](#), 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.appenders.A2=org.apache.log4j.RollingFileAppender
log4j.appenders.A2.File=${lisa.tmpdir}/log.log
log4j.appenders.A2.MaxFileSize=10MB
log4j.appenders.A2.MaxBackupIndex=5
log4j.appenders.A2.layout=org.apache.log4j.PatternLayout
log4j.appenders.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.

Monitoring

You can monitor DevTest Solutions by using the Service Manager command-line utility, the web-based **Server Console**, the Registry Monitor, or the Enterprise Dashboard.

This section contains the following pages :

- [Use the ServiceManager \(see page 1452\)](#)
- [Use the Server Console \(see page 1455\)](#)
- [Use the Registry Monitor \(see page 1458\)](#)
- [Use the Enterprise Dashboard \(see page 1460\)](#)

Use the ServiceManager

Contents

- [Service Manager Options \(see page 1453\)](#)
- [Service Manager Examples \(see page 1454\)](#)

The ServiceManager command-line utility lets you perform various actions on a registry, coordinator, simulator, or VSE server.

`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.
- **-b simulator-name, --attached=simulator-name**
Instructs simulator-name to return a list of attached mobile devices.
- **-e service-name, --heapprof=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.

- **-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%) Our cp
u 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 Server Console

Open the Server Console

You can open the **Server Console** from DevTest Workstation or from a web browser.

To open the Server Console from DevTest Workstation:

Select **View, Server Console** from the main menu.

To open the Server Console from a web browser:

1. Ensure the [registry \(see page 319\)](#) is running.
2. Enter **http://localhost:1505/** in a web browser.
If the registry is on a remote computer, replace **localhost** with the name or IP address of the computer.
The DevTest Console appears.
3. Click **Server Console**.

View the Component Health Summary

The **Component Health Summary** tab in the **Server Console** provides an indicator of the overall health of running components.

The health indicator appears on a scale from 0 through 100. The value is derived from the following statistics:

- Overall CPU load
- Amount of CPU the server is using
- How much of its own heap the server is using

A value of 0 represents an idle system. A value of 100 represents a system that is maxed out.

The following image shows the **Component Health Summary** tab. The X-axis represents the time. The Y-axis represents the value of the health indicator.

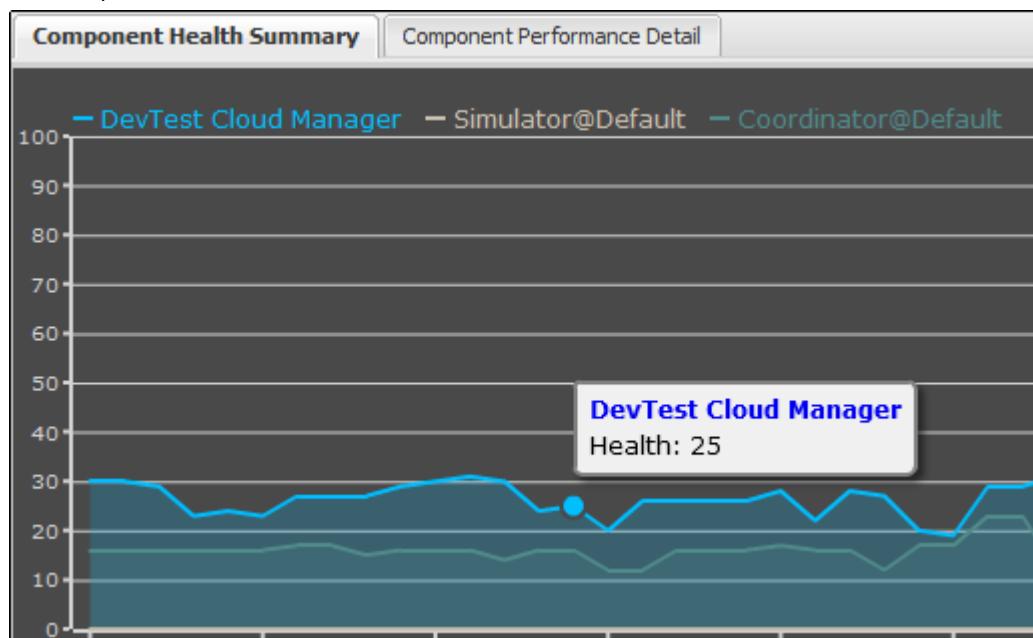


Image of the Component Health Summary page

To view the component health summary:

1. Go to the **Server Console**.
2. Click the **DevTest Cloud Manager** node in the **DevTest Network** panel.
The data appears in the **Component Health Summary** tab.

View the Component Performance Detail

The **Component Performance Detail** tab in the **Server Console** lets you view the following performance statistics about running lab members:

- **Average load:** The value is shown as an integer.
- **Heap:** The value is shown as a percentage.
- **CPU:** The value is expressed as a percentage.

The following image contains performance data for a coordinator in the **Default** lab. Tooltips for the average load and CPU metrics are shown.

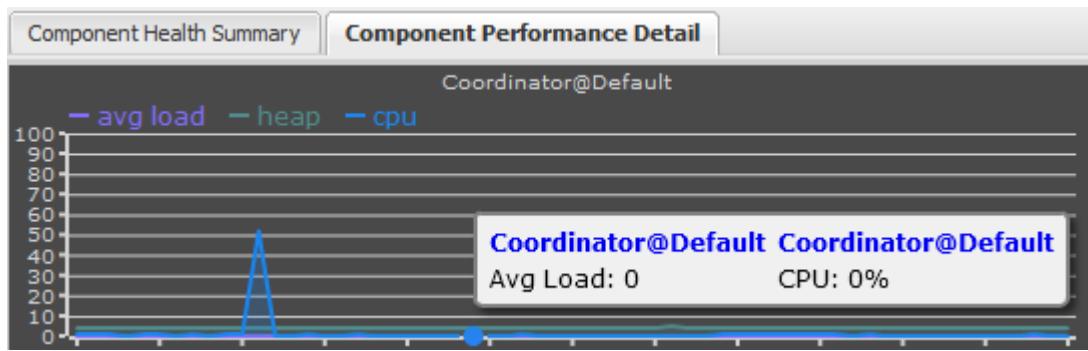


Image of the Component Performance Detail page

To view the component performance detail:

1. Go to the **Server Console**.
2. Click the **DevTest Cloud Manager** node in the **DevTest Network** panel.
3. Click the **Component Performance Detail** tab.
4. Right-click a lab member in the network graph and select **View Performance Data**.
The lab member is added to the **Component Performance Detail** tab. The metrics appear with different colors. Each metric provides a tooltip.

Create Heap Dumps and Thread Dumps

The **Server Console** lets you create heap dumps and thread dumps for a server component.

Typically, you perform these procedures only when requested to do so by customer support.

An alternate way to create a heap dump is by using the Service Manager command-line utility.

To create a heap dump:

1. Go to the **Server Console**.
2. Right-click a server component in the network graph and select **Dump Heap**.
3. When prompted, save the file.
4. Open the file.
The file contains the fully qualified path name of the .hprof file, for example, C:\Users\myusername\isatmp_6.0.7\Coordinator_Server_HeapDump_2012-02-28_08-36-55.hprof.

Note: This functionality is available only on Sun Java 6 Java runtimes. This diagnostic tool can help Support determine the causes of OutOfMemory conditions. The heap is automatically dumped if an OutOfMemory condition occurs; this button is for manually triggering a heap dump.

To create a thread dump:

1. Go to the **Server Console**.

2. Right-click a server component in the network graph and select **Dump Threads**.
3. When prompted, save the thread dump file.
4. Open the thread dump file and view the contents.

Force Garbage Collection

The Server Console lets you force a server component to perform garbage collection.

In addition to performing the garbage collection, this procedure creates a file that contains memory statistics. For example:

```
Before: Memory used 35mb, allocated 97mb, max 227mb (15%) ;
After: Memory used 19mb, allocated 95mb, max 227mb (8%)
```

Typically, you perform this procedure only when requested to do so by customer support.

To force garbage collection:

1. Go to the **Server Console**.
2. Right-click a server component in the network graph and select **Force GC**.
3. When prompted, save the statistics file.
4. Open the statistics file and view the contents.

Use the Registry Monitor

Contents

- [Registry Monitor - Test Tab \(see page 1459\)](#)
- [Registry Monitor - Simulators Tab \(see page 1459\)](#)
- [Registry Monitor - Coordinator Servers Tab \(see page 1460\)](#)
- [Registry Monitor - Virtual Environments Tab \(see page 1460\)](#)

The **Registry Monitor** lets you monitor the test cases, simulators, coordinators, and virtual environments for a test suite.



To open the **Registry Monitor**, click **Toggle Registry** on the main toolbar of DevTest Workstation.

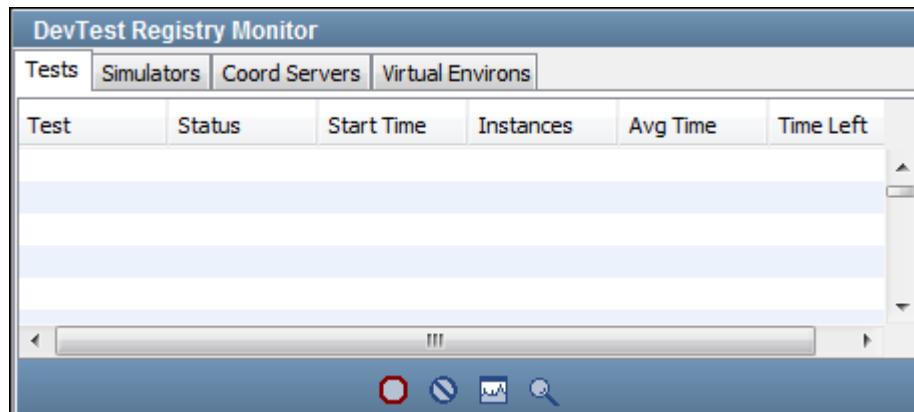


Image of the Registry Monitor

The **Registry Monitor** has the following tabs:

- [Tests Tab \(see page 1459\)](#)
- [Simulators Tab \(see page 1459\)](#)
- [Coordinator Servers Tab \(see page 1460\)](#)
- [Virtual Environments Tab \(see page 1460\)](#)

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 \(see page 562\)](#).
- 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 :

- [Reactivate a Registry or Enterprise Dashboard \(see page 1464\)](#)
- [Maintain Registries \(see page 1464\)](#)
- [Export Dashboard Data \(see page 1466\)](#)
- [Purge Dashboard Data \(see page 1468\)](#)

Open the Enterprise Dashboard

To open the Enterprise Dashboard from a web browser:

1. Ensure that the [registry \(see page 319\)](#) is running.
2. Navigate to the directory where you installed the Enterprise Dashboard.
3. Navigate to the **bin** directory, and run **EnterpriseDashboard.exe**, or select **Enterprise Dashboard** from the **Start, Programs** menu.
4. Enter **http://localhost:1506/** in a web browser.
If the **Enterprise Dashboard** is running on a remote computer, replace **localhost** with the name or IP address of that computer.
The **Enterprise Dashboard** opens.

Enterprise Dashboard Main Window

The main window in the **Enterprise Dashboard** lets you view the following performance statistics about running registries:

- **Display Name**

The Display Name (also called the "registry name" on the configure window) is a name that you assign when you configure the registry.

- **Name**

The URL of the registry.

- **Status**

Values can be *Running* or *Stopped*.

- **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](#) (see page 1272).

- **DevTest Console URL**

The address of the DevTest Console.

To display a **Registry Summary** window with more information, hover your mouse over the **Display Name** of a registry.

- **Display Name**

The Display Name (also called the "registry name" on the configure window) is a name that you assign when you configure the registry.

- **URL**

The URL of the registry.

- **DevTest Solutions Version**

Version of DevTest running on the computer where the registry runs.

- **Java Version**

Version of Java running on the computer where the registry runs.

- **Operating System**

Version of the operating system running on the computer where the registry runs.

- **Security**

Enabled or Disabled, depending on whether the registry uses ACL.

- **Status**

Values can be *Running* or *Stopped*.

- **Uptime**

The number of days, hours, minutes, and seconds since the registry was started.

- **Last Start Time**

The date and time the registry was last started.



To display information about the database that the **Enterprise Dashboard** uses, click  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.



Note: All information is not available for releases earlier than 7.1. The following table shows what feature information displays for earlier releases.

Feature	Available in Registries Earlier than 7.1	Available for 7.1 Registries	New for 7.5 Registries
Registry display name		x	
URL	x		
Status	x		
Uptime		x	
Security	x		
DevTest version		x	
Java version		x	
Uptime		x	
Operating system		x	
DevTest Console URL		x	
Coordinator names	x		
Simulator names	x		
VSE server names	x		
Lab names	x		
Metrics			x
Historical Data			x

Enterprise Dashboard Registry Details Window

The registry details window in the **Enterprise Dashboard** lets you view details about registries.



Note: This window only displays data for registries running version 7.5 or later.

The screenshot shows a dashboard titled "Registry@utilisa-xp64.ca.com:2010". At the top, there are six tabs: "Last Hour", "7/3/14", "11:00 AM", "7/3/14", "12:00 PM", and a refresh button. Below the tabs, there are six categories with their respective counts and icons: Coordinators (2), Simulators (2), VSE Servers (1), Labs (2), Workstations (1), and Agents (1). Each category has a detailed table below it. The "Coordinator Metrics" table shows two entries: Coordinator1@DevLab and Coordinator1@Default, both with 0 tests started and 0 count. The "VSE Server Metrics" table shows one entry: VSE@Default with 0 maximum number of models deployed. The "Simulator Metrics" table shows two entries: Simulator1@Default and Simulator1@DevLab, both with 0 maximum number of VUs allocated. The "Coordinator Metrics" table at the bottom shows two entries: Coordinator1@DevLab and Coordinator1@Default, both with 0 tests started and 0 count. The "Simulator Metrics" table at the bottom shows two entries: Simulator1@Default and Simulator1@DevLab, both with 0 maximum number of VUs allocated.

The top part of the window lists all active coordinators, simulators, VSE servers, labs, Workstations, and agents for the registry.

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 largest and smallest number of workstations, simulators, coordinators, and VSEs running in the designated time period.

The **VSE Server 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 **Go back to registry view** or click **Overview** in the breadcrumbs in the top left corner of the page.

Reactivate a Registry or Enterprise Dashboard

The DevTest Installation Setup Wizard for 8.0 and above configures new registries. When you start the Enterprise Dashboard and the registries, the registries are automatically activated. However, if there are subsequent changes to any of the following, reactivation is needed.

- Host name of the server where the registry is installed
- Registry name
- Registry port
- Enterprise Dashboard URL

Follow these steps:

1. If the host name or port of the Enterprise Dashboard changes:
 - a. Log on to the computer with the Enterprise Dashboard.
 - b. Navigate to LISA_HOME and open **site.properties**.
 - c. Update the following line if the host name or port of the Enterprise Dashboard changes.
`lisa.enterprisedashboard.service.url=tcp://somehost:2003
/EnterpriseDashboard`
 - d. Save the file.
 - e. Restart the Enterprise Dashboard.
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 \(see page 96\).](#)

Maintain Registries

Maintaining registries involves the following procedures:

- Add the registry of a DevTest Server and then validate the registry.



Note: See [Configure Existing Registries \(see page 96\)](#) for information about adding a registry from a release earlier than DevTest 7.5. For versions beginning with 8.0, the installation process automatically configures new registries.

- Update an existing registry and then validate the registry.
- Delete a registry from an DevTest Server that is no longer in service.
- Periodically, validate each registry that is listed in the Enterprise Dashboard.

To add a registry (DevTest 7.5.x):

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.
`lisa.enterprisedashboard.service.url=tcp://localhost:2003/EnterpriseDashboard`
5. Save local.properties and exit.

To delete a registry:

1. [Open the Enterprise Dashboard \(see page 1460\)](#).
2. Select **Configure** from the **Options** menu.
3. Click the registry that you want to delete from the list of registries in the **Registry** column.
4. Click **Delete**.



Note: If you delete a registry that runs on version 7.5 or later, you must also update the `lisa.enterprise.dashboard.url` property in the `local.properties` file for that registry.



Note: Deleting a registry through the Registry Configuration page removes the registry from the dashboard. However, the historical data for that registry is not automatically purged from the database. For more information about purging historical data, see [Purge Dashboard Data \(see page 1468\)](#).

To update a registry:

1. [Open the Enterprise Dashboard \(see page 1460\).](#)
2. Select **Configure** from the **Options** menu.
3. Click the registry that you want to update from the list of registries in the **Registry** column.
4. Edit the **Registry Name** and **Display Registry** fields.
5. Click **Save**.

To validate a registry:

1. [Open the Enterprise Dashboard \(see page 1460\).](#)
2. Select **Configure** from the **Options** menu.
3. Click the registry that you want to validate from the list of registries in the **Registry** column.
4. Click **Validate**.
The dashboard server queries the selected registry and displays the status, for example, is running.
5. Click **OK**.

Export Dashboard Data

Contents

- [Export Usage Audit Data \(see page 1467\)](#)

The Enterprise Dashboard allows you to 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 Registry Details window, skip to step 4.
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. Click **OK**.
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.

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:

- PF Power User
- SV Power User
- Test Power User
- Runtime User

The report can include the Admin user, which is displayed in the Roles grid in the Service Console as a combination of PF Power User and SV Power User. Licensing ignores this user type combination.

You can generate the Usage Audit Report on demand. The default is every three months.

Follow these steps:

1. Browse to the Enterprise Dashboard.

`http://hostname:1506`

2. Click the **Options** list and select **Export Usage Audit Data**.

The Export Usage Audit Data dialog opens.

3. To select a start date 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 \(see page 1401\)](#) for details on each tab.

Purge Dashboard Data

The Enterprise Dashboard lets you purge historical event logs and metrics that are older than a specified date and time. You can also purge registries from the database that you deleted through the Registry Configuration page. For more information, see [Configure Registries \(see page 96\)](#).

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. Select the **Purge registries marked for deletion** check box to permanently delete all registries from the database that you deleted through the Registry Configuration page.



Note: Deleting a registry through the Registry Configuration page removes the registry from the dashboard. However, the historical data for that registry is not automatically purged from the database. Selecting this check box permanently removes deleted registries from the database and all event logs, metrics, and components that are associated with the registry.

4. Click **OK**.

A confirmation dialog opens.

5. Click **Yes**.

The selected data is purged from the Enterprise Dashboard.

Reference

This section provides detailed descriptions of elements 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 \(see page 1470\)](#)
- [Asset Descriptions \(see page 1511\)](#)
- [Companion Descriptions \(see page 1533\)](#)
- [Correlation Schemes \(see page 1555\)](#)
- [Data Set Descriptions \(see page 1558\)](#)
- [Event Descriptions \(see page 1581\)](#)
- [Filter Descriptions \(see page 1587\)](#)
- [Metrics Descriptions \(see page 1627\)](#)
- [Property Descriptions \(see page 1638\)](#)
- [REST Invoke API \(see page 1698\)](#)
- [Test Step Descriptions \(see page 1701\)](#)

Assertion Descriptions

This section describes the following assertions:

- [HTTP Assertions \(see page 1470\)](#)
- [Database Assertions \(see page 1477\)](#)
- [XML Assertions \(see page 1478\)](#)
- [JSON Assertions \(see page 1489\)](#)
- [Virtual Service Environment Assertion \(see page 1493\)](#)
- [Other Assertions \(see page 1494\)](#)
- [Mobile Testing Assertions \(see page 1507\)](#)

Regular expressions are used for comparison purposes in many assertions. For more information about regular expressions, see [Regular Expressions \(http://psoug.org/reference/regexp.html\)](http://psoug.org/reference/regexp.html).

HTTP Assertions

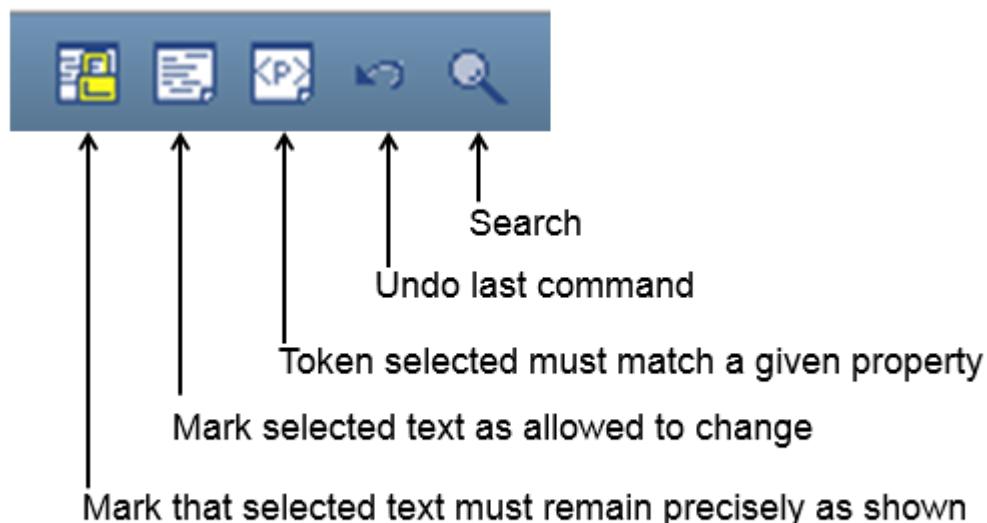
The following assertions are available in the HTTP assertions list for any test step:

- [Highlight HTML Content for Comparison \(see page 1471\)](#)
- [Check HTML for Properties in Page \(see page 1473\)](#)
- [Ensure HTTP Header Contains Expression \(see page 1475\)](#)
- [Check HTTP Response Code \(see page 1475\)](#)
- [Simple Web Assertion \(see page 1476\)](#)
- [Check Links on Web Responses \(see page 1476\)](#)

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.

The screenshot displays the DevTest Solutions interface for managing assertions. At the top, a toolbar includes icons for New, Open, Save, Print, and Help. Below the toolbar, a navigation bar shows the current location: [examples overview](#). The main content area is titled "Highlight HTML Content for Comparison - Assert369". The assertion details are as follows:

- Name:** Assert369
- If:** True then Fail the Test
- Log:** (empty)
- Run Assertion** button

A "Select Property" dialog box is overlaid on the browser window. It contains the following fields:

- Property:** {{correctCompany}}
- OK** and **Cancel** buttons

The browser window displays the ITKO examples overview page. The "Welcome to ITKO examples." text is highlighted in yellow, indicating it is the content being compared. The raw HTML code for the page is shown in the bottom panel:

```

<class="body">
    Welcome to ITKO examples. These are small applications used to make it easy to
    using LISA. </span>
    <p />
    <span class="body">
        You can now go through the examples to the left. Use this server in combination with the example
        test cases in the LISA examples directory in your installation. The source and everything you would need

```

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 by 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.

The screenshot shows the 'Check HTML for Properties in Page' window within the DevTest Solutions application. The window has a title bar 'Check HTML for Properties in page - Check HTML for Properties in page~1'. Below the title bar, there are fields for 'Name' (set to 'Check HTML for Properties in page~1'), 'If' (set to 'True'), and 'then' (set to 'Fail the Test'). A 'Log:' text input field and a 'Run Assertion' button are also present. The main area is titled 'Properties Referenced on Page' and contains a table with four rows:

Property	Observed Value	Expression
user_profile.userid	user-1398721607	
title	LisaBank - Account Activity	
user_profile.action	Withdraw	
user_profile.accountid	-2d085326:1157caf4ae9:-ffff	

At the bottom of the window are standard toolbar icons and a 'Find:' search bar.

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 then this 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 \(see page 1477\)](#)

- [Ensure Result Set Contains Expression \(see page 1478\)](#)

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

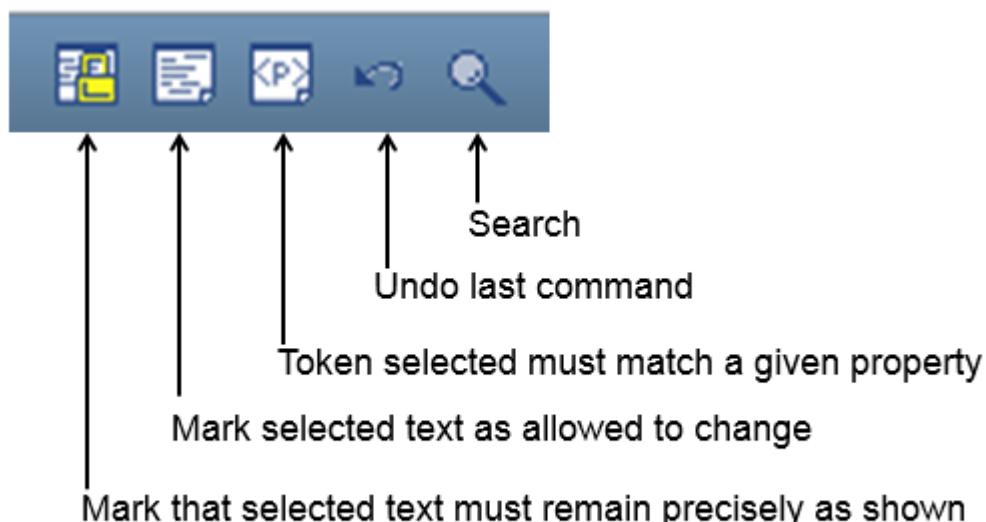
The following assertions are available in the XML assertions list for any test step:

- [Highlight Text for Comparison \(see page 1479\)](#)
- [Ensure Result Contains String \(see page 1481\)](#)
- [Ensure Step Response Time \(see page 1481\)](#)
- [Graphical XML Side-by-Side Comparison \(see page 1482\)](#)
- [XML XPath Assertion \(see page 1486\)](#)
- [Ensure XML Validation \(see page 1487\)](#)

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:

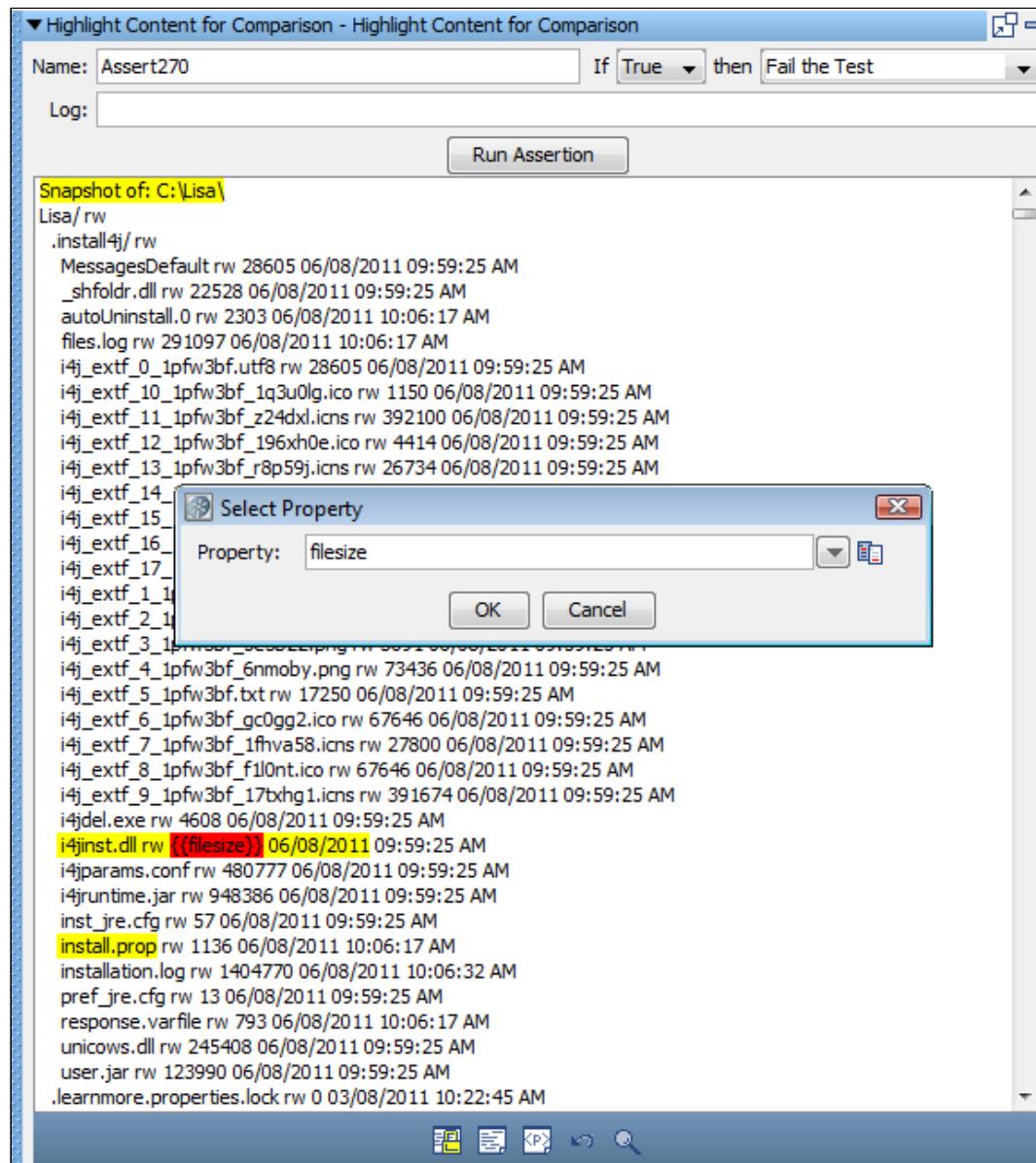


In the following example, we want to:

- 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.

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.

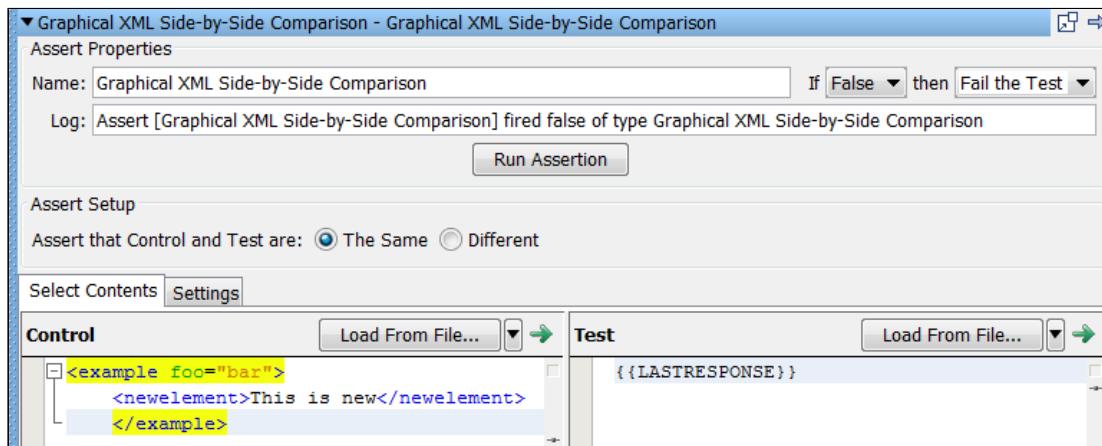
The assertion editor works by comparing a left and right side of XML to each other. The left side is known as Control Content. The right side is known as Test Content. For example, Control Content is the expected XML returned from a web service in the application under test, while the Test Content is 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

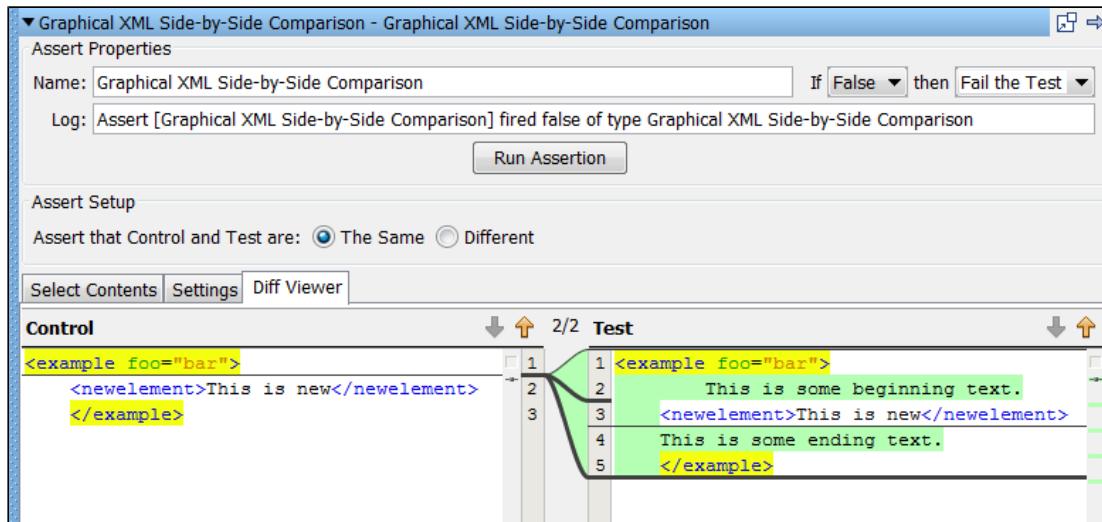


To perform the diff, click the green arrow.



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).

- **Collapse whitespace**

In addition to trimming whitespace, any sequence of one or more whitespace characters inside text is converted to a single space character.

- **Normalize whitespace**

Any sequence of one or more whitespace characters is converted to a single space character.

- **Ignore all whitespace**

All whitespace is ignored during the comparison.

- **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.

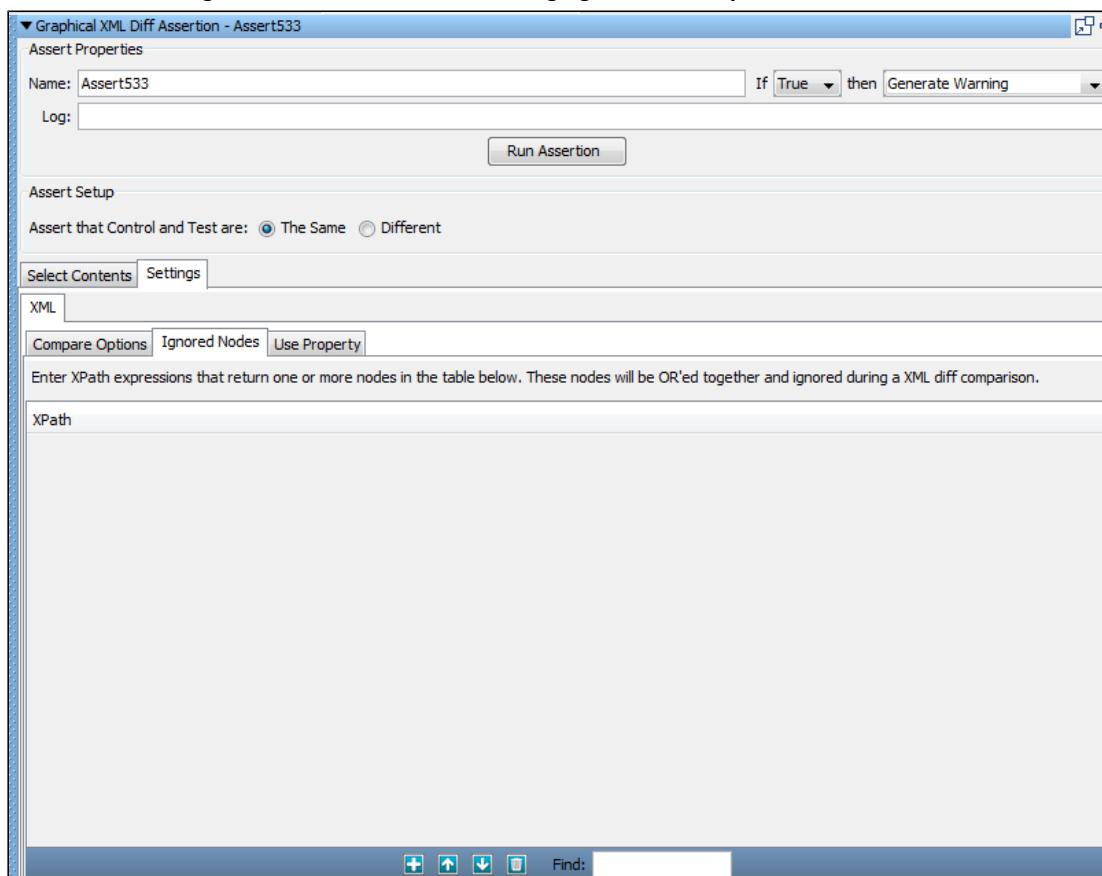
- **Ignore attributes**

Ignore attribute names and values.

- **Ignored Nodes**

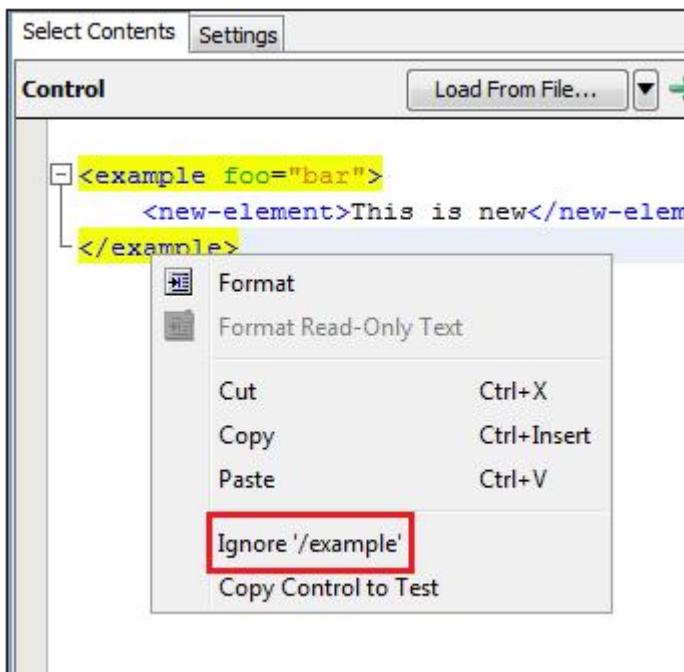
Ignored nodes are created from a list of XPaths that are run against the left and right documents. Each evaluated XPath that returns a node set is aggregated. When the diff occurs, any node that is found in the aggregate set is ignored.

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 "ITKO" 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

Namespace	Schema URL
http://www.w3.org/2001/XMLSchema	http://www.w3.org/2001/XMLSchema.xsd
http://schemas.xmlsoap.org/soap/envelope/	http://schemas.xmlsoap.org/soap/envelope...

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 \(see page 1489\)](#)
- [Ensure Result Contains \(see page 1491\)](#)
- [Ensure JSON Schema \(see page 1491\)](#)

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 for this assertion.

Name	Type	Value
<top>	Object	
store	Object	
book	Array	
[]	Object	
category	String	reference
author	String	Nigel Rees
title	String	Sayings of the Century
price	Unquoted	8.95
[]	Object	
category	String	fiction
author	String	Evelyn Waugh
title	String	Sword of Honour
price	Unquoted	12.99
[]	Object	
category	String	fiction
author	String	Herman Melville
title	String	Moby Dick
isbn	String	0-553-21311-3
price	Unquoted	8.99
[]	Object	
category	String	fiction
author	String	J. R. R. Tolkien
title	String	The Lord of the Rings
isbn	String	0-395-19395-8
price	Unquoted	22.99
bicycle	Object	

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.

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.

If you click **Run Assertion** and the assertion result is false, the **Run Assertion Result** area displays information about the failure.

Ensure Result Contains

The Ensure Result Contains assertion lets you ensure that a **JSON Path** result that is either 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**.

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 assertions are available in the Virtual Service Environment assertions list for any test step:

- [Assert on Execution Mode \(see page 1493\)](#)

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. For more information about execution modes, see the section [Specify How the Selected Model Should Behave \(see page 978\)](#).

Click **Run Assertion** to execute the assertion.

Other Assertions

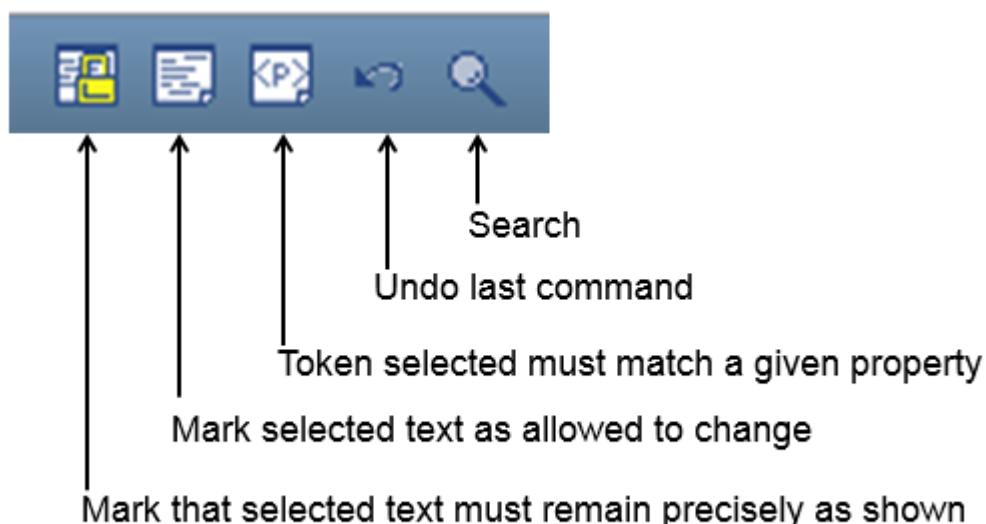
The following assertions are available in the Other assertions list for any test step:

- [Highlight Text Content for Comparison \(see page 1494\)](#)
- [Ensure Non-Empty Result Assertion \(see page 1497\)](#)
- [Ensure Result Contains String Assertion \(see page 1498\)](#)
- [Ensure Result Contains Expression Assertion \(see page 1498\)](#)
- [Ensure Property Matches Expression Assertion \(see page 1498\)](#)
- [Ensure Step Response Time Assertion \(see page 1499\)](#)
- [Scripted Assertion \(see page 1499\)](#)
- [Ensure Properties Are Equal Assertion \(see page 1501\)](#)
- [Assert on Invocation Exception Assertion \(see page 1501\)](#)
- [File Watcher Assertion \(see page 1502\)](#)
- [Check Content of Collection Object Assertion \(see page 1502\)](#)
- [WS-I Basic Profile 1.1 Assertion \(see page 1503\)](#)
- [Messaging VSE Workflow Assertion \(see page 1505\)](#)
- [Validate SWIFT Message Assertion \(see page 1505\)](#)

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:



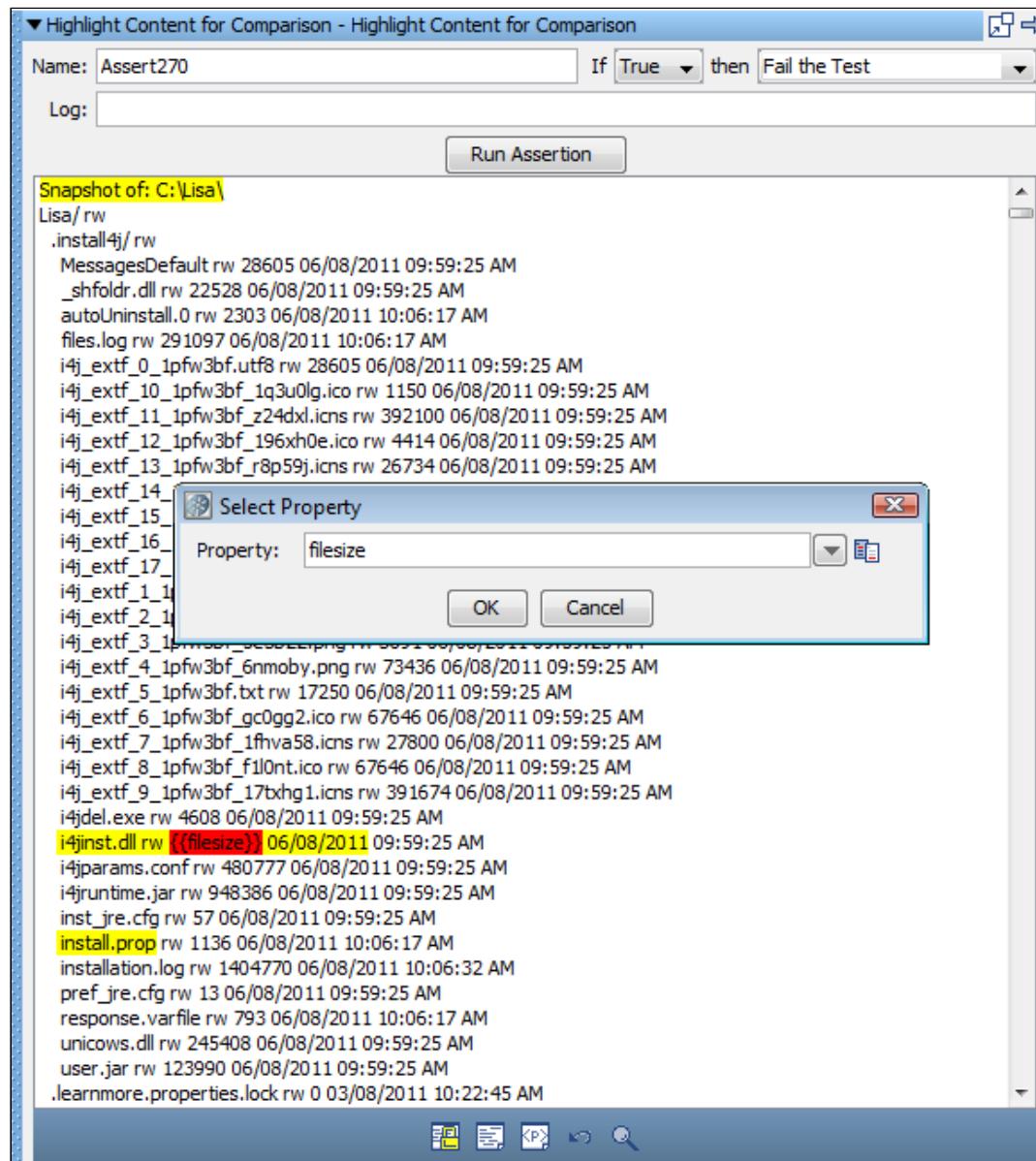
This technique is best explained by using an example.

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 it finds the value being searched anywhere in the response, the Ensure Result Contains String 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 \(see page 1481\)](#).

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 \(see page 1481\)](#).

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 \(see page 1385\)](#)".

- **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 \(see page 383\)](#).

▪ **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 \(see page 383\)](#).

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.

- **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 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 1.1 Assertion - WS-I Basic Profile 1.1 Assertion

Name:	Basic Profile 1.1 Assertion	If	True	then	Fail the Test
Log:					
Report Type	Display Only Not Passed Assertions				
Auto-Select Port	Don't Auto-select				
Service Name	EJB3AccountControlBeanService				
Service Namespace	http://ejb3.examples.itko.com/				
Port Name	EJB3AccountControlBeanPort				
<input type="button" value="Test"/>					

WS-I Profile Conformance Report

WS-I Profile Conformance Report



Report: WS-I Basic Profile Conformance Report.
Timestamp: 2012-05-08T13:26:05.808

Copyright (c) 2002-2004 by [The Web Services-Interoperability Organization](#) (WS-I) and Certain of its Members. All Rights Reserved.

Analyzer Tool Information

Version	1.0.0
---------	-------

Done

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 Cancellation 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](http://www.iso20022.org/full_catalogue.page) (http://www.iso20022.org/full_catalogue.page), 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. Please 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 decimal 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.

Mobile Testing Assertions

There are two types of mobile assertions that apply to a test step, depending on the type of test you are creating:

- Mobile assertions strictly for mobile tests

These assertions only apply to mobile tests and cannot be obtained outside of the Mobile Test Step Editor.

- Mobile Target Predicate

The Mobile Target predicate assertion can apply to any DevTest Solutions step, but a mobile step must come before it to initialize the mobile session. So in that sense, the predicate depends on the presence of a mobile step.

Mobile-Specific Assertions

These assertions are only available for a mobile test step that you edit within the Mobile test step editor.

The following assertions are available in the Mobile assertions list for any test step:

- [Ensure Same Mobile Screen \(see page 1508\)](#)
- [Ensure Screen Element Matches Expression \(see page 1509\)](#)
- [Ensure Screen Element is Removed \(see page 1509\)](#)

Ensure Same Mobile Screen

The Mobile Ensure Same Mobile Screen assertion verifies that all screen elements on the mobile device match the original recording. By default, if any element on the screen is different, the test step fails.

Complete the following fields:

- **Name**

The name of the assertion. The default value is **Assert same screen**.

- **If**

Select one of the following values.

- **True:** The action that you define in the **Then** field is executed when all screen elements on the mobile device match the original recording.

- **False:** The action that you define in the **Then** field is executed when the screen elements on the mobile device do not match the original recording. **False** is the default value.

- **Then**

Select the action to take when the conditions of the selected If statement are met:

- Generate a warning or error.

- End, fail, or abort the test.

- Select the next test step to run.

- **Log**

The text that appears as log event text when the assertion runs.

Click **Run Assertion** to execute the assertion.

- **Difference Threshold**

The difference threshold lets you define the level of precision to use when comparing screens. Subtle differences can exist between the application and the recorded screen shots, even when the application has not changed. For example, your test could compare a table in your application to a screenshot of that same table. If the screenshot included a highlighted row, but no rows were highlighted in the application, a precise pixel-by-pixel comparison shows these tables as nonmatching.

The default difference threshold is **1000**, but you can adjust this number if you find that your test case is returning incorrect matches. The following values provide some guidance for your settings:

- **0:** Indicates an exact match where every pixel on the screen must match every pixel on the recording.

- **1000:** Indicates that the screen and the recording are close enough to each other to be considered a match.

- **2000+:** Indicates that the screen and the recording can almost certainly be classified as nonmatching.

- **Start of Step**

Selecting this check box indicates that the screen comparison occurs at the beginning of the step.

- **End of Step**

Selecting this check box indicates that the screen comparison occurs at the end of the step.

- **Consider image pixels**

Selecting this check box indicates that pixel comparisons are used to determine the screen matches.

- **Consider screen structure**

Selecting this check box indicates that the structure of the screen is used instead of a pixel comparison to determine screen matches. For example, you might have a screen that displays the current date and time. The date and time in a recording that was performed two days ago do not match a test that you run today. This option lets you ensure that the same screen structure is in place, even though the specific values on the screen do not match.

- **Consider component bounds**

Ensure Screen Element Matches Expression

The Mobile Ensure Screen Matches Expression assertion verifies that the data entered for a screen element matches your specified regular expression. By default, if the value entered for the screen element does not match the specified expression, the test step fails.

When you select the Ensure Screen Element Matches Expression assertion, the system prompts you to enter an expression that you want the screen element to match. Once entered, the name and expression appear as a Key/Value pair in the Actions section of the tab. You can double-click these fields to modify them.

For example, the regular expression "ame" would match the names "Cameron" and "Pamela". If you want to ensure that a value entered for a screen element starts with a capital letter, and then three lower-case letters, enter this regular expression:

```
'[A-Z][a-z]{3}'
```

For more information about regular expressions, see [Assertion Descriptions \(see page 1470\)](#).

Complete the following fields:

- **Expression to Match (Value)**

The regular expression to match for the specified screen element.

- **If no match, execute step**

Select a step from the drop-down list to execute if a match is not made.

Ensure Screen Element is Removed

The Ensure Screen Element is Removed assertion verifies that a specified screen element on the mobile device has been removed before proceeding to the next step. By default, if the specified element is present, the test step fails.

When you [add this assertion \(see page 430\)](#), a pop-up dialog lets you select the screen to test that the element has been removed.

For example, you could have a test case that includes three steps. The last step involves clicking the Back button. To ensure the Back button is *not* included on the first step, you would add the Ensure Screen Element is Removed assertion for the Back button on the last step, and select that first step in the drop-down list.

Mobile Target Predicate

The Mobile Target Predicate assertion exists outside of the Mobile Test Step Editor and provides the ability to find and check values of a mobile element. This assertion can apply to any test within DevTest Workstation that has a mobile step.

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. For more information about execution modes, see the section [Specify How the Selected Model Should Behave \(see page 978\)](#).

The top drop-down list and text field define the search terms for the mobile target:

- **XPath**

Takes an XPath to the element in the page source.

- **Name**

Takes the name of the element, which corresponds to the name attribute in HTML/Hybrid page sources.

- **Id**

Takes the ID of the element, which corresponds to the ID attribute in HTML/Hybrid page sources.

- **Text**

Matches the full text value of an element.

- **PartialText**

Matches a substring of the full text value of an element.

The second drop-down list and true/false option defines the attribute of the element to match once the element has been found. The following attributes are supported:

- **Displayed**

True if the element is visible.

- **Enabled**

True if the element is enabled.

- **Selected**

True if the element has focus.

- **Text**

Matches the text of the element.

- **TagName**

Matches the tag name of the element.

The two check boxes apply if you use Text or TagName:

- **Partial Match**

Returns a partial match of any Text or TagName you enter.

- **Ignore Case**

Ignores the case of the text or TagName.

- **Timeout**

This field defines a timeout for the full operation. In other words, if the element is found but fails the predicate, the assertion will retry until the timeout is reached or the predicate succeeds.

Asset Descriptions

This section describes the following assets:

- [Email Connection Asset \(see page 1511\)](#)
- [IBM MQ Native Assets \(see page 1513\)](#)
- [JDBC Connection Assets \(see page 1514\)](#)
- [JMS Client Assets \(see page 1516\)](#)
- [JNDI Initial Context Asset \(see page 1519\)](#)
- [Mobile Assets \(see page 1519\)](#)
- [RabbitMQ Assets \(see page 1522\)](#)
- [SAP JCo Destination Assets \(see page 1526\)](#)
- [SSL Assets \(see page 1528\)](#)

Email Connection Asset

To define the connection information to the SMTP mail server, use a [connection asset \(see page 398\)](#).

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 \(see page 402\)](#) 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.

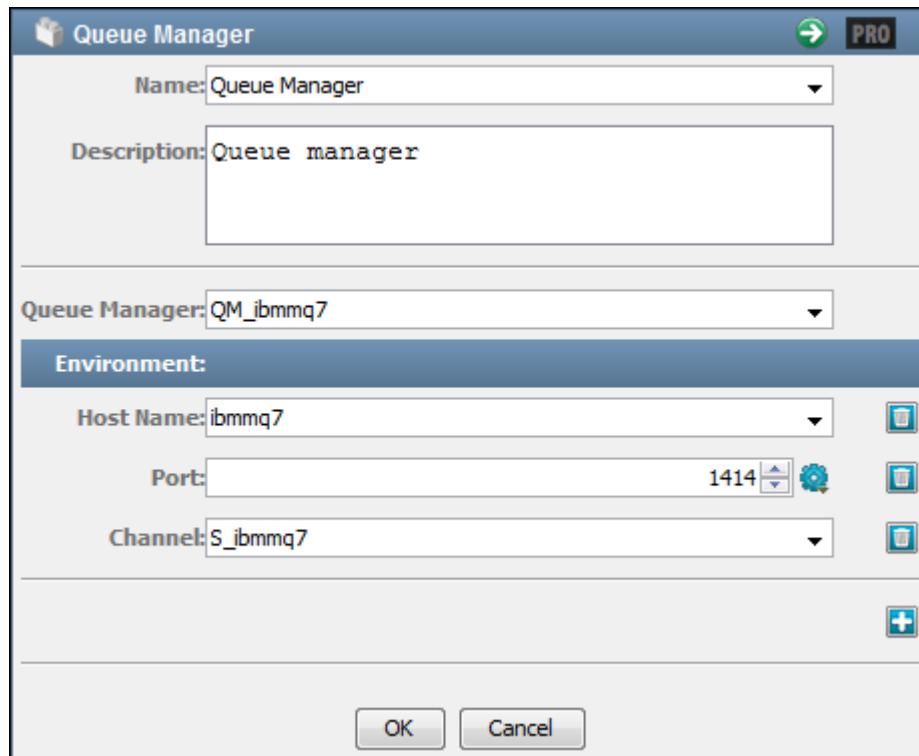


Note: These assets are for WebSphere MQ in native mode. If you are using WebSphere MQ in JMS mode, we recommend that you use the **IBM MQ JMS** assets.

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.

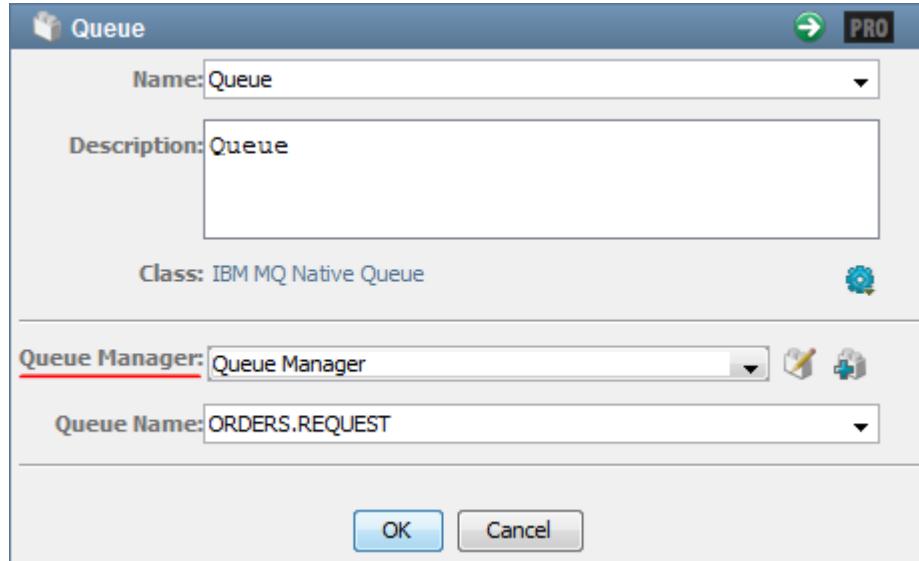
The following graphic shows a queue manager asset.



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 \(see page 404\)](#).

JDBC Connection Assets

To define the connection information to your JDBC system, use a [destination asset \(see page 398\)](#). 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**

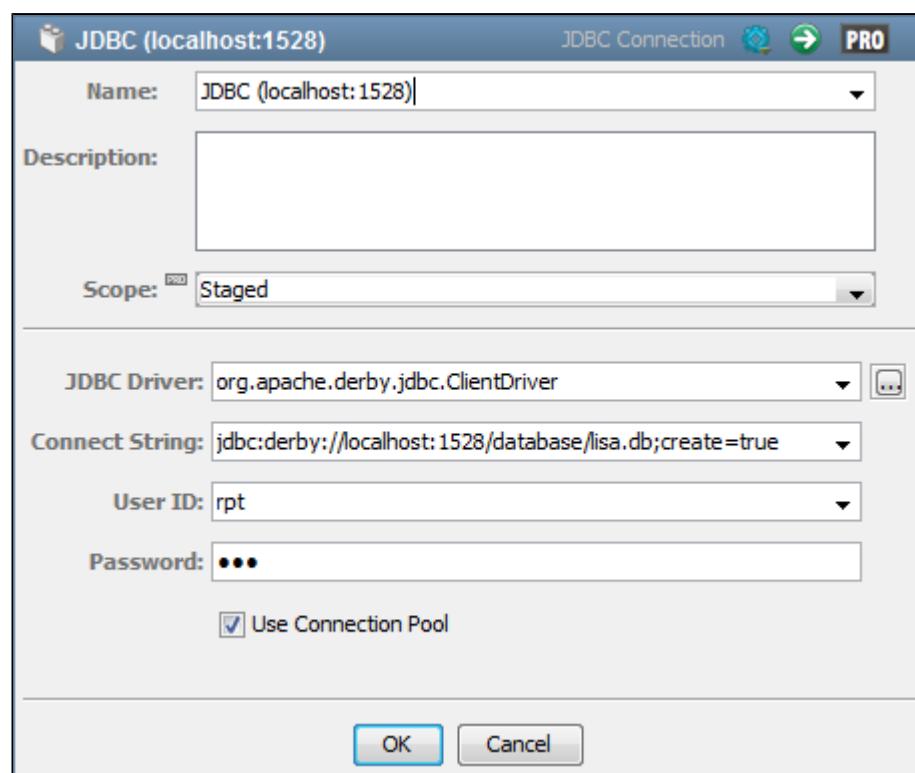
Enter a user ID (if the database requires it).

▪ **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](#) (see page 1666) file.



JDBC Connection assets panel

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](#) (see page 402) 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:

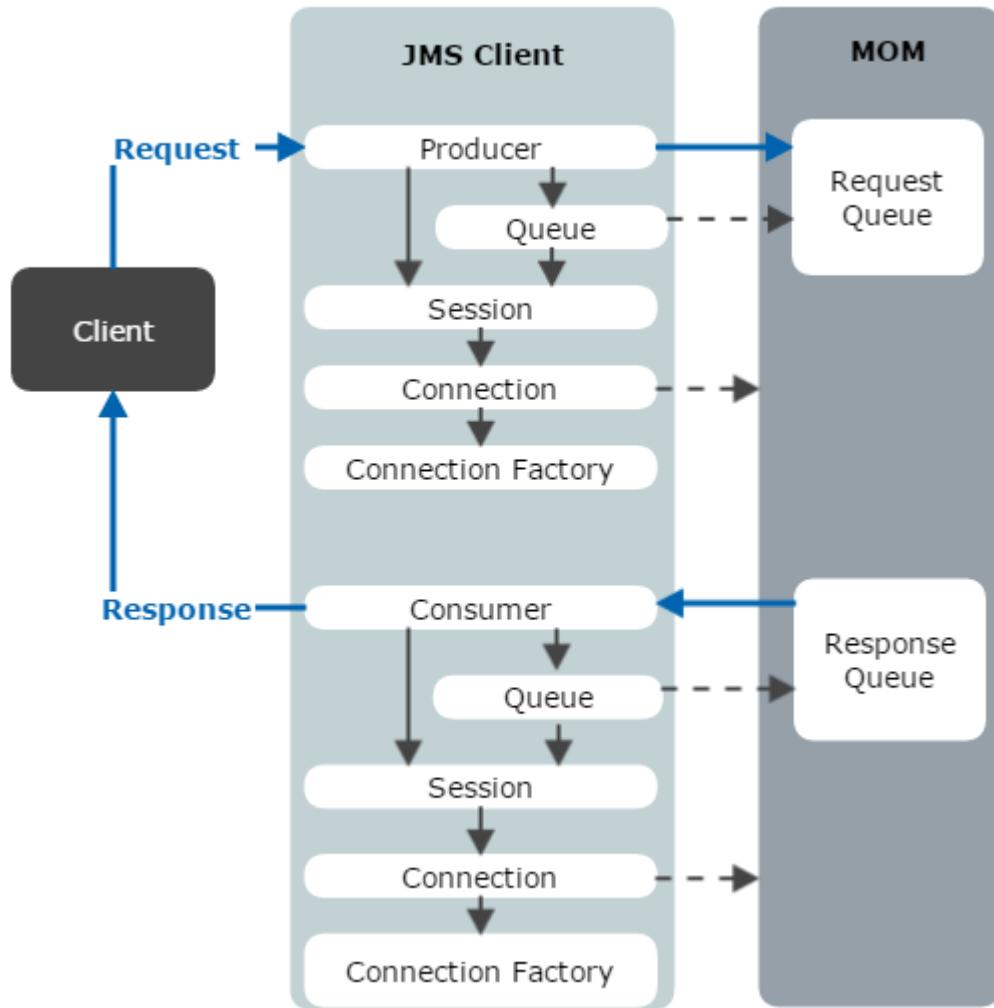
- **JMS client:** A Java program that uses JMS to communicate with a messaging system.
- **JMS provider:** A messaging system that implements JMS.

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.

Many 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.

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.

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.

Producers

A producer is used to send a message to a destination.

DevTest has only one type of producer.

Consumers

A consumer is used to receive a message from a destination.

DevTest has only one type of consumer. However, you can use a consumer 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.

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 \(see page 1516\)](#) 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.

Mobile Assets

You can create a mobile asset with various devices using the Mobile Session dialog.

Follow these steps:

1. Open the configuration file where you want to create the asset.
2. In the [Asset Browser \(see page 399\)](#), click **Add**  at the bottom of the pane.
3. Click **Mobile Session**.
The Mobile Session dialog displays.

4. Enter a **Name** for the asset. This name appears in the Destination field in the step editor. Use a name that is meaningful.
5. Enter a **Description** that helps you identify the asset.
6. Select a **Platform: iOS or Android**.
7. In the **Application** field, enter the full path to your application package file (.apk or .app). Or, click the Folder icon to locate and select the file on your computer.
8. (iOS only) In the Family field, select the iOS device type: **iPhone or iPad, iPhone only, or iPad only**.



Note: All iOS apps must be code signed and provisioned to launch on a device. See [Application Signing \(see page 655\)](#) for more information.

9. Select a Target:

- **Emulator (Android)**
- **Simulator (iOS)**
- **Attached Device**
- **SauceLabs**

Fields then vary depending upon the target you select.

10. Define the following fields:

- **For an Emulator (Android only)**

Emulator assets specify the mobile emulator that is used for testing on an Android device.

- **AVD**

Click the folder icon to locate the Android AVD that you defined when [configuring mobile testing \(see page \)](#).

- **SDK Version**

Select the version of the SDK to use with your test case.

- **For a Simulator (iOS only)**

Simulator assets specify the mobile simulator that is used for testing on an iOS device.

- **Simulator**

Click the folder icon to locate the simulator on your computer.

- **iOS Version**

Select the version of the iOS to use with your test case.

- **For an Attached Device**

Attached Device assets specify the mobile device that is used for your test cases. This asset is used for physical iOS and Android mobile devices that are connected to your computer.

- **Attached**

Click the folder icon to locate the simulator of your choice on your computer.



Note: If you connect or disconnect a device, click Refresh in the Attached Device dialog to view the latest devices.

- **SDK Version (Android only)**

Select the version of the SDK to use with your test case.

- **iOS Version (iOS only)**

Select the version of the iOS to use with your test case.

- **For SauceLabs**

SauceLabs is a cloud provider of scalable iOS and Android simulators. SauceLabs assets specify the SauceLabs account information for mobile cloud testing.



Note: Before you can create a SauceLabs asset, you must have a SauceLabs account with a unique access key.

11. To define advanced options, click . The **Scope** field is now available. For more information, see [Runtime Scope \(see page 402\)](#).

12. Once you enter an application in the **Application** field, specific information about the application displays in the **Details** window.

13. If your application is built using both native and browser elements, select the **Mixed Mode** check box.

14. To verify the asset ([see page 404](#)) before saving, click .

15. Click **OK**.

The new asset appears in the Asset Browser.



Note: If you defined multiple mobile assets, you must select one before you can record a test case.



Important! Android testing requires the correct version of Android SDK Build-tools. If you encounter an error message regarding zipalign or aapt, see [Android SDK Build-tools](#) under [Set Up the Android SDK \(see page \)](#) before you continue.

RabbitMQ Assets

You can create the following types of assets for RabbitMQ:

- RabbitMQ Queue
- RabbitMQ Temp Queue
- RabbitMQ Exchange
- RabbitMQ Connection

In the editor for each asset, each parameter has a tooltip that describes the purpose of the parameter.



Note: This feature is in preview status. To learn how to access this feature and provide feedback, you must be a member of the DevTest Customer Validation Group. For information about becoming a member, see <https://communities.ca.com/community/ca-devtest-community/blog/2015/02/18/thank-you-to-the-devtest-customer-validation-team>.

Contents

- [Connections \(see page 1522\)](#)
- [Queues \(see page 1523\)](#)
- [Exchanges \(see page 1524\)](#)
- [Temp Queues \(see page 1525\)](#)

Connections

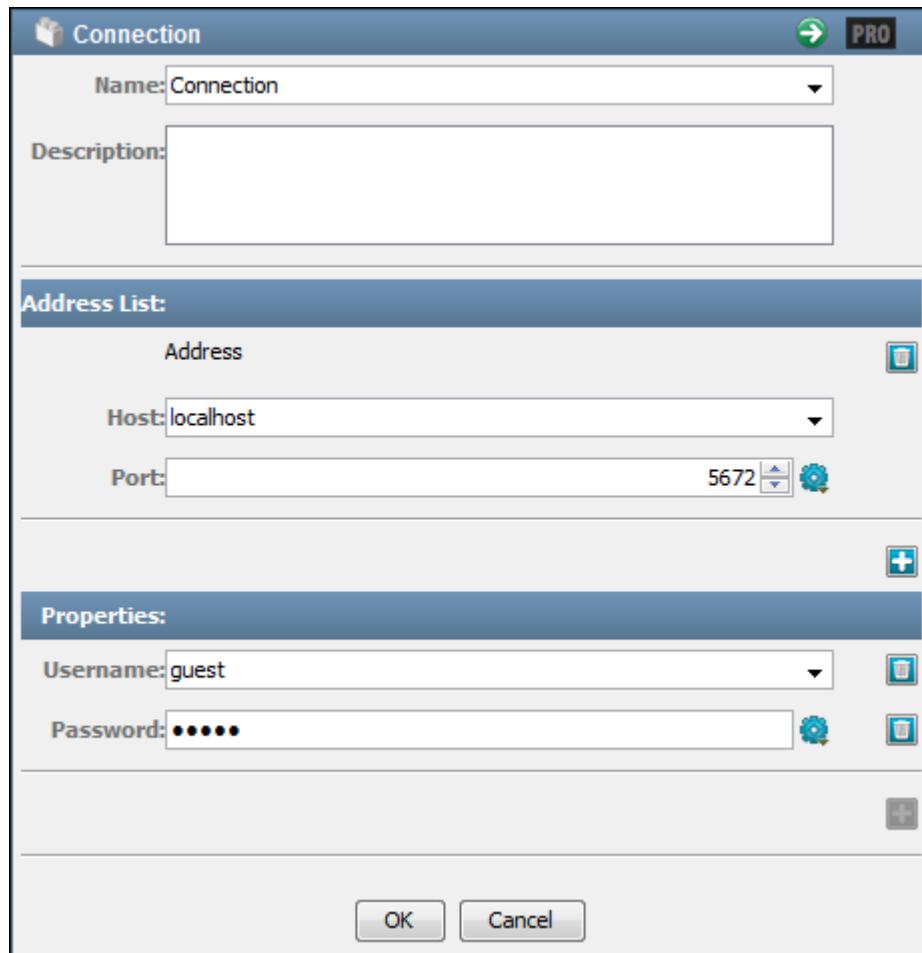
The connection asset lets you establish a connection to the RabbitMQ server.

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.

Queues

A queue represents a location where a message can wait to be picked up by a consumer.

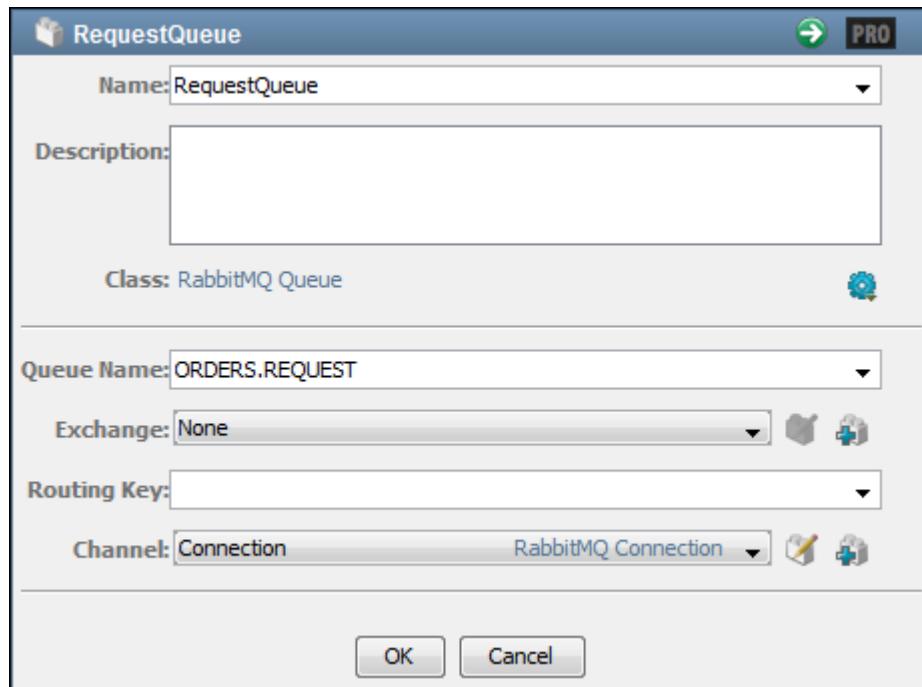
Like JMS and WebSphere MQ, more than one client can listen on the same queue. In this situation, each message is delivered to only one consumer.

Like JMS and WebSphere MQ, a client can create a temporary queue on demand.

Like WebSphere MQ, the 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.

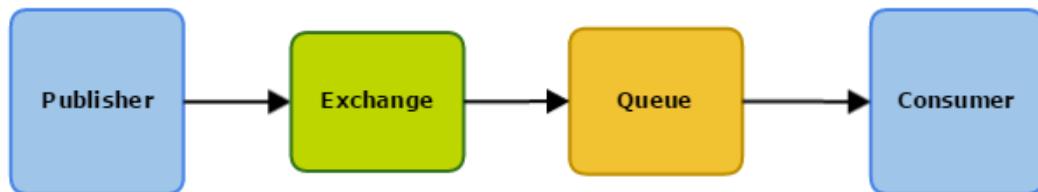
The following graphic shows a queue asset.



Screen capture of queue asset.

Exchanges

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 that 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 acts like a JMS queue, where a message waits to be consumed, the exchange side acts 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 there is more than one queue whose binding parameters 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 work 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.

Temp Queues

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.

SAP JCo Destination Assets

To define the connection information to your SAP system, use a [destination asset \(see page 398\)](#). 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**

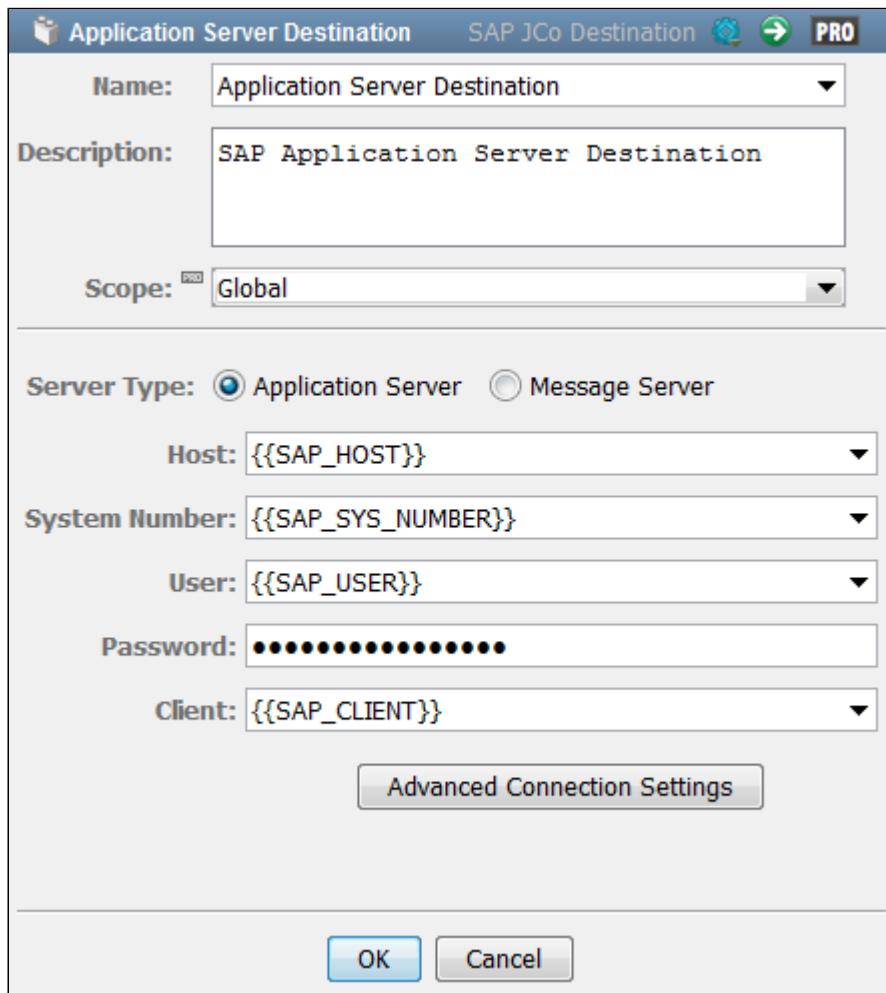
SAP client

The following graphics illustrate destination asset definitions for both application and message servers.

Message Server Destination SAP JCo Destination 

Name:	Message Server Destination
Description:	SAP Message Server Destination
Scope:	Global
Server Type: <input checked="" type="radio"/> Application Server <input type="radio"/> Message Server	
Host:	{{{SAP_HOST}}}
R/3 Name:	{{{SAP_R3NAME}}}
Group:	{{{SAP_GROUP}}}
User:	{{{SAP_USER}}}
Password:	*****
Client:	{{{SAP_CLIENT}}}
<input type="button" value="Advanced Connection Settings"/>	
<input type="button" value="OK"/> <input type="button" value="Cancel"/>	

SAP JCo Destination 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](#) (see page 402) 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. It affects every SSL connection that is made, whether from messaging, the web service step, or even contacting the license server.

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

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 covers the scenarios in order from the simplest to the most complex.

Contents

- [Basic Scenarios \(see page 1529\)](#)
 - [Just a trust store: The Key Store File Asset \(see page 1529\)](#)
 - [A trust store and key store using the default key: Two Key Store File Assets \(see page 1530\)](#)
- [Advanced Scenarios \(see page 1531\)](#)
 - [Forgoing the trust store: The "Trust All" Trust Manager Asset \(see page 1531\)](#)
 - [Using a specific client key: The Key Manager Asset \(see page 1532\)](#)
 - [Full configuration: The "Generic" Trust Manager Asset \(see page 1532\)](#)
- [SSL Context Parameters \(see page 1533\)](#)

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.

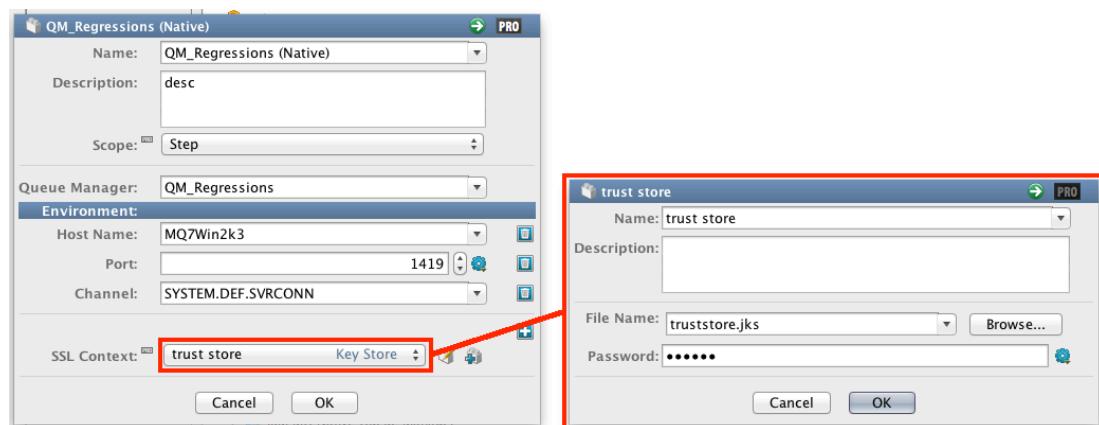
Historically, this scenario would have required the following global configuration in the **local.properties** file:

```
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.

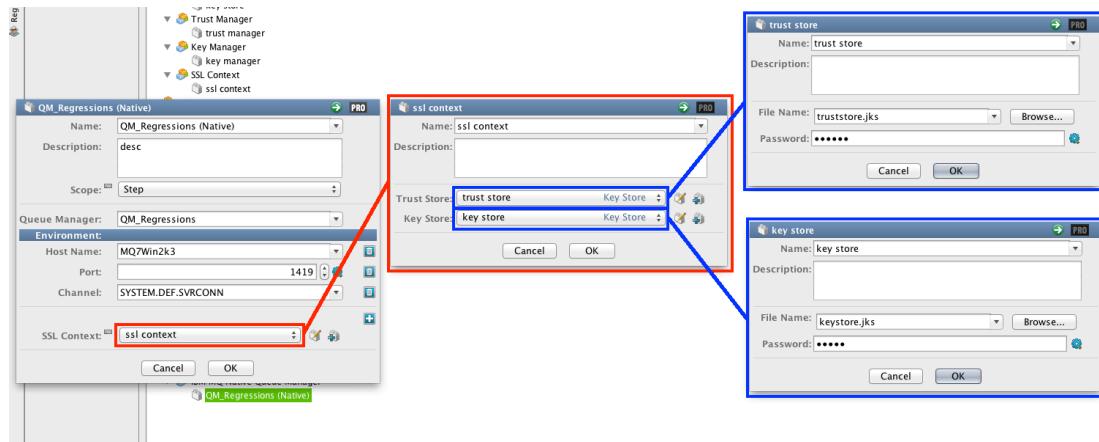
Historically, this scenario would have required the following global configuration in the **local.properties** file:

```
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.

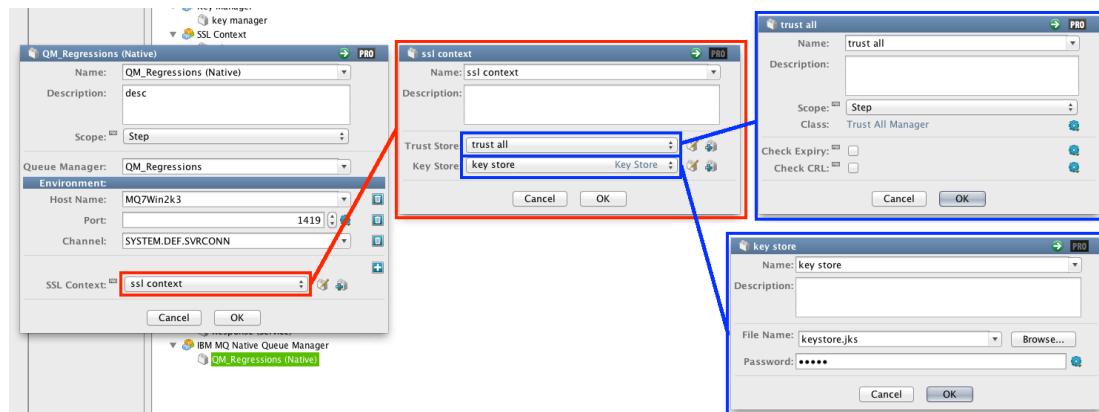
Advanced Scenarios

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.

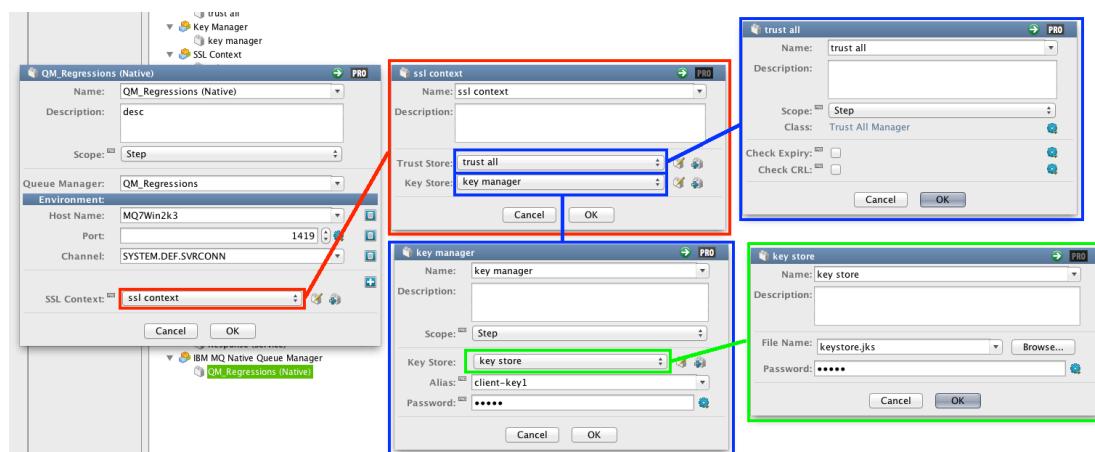


Note: If you want to use the Trust All Manager without a client-side key store, you can leave the Key Store field in the SSL Context blank. However, you cannot select a Trust All Manager directly from the SSL Context field in the IBM MQ Native Queue Manager asset.

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.



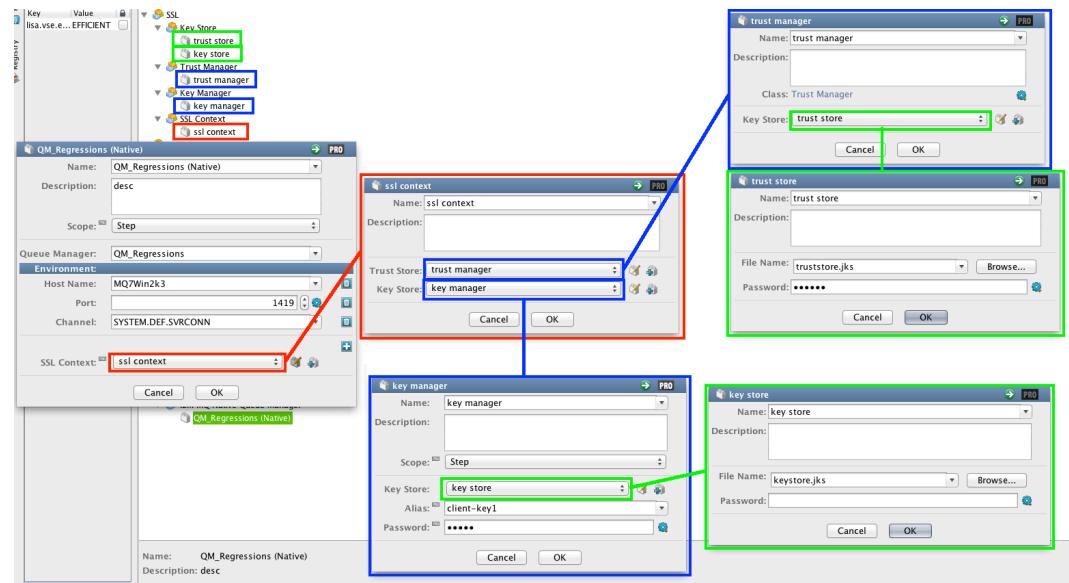
Screen capture of Key Manager asset scenario.

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

This section describes the following companions:

- [Web Browser Simulation Companion \(see page 1534\)](#)
- [Browser Bandwidth Simulation Companion \(see page 1535\)](#)
- [HTTP Connection Pool Companion \(see page 1535\)](#)

- [Configure DevTest to Use a Web Proxy Companion \(see page 1536\)](#)
- [Set Up a Synchronization Point Companion \(see page 1537\)](#)
- [Set Up an Aggregate Step Companion \(see page 1537\)](#)
- [Execute Event-driven Subprocess Companion \(see page 1538\)](#)
- [Observed System VSE Companion \(see page 1538\)](#)
- [VSE Think Scale Companion \(see page 1547\)](#)
- [Batch Response Think Time Companion \(see page 1549\)](#)
- [Recurring Period Think Time Companion \(see page 1550\)](#)
- [Create a Sandbox Class Loader for Each Test Companion \(see page 1552\)](#)
- [Set Final Step to Execute Companion \(see page 1552\)](#)
- [Negative Testing Companion \(see page 1553\)](#)
- [Fail Test Case Companion \(see page 1553\)](#)
- [XML Diff Ignored Nodes Companion \(see page 1553\)](#)
- [JSON Ignored Nodes Companion \(see page 1553\)](#)

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 enables you to limit the number of HTTP connections for each target server. This companion applies only to the HTTP/HTML Request, REST, and Raw SOAP Request test steps.

DevTest typically uses one HTTP connection for each virtual user. For example, if you run a test with 100 virtual users, then the client and the server each have 100 sockets open.

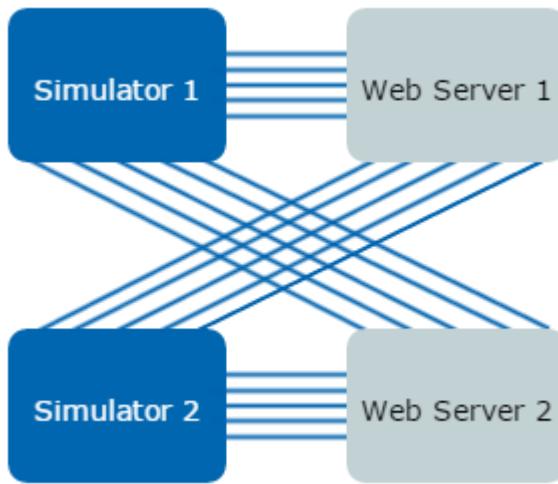
If you run a load test with thousands of virtual users for each simulator, the underlying operating system can run out of available sockets. In this scenario, use the HTTP Connection Pool companion.

The ConnectionsPerTargetHost 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, and a second 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 in the Web Proxy companion editor.

- **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

Use the Create a Synchronization Point companion to 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 Create 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 in the **Set Up an Aggregate Step** editor:

- **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 a specified event or events happens.

Enter the following parameters in the **Execute Event-driven Subprocess** editor:

- **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 may 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 \(see page 1788\)](#).

The default name for the Execute Event-driven Subprocess companion is *Execute Event-driven Subprocess Companion - name of subprocess test case*.

Observed System VSE Companion

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 VSE Observed System companion supports these requirements. If a virtual service will provide this behavior, add and configure this companion.

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 three 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.

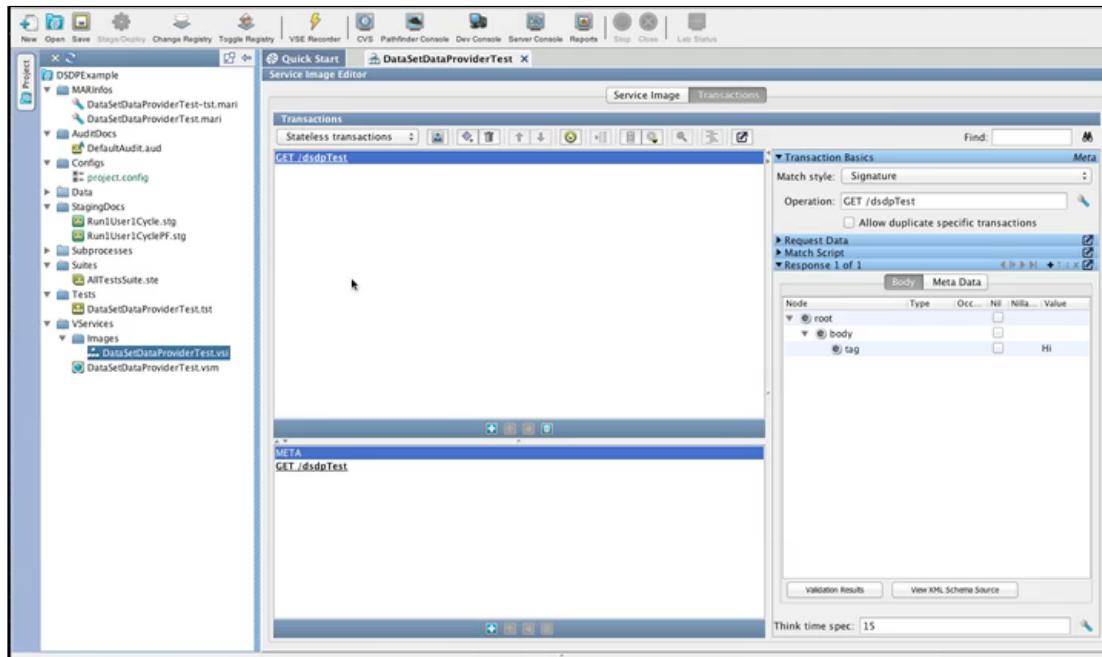


Note: For DevTest 8.2 and later, the **metric-data-service.jar** from CA APM must be in the lib /core folder in the DevTest directory.

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.

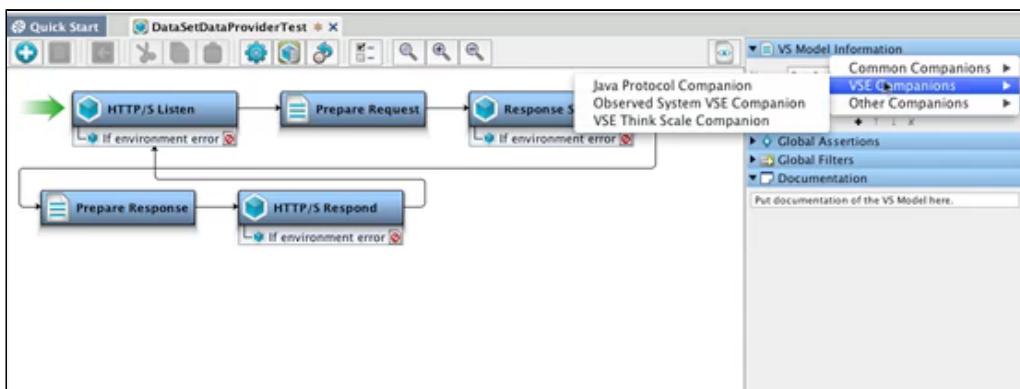
The service image that is shown contains one transaction, and the **Think time spec** is set to 15 milliseconds.



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.

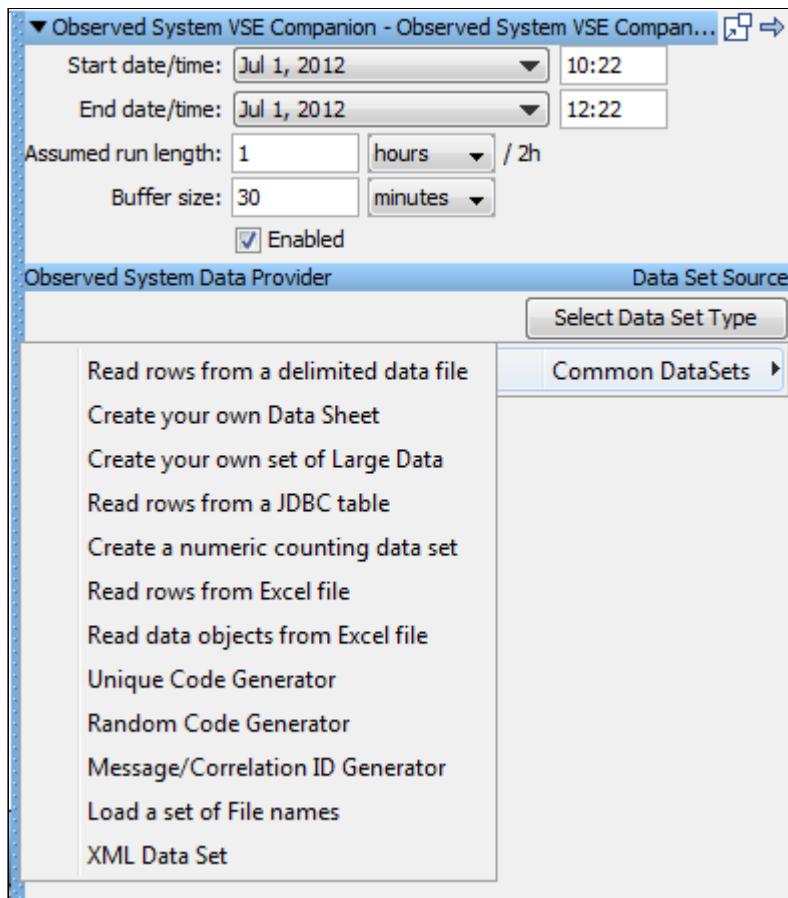
In this example, the Start date/time and End date/time define a two-hour window, but the Assumed run length is set to one hour. Therefore, for every one hour of VSE run time, the companion goes through two hours of data from the data set data provider. The buffer size of 30 minutes means that VSE accesses 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.

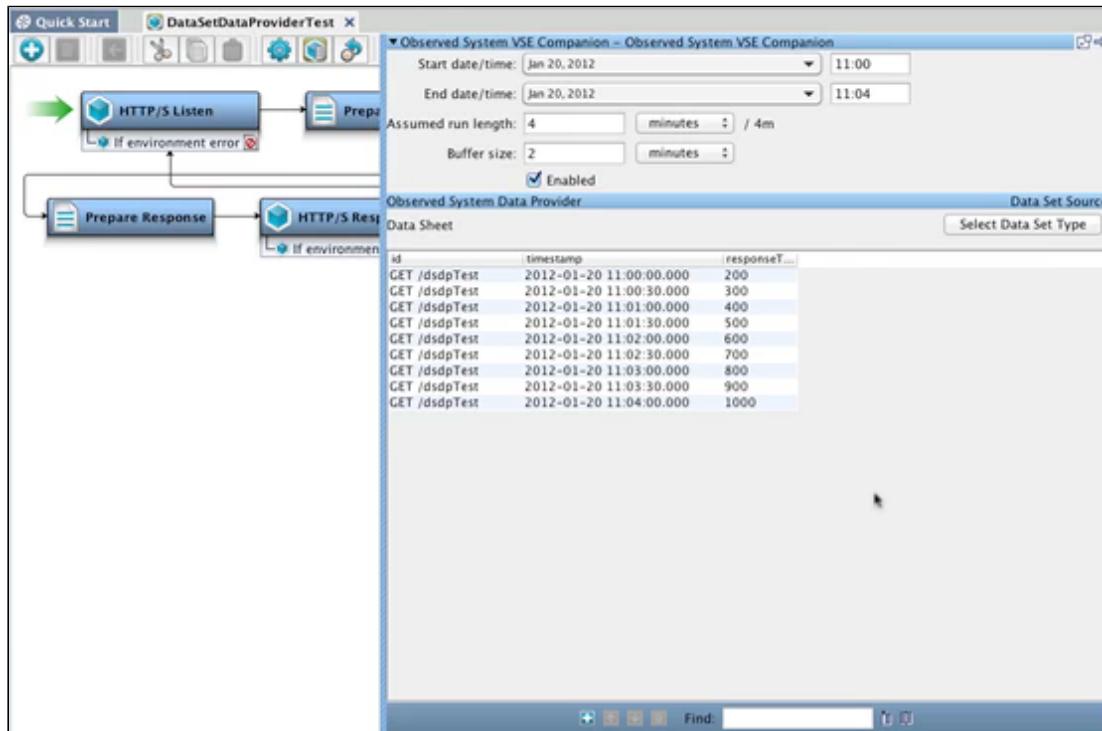
The screenshot shows the 'Observed System VSE Companion - Observed System VSE Companion' window. At the top, there are fields for 'Start date/time' (Jul 1, 2012, 10:22) and 'End date/time' (Jul 1, 2012, 12:22). Below these are 'Assumed run length' (1 hours / 2h) and 'Buffer size' (30 minutes). A checked checkbox labeled 'Enabled' is present. The main area is divided into 'Observed System Data Provider' and 'Data Set Source'. Under 'Data Sheet', there is a table:

id	timestamp	responseTi...
GET / dsdpTest	2012-07-01 10:30:00:00.000	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.

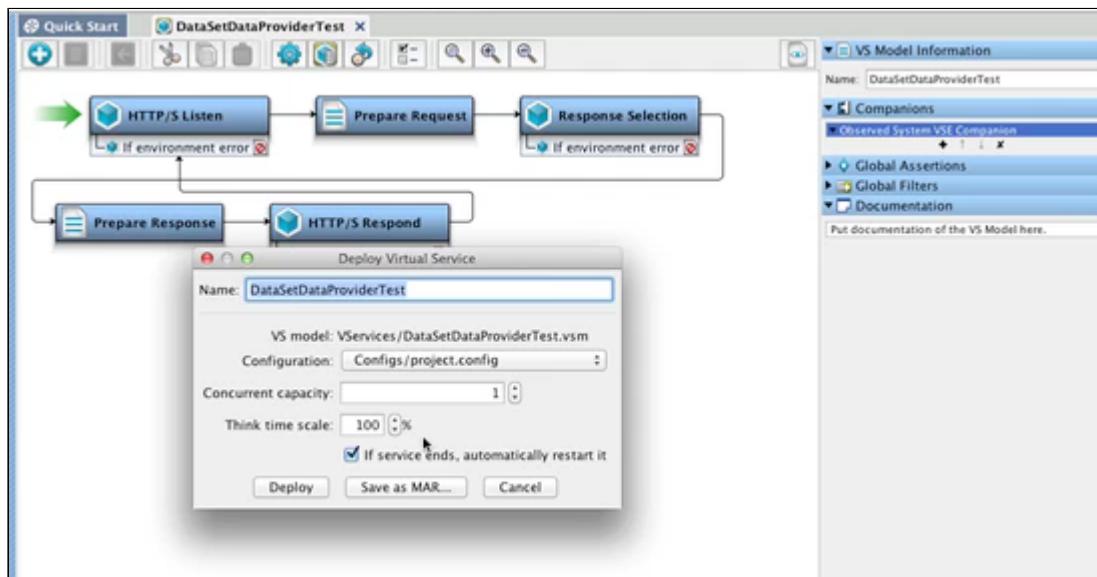


Observed System Companion example using data set as source

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.

Timestamp	Event	Simulator	Instance	Short Info	Long Info
2012-02-01 08:48:34,707	Test ended	0/0	32396239626...		
2012-02-01 08:48:33,998	Cycle ending	local	0/0	35366331303...	Sigaled to sto...
2012-02-01 08:48:33,997	Log message	local	0/0	Will execute th...	
2012-02-01 08:48:33,363	Log message	local	0/0	APMDataprovid...	N/A
2012-02-01 08:48:33,206	Cycle started	local	0/0	35366331303...	
2012-02-01 08:48:33,203	Test started	N/A	0/0	32396239626...	N/A

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.

Key	Value	Encrypt
WSERVER	192.168.168.129	<input type="checkbox"/>
lisa.vse.execution.mode	EFFICIENT	<input type="checkbox"/>
WSPORT	9081	<input type="checkbox"/>
DEBUG	true	<input type="checkbox"/>
lisa.http.pass.encrypted	f5504e2d23a7888253a27e8ef5260	<input type="checkbox"/>
LIVE_INVOCATION_SERVER	192.168.168.129	<input type="checkbox"/>
lisa.http.user.encrypted	e20fea06fd51ae896429ff7496ba4b0e	<input type="checkbox"/>
LIVE_INVOCATION_PORT	9081	<input type="checkbox"/>

Observed System Companion example Properties Editor

Run Time Priorities

The process that is followed at run time is:

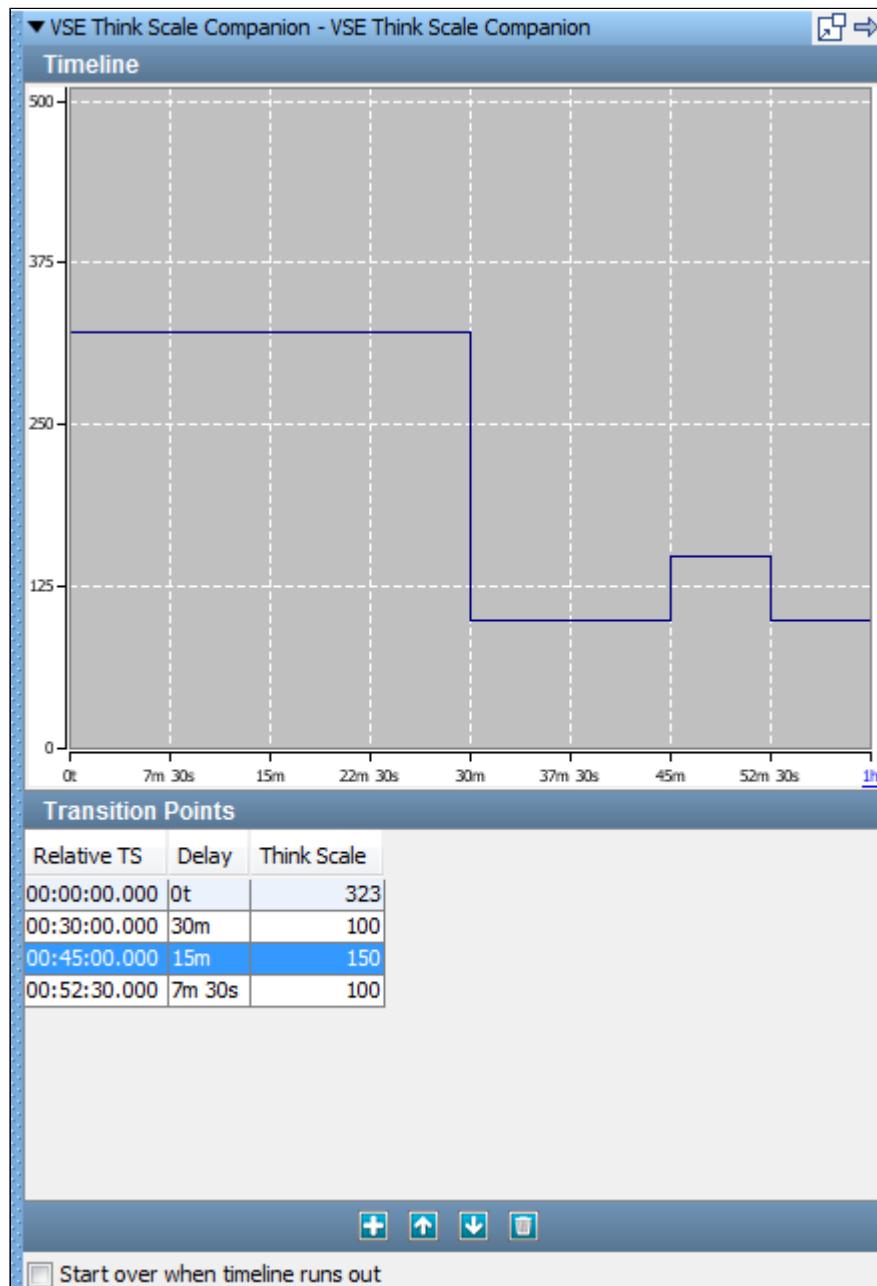
- The think time specification from the VSE response is examined when the delay factor is being determined.
- If the think time specification contains state (that is, a double-brace expression), it is evaluated directly and the result is used as the response time for the response.
- If the think time 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.

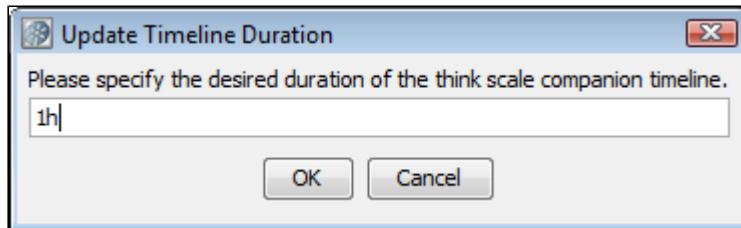
- **Think Scale**

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.

Create a Sandbox Class Loader for Each Test Companion

The Create a Sandbox Class Loader 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

Use the Set Final Step to Execute companion to 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 normally-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.

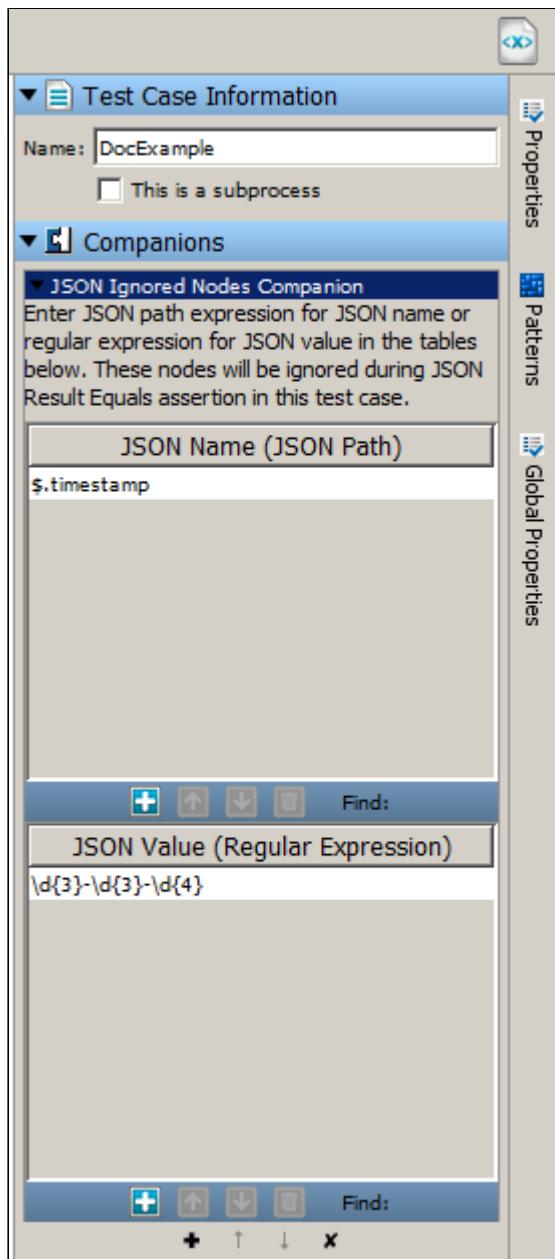
This companion has no configuration parameters.

XML Diff Ignored Nodes Companion

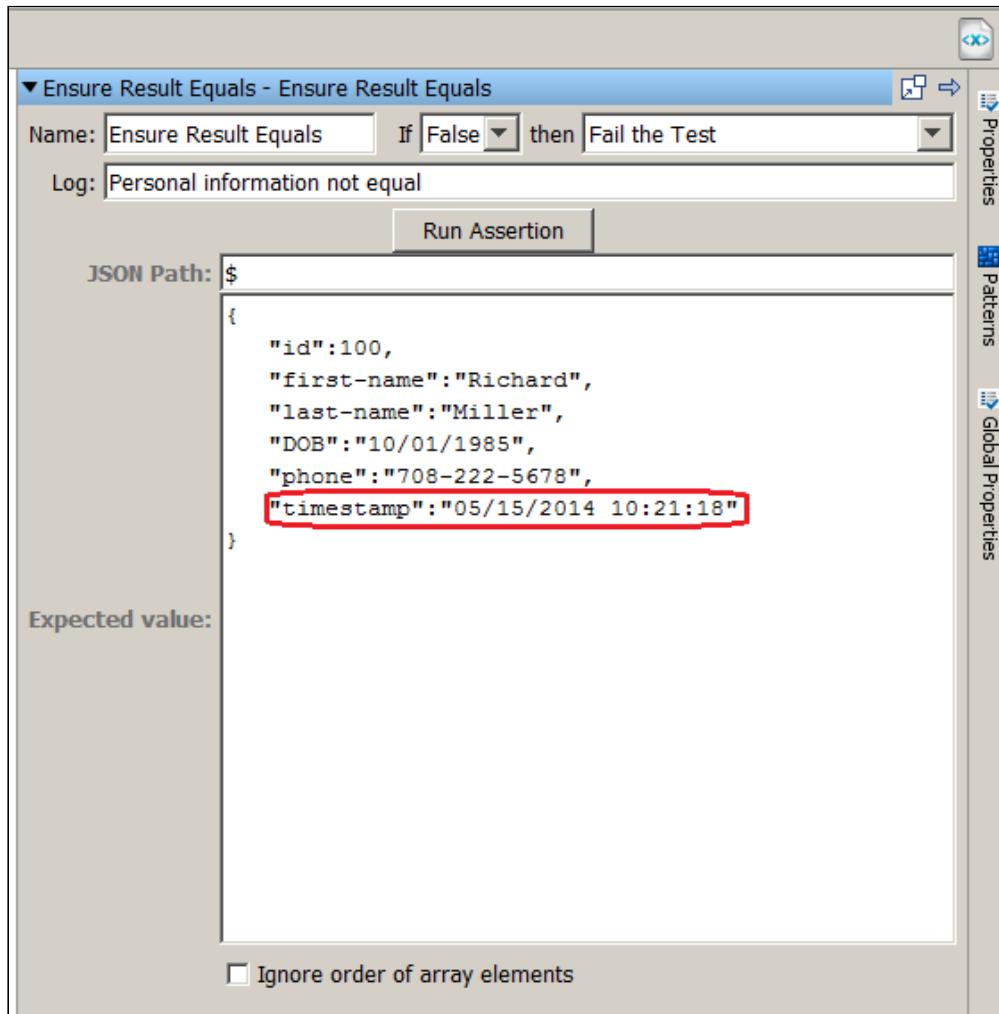
The XML Diff Ignored Nodes companion lets you enter XPath expressions that return one or more nodes in the following table. The companion uses an OR operation to collect these nodes, then ignores them during all XML diff comparisons for this test case.

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

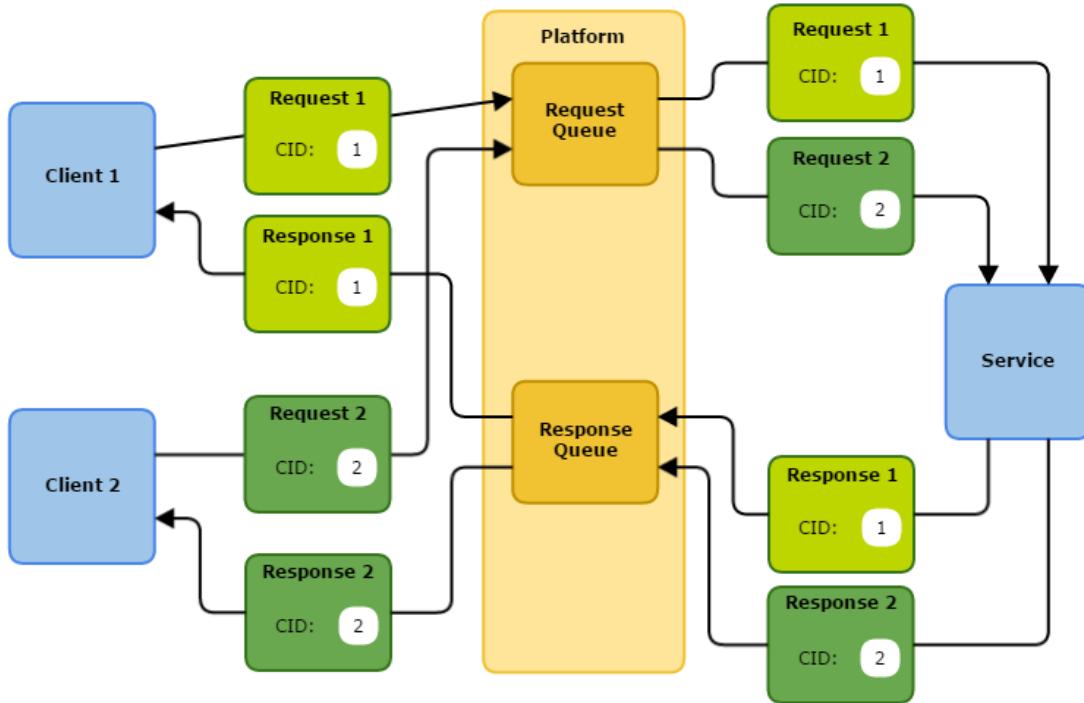
Contents

- [Correlation ID \(see page 1556\)](#)
- [Message ID to Correlation ID \(see page 1557\)](#)
- [Payload \(see page 1557\)](#)
- [ReplyTo and Temporary Response Queues \(see page 1558\)](#)

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.



You can enable correlation in the following test steps:

- [JMS Send Receive \(see page 1803\)](#)
- [IBM MQ Native Send Receive \(see page 1851\)](#)

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.

Correlation Scheme:	<input checked="" type="checkbox"/> Enabled	JMS Payload
Manual Value:	<input type="text"/>	
Reuse ID:	No	
Auto Generate:	Yes	
Payload Scheme:		
Request XPath:	request/id/getText()	
Response XPath:	response/id/getText()	

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

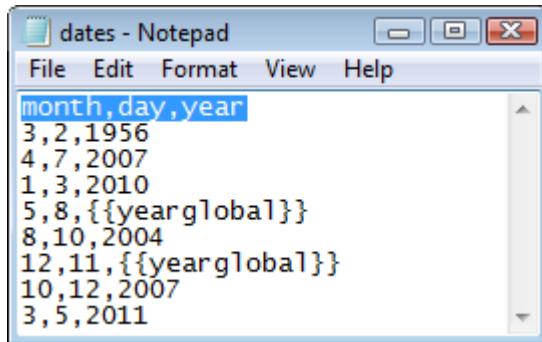
This section describes the following data sets:

- [Read Rows from a Delimited Data File Data Set \(see page 1558\)](#)
- [Create Your Own Data Sheet Data Set \(see page 1560\)](#)
- [Create Your Own Set of Large Data Data Set \(see page 1562\)](#)
- [Read Rows from a JDBC Table Data Set \(see page 1564\)](#)
- [Create a Numeric Counting Data Set \(see page 1565\)](#)
- [Read Rows from Excel File Data Set \(see page 1566\)](#)
- [Read DTOs from Excel File Data Set \(see page 1567\)](#)
- [Unique Code Generator Data Set \(see page 1573\)](#)
- [Random Code Generator Data Set \(see page 1573\)](#)
- [Message-Correlation ID Generator Data Set \(see page 1574\)](#)
- [Load a Set of File Names Data Set \(see page 1575\)](#)
- [XML Data Set \(see page 1576\)](#)

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.

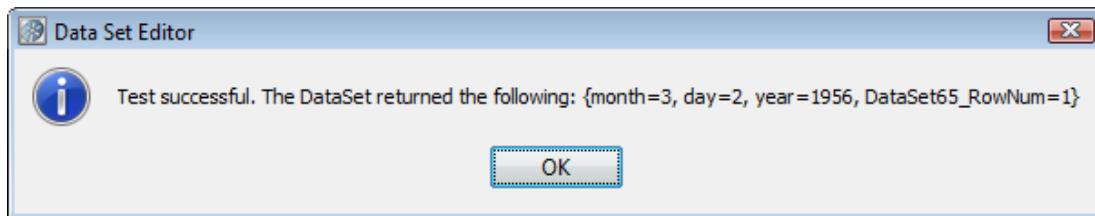
- **File Encoding**

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 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.

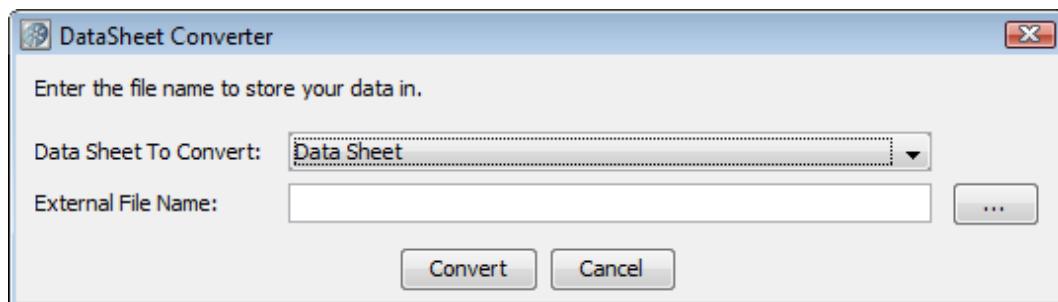


Test successful message dialog

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. A backing file name is the file in which all the data is stored.

▼ Create Your Own Set of Large Data - Create Your Own Set of Large Data

Name:	Create Your Own Set of Large Data																																						
<input checked="" type="checkbox"/> Local	<input checked="" type="checkbox"/> Random	Max Records To Fetch:	100																																				
At end of data, <input checked="" type="radio"/> Start over		<input type="radio"/> Execute																																					
Test and Keep																																							
Backing File Name: [LISA_PROJ_ROOT}\Data\Create Your Own Set of Large Data.lds		Browse	▼																																				
Create		Open																																					
<table border="1"> <thead> <tr> <th>Enabled</th> <th>firstName</th> <th>lastName</th> <th>userid</th> </tr> </thead> <tbody> <tr><td><input checked="" type="checkbox"/></td><td>Art</td><td>Katect</td><td>katar03</td></tr> <tr><td><input checked="" type="checkbox"/></td><td>Bruce</td><td>Adams</td><td>adabr02</td></tr> <tr><td><input checked="" type="checkbox"/></td><td>Cat</td><td>Taylor</td><td>tayca05</td></tr> <tr><td><input checked="" type="checkbox"/></td><td>Colter</td><td>Ames</td><td>ameco01</td></tr> <tr><td><input checked="" type="checkbox"/></td><td>Erin</td><td>Gillian</td><td>giler03</td></tr> <tr><td><input checked="" type="checkbox"/></td><td>Heather</td><td>Marley</td><td>marhe10</td></tr> <tr><td><input checked="" type="checkbox"/></td><td>Iris</td><td>Meier</td><td>meiir01</td></tr> <tr><td><input checked="" type="checkbox"/></td><td>Jeff</td><td>Hardy</td><td>harje09</td></tr> </tbody> </table>				Enabled	firstName	lastName	userid	<input checked="" type="checkbox"/>	Art	Katect	katar03	<input checked="" type="checkbox"/>	Bruce	Adams	adabr02	<input checked="" type="checkbox"/>	Cat	Taylor	tayca05	<input checked="" type="checkbox"/>	Colter	Ames	ameco01	<input checked="" type="checkbox"/>	Erin	Gillian	giler03	<input checked="" type="checkbox"/>	Heather	Marley	marhe10	<input checked="" type="checkbox"/>	Iris	Meier	meiir01	<input checked="" type="checkbox"/>	Jeff	Hardy	harje09
Enabled	firstName	lastName	userid																																				
<input checked="" type="checkbox"/>	Art	Katect	katar03																																				
<input checked="" type="checkbox"/>	Bruce	Adams	adabr02																																				
<input checked="" type="checkbox"/>	Cat	Taylor	tayca05																																				
<input checked="" type="checkbox"/>	Colter	Ames	ameco01																																				
<input checked="" type="checkbox"/>	Erin	Gillian	giler03																																				
<input checked="" type="checkbox"/>	Heather	Marley	marhe10																																				
<input checked="" type="checkbox"/>	Iris	Meier	meiir01																																				
<input checked="" type="checkbox"/>	Jeff	Hardy	harje09																																				

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.

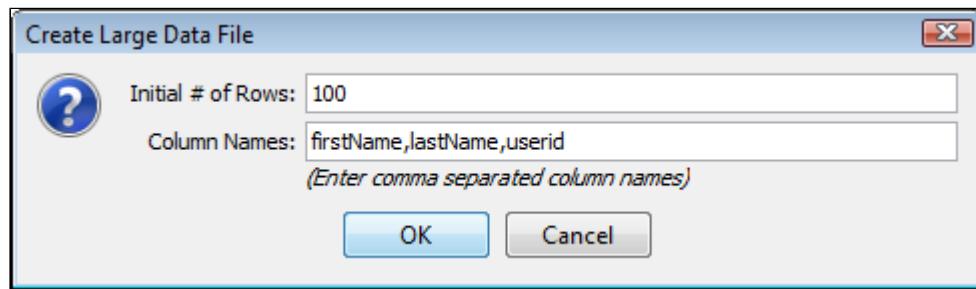
▪ **At End Of Data**

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 data is stored. This file is created automatically and data that you supply is inserted in it.

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.

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.

	A	B	C	D	E	F
1						
2		firstName	lastName	id	password	
3	Lynn	Parker	parly03	parkpass		
4	Paul	Martin	marpa02	martpass		
5	Jason	Sauer	sauja01	saupass		
6						

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 nonnullable types. If not used in the case of nonnullable 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.

The screenshot shows two tabs in an Excel application. The active tab is the 'Spec' tab, which contains the following information:

1	Spec:	com.itko.examples.dto.Customer									
2	DTO:	com.itko.examples.dto.Customer									
3	Docs:	No docs available									
4	Static Constructor Methods:	No methods available									
5	Static Constructor Fields:	No fields available									

The second tab is a 'Data' sheet, which contains the following data:

	Primary Key	balance	id	name	poAddr.city	poAddr.line1	poAddr.line2	poAddr.state	poAddr.zip	since	types
8	(unique per row)	double	int	String	String	String	String	String	String	Date	int[]
9											
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 two special fields in this sheet: "Primary Key", and the "reference the containing DTO" field that is used to link the rows in this sheet to the rows in the primary sheet.

	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 Key	com.itko.e.city	line1	line2	state	zip				
9	(unique per row)	(reference the containing DTO)	String	String	String	String				
10										
11										
12										
13										

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.

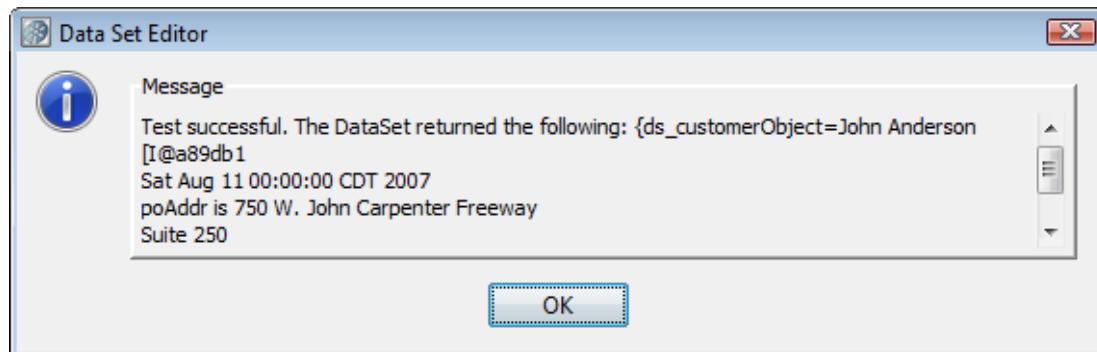
	A	B	C	D	E	F	G	H	I	J	K	L
1												
2	Spec:	com.itko.examples.dto.Customer										
3	DTO:	com.itko.examples.dto.Customer										
4	Docs:	No docs available										
5	Static Constructor Methods:	No methods available										
6	Static Constructor Fields:	No fields available										
7												
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 C Suite 250	TX			75024	8/11/2007	1,5,10	
11		2	250	102 Dirk Mar Plano	5800 Granite Pz Apt. 3455	TX			75031	6/29/2010	2,4,9	
12		3	800	103 Francis I Southlake	1376 Lakeview I 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 Key	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 \(see page 1701\)](#).

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 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.

Node	Occurs	Nil	Nillable	Value
addUserObject				
userObject				
accounts				
balance				101.01
name				Basic Checking
type				CHECKING
accounts				
balance				103.22
name				Orange Savings
type				SAVINGS
addresses				
city				Santa Clara
line1				USS Enterprise, NCC
state				California
zip				95343
addresses				
city				Dallas
line1				Deep Space Nine

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.

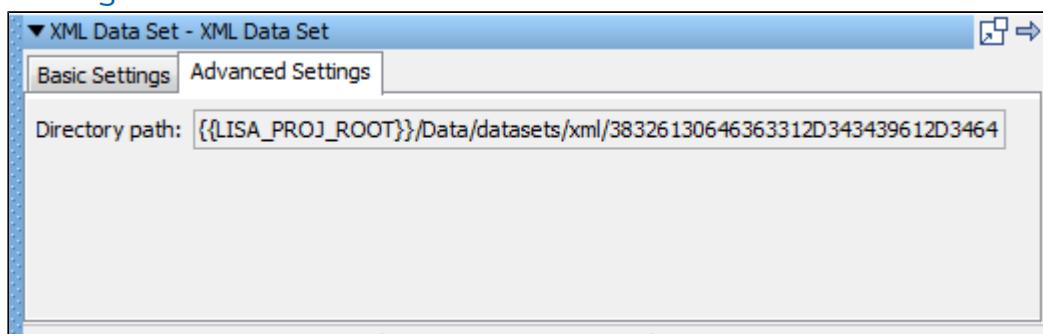
- **At End Of Data**

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

Node	Occurs	Nil	Nullable	Value
e addUserObject	<input type="checkbox"/>			
e userObject	<input type="checkbox"/>			
e accounts	<input type="checkbox"/>			
e balance	<input type="checkbox"/>			101.01
e name	<input type="checkbox"/>			Basic Checking
e type	<input type="checkbox"/>			CHECKING
e accounts	<input type="checkbox"/>			
e balance	<input type="checkbox"/>			103.22
e name	<input type="checkbox"/>			Orange Savings
e type	<input type="checkbox"/>			SAVINGS
e addresses	<input type="checkbox"/>			
e city	<input type="checkbox"/>			Santa Clara
e line1	<input type="checkbox"/>			USS Enterprise, NCC
e state	<input type="checkbox"/>			California
e zip	<input type="checkbox"/>			95343
e addresses	<input type="checkbox"/>			
e city	<input type="checkbox"/>			Dallas
e line1	<input type="checkbox"/>			Deep Space Nine

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

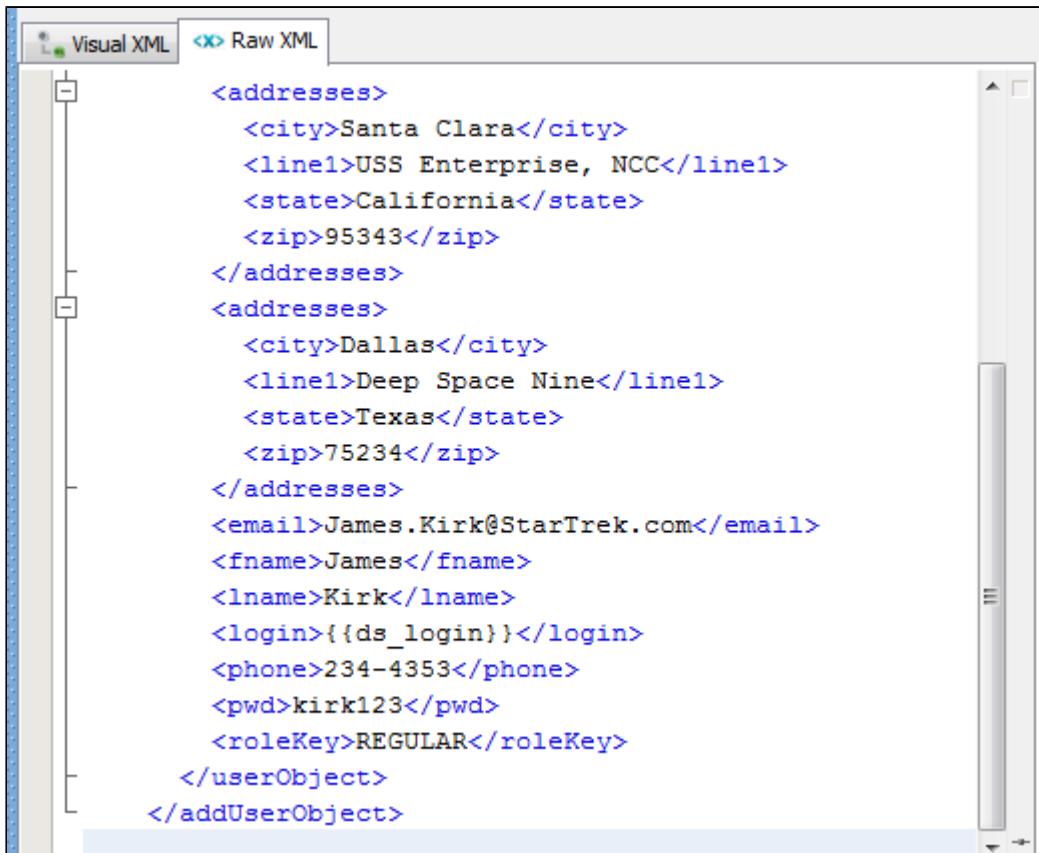
Node	Occurs	Nil	Nillable	Value
addUserObject	<input type="checkbox"/>			
userObject	<input type="checkbox"/>			
accounts	<input type="checkbox"/>			
balance	<input type="checkbox"/>			101.01
name	<input type="checkbox"/>			Basic Checking
type	<input type="checkbox"/>			CHECKING
accounts	<input type="checkbox"/>			
balance	<input type="checkbox"/>			103.22
name	<input type="checkbox"/>			Orange Savings
type	<input type="checkbox"/>			SAVINGS
addresses	<input type="checkbox"/>			
city	<input type="checkbox"/>			Santa Clara
line1	<input type="checkbox"/>			USS Enterprise, NCC
state	<input type="checkbox"/>			California
zip	<input type="checkbox"/>			95343
addresses	<input type="checkbox"/>			
city	<input type="checkbox"/>			Dallas
line1	<input type="checkbox"/>			Deep Space Nine

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



```

<addresses>
    <city>Santa Clara</city>
    <line1>USS Enterprise, NCC</line1>
    <state>California</state>
    <zip>95343</zip>
</addresses>
<addresses>
    <city>Dallas</city>
    <line1>Deep Space Nine</line1>
    <state>Texas</state>
    <zip>75234</zip>
</addresses>
<email>James.Kirk@StarTrek.com</email>
<fname>James</fname>
<lname>Kirk</lname>
<login>{ds_login}</login>
<phone>234-4353</phone>
<pwd>kirk123</pwd>
<roleKey>REGULAR</roleKey>
</userObject>
</addUserObject>

```

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.

Event Descriptions

The following table describes the standard events.

Event Description	Short Info	Long Info
Name		
Abort This event ends a test in a "cannot finish" state. It is a failing type of end event.	Event Aborted	
Assert A non-embedded assertion will generate this event if it does not fire. These events are the evaluated assertions that did not get to execute their asserted consequence.	The name of the assertion	

Asser	A non-embedded assertion will generate this event if it does fire. Firing means it is true and assertion fired 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	Steps that perform object calls like web made 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	Steps that perform object calls like web result 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.
Config ure for load test		Configure for load test	The test run has been configured as a load test. Selecting individual events has been disabled.
Coord inato nde d	A coordinator has been removed.	The name of the coordinator server	
Coord inato serve r ende d	The coordinator server was ended.	The name of the coordinator server	
Coord inato serve r starte d	The coordinator server was created.	The name of the coordinator server	
Coord inato starte d	A new coordinator was created.	The name of the coordinator server	
Cycle ende d norm ally	The test execution completed in a successful state.	The name of the test case	
Cycle g	The instances have been instructed to stop testing.		
Cycle failed		The name of the test case	

	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.	
Cycle	Every model that runs will generate one of histor these events. It is the final trace of all the details of its run.	Cycle History
Cycle	The test has been initialized (loaded).	The name of the test case
Cycle	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
Cycle	This test instance and cycle have just started.	The name of the test case
Data	Data sets generate an event that makes it clear what values are about to be used.	Data set read
HTTP	This event is generated for every HTTP transaction that DevTest executes to capture the performance statistics.	HTTP performance statistics
Info	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
Instance	Sent when a simulator instance has finished.	The name of the simulator
Instance	Sent when a simulator creates an instance.	The name of the simulator
Log	Basic logging data. Can be turned off to minimize overhead when filtering events.	The message sent to the log
Metric	A metric has been collected and is reporting its value.	The short name of the metric, for example, D evTest: Avg Response Time
	Metrics that are collected generate real-time events of their values.	Metric Started

Metrics	Metrics that are collected generate real-time events of their values.	Metric value
ModeError	A test case error was discovered during execution of the test. For example, the name definition 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
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
PropertyRemoved	A property was removed.	The property key The word "<removed>"
PropertySet	A property was set.	The property key The property value
SimulatorEnded	Sent when a simulator has ended.	The name of the simulator
SimulatorStarted	Sent when a simulator has started.	The name of the simulator
DataConsumed	Approximate amount of data sent and received from the system under test for the width step execution.	The name of the step Actual number of bytes read/received
ErrorTestFailed	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
StepHistory	Every step has a history event that fires off its long info.	Step history
RequestReported	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 status bandwidth			
Step completed against the system response under test.	The name of the step	The response data as a string	
Step response status bandwidth			
Step The amount of time a step took to execute response against the system under test.	The name of the step	The number of milliseconds to execute the step	
Step A step is being executed.	The name of the step		
Step Every step has a target, such as the URL for a target web request or the JNDI name of an EJB.			
Step A warning was recorded. For example, a filter warning took the default value because it could not find the current value.	Step warning		
Subprocess A subprocess has finished executing.			
Subprocess A subprocess has started.			
Suite 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 All the tests running as part of a suite have ended.	The name of the suite		
Suite 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 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	
A suite is starting to run.	The name of the suite		

Suite starte d		
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 ende d	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 starte d	Sent when the coordinator starts the test.	The name of the test case
VS log message	VSE internal logging	VS log message
VS trans actio n matc h	A virtual service did not match a transaction request to at least one recorded response.	VS transaction no match
VS servic e ende d	A virtual service was stopped.	VS service stopped
VS servic e starte d	A virtual service was started.	VS service started
	A virtual service finished processing a transaction request.	

VS
trans
actio
n
finish
ed

VS A virtual service matched a transaction VS transaction match
trans request to at least one recorded response.

actio
n
matc
h

VSE A service reset request was made of a server. VSE server resets
serve
r
reset

VSE A virtual service environment was asked to VSE server shutdown
serve shut down.
r
shutd
own

VSE A service stop request was made of a server. VSE server stops
serve
r
stop

Filter Descriptions

This section describes the following filters:

- [Utility Filters \(see page 1588\)](#)
- [Database Filters \(see page 1593\)](#)
- [Messaging/ESB Filters \(see page 1599\)](#)
- [HTTP/HTML Filters \(see page 1601\)](#)
- [XML Filters \(see page 1616\)](#)
- [JSON Filters \(see page 1621\)](#)
- [Java Filters \(see page 1622\)](#)
- [VSE Filters \(see page 1624\)](#)
- [CAI Filters \(see page 1625\)](#)
- [Copybook Filters \(see page 1626\)](#)

Regular expressions are used for comparisons in several filters. For more information about regular expressions, see [Regular Expressions \(http://psoug.org/reference/regexp.html\)](http://psoug.org/reference/regexp.html).

Utility Filters

The following filters are available in the Utility Filters list for any test step:

- [Create Property Based on Surrounding Values \(see page 1588\)](#)
- [Store Step Response \(see page 1590\)](#)
- [Override Last Response Property \(see page 1591\)](#)
- [Save Property Value to File \(see page 1591\)](#)
- [Parse Property Value As Argument String \(see page 1592\)](#)
- [Save Property from One Key to Another \(see page 1592\)](#)
- [Time Stamp Filter \(see page 1593\)](#)

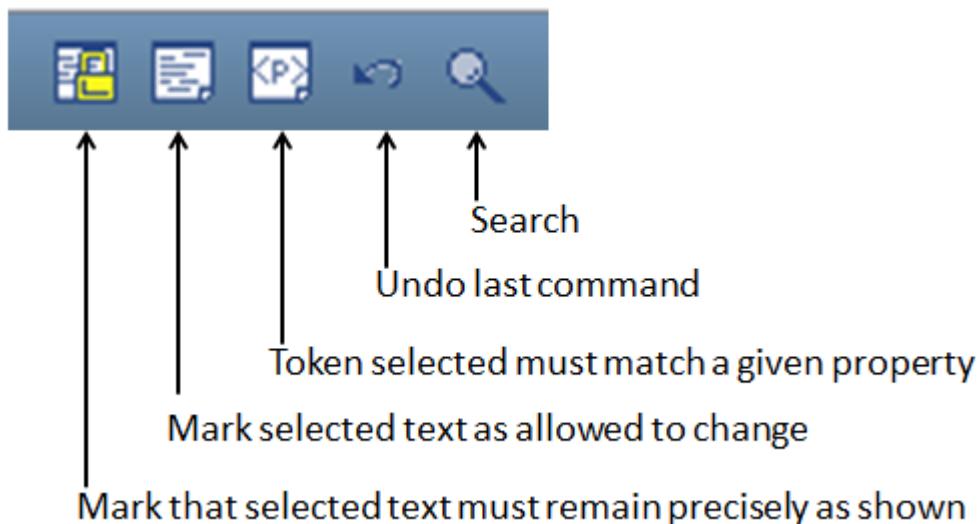
Create Property Based on Surrounding Values

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

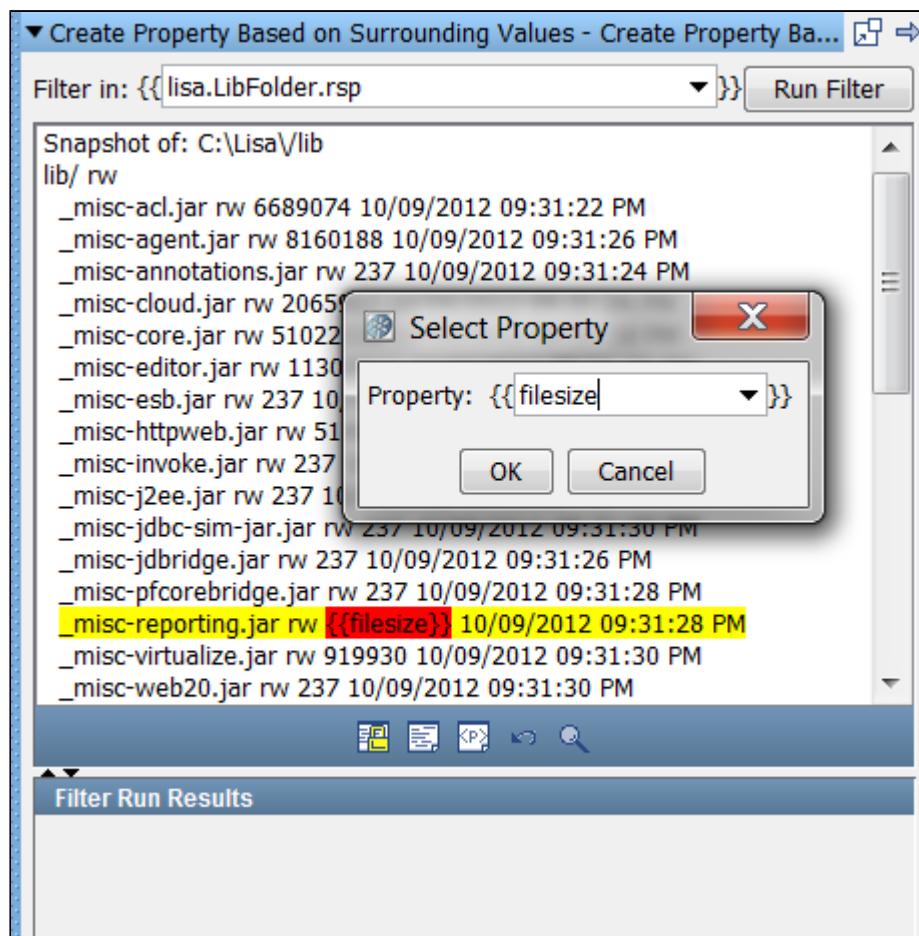
In the following example, the goal is to store the size of a specific file in a property.

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.

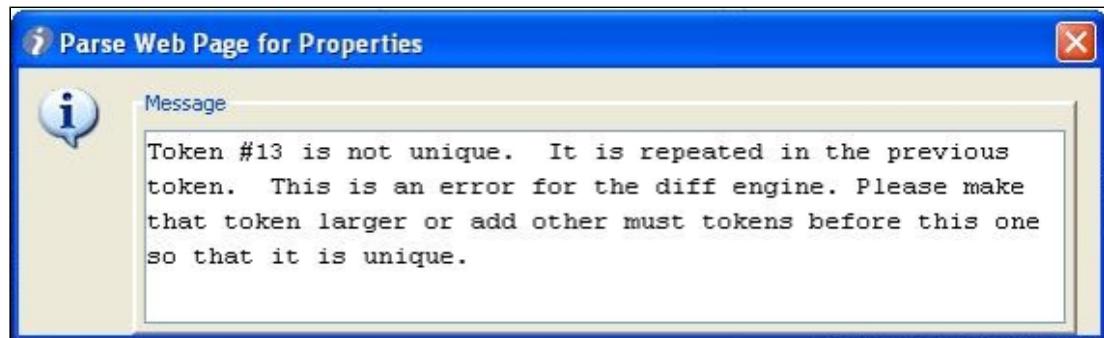
The property name was then entered into the dialog. The actual value of the file size has been replaced with the name of the property.

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. The example shows the JSESSIONID attribute.

- **Property**

The name of the property to store the text of the attribute. The example shows sessionID.

- **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 \(see page 1594\)](#)
- [Size of JDBC Result Set \(see page 1596\)](#)
- [Set Size of a Result Set to a Property \(see page 1596\)](#)
- [Get Value For Another Value in a ResultSet Row \(see page 1597\)](#)

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.

▼ SQL Database Execution (JDBC) - Verify User Added

Result Set

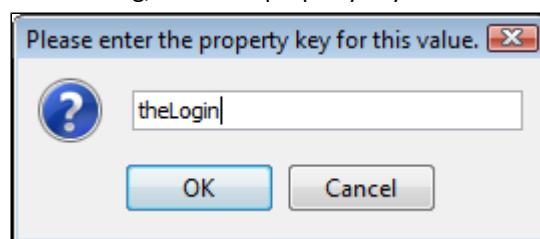
LOGIN	PWD	NEWFLAG	FNAME	LNAME	EMAIL	PHONE	ROLEKEY	SSN
dmxxxx-009	tIDAdNa3...	1	first-9	last-9	test@test...		1	
Testuser	IGyAQTu...	0	Test	User	anne@itk...	817-433-...	1	433-87-3...
TEST1	nU4eI71b...	1					1	
First1	8FePHnF0...	1	Firstname1	Lastname1	first1@itk...	555-555-...	1	555-55-8...
Last1	i+UhJqb9...	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	ODPiKuNir...	1	ITKO	Admin	lisabank-a...	123-4567	2	434-47-5...
sbellum	26yJsXNp...	1	Sara	Bellum	sbellum@...	232-4345	1	614-40-1...
wpiece	/JuJ0Me...	1	Warren	Piece	wpiece@...	455-3232	1	546-71-4...
areck	AHDRRjD...	1	Amanda	Reckonwith	areck@my...	555-2244	1	350-02-1...
boaty	RQIi0Ldp...	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...

Generate Filter for Current Col/Row Value

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 ResultSet Row

The Get Value for Another Value in a ResultSet 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:

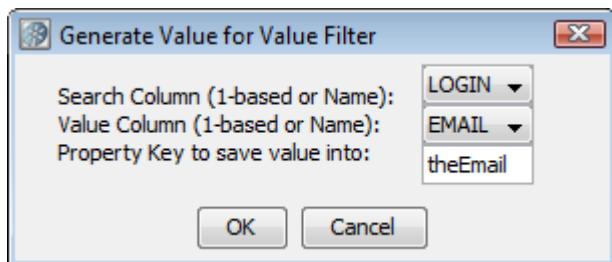
1. Display the step response that contains the result set.
2. From the result set select the two values in different columns, using the Ctrl key.

▼ 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@tk...	817-433...	1	433-87-3...	
TEST1	nU4eI71b...	1					1		
First1	8FePHnF0...	1	Firstname1	Lastname1	first1@tk...	555-555...	1	555-55-8...	
Last1	i+UhJqb9...	1	Firstname2	Lastname2	last1@tko...	333-448...	1	533-88-9...	
test1	nU4eI71b...	1	First1	Last1	test1@tk...	214-111...	1	111-11-1...	
test2	nU4eI71b...	1	First2	Last2	test2@tk...	215-222...	1	222-22-2...	
demo	89yJPVNn...	0	DEMOFIRST	DEMOLAST	demo@tk...	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	/JuJ0Me...	1	Warren	Piece	wpiece@...	455-3232	1	546-71-4...	
areck	AHDRRjD...	1	Amanda	Reckonwith	areck@my...	555-2244	1	350-02-1...	
boaty	RQIi0Ldp...	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 \(see page 1599\)](#)
- [Convert an MQ Message to a VSE Request \(see page 1600\)](#)
- [Convert a JMS Message to a VSE Request \(see page 1600\)](#)

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:

- [Create Resultset from HTML Table Rows \(see page 1601\)](#)
- [Parse Web Page for Properties \(see page 1603\)](#)
- [Parse HTML/XML Result for the Value of a Specific Tag or Attribute \(see page 1606\)](#)
- [Parse HTML Result for Specific Value and Parse It \(see page 1608\)](#)
- [Parse HTML Result for the Child Text of a Tag \(see page 1609\)](#)
- [Parse HTML Result for HTTP Header Value \(see page 1609\)](#)
- [Parse HTML Result for the Value of an Attribute \(see page 1610\)](#)
- [Parse HTML Result for DevTest Tags \(see page 1611\)](#)
- [Choose Random HTML Attribute \(see page 1611\)](#)
- [Parse HTML into List of Attributes \(see page 1612\)](#)
- [Parse HTTP Header Cookies \(see page 1613\)](#)
- [Dynamic Form Filter \(see page 1613\)](#)
- [Parse HTML Result by Searching Tag Attribute Values \(see page 1614\)](#)
- [Inject HTTP Header \(see page 1615\)](#)

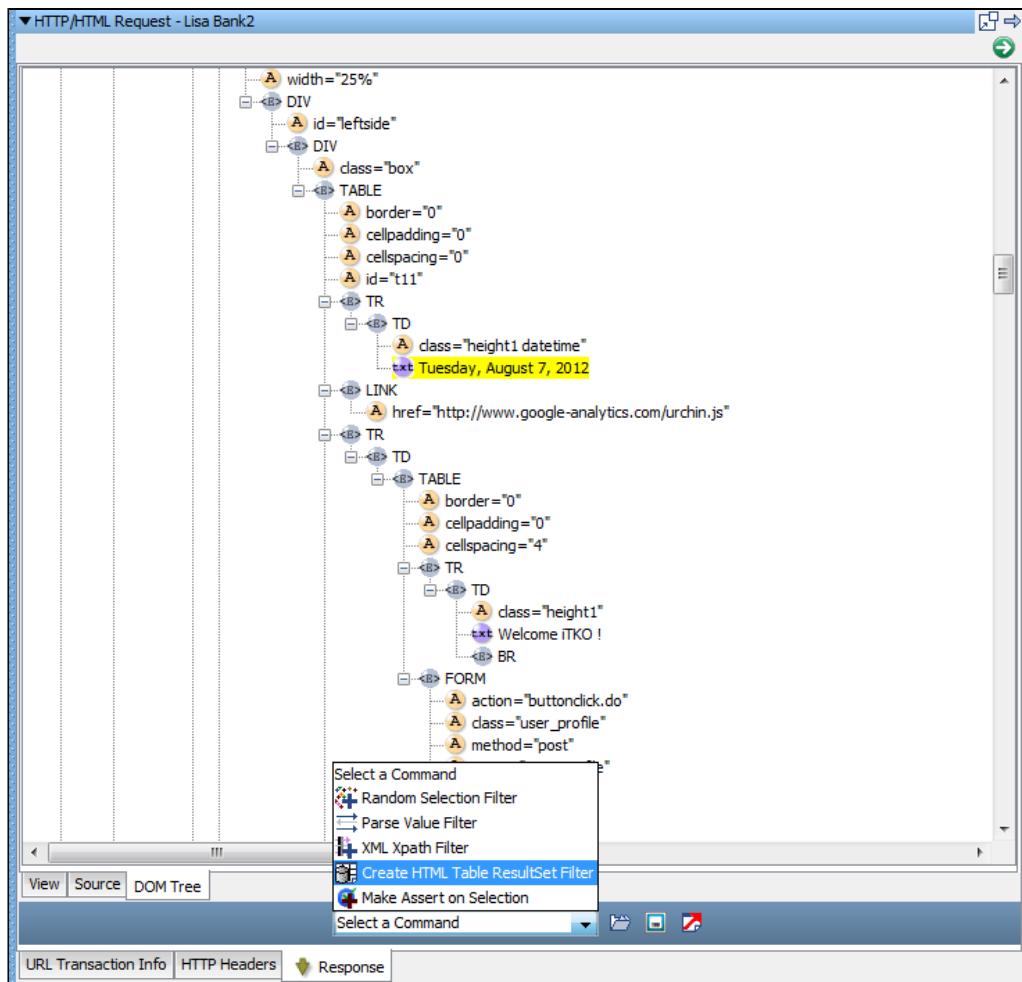
Create Resultset from HTML Table Rows

To create a result set (for example, a JDBC result set) from an HTML table that the response returns, use the Create Resultset from HTML Table Rows filter. 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.

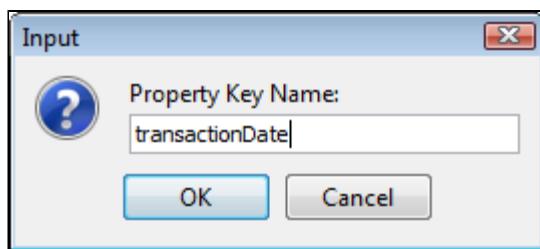
To create a filter on a table:

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 - transactionDate

Filter in: {{lisa.Lisa Bank2.rsp}}

Attributes

Save Into Property Key: {{transactionDate}}

HTML Table for Search: TABLE[3]

Defined Element Paths: 2.null.TD[0].TD[0]

Run Filter

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.

Save Property as Last Response - Save Property as Last Response

Property Key to Save as Step Result: {{transactionDate}}

Result Set

TD1-TXT-VALUE
Tuesday, August 7, 2012
Welcome ITKO !

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.

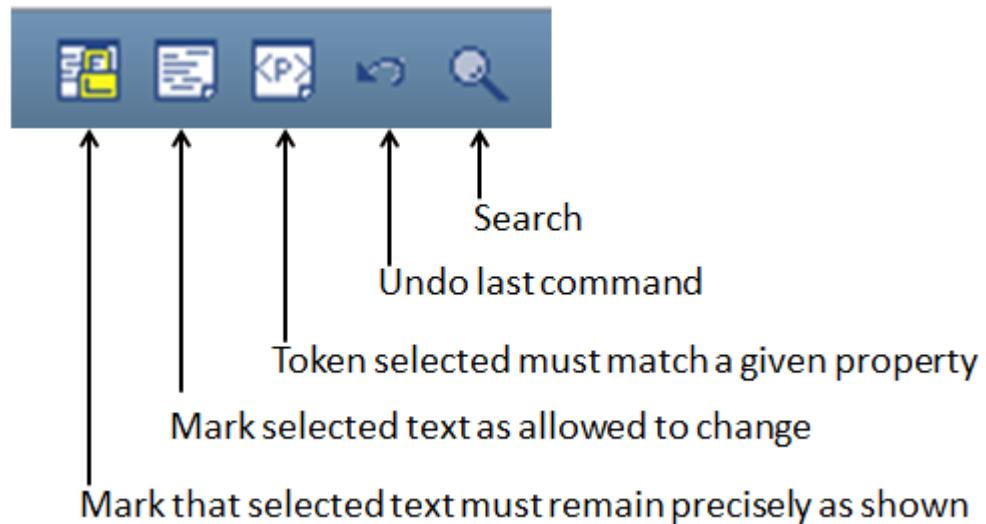
Parse Web Page for Properties

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.

The screenshot illustrates the 'Parse Web Page for Properties' feature. The top panel shows a browser window with a login dialog box. The dialog box has a title 'Select Property' and a field 'Property: {{Website Title}}'. The bottom panel shows the raw HTML code of the page, which includes the title tag <title>LisaBank - Home</title> and a link tag <link href="/isabank/css/financial.css" rel="stylesheet" type="text/css">. The browser window shows a globe graphic and a menu bar with 'Help | Log Out'.

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** .

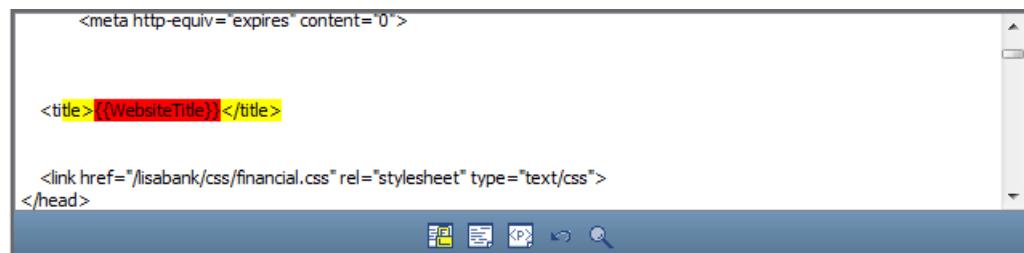
In this example, the website name text, "LisaBank - Home," is selected.

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.



A screenshot of the DevTest Solutions interface showing the HTML editor. The code in the editor is:

```
<meta http-equiv="expires" content="0">

<title>{{WebsiteTitle}}</title>

<link href="/lisabank/css/financial.css" rel="stylesheet" type="text/css">
</head>
```

The text "{{WebsiteTitle}}" is highlighted with a red background, indicating it is a property block. The toolbar at the bottom of the editor window includes icons for file operations, copy/paste, and search.



Note: All **Property** blocks must always be bounded by **Must** blocks.

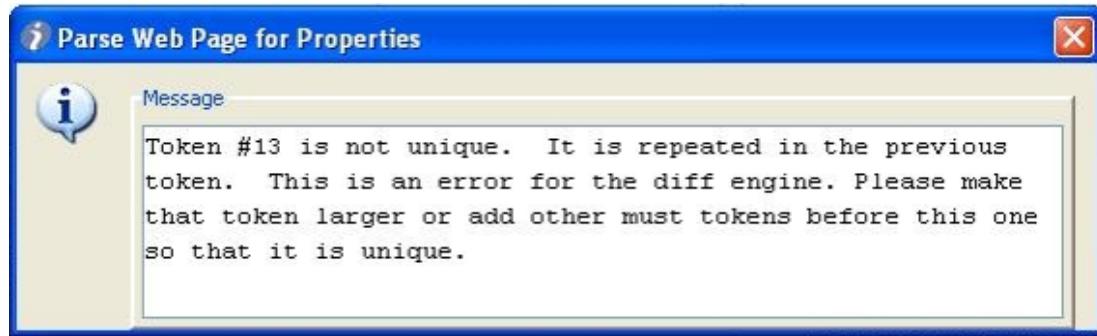
Frequently you can do this action purely from the web page view by selecting the content in the web browser. It can be easier to click the web browser in the area that you want to select, and then make your selection in the HTML panel.

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

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**

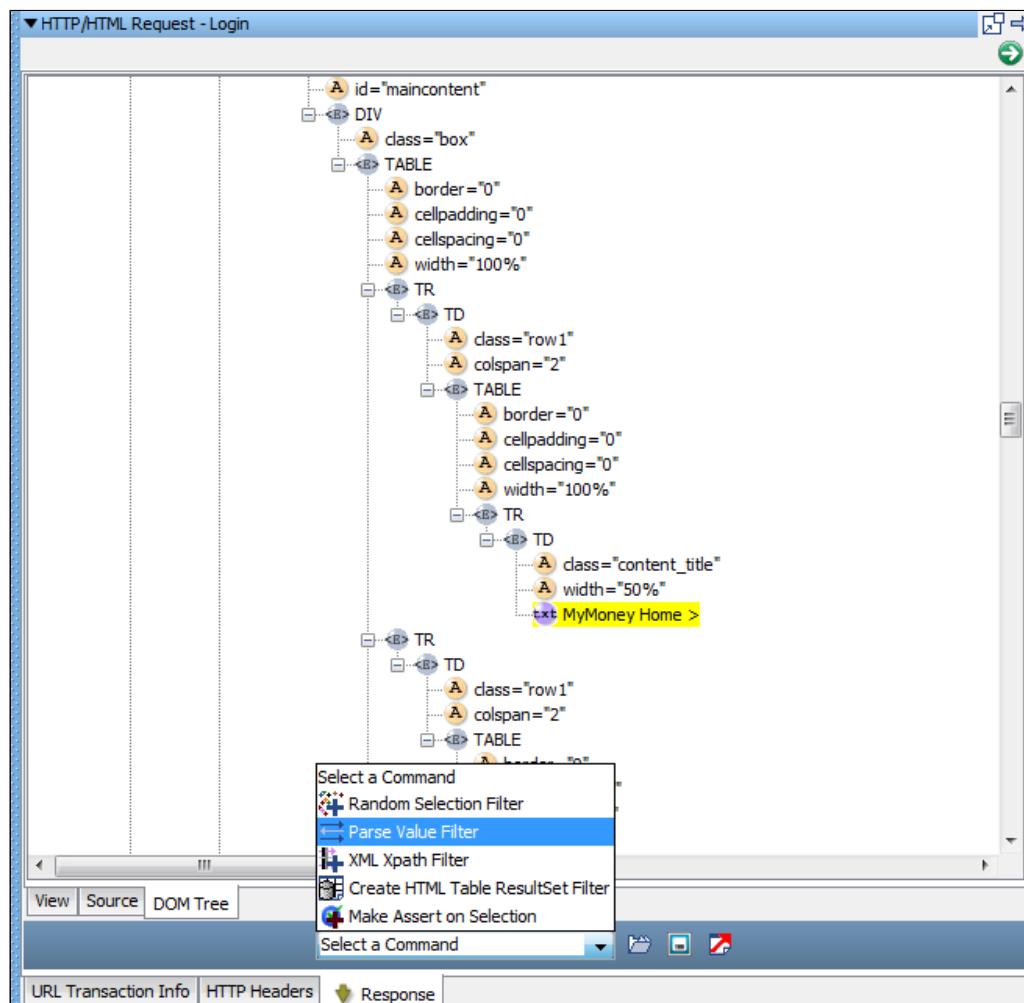
Displays the property and values that result from running the filter.

■ Run Filter

To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

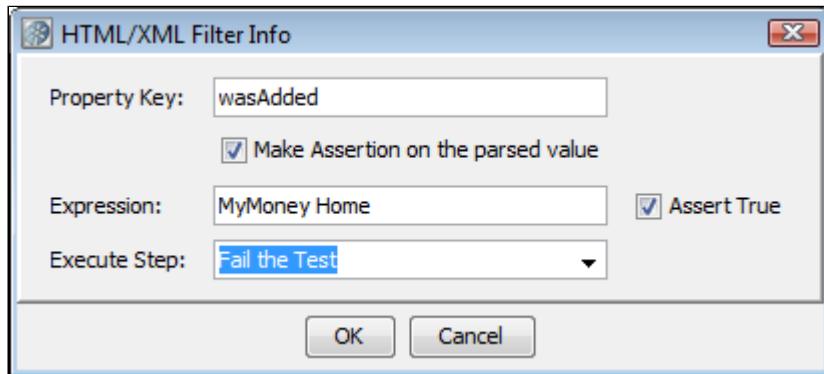
To create the filter from the HTTP/HTML request step response page:

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 Tags

The Parse HTML Result for DevTest Tags 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**

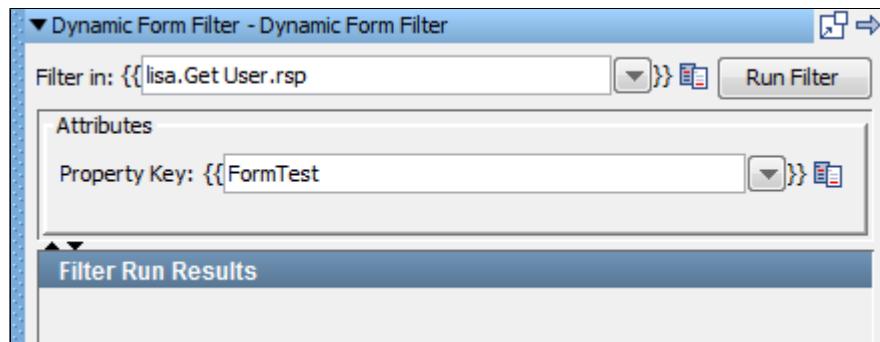
To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

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

To filter an attribute value by searching the tag for the name and value of another attribute, use the Parse HTML Result by Searching Tag/Attribute Values filter. 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.

This filter is a global filter.

The following test steps are supported:

- HTTP/HTML Request
- Raw SOAP 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.



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 \(see page 1616\)](#)
- [Read Attribute from XML Tag \(see page 1618\)](#)
- [Parse XML Result for DevTest Tags \(see page 1620\)](#)
- [Choose Random XML Attribute \(see page 1620\)](#)
- [XML XPath Filter \(see page 1620\)](#)

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".

▪ **Tag Count**

The occurrence of the tag. For example, if you want the child text of the first multiRef tag, set the count to "1".

▪ **Property**

The property in which to store the value.

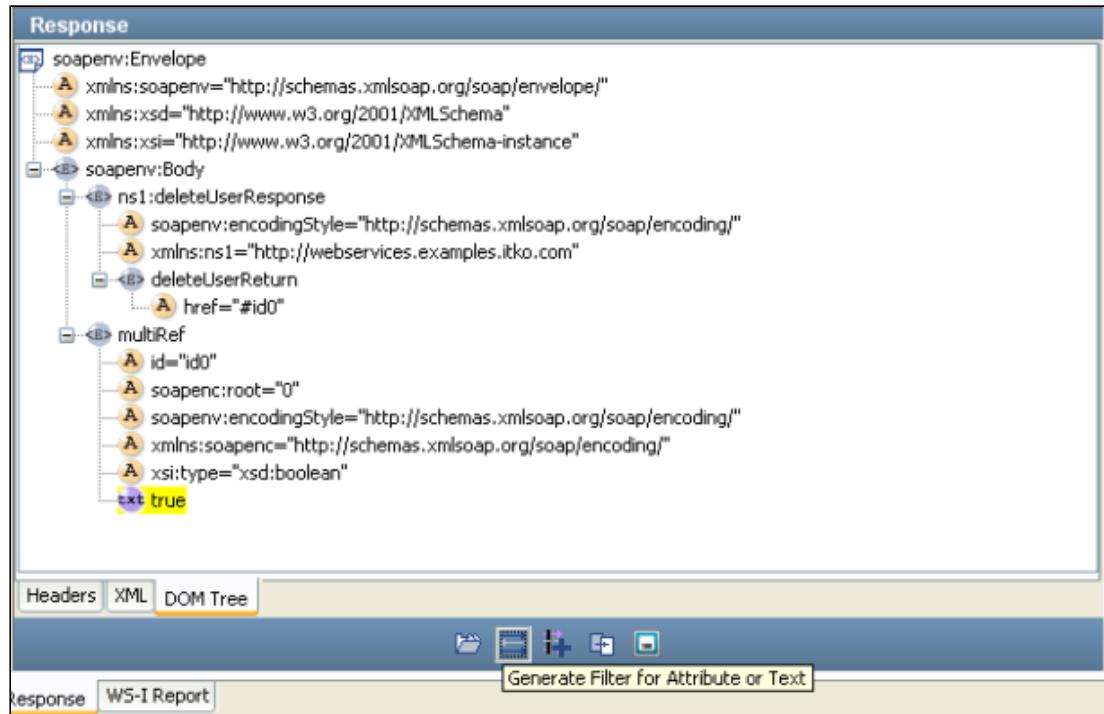
▪ **Filter Run Results**

Displays the property and values that result from running the filter.

▪ Run Filter

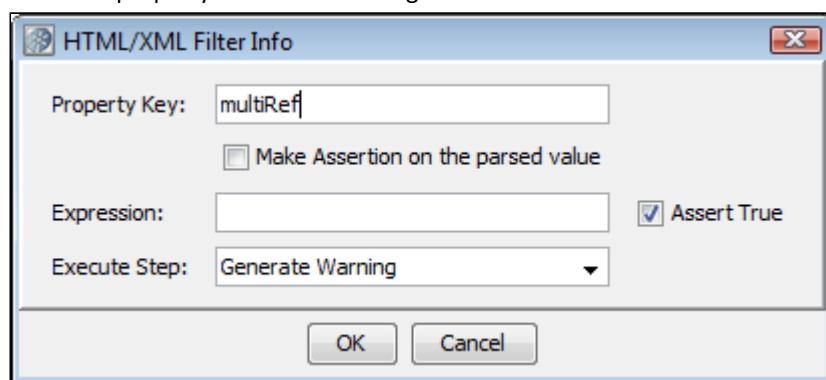
To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

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**

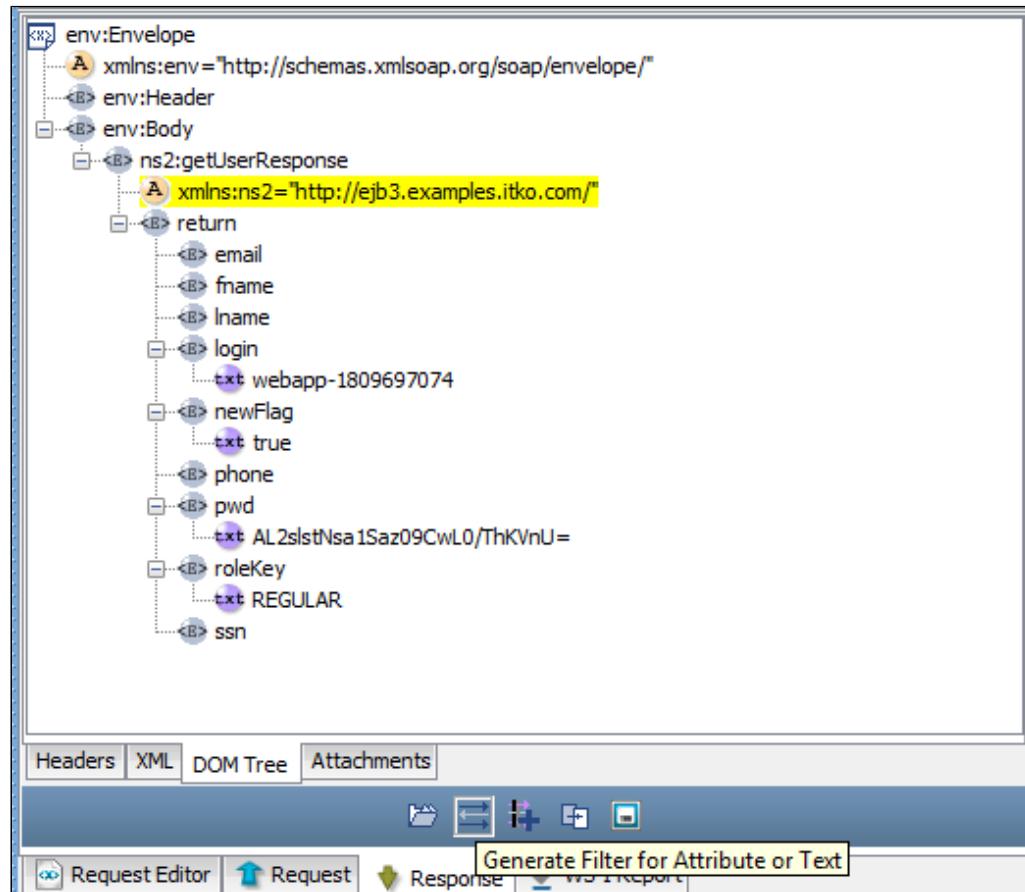
Displays the property and values that result from running the filter.

- **Run Filter**

To run and execute the filter, click **Run Filter**. The results appear in the **Filter Run Results** section.

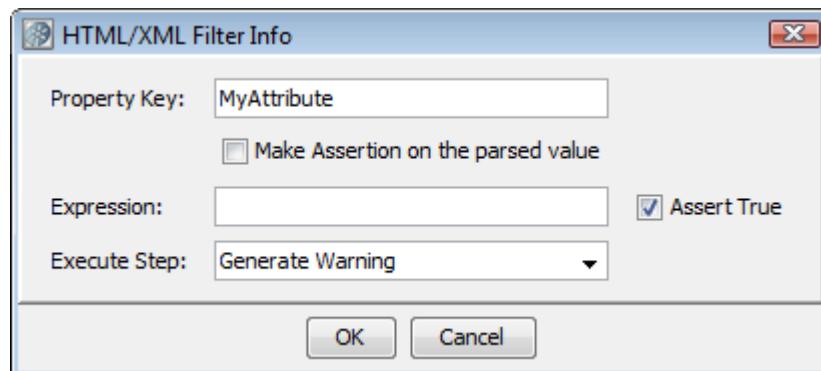
To create the filter from the response page:

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 Tags

The Parse XML Result for DevTest Tags 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 \(see page 1611\)](#).

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. The property must exist.

▪ Save To Property

The property in which to store the result of the XPath query.

Now construct the XPath query using one of the methods described earlier.

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 \(<http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/level-one-html.html>\)](http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/level-one-html.html) spec that maintains that all DOM element names must be upper case, while attribute names must be lower case.

JSON Filters

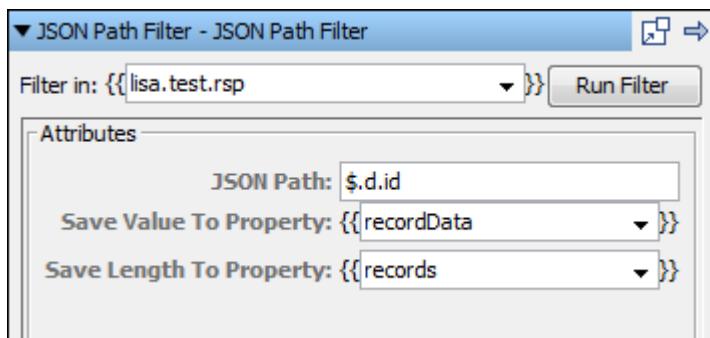
The following filter is available in the JSON filters list for any test step:

- [JSON Path Filter \(see page 1621\)](#)

JSON Path Filter

The JSON Path filter lets you extract a JSON property value from a JSON object and save it to a property. To open its editor, click the filter.

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 \(see page 1489\)](#) assertion.

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.

- **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.

An array filter, denoted by a `?()` expression, can be applied to an array as a select criteria to select certain array elements. For example, `?(@.age > 20)` can be used to select array members whose age is greater than 20. Another example is `?(@.name in ('Mary', 'John'))`, which selects array members whose name is either Mary or John.

The following table lists supported filter operators that are available for JSON data types.

Operator	String	Number	Boolean
<code>== (equal)</code>	supported	supported	supported
<code>!= (not equal)</code>	supported	supported	supported
<code>>=</code>		supported	
<code><=</code>		supported	
<code>></code>		supported	
<code><</code>		supported	
<code>in</code>	supported	supported	
<code>not in</code>	supported	supported	

- **Save Value to Property**

Designates the name of the property where the property value of the JSON path is saved.

- **Save Length to Property**

Defines the name of the property where the number of components in the attribute value of the JSON path is saved. 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 \(see page 1623\)](#)
- [Java Store Step Response Filter \(see page 1623\)](#)
- [Java Save Property Value to File Filter \(see page 1624\)](#)

Java Override Last Response Property Filter

A special property (Last Response) 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.

If you want the last response to 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 \(see page 1591\)](#).

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 filters are available in the VSE Filters list for any test step:

- [Data Protocol Filter \(see page 1624\)](#)

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 functionality for the VSE run-time side.

To open its editor, click the 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 CAI Filters list for any test step:

- [Integration for CA Continuous Application Insight \(see page 1625\)](#)
- [Integration for webMethods Integration Server \(see page 1625\)](#)

Integration for CA Continuous Application Insight

The DevTest Integration Support for CA CAI filter is a common filter to enable CAI for all the technologies that DevTest supports. This filter collects extra information from a CAI application.

Currently 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 CAI 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.

Integration for webMethods Integration Server

The DevTest Integration Support for webMethods Integration Server filter collects more information from CAI-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 CAI warnings after filter is set to run.

Copybook Filters

The following filters are available in the Copybook Filters list for any test step:

- [Copybook Filter \(see page 1626\)](#)

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

This section describes the following metrics:

- [DevTest Whole Test Metrics \(see page 1627\)](#)
- [DevTest Test Event Metrics \(see page 1628\)](#)
- [SNMP Metrics \(see page 1629\)](#)
- [JMX Metrics \(see page 1630\)](#)
- [TIBCO Hawk Metrics \(see page 1633\)](#)
- [Windows Perfmon Metrics \(see page 1634\)](#)
- [UNIX Metrics Via SSH \(see page 1636\)](#)

DevTest Whole Test Metrics

Whole Test Metrics, as the name suggests, collect all the basic information about the test case and provide six sub metrics.

The following sub metrics are collected. The response time metrics are reported in milliseconds.

- **Instances** (the number of virtual users)
- **Avg Resp Time**
- **Max Resp Time**

- **Min Resp Time**
- **Last Resp Time**
- **Steps Per second**



Note: The number of virtual users (Instances), average response times (Avg Resp Time), and Steps Per Second sub metrics are added by default to the metric list in a staging document.

DevTest Test Event Metrics

Test Event Metrics provide metrics in regard to DevTest events. These metrics include both 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 eight 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 \(see page 122\)](#).

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.

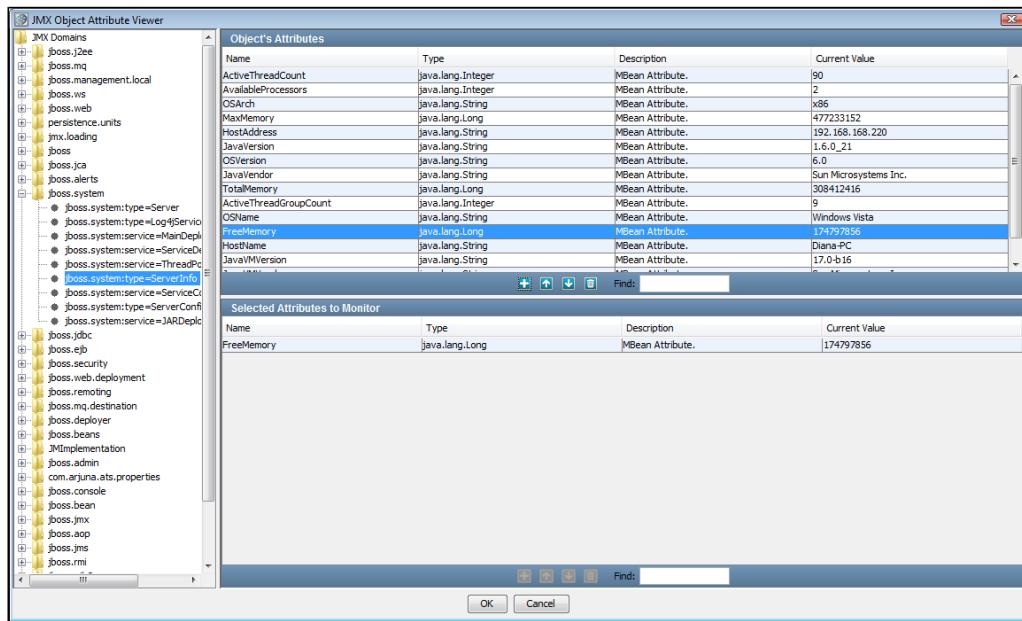
Follow these steps:

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 \(see page 1380\)](#).

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.

Setting up Perfmon

Before you can collect the Perfmon metrics, configure Perfmon on your Windows computer.

For information about setting up Perfmon, see [Install Performance Monitor \(Perfmon\) \(see page 121\)](#).



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

This metric is used to collect command-line metrics. This metric gathers inputs like 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. However, you can define a new location by updating the key **stats.unix.xml.folder** in **local.properties**.

Each file in that directory 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 \(lisa.properties\) \(see page 1638\)](#)
- [Custom Property Files \(see page 1671\)](#)
 - [Local Properties File \(see page 1671\)](#)
 - [Site Properties File \(see page 1690\)](#)
 - [logging.properties \(see page 1694\)](#)

DevTest Property File (lisa.properties)

The **lisa.properties** property file stores initialization and configuration information.

Property files are stored in the DevTest install directory.

You can display the contents of this file in DevTest Workstation.



Note: Do not add your custom properties to this file. CA Technologies reserves the right to replace this file at any time.

To open the **lisa.properties** file, select **System, Edit Properties** from the main menu.

This section contains the following pages :

- [Comma-Separated List of Paths for Javadoc and Source Code \(see page 1639\)](#)
- [System Properties \(see page 1639\)](#)
- [Server Properties \(see page 1640\)](#)
- [OS X Properties \(see page 1640\)](#)
- [Update Notifications \(see page 1641\)](#)
- [Basic Defaults \(see page 1641\)](#)
- [HTTP Header Keys Properties \(see page 1642\)](#)
- [HTTP Field Editor Properties \(see page 1643\)](#)
- [Test Case Execution Parameters \(see page 1643\)](#)
- [TestEvent Handling Customizations \(see page 1644\)](#)
- [Test Manager/Editor Properties \(see page 1645\)](#)
- [J2EE Server Parameters \(see page 1647\)](#)
- [Native Browser Information to Use for Internal Rendering \(see page 1648\)](#)
- [Test Manager/Monitor Properties \(see page 1649\)](#)
- [Built-In String-Generator Patterns \(see page 1649\)](#)
- [JMX Information \(see page 1649\)](#)

- [Test Manager/ITR Properties \(see page 1651\)](#)
- [External Command Shells \(see page 1651\)](#)
- [Testing Parameters \(see page 1651\)](#)
- [Properties for Use by StdSchedulerFactory to Create a Quartz Scheduler Instance \(see page 1651\)](#)
- [VSE Properties \(see page 1653\)](#)
- [Network Port Properties \(see page 1661\)](#)
- [CA Continuous Application Insight Properties \(see page 1662\)](#)
- [Database Properties \(see page 1666\)](#)
- [Mainframe Properties \(see page 1667\)](#)
- [Selenium Integration Properties \(see page 1668\)](#)
- [VSEasy Properties \(see page 1670\)](#)
- [Solr Properties \(see page 1670\)](#)
- [CAI Stateful Baseline Parameterization \(see page 1670\)](#)

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 (http://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|\._IJ|..*|.*\.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-bitmap, 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 DevTest data into that tool and 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.FioranolInitialContextFactory
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.git.mm.mysql.Driver org.apache.derby.jdbc.ClientDriver
lisa.prefill.jdbc.jdbcConnectionURLs=	jdbc:oracle:thin:@SERVER:1521:SIDNAME jdbc:sqlserver://SERVER:PORT;datasasename=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=yes

lisa.internal.browser.win=com.itko.lisa.web.ie.IEUtils

lisa.internal.browser.osx=com.itko.lisa.web.jxbrowser.JxBrowserUtils

lisa.internal.browser.linux=com.itko.lisa.web.jxbrowser.JxBrowserUtils

lisa.internal.browser.sol-sparc=null

lisa.internal.browser.imgs=false

lisa.internal.browser=msie

WR type: mozilla, safari, msie

lisa.internal.browser.swing.heavy=false

lisa.internal.browser.usejsinviews=yes

lisa.example.wsdl=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=LII-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.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/ITR 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 shutdown the registry.

Default: false

Property Used for the Eclipse Connector

- **lisa.eclipse.connector.port**

Default: 8546

Property Used for Example Test Suites

- **EXAMPLES_HOME**

Default: LISA_HOME/examples

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.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.ddmmmmregex**
`((\[12\]\\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.ddmmmyyy.separatorformat**
Format: dd*MM*yyyy

- **lisa.vse.datechecker.yyyymmdd.separatorformat**
Format: yyyy*MM*dd

- **lisa.vse.datechecker.yyyymmdd.separatorformat**
Format: yyyy*MM*dd

- **lisa.vse.datechecker.ddmmmyyyyformat**
Format: ddMMMMyyyy

- **lisa.vse.datechecker.mmmddyyyyformat**
Format: MMMddyyyy

- **lisa.vse.datechecker.yyyddmmmmformat**
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.ddmmmyyyformat&\
lisa.vse.datechecker.yyyddmmmformat&\
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{1,2}(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

CA Continuous Application Insight Properties

- **lisa.pathfinder.broker.host**

Default: 0.0.0.0

- **lisa.pathfinder.broker.port**

Default: 2009

- **lisa.webserver.port**

Our embedded web server.

Default: 1505

- **lisa.webserver.https.enabled**

Default: false

- **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/

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.threshold**
Default: 100
- **rpt.lisa.graph.view.infomessage**
Results more than maximum threshold. Modify a filter to fetch the details.
- **rpt.lisa.graph.scatter.view.threshold**
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> (<http://www.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 **logging.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. The metrics that are collected can be viewed in the web-based VSE dashboard.

- **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, request

- **Service:** The name of the service names transaction counts

- **Request:** The service name plus the request operation names transaction counts

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.maxPoolSize**

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

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

For more information, see [Mainframe Bridge \(see page 1316\)](#).

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 \(see page 1846\)](#) 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 \(see page 712\)](#).

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

- **lisa.vseeasy.http.max.dynamic.port**

Default: 65535

- **lisa.vseeasy.default.group.tag**

Specifies the name of the default virtual service group.

Default: VSEasy

Solr Properties

CAI 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.

CAI 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 CAI 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

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 \(see page 1671\)](#)
- [Site Properties File \(see page 1690\)](#)
- [logging.properties \(see page 1694\)](#)

Local Properties File

Contents

- [Autoconnection Properties \(see page 1672\)](#)
- [License Properties \(see page 1672\)](#)
- [VSE Properties \(see page 1673\)](#)

- [Enterprise Dashboard Properties \(see page 1677\)](#)
 - [Oracle Properties \(see page 1677\)](#)
 - [MS SQL Server Properties \(see page 1678\)](#)
 - [MySQL Properties \(see page 1678\)](#)
 - [Derby Properties \(see page 1678\)](#)
 - [Enterprise Dashboard Pool Properties \(see page 1679\)](#)
- [Interconnectivity Properties \(see page 1679\)](#)
- [License Properties if Using an HTTP Proxy Server \(see page 1680\)](#)
- [SDK Properties \(see page 1681\)](#)
- [SSL Properties \(see page 1681\)](#)
- [HTTP Authorization Properties \(see page 1682\)](#)
- [Kerberos Authentication Properties \(see page 1683\)](#)
- [HTTP Proxy Server Properties \(see page 1683\)](#)
- [XML Serialization Settings \(see page 1684\)](#)
- [Workstation Management Properties \(see page 1684\)](#)
- [IP Spoofing \(see page 1685\)](#)
- [Quick Start Recent Items Properties \(see page 1685\)](#)
- [lisa-invoke Properties \(see page 1685\)](#)
- [Server Host Name Properties \(see page 1686\)](#)
- [DevTest to DevTest Communication Encryption Properties \(see page 1686\)](#)
- [IBM WebSphere MQ Properties \(see page 1687\)](#)
- [JMS Properties \(see page 1687\)](#)
- [Miscellaneous Properties \(see page 1687\)](#)
- [User Session Lifetime Properties \(see page 1689\)](#)
- [Mobile Properties \(see page 1690\)](#)

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 rest 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 in the charts list on the VSE Console 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.

Errors are still reported in the VSE console with a red circle and an error count, regardless of the value of this property.

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.conversation.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 \(see page 661\)](#).

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

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.

The Enterprise Dashboard does not support IBM DB2.

Because the site.properties is handled solely by a registry process, a custom Enterprise Dashboard database must be configured in the local.properties file.

These are the properties that DevTest uses to connect to the Enterprise Dashboard database. The default connection is to the internal Derby database. For information about connecting to external databases, see [External Database Configuration \(see page 1386\)](#).

Oracle Properties

- **lisadb.pool.dradis.driverClass**

Default: oracle.jdbc.driver.OracleDriver

- **lisadb.pool.dradis.url**

Default: jdbc:oracle:thin:@[HOST]:1521:[SID]

- **lisadb.pool.dradis.user**

Default: [USER]

- **lisadb.pool.dradis.password**

Default: [PASSWORD]

MS SQL Server Properties

- **lisadb.pool.dradis.driverClass**

Default: com.microsoft.sqlserver.jdbc.SQLServerDriver

- **lisadb.pool.dradis.url**

Default: jdbc:sqlserver://[SERVER]:[PORT];databaseName=[DATABASENAME]

- **lisadb.pool.dradis.user**

Default: [USER]

- **lisadb.pool.dradis.password**

Default: [PASSWORD]

MySQL Properties

- **lisadb.pool.dradis.driverClass**

Default: com.mysql.jdbc.Driver

- **lisadb.pool.dradis.url**

Default: jdbc:mysql://[DBHOST]:[DBPORT]/[DBNAME]

- **lisadb.pool.dradis.user**

Default: [USER]

- **lisadb.pool.dradis.password**

Default: [PASSWORD]

Derby Properties

- **lisadb.dradis.poolName**

Default: common

- **lisadb.pool.common.driverClass**

Default: org.apache.derby.jdbc.ClientDriver

- **lisadb.pool.common.url**

Default: jdbc:<derby://localhost:1530/database/dradis.db>;create=true

- **lisadb.pool.common.user**

Default: app

Enterprise Dashboard Pool Properties

- **lisadb.dradis.poolName**

Default: dradis

Default pool properties to keep the number of connections to a minimum if we are idle.

- **lisadb.pool.dradis.minPoolSize**

Default: 0

- **lisadb.pool.dradis.initialPoolSize**

Default: 0

- **lisadb.pool.dradis.maxPoolSize**

Default: 10

- **lisadb.pool.dradis.acquireIncrement**

Default: 1

- **lisadb.pool.dradis.maxIdleTime**

Default: 45

- **lisadb.pool.dradis.idleConnectionTestPeriod**

Default: 5

Should the internal Derby database instance in the Enterprise Dashboard be started?

- **dradisdb.internal.enabled**

Controls whether the Enterprise Dashboard starts an internal Derby database. If you configured to use an external database, you can save resources by setting this property to false.

Default: true

Interconnectivity Properties

- **lisa.default.keystore**

Default: {{LISA_HOME}}/webreckeys.ks

- **lisa.default.keystore.pass**
Default: passphrase

DevTest to DevTest communication encryption. Normally 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\webreckey.ks`). For a stronger certificate, specify the `lisa.net (http://lisa.net).keyStore` property and the plaintext password in `lisa.net (http://lisa.net).keyStore.password`.

The next time DevTest starts, an encrypted string (`lisa.net (http://lisa.net).keyStore.password_enc`) replaces the plaintext password.

- **lisa.net (http://lisa.net).keyStore**
Default: {{LISA_HOME}}lisa.ks
- **lisa.net (http://lisa.net).keyStore.password**

This is where we keep our identification certificate. If we are a server installation, this is the certificate that clients need to 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 needs to 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 (http://lisa.net).trustStore**
Default: {{LISA_HOME}}lisa.ts
- **lisa.net (http://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 (http://lisa.net).clientAuth**
 Indicates whether to use mutual authentication.
Default: false
- **lisa.net (http://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*](http://my_proxyserver.com) ([*http://my_proxyserver.com*](http://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*](mailto:recipient@mycompany.com) ([*mailto:recipient@mycompany.com*](mailto:recipient@mycompany.com)) with the target email address. Replace [*from@mycompany.com*](mailto:from@mycompany.com) ([*mailto:from@mycompany.com*](mailto:from@mycompany.com)) with the email address from which the email is sent.

Example: [*mailhost,recipient@mycompany.com*](mailto:mailhost,recipient@mycompany.com) ([*http://mycompany.com*](http://mycompany.com)),[*from@mycompany.com*](mailto:from@mycompany.com) ([*http://mycompany.com*](http://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 and Raw SOAP 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**.

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

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=NOREFERENCE**

NOREFERENCE 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 re-read fails and could require you to set a default of XPATHREFERENCES.

Workstation Management Properties

- **lisa.ui.admin.tr (<http://lisa.ui.admin.tr>).control=no**
- **lisa.ws (<http://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 (<http://lisa.ws>).ser.dateFormat=yyyy-MM-dd**
- **lisa.ws (<http://lisa.ws>).ser.timeFormat=HH:mm:ss.SSS'Z'**

- **lisa.ws (http://lisa.ws).ser.timeFormat.timeZone=GMT**
- **lisa.ws (http://lisa.ws).ser.dateFormat=yyyy-MM-dd'T'HH:mm:ss.SSS'Z'**
- **lisa.ws (http://lisa.ws).ser.dateFormat.timeZone=GMT**
- **ws.raw.format=true**

To format the SOAP response for RAW Soap Steps, set this property to **true**. This formatting can also be done dynamically at run time.

- **lisa.ws (http://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 WSPOST 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 (http://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

Normally 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\webreckkeys.ks** is a default internal self-signed certificate. For a stronger certificate, specify the **lisa.net (http://lisa.net).keyStore** property and the plaintext password in **lisa.net (http://lisa.net).keyStore.password**. The next time DevTest starts, an encrypted string **lisa.net (http://lisa.net).keyStore.password_enc** replaces the plain text password.
- **lisa.default.keystore={{LISA_HOME}}webreckkeys.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 (http://lisa.net).keyStore={{LISA_HOME}}lisa.ks**
- **lisa.net (http://lisa.net).keyStore.password=PlainTextPasswordWillBeConvertedToEncrypted**

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 (http://lisa.net).trustStore={{LISA_HOME}}lisa.ts**
- **lisa.net (http://lisa.net).trustStore.password_enc=079f6a3d304a978146e547802ed3f3a4**

Whether or not to use mutual authentication.

- **lisa.net (http://lisa.net).clientAuth=false**

Defines the default protocol for ActiveMQ connections. The default is **tcp**.

- **lisa.net (http://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 (<http://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 (<http://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 (<http://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 (<http://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 (<http://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 causes the JMS step to always 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: 15
- [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.

Default: 50

▪ `lisa.metric.initialization.timeout`

Initialization of the Metric Collector occurs during the staging of the test/suite and the test/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` (<http://lisa.gui.dashboard.sleep.ms>)

There is a small sleep() just 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

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 DevTest Workstation to bypass authentication for DevTest console web applications (for example: reporting, CVS, CAI, Server Console), the DevTest Portal, and DevTest Workstation.

When you click a button to access a console web application, DevTest bypasses ACL authentication by using an SSO security token. That SSO security token is stored in memory for reuse. However, if DevTest Workstation 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 DevTest Workstation, you can click these buttons and go directly to the web application without entering a userid and password. But after DevTest Workstation is idle (no activity) for 20 minutes, if you click these buttons, DevTest Workstation 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.

Mobile Properties

- **lisa.mobile.custom.arguments**

If set, then the string value of the property is added to the command line of the Appium process. It can be anything meaningful to Appium. A full list of Appium arguments can be found [here](https://github.com/appium/dmg-template/blob/master/Appium.app/Contents/Resources/node_modules/appium/docs/server-args.md) (https://github.com/appium/dmg-template/blob/master/Appium.app/Contents/Resources/node_modules/appium/docs/server-args.md).

For example: lisa.mobile.custom.arguments="--no-reset"

This can be used with applications that require setup configuration or exhibit distinct behavior on the first run. The no-reset assures that the application would not require setup on every test run.

- **lisa.mobile.remote.user**

When running remote tests through a headless app like TestRunner, this property is required if you want to prompt for a User name for the remote host during a playback of a recording.

- **lisa.mobile.remote.password**

When running remote tests through a headless app like TestRunner, this property is required if you want to prompt for a Password for the remote host during a playback of a recording.

Make sure you encrypt this password by clicking the check box for the lock icon in DevTest Workstation .

Site Properties File

Contents

- [DevTest Enterprise Dashboard Properties \(see page 1691\)](#)
- [Database Properties \(see page 1691\)](#)
- [Oracle Properties \(see page 1692\)](#)
- [MS SQL Server Properties \(see page 1693\)](#)
- [DB2 Properties \(see page 1693\)](#)
- [MySQL Properties \(see page 1693\)](#)

- [Derby Properties \(see page 1694\)](#)
- [DCM Settings \(see page 1694\)](#)

The properties in **site.properties** 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

- **lisa.enterprisedashboard.service.url**

Designates the Service URL to use for sending component and metric information to the DevTest Enterprise Dashboard.



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 will need to be configured in the local.properties file. For more details on configuration of the Enterprise Dashboard database, look at the _local.properties template in the home directory of your DevTest installation.

Default: <tcp://somehost:2003/EnterpriseDashboard>

Database Properties

These are the default properties 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

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]

- lisadb.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

- **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.appenders.A1**
Default: com.itko.util.log4j.TimedRollingFileAppender
- **log4j.appenders.A1.File**
Default: \${lisa.tmpdir}/\${LISA_LOG}

- **log4j.appendер.A1.MaxFileSize**

Default: 10MB

- **log4j.appendер.A1.MaxBackupIndex**

Default: 5

- **log4j.appendер.A1.layout**

Default: org.apache.log4j EnhancedPatternLayout

- **log4j.appendер.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.appendер.VSEAPP**

Default: com.itko.util.log4j.TimedRollingFileAppender

- **log4j.appendер.VSEAPP.File**

Default: \${lisa.tmpdir}/vse_matches.log

- **log4j.appendер.VSEAPP.MaxFileSize**

Default: 10MB

- **log4j.appendер.VSEAPP.MaxBackupIndex**

Default: 20

- **log4j.appendер.VSEAPP.layout**

Default: org.apache.log4j EnhancedPatternLayout

- **log4j.appendер.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.appendер.ADVICE_APP**

Default: org.apache.log4j RollingFileAppender

- **log4j.appenders.ADVICE_APP.File**
Default: \${lisa.tmpdir}/advice.log
- **log4j.appenders.ADVICE_APP.MaxFileSize**
Default: 10MB
- **log4j.appenders.ADVICE_APP.MaxBackupIndex**
Default: 20
- **log4j.appenders.ADVICE_APP.layout**
Default: org.apache.log4j.EnhancedPatternLayout
- **log4j.appenders.ADVICE_APP.layout.ConversionPattern**
Default: %d{ISO8601}{UTC}Z (%d{HH:mm}) %-5p - %m%n

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_DUMP
- **log4j.additivity.threadDumpLogger**
Default: false
- **log4j.appenders.THREAD_DUMP**
Default: org.apache.log4j.RollingFileAppender
- **log4j.appenders.THREAD_DUMP.File**
Default: \${lisa.tmpdir}/threadDumps/TD_\${LISA_LOG}
- **log4j.appenders.THREAD_DUMP.MaxFileSize**
Default: 10MB
- **log4j.appenders.THREAD_DUMP.MaxBackupIndex**
Default: 20
- **log4j.appenders.THREAD_DUMP.layout**
Default: org.apache.log4j.EnhancedPatternLayout
- **log4j.appenders.THREAD_DUMP.layout.ConversionPattern**
Default: %d{ISO8601}{UTC}Z (%d{HH:mm}) [%t] %-5p - %m%n
- **log4j.appenders.registry**
Mirrors DevTest logging to the (remote) registry.
Default: com.itko.lisa.net.LoggingToRegistryAppender

- **log4j.appenders.registry.layout**
Default: org.apache.log4j.EnhancedPatternLayout
- **log4j.appenders.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

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
- Labs
- VSE servers

The full API documentation is available [here](https://support.ca.com/cadocs/0/CA%20DevTest%20Solutions%208%200-ENU/Bookshelf_Files/Invoke/index.html) (https://support.ca.com/cadocs/0/CA%20DevTest%20Solutions%208%200-ENU/Bookshelf_Files/Invoke/index.html). The API documentation provides details on how to interact with the various RESTful services.

You can also view a set of examples [here](http://github.com/ianakelly/invoke) (<http://github.com/ianakelly/invoke>).



Note: You must use Google Chrome or Mozilla Firefox to access the API documentation.

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

This page describes how to use the REST Invoke API to change the execution mode of a virtual service from Most Efficient to Transaction Tracking.

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>
```

Test Step Descriptions

This section describes the following test steps:

- [Test Step Information \(see page 1701\)](#)
- [Web-Web Services Steps \(see page 1702\)](#)
- [Java-J2EE Steps \(see page 1756\)](#)
- [Other Transaction Steps \(see page 1768\)](#)
- [Utilities Steps \(see page 1774\)](#)
- [External-Subprocess Steps \(see page 1785\)](#)
- [JMS Messaging Steps \(see page 1792\)](#)
- [BEA Steps \(see page 1810\)](#)
- [Sun JCAPS Steps \(see page 1816\)](#)
- [Oracle Steps \(see page 1819\)](#)
- [TIBCO Steps \(see page 1829\)](#)
- [Sonic Steps \(see page 1835\)](#)
- [webMethods Steps \(see page 1837\)](#)
- [IBM Steps \(see page 1846\)](#)
- [SAP Steps \(see page 1855\)](#)
- [Selenium Integration Steps \(see page 1860\)](#)
- [Virtual Service Environment Steps \(see page 1865\)](#)
- [CAI Steps \(see page 1865\)](#)
- [Mobile Steps \(see page 1868\)](#)
- [Custom Extension Steps \(see page 1872\)](#)
- [RabbitMQ Steps \(see page 1875\)](#)

Test Step Information

The following test steps are standard:

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:

- [HTTP-HTML Request Step \(see page 1702\)](#)
- [REST Step \(see page 1710\)](#)
- [Web Service Execution \(XML\) Step \(see page 1711\)](#)
- [WSDL Validation \(see page 1748\)](#)
- [Web-Raw SOAP Request \(see page 1749\)](#)
- [Base64 Encoder \(see page 1750\)](#)
- [Multipart MIME Step \(see page 1750\)](#)
- [SAML Assertion Query \(see page 1752\)](#)

HTTP-HTML Request Step

This 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 (similar to the WS step). 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 \(see page 1703\)](#)
- [HTTP Headers Tab \(see page 1706\)](#)
- [Response Tab \(see page 1706\)](#)
- [SSL Tab \(see page 1706\)](#)

URL Transaction Info Tab

Specify the information that is used to construct the URL on the **URL Transaction Info** tab.

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 the **Specify URL in parts** option (default) 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` in the **Path** field.

▪ **User**

Enter if a user ID is required for the application server.

▪ **Password**

Enter if a password is required for the application server.

▪ **Encoding**

The property governs the **Encoding** drop-down: `lisa.supported.html.request.encodings=ISO-8859-1, UTF-8, Shift_JIS, EUC-JP, Windows-31J`

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, then 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, then the DevTest default encoding is assumed.

▪ **URL Parameters**

GET (or URL) Request Parameters: these request parameters are passed as part of the URL, and so they are exposed to the user in address bar of the web browser.

▪ POST Parameters

POST Request Parameters. The request parameters are passed as part of the body of the page request. 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 that is used is application/form-urlencoded.

▪ All Known State

All known properties, such as test case properties, data sets, and filters, are listed.

▪ Download images referenced (test bandwidth)

If this element is selected, the step 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 a request parameter Add icon
Up		Move an existing parameter up in the list of parameters Up icon
Down		Move an existing parameter down in the list of parameters Down icon
Delete		Delete an existing parameter ground-ICO
Find:		Find text
Auto Generate a Filter		From the referring step to make this parameter dynamic. Create a new filter to auto-populate this property at run time. For more information on filters, see the Filters section. Auto Generate a Filter icon
Apply selected All Known State property		Apply state to the parameter. For more information on applying state, see the All Known State section that follows. Apply selected All Known State property
Auto Apply all Known State properties		Apply all state to all properties possible by patterns. For more information on applying state, see the All Known State section that follows.



Auto Apply all
Known State
properties icon

All Known State

All known properties, such as test case properties, data sets, and filters, are displayed in the **All Known State** panel.

URL Parameters		All Known State	
Key	Value	Key	Value
u_login	vvv	lisa.Verify User ...	<list> <list> ...
LISA_TC_PATH	C:\Lisa\example...	robot	0
lisa.hidden.ejb....	SerialNum=139...	lisa.Account Act...	<!- http://loca...
LISA_PROJ_NAME	examples	LISA_HOST	Diana-PC
lisa.Verify User ...	<list> <list> ...	LISA_USER	arhoades@Dian...
instance	0	lisa.Delete User...	<?xml version...
testCaseId	656538366462...		

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 Option

If the **Use property** option button is selected, you can specify the following parameters:

- **Property Key**
Specify a property that contains the connection information.

- **Download images referenced (test bandwidth)**

If this element 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 top section, **Custom HTTP Headers (Current Only)**, is for headers that are only sent to the server for this request.
- The bottom section, **Custom HTTP Headers (Persist)**, is for headers that are sent on this transaction and every other transaction in the test.

To create a request parameter in either section, click **Add**  and change the key and value to the 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.

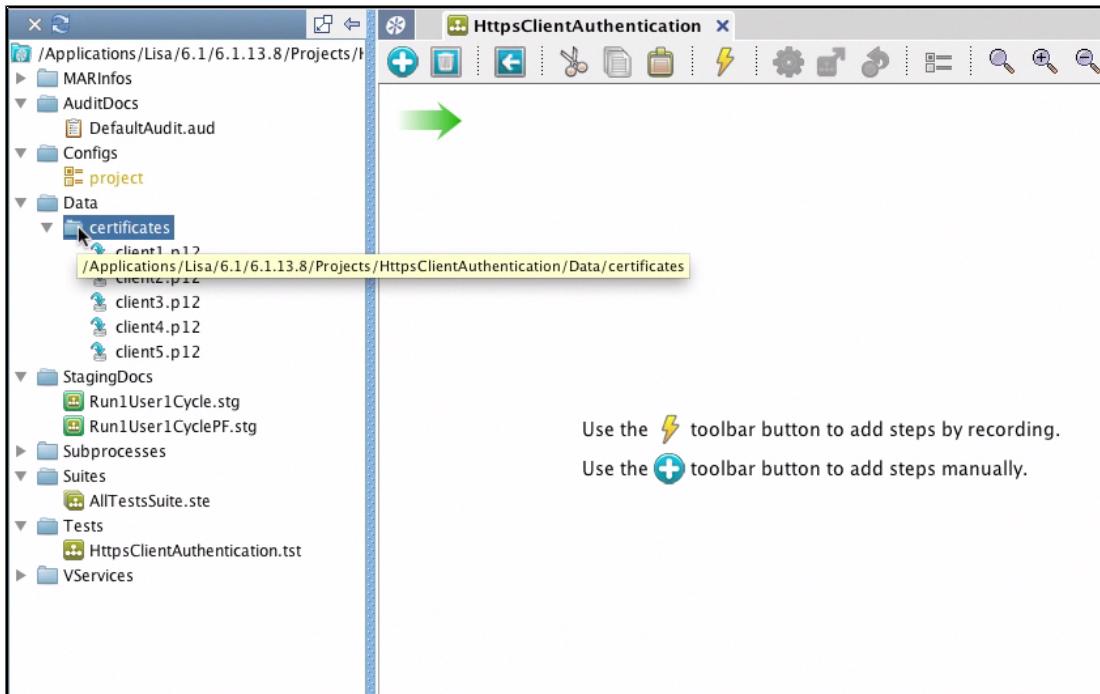
- **SSL Key password**

An optional password for the key entry if using a JKS keystore, and key has a different password from keystore.

Using Multiple SSL Certificates

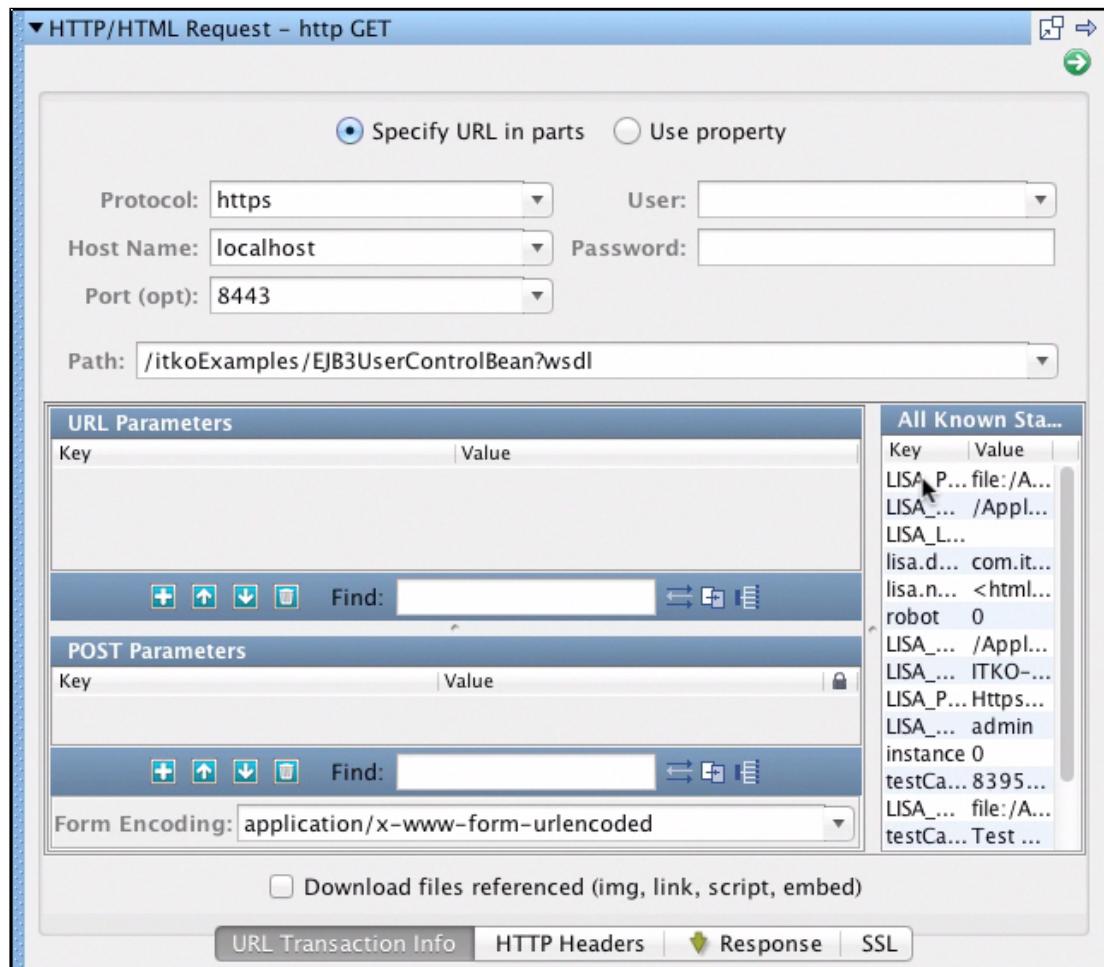
To use multiple certificates on one step, store your SSL information in a data set. The following example shows how you can use a data sheet to store information about multiple certificates.

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** with the name **certificates**. 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 your own Data Sheet - Create your own Data Sheet

Name:	<input type="text" value="Certificate Information"/>		
<input type="checkbox"/> Local	<input type="checkbox"/> Random	Max Records To Fetch:	<input type="text" value="100"/>
At end of data, <input type="radio"/> Start over <input checked="" type="radio"/> Execute <input type="button" value="End the Test"/>			
<input type="button" value="Test and Keep"/>			
Enter Data Sheet Params			
Number of Rows: <input type="text" value="5"/>			
Column Names: <input type="text" value="keystoreFile, keystorePassword, keyPassword, alias"/> (Enter comma separated column names)			
<input type="button" value="Create Data Sheet Skeleton"/> <input type="button" value="Import"/>			

Create Data Sheet editor

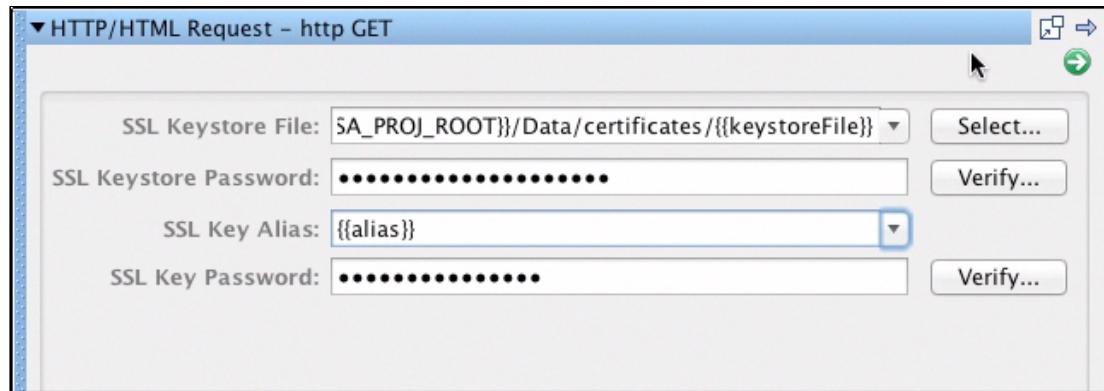
For **keystoreFile**, we use the short file name of the keystore so that relative paths can be used later. To encrypt the password columns, right-click the column label and select **Encrypt**.

▼ Create your own Data Sheet - Create your own Data Sheet

Name:	<input type="text" value="Certificate Information"/>																										
<input type="checkbox"/> Local	<input type="checkbox"/> Random	Max Records To Fetch:	<input type="text" value="100"/>																								
At end of data, <input type="radio"/> Start over <input checked="" type="radio"/> Execute <input type="button" value="End the Test"/>																											
<input type="button" value="Test and Keep"/>																											
<table border="1"> <thead> <tr> <th>keystoreFile</th> <th>keystorePa...</th> <th>keyPassword</th> <th>alias</th> </tr> </thead> <tbody> <tr> <td>client1.p12</td> <td>123456</td> <td>123456</td> <td>client1key</td> </tr> <tr> <td>client2.p12</td> <td>123456</td> <td>123456</td> <td>client2key</td> </tr> <tr> <td>client3.p12</td> <td>123456</td> <td>123456</td> <td>client3key</td> </tr> <tr> <td>client4.p12</td> <td>123456</td> <td>123456</td> <td>client4key</td> </tr> <tr> <td>client5.p12</td> <td>123456</td> <td>123456</td> <td>client5key</td> </tr> </tbody> </table>				keystoreFile	keystorePa...	keyPassword	alias	client1.p12	123456	123456	client1key	client2.p12	123456	123456	client2key	client3.p12	123456	123456	client3key	client4.p12	123456	123456	client4key	client5.p12	123456	123456	client5key
keystoreFile	keystorePa...	keyPassword	alias																								
client1.p12	123456	123456	client1key																								
client2.p12	123456	123456	client2key																								
client3.p12	123456	123456	client3key																								
client4.p12	123456	123456	client4key																								
client5.p12	123456	123456	client5key																								

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.

Timestamp	Event	Simulator	Instance	Short Info	Long Info
12:36:33,662	Test ended		0/0	839560C54790EC947542C4...	
12:36:29,946	Cycle ending	local	0/6	839560C54790EC947EE1FE9...	Signaled to stop test
12:36:29,946	Cycle ended normally	local	0/6	839560C54790EC947EE1FE9...	N/A
12:36:29,946	Cycle started	local	0/6	839560C54790EC947EE1FE9...	
12:36:29,946	Cycle ending	local	0/5	839560C54790EC947EE1B0...	Signaled to stop test
12:36:29,946	Cycle ended normally	local	0/5	839560C54790EC947EE1B0...	N/A
12:36:29,945	Info message	local	0/5	Dataset Ending Test	Data set 'Certificate I...
12:36:29,945	Cycle started	local	0/5	839560C54790EC947EE1B0...	
12:36:29,944	Cycle ending	local	0/4	839560C54790EC947E408E...	Signaled to stop test
12:36:29,944	Cycle ended normally	local	0/4	839560C54790EC947E408E...	N/A
12:36:29,684	Step response time	local	0/4	http GET	96
12:36:29,684	Info message	local	0/4	http GET	Requested URL:https://...
12:36:28,888	Cycle started	local	0/4	839560C54790EC947E408E...	
12:36:28,887	Cycle ending	local	0/3	839560C54790EC947DBC8...	Signaled to stop test
12:36:28,887	Cycle ended normally	local	0/3	839560C54790EC947DBC8...	N/A

Test staged with Quick Stage Test

REST Step

Use the REST step when testing REST applications.

This step is used to send and receive HTTP(S) requests, including GET and POST parameters.

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 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.

At the bottom of the **Response** panel is a selection of filters and an assertion that you can add to the response.

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/>.

An example test case for the REST step is in the examples project. The name of the test case is **rest-example.tst**.

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.

Name	Type	Value
<top>	Object	
user	Object	
emailAddress	String	test@test.com
firstName	String	first-9
lastName	String	last-9
password	String	aaaaaaaa
username	String	dmxxx-009

Screen capture of Visual JSON tab.

Web Service Execution (XML) Step

Contents

- [Connection Tab \(see page 1712\)](#)
- [Basic Configuration \(see page 1712\)](#)

The Web Service Execution (XML) step is designed to execute an operation on a SOAP-based Web Service using an HTTP POST or JMS message.

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

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/wsdl directory for the project. This action caches the WSDL locally for quicker access. DevTest parses the WSDL and uses its schema 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/wsdl directory 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

Contents

- [Visual XML Tab \(see page 1714\)](#)
- [Raw XML Tab \(see page 1717\)](#)
- [Headers Tab \(see page 1718\)](#)
- [Attachments Tab \(see page 1720\)](#)
- [Addressing Tab \(see page 1723\)](#)
- [Security Tab \(see page 1723\)](#)

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.

The screenshot shows the Visual XML Editor interface. At the top, there are tabs: Visual XML (selected), Raw XML, POST SOAP Headers, and Attachments. Below the tabs is a table with columns: Node, Type, Occurs, Nil, Nillable, and Value. The table displays the structure of a SOAP message:

Node	Type	Occurs	Nil	Nillable	Value
soapenv:Envelope	Envelope	1..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
soapenv:Body	Body	1..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
addAddress	addAddress	1..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
username	string	0..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	{:=[:username]}
addressObject	address	0..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	...
city	string	0..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	{:=[:city:A*]}
id	string	0..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	{:=[:id:A*(5-)}
line1	string	0..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	{:=[:line1:A*]}
line2	string	0..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	{:=[:line2:A*]}
state	string	0..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	{:=[:state:A*]}
zip	int	1..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	{:=[:zip:1]}

At the bottom of the editor, there are buttons for Validation Results, View XML Schema Source, Error Log, Request Editor, Request, and Response.

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 \(see page 1720\)](#).

- **Convert to XOP Attachment**

A shortcut method for creating a standard referenced XOP Include attachment. For more information, see [Attachments Tab \(see page 1720\)](#).

- **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

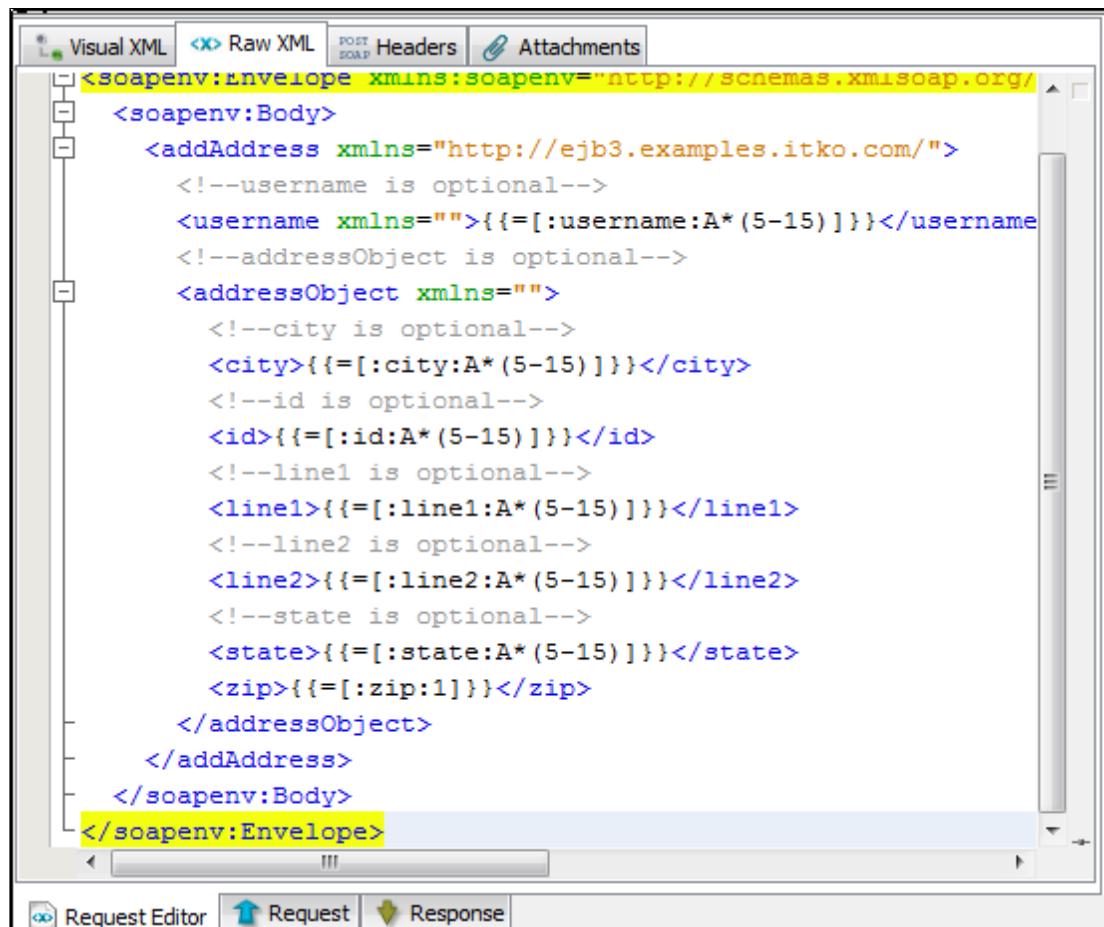
The **Type** column shows the XML schema type (local name) and has a tool tip to show the fully qualified name (qName) with namespace. This column can be edited for derived XSD types. Only base types with derived types can be edited.

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](#) ([see page 1714](#)) (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).

Transport Headers	
Key	Value
Key	HeaderValue
Accept	
Accept-Language	
User-Agent	
Connection	
Authorization	

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.

cid	content type	type	value
4AFE682D0AB9453B278...	text/plain	Text	

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.

Addressing Tab

For information about the Addressing Tab, see [Advanced Settings \(see page 1732\)](#).

Security Tab

For more details on the Security Tab, see [Advanced Settings \(see page 1732\)](#).

Design Time Execution

Contents

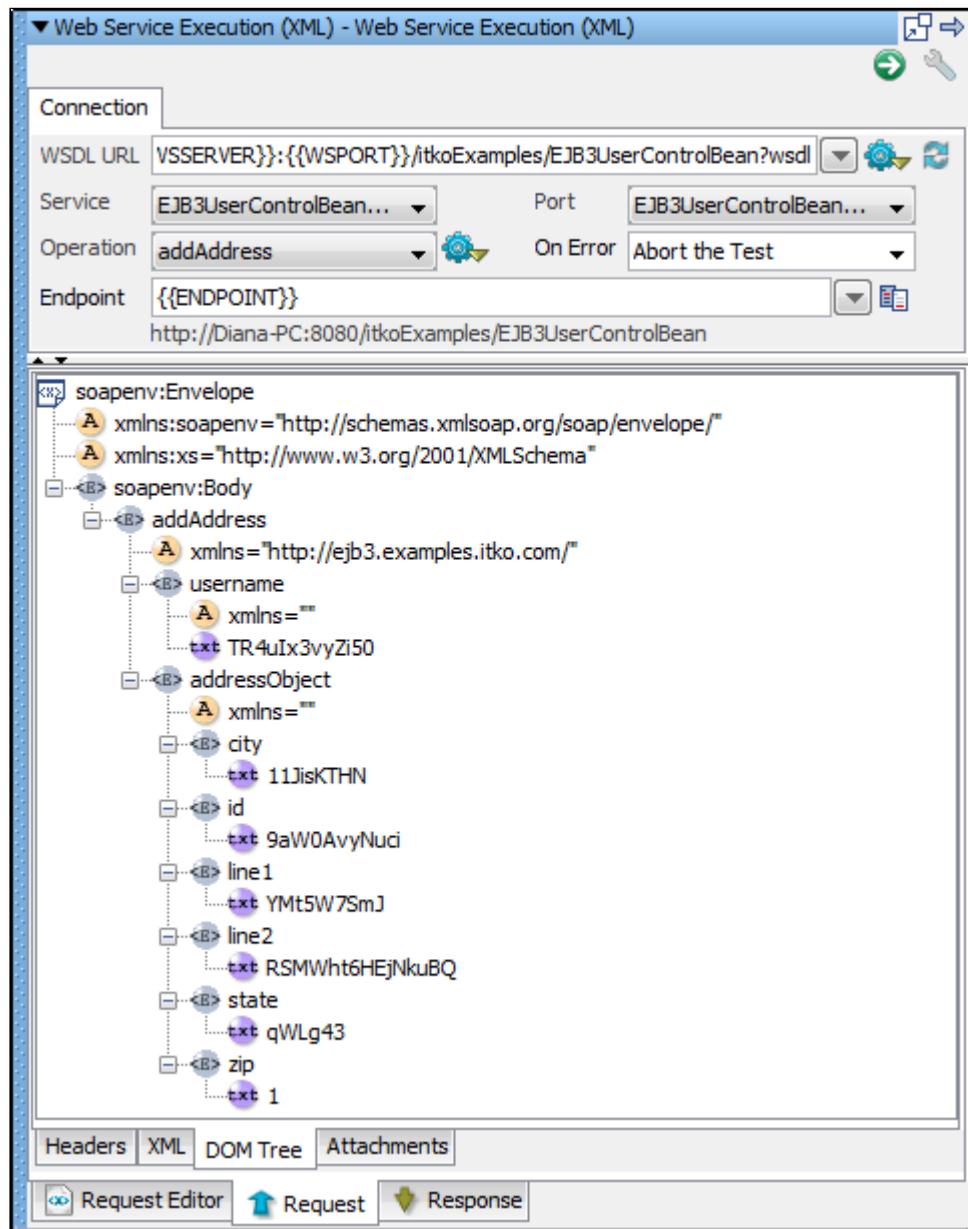
- [Request Tab \(see page 1723\)](#)
- [Response Tab \(see page 1727\)](#)

After you finish configuring the connection information and building the SOAP Request Message, you can run the 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 will be populated and it will switch to the **Response** tab automatically.

Request Tab

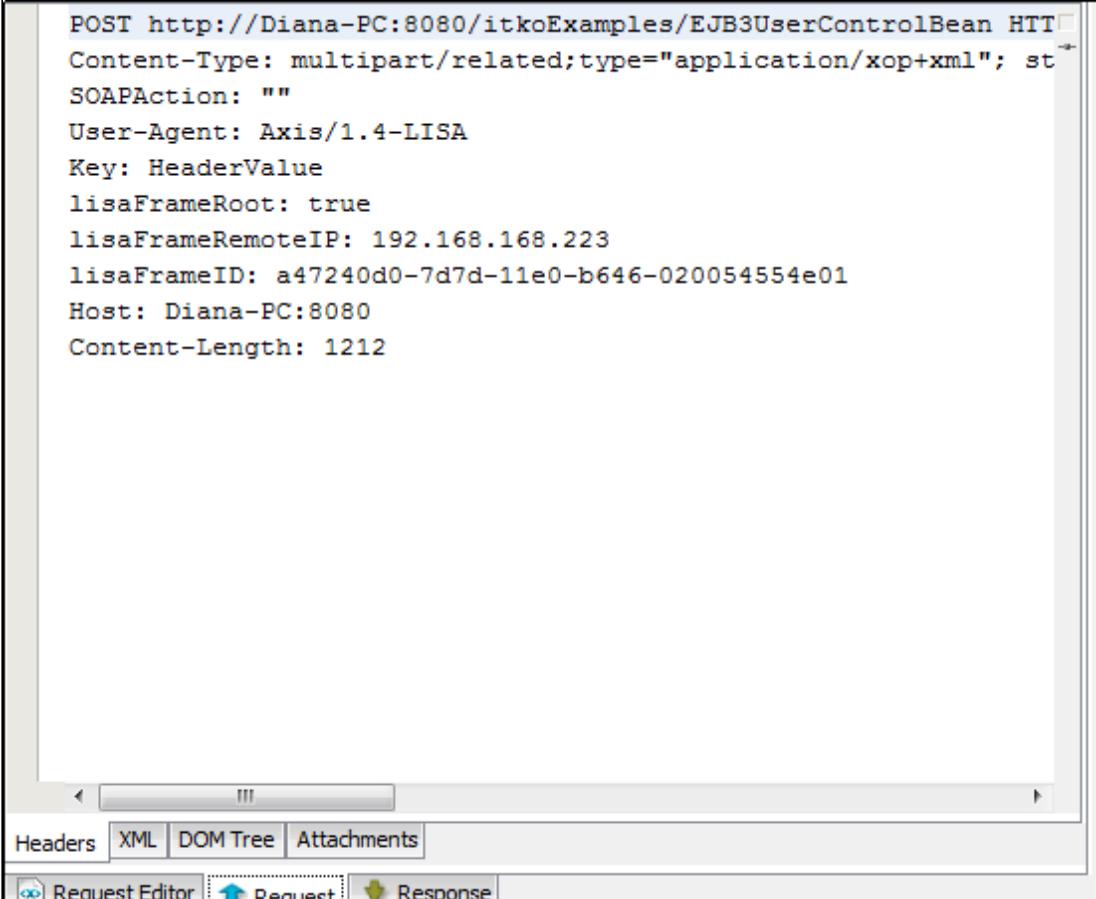
The **Request** tab shows the resulting request data that was sent after any post processing (for example, substituting DevTest properties). If the message contained any attachment, it does not show the raw MIME or DIME encoded message, but rather the processed message and attachments. To see the raw 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.



POST http://Diana-PC:8080/itkoExamples/EJB3UserControlBean HTTP/1.1
Content-Type: multipart/related;type="application/xop+xml"; start=0
SOAPAction: ""
User-Agent: Axis/1.4-LISA
Key: HeaderValue
lisaFrameRoot: true
lisaFrameRemoteIP: 192.168.168.223
lisaFrameID: a47240d0-7d7d-11e0-b646-020054554e01
Host: Diana-PC:8080
Content-Length: 1212

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.

The screenshot shows the 'Request' tab of the 'Web Service Execution (XML)' step in DevTest Solutions. The main area displays an XML document structure:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope">
  <soapenv:Body>
    <addAddress xmlns="http://ejb3.examples.itko.com/">
      <!--username is optional-->
      <username xmlns="">hg6SE</username>
      <!--addressObject is optional-->
      <addressObject xmlns="">
        <!--city is optional-->
        <city>EXqmVcRwGpum5i</city>
        <!--id is optional-->
        <id>uLWQ6XvyBYDYOb5</id>
        <!--line1 is optional-->
        <line1>HA5JgL356</line1>
        <!--line2 is optional-->
        <line2>y58auPo</line2>
        <!--state is optional-->
        <state>Nkm9EfE0yelkJ</state>
        <zip>1</zip>
      </addressObject>
    </addAddress>
  </soapenv:Body>
</soapenv:Envelope>
```

The XML code is color-coded for syntax highlighting. The 'Headers', 'XML', 'DOM Tree', and 'Attachments' tabs are visible at the bottom. Below the tabs, there are three buttons: 'Request Editor', 'Request', and 'Response'.

Web Service Execution (XML) step Request tab XML tab

DOM Tree Tab

The **DOM Tree** tab shows a DOM tree for the SOAP message.

The screenshot shows the DevTest Solutions interface with the 'Web Service Execution (XML)' step open. The 'Request' tab is selected, displaying a detailed XML DOM tree. The tree structure includes the following nodes:

- soapenv:Envelope**
 - xmlns:soapenv** = "http://schemas.xmlsoap.org/soap/envelope/"
 - xmlns:xs** = "http://www.w3.org/2001/XMLSchema"
 - Body**
 - addAddress**
 - username**
 - txt** hg6SE
 - addressObject**
 - city**
 - txt** EXqmVcRwGpum5i
 - id**
 - txt** uLWQ6XvyBYDYOb5
 - line1**
 - txt** HA5JgL356
 - line2**
 - txt** y58auPo
 - state**
 - txt** Nkm9EfE0yelkJ
 - zip**
 - txt** 1

Below the tree view, there are tabs for Headers, XML, DOM Tree, and Attachments. The DOM Tree tab is currently active. At the bottom, there are buttons for Request Editor, Request, and Response.

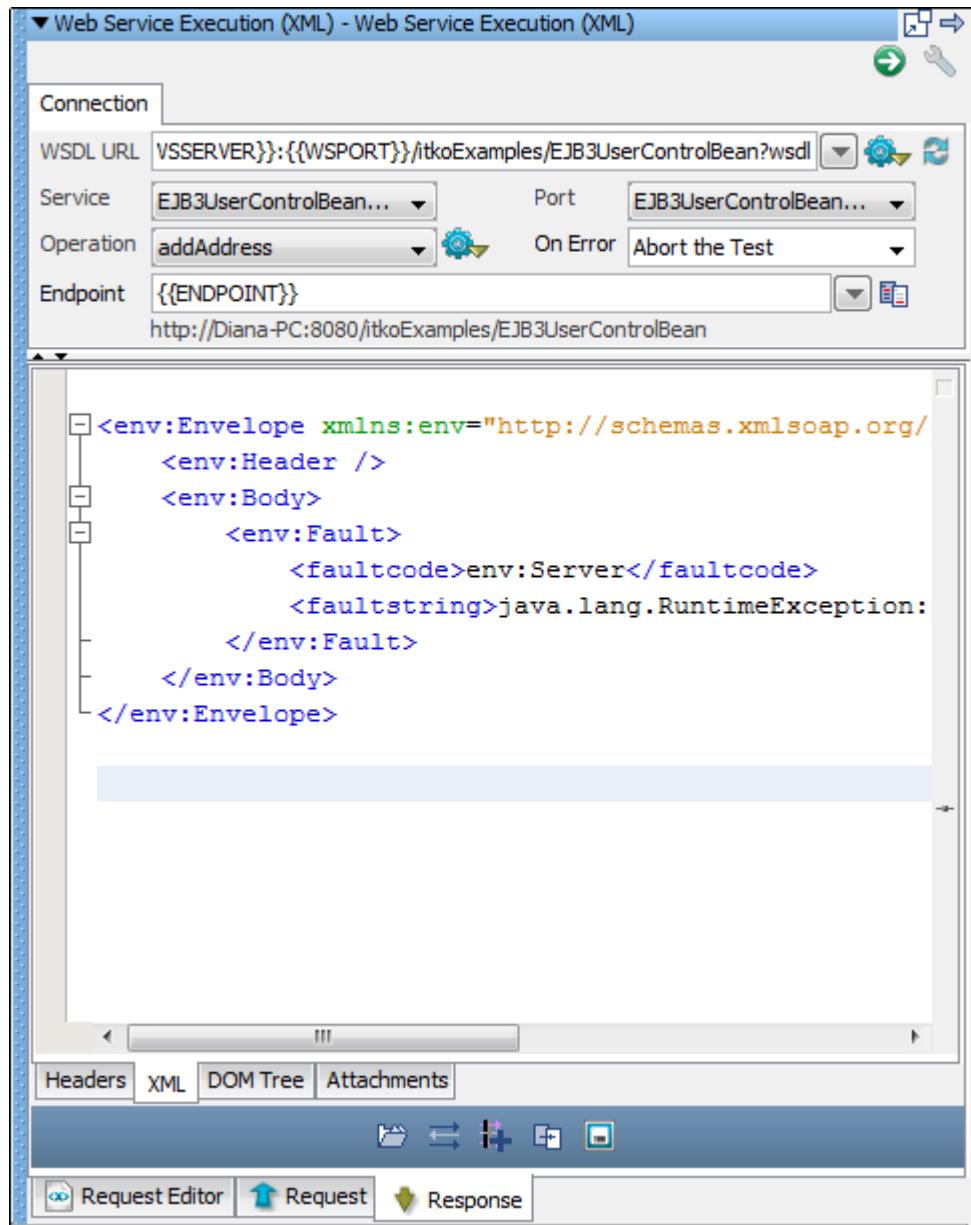
Web Service Execution (XML) step Request tab DOM Tree tab

Attachments Tab

The **Attachments** tab shows any attachments that were sent.

Response Tab

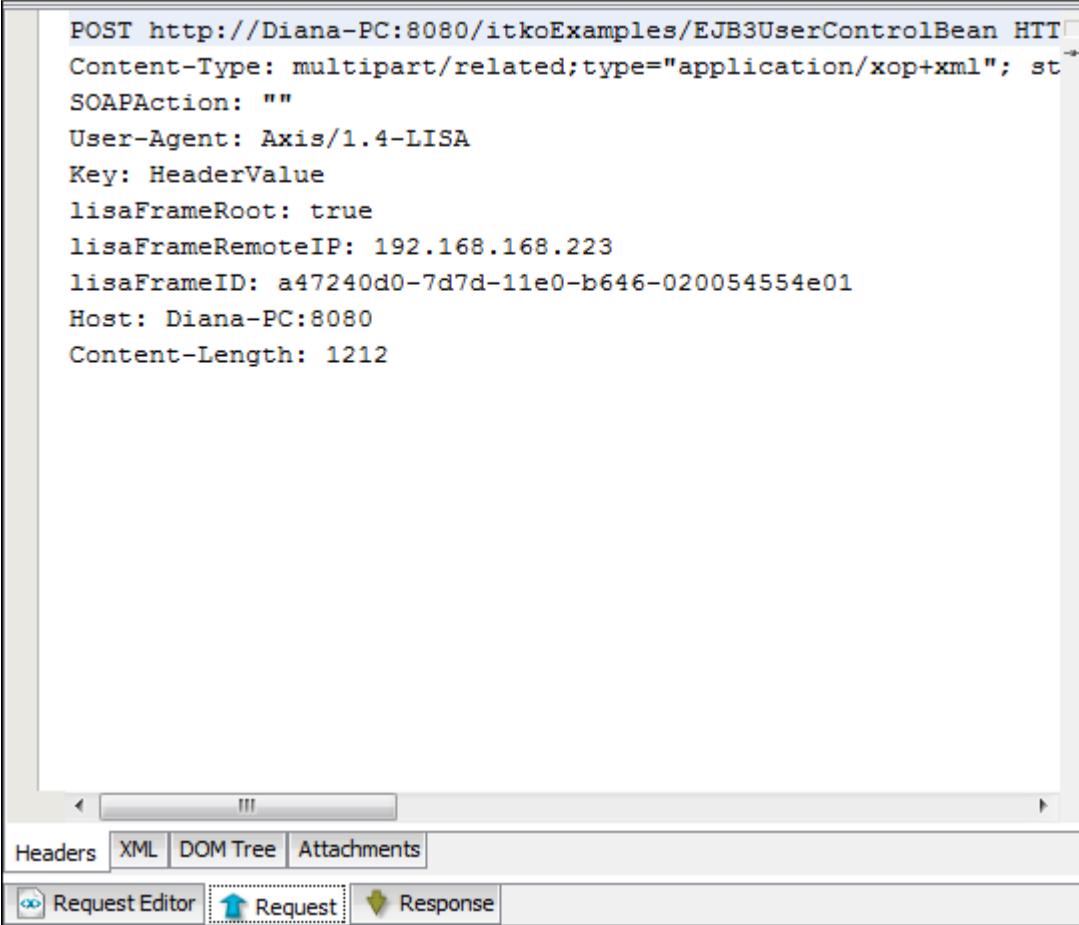
The **Response** tab shows the resulting response data that was received. If the message contained any attachment, it does not show the raw MIME or DIME encoded message, but rather the processed message and attachments. To see the raw message, use a tool like TCPMon. If any advanced post-processing options 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.



POST http://Diana-PC:8080/itkoExamples/EJB3UserControlBean HTTP/1.1
Content-Type: multipart/related;type="application/xop+xml"; start=1
SOAPAction: ""
User-Agent: Axis/1.4-LISA
Key: HeaderValue
lisaFrameRoot: true
lisaFrameRemoteIP: 192.168.168.223
lisaFrameID: a47240d0-7d7d-11e0-b646-020054554e01
Host: Diana-PC:8080
Content-Length: 1212

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.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope">
  <soapenv:Body>
    <addAddress xmlns="http://ejb3.examples.itko.com/">
      <!--username is optional-->
      <username xmlns="">hg6SE</username>
      <!--addressObject is optional-->
      <addressObject xmlns="">
        <!--city is optional-->
        <city>EXqmVcRwGpum5i</city>
        <!--id is optional-->
        <id>uLWQ6XvyBYDYOb5</id>
        <!--line1 is optional-->
        <line1>HA5JgL356</line1>
        <!--line2 is optional-->
        <line2>y58auPo</line2>
        <!--state is optional-->
        <state>Nkm9EfE0yelkJ</state>
        <zip>1</zip>
      </addressObject>
    </addAddress>
  </soapenv:Body>
</soapenv:Envelope>
```

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.

The screenshot shows the DevTest Solutions interface with the 'DOM Tree' tab selected. The tree view displays an XML envelope structure. The root node is 'soapenv:Envelope'. It contains two namespace declarations: 'xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"' and 'xmlns:xs="http://www.w3.org/2001/XMLSchema"'. The 'Body' node contains an 'addAddress' node, which has several child nodes: 'username' (with value 'hg6SE'), 'addressObject' (with 'city' value 'EXqmVcRwGpum5i'), 'id' (with value 'uLWQ6XvyBYDYOb5'), 'line1' (with value 'HA5JgL356'), 'line2' (with value 'y58auPo'), 'state' (with value 'Nkm9EfE0yelkJ'), and 'zip' (with value '1'). Below the tree view are tabs for 'Headers', 'XML', 'DOM Tree', and 'Attachments', with 'DOM Tree' being the active tab. At the bottom are buttons for 'Request Editor', 'Request', and 'Response'.

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

Contents

- [Transport Tab \(see page 1734\)](#)
- [SSL Tab \(see page 1734\)](#)
- [UDDI Tab \(see page 1736\)](#)
- [WS-I Tab \(see page 1736\)](#)
- [Advanced Tab \(see page 1737\)](#)
- [Request Editor Tabs \(see page 1738\)](#)

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) - Add User

Connection Transport PRO SSL PRO UDDI PRO WS-I PRO Advanced PRO

WSDL URL: http://{{WSSERVER}}:{{WSPORT}}/itkoExamples/EJB3UserControlBean?wsdl

Service: EJB3UserControlBeanService Port: EJB3UserControlBeanPort

Operation: addAddress On Error: Abort the Test

Endpoint: {{ENDPOINT1}}
http://localhost:8080/itkoExamples/EJB3UserControlBean

Visual XML Raw XML POST SOAP Headers Attachments Addressing PRO Security PRO

Node	Type	Occurs	Nil	Nillable	Value
soapenv:Envelope	Envelope	1..1	<input type="checkbox"/>		
soapenv:Body	Body	1..1	<input type="checkbox"/>		
addUser	addUser	1..1	<input type="checkbox"/>		
username	string	0..1	<input type="checkbox"/>		{{{unique...}}
password	string	0..1	<input type="checkbox"/>		{{{pass...}}

Validation Results View XML Schema Source Error Log

Request Editor Request Response WS-I Report PRO

Web Service Execution (XML) step Advanced Settings tabs

Transport Tab

HTTP Version 1.1 1.0

SOAP Version 1.1 1.2 {}

Call Timeout (ms) 30,000

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

SSL Keystore File:

SSL Keystore Password:

SSL Key Alias:

SSL Key Password:

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 \(see page 1702\)](#).

Specify global certificates properties for SSL in your **local.properties** file.

For global certificates (web server, raw SOAP, and web service steps):

- **ssl.client.cert.path**

A full path to the keystore.

- **ssl.client.cert.pass**

Password for the keystore (this password 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.



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

The screenshot shows the UDDI tab selected in a ribbon-style menu. Below the menu, there is a section titled "UDDI" with a checkbox labeled "Perform Access Point (Web Service URL) Lookup". Underneath this, there are two lines of text: "inquiry url not selected" and "binding template key not selected". To the right of the text is a magnifying glass icon.

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** 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

The screenshot shows the WS-I tab selected in a ribbon-style menu. Below the menu, there is a dropdown menu labeled "WS-I Basic Profile 1.1" with the option "Display Only Failed Assertions" selected. There are also sections for "Validate" (checkboxes for "WSDL" and "SOAP Message") and "On Failure Go To" (dropdown menu set to "Generate Error").

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

Connection	Transport	SSL	UDDI	WS-I	Advanced
<input type="text" value="getItemsByTitle"/> ▼ 					
Style	<input checked="" type="radio"/> Document <input type="radio"/> RPC				
Use	<input checked="" type="radio"/> Literal <input type="radio"/> Encoded <input type="text"/> ▼ 				
<input checked="" type="checkbox"/> SOAP Fault is Error <input type="checkbox"/> Do not send request <input checked="" type="checkbox"/> Maintain Session <input checked="" type="checkbox"/> Clear Session					

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.

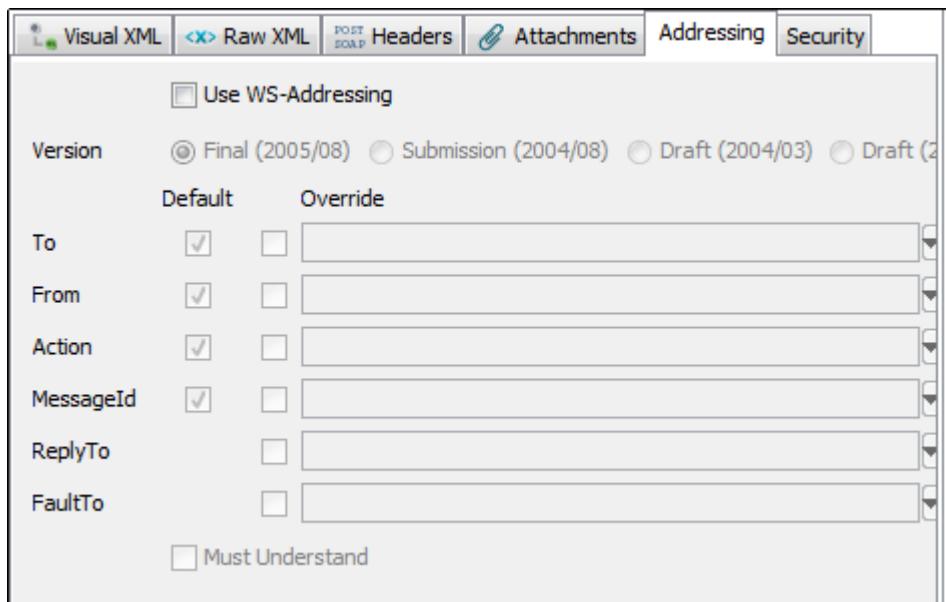
- **Clear Session**

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.

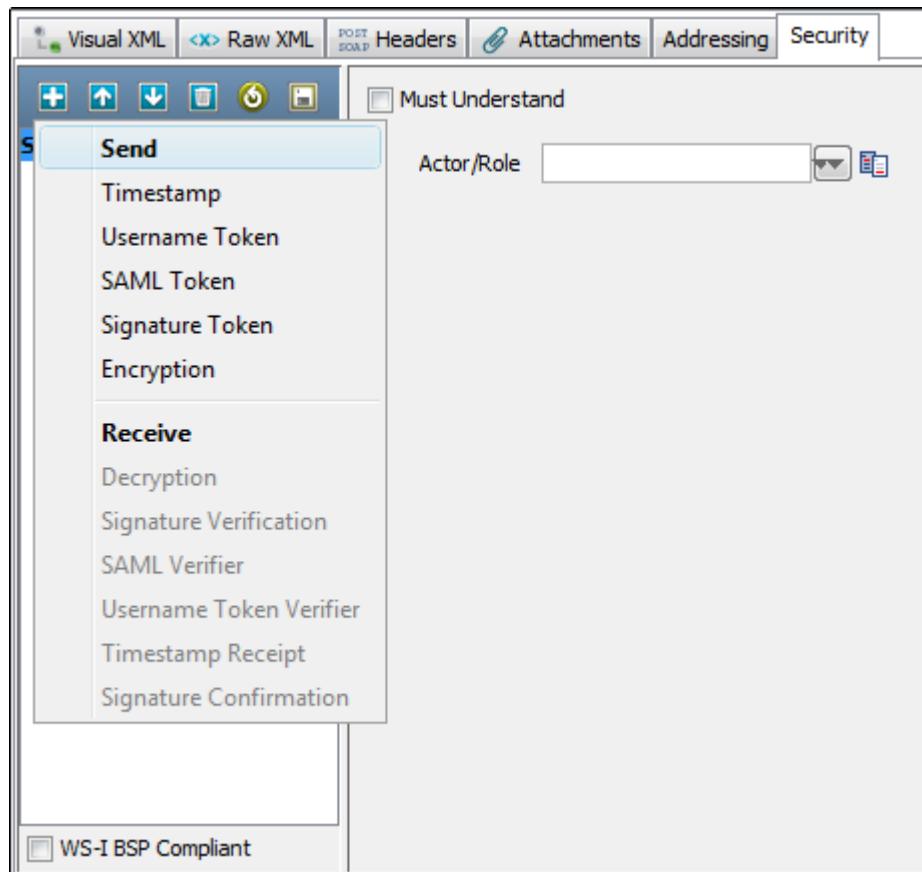
- **Version**

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.



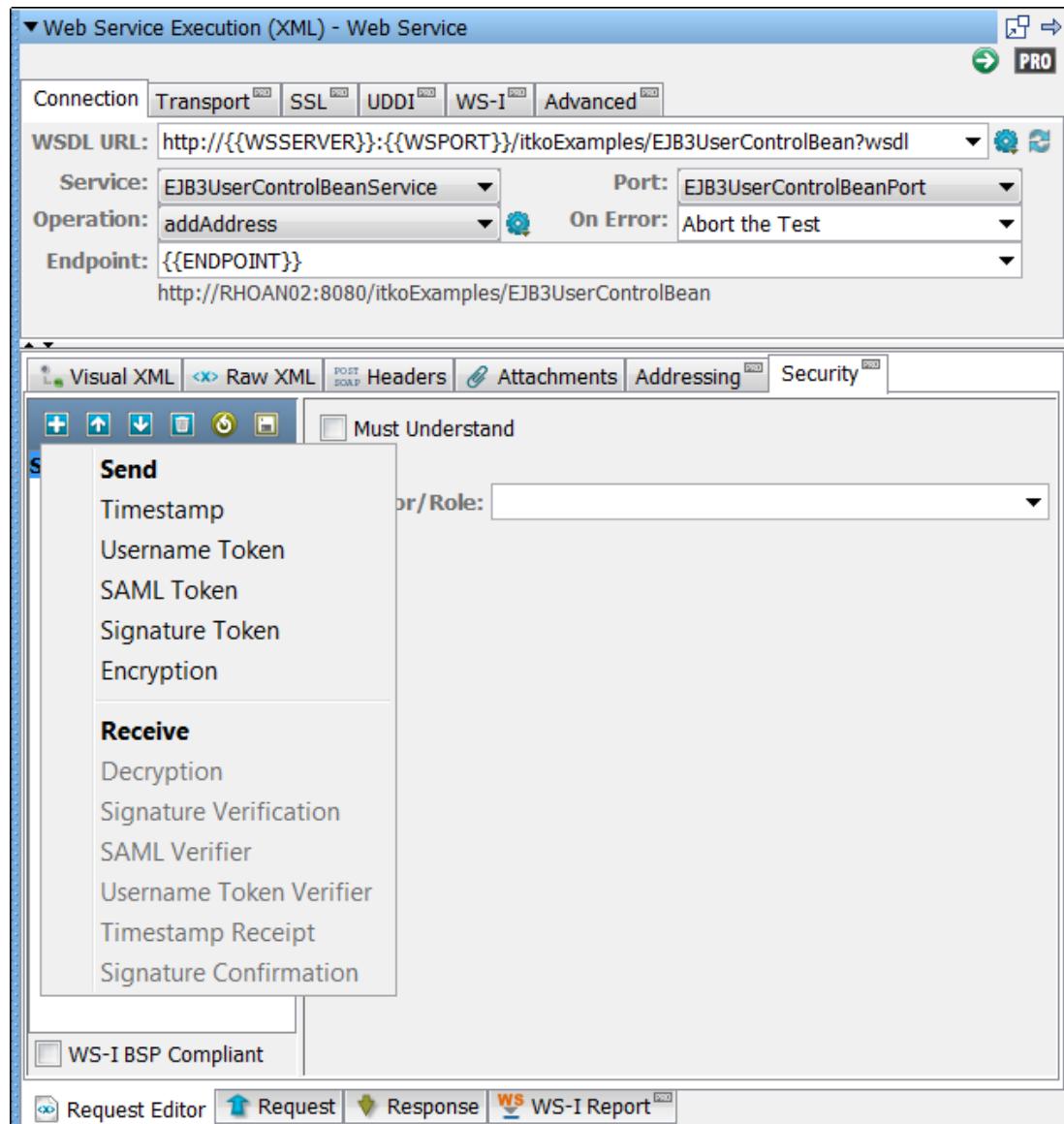
Note: You can load and save the security configuration information from and to a .wss file, using the **Load** and **Save** icons. This capability allows for quick and easy creation of new steps connecting to the same service.

Security Example

Contents

- [XML Encryption-Decryption \(see page 1741\)](#)
- [XML Signature Token-Signature Verification \(see page 1743\)](#)
- [Timestamp-Timestamp Receipt \(see page 1744\)](#)
- [Username Token-Username Token Verifier \(see page 1744\)](#)
- [SAML Assertion Token-SAML Assertion Receipt \(see page 1746\)](#)

This section describes the parameters that are necessary to run the WS-security example.



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 the assertion 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 a little 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 \(see page 1470\)](#).

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.

Web-Raw SOAP Request

The Raw SOAP Request step lets you test a web service by sending a raw SOAP request (raw XML). You can use this step to test legacy SOAP calls or web services that do not have a WSDL. This step also lets you test the reaction of a web service to data of an incorrect type. For example, sending a string when it expects a number, which is not allowed in the Web Service Execution step. Another use for the Raw SOAP request is to reduce overhead during intense load tests. The regular web service step has some additional overhead because it marshals an object into XML to make the request. This step then unmarshals the SOAP XML response back into an object. The Raw SOAP step avoids this overhead and only deals with the raw SOAP XML. Because it has less work to do, it executes faster.

You can type or paste the SOAP request into the editor. The request can also be read from a file, and then parameterized using DevTest properties.

Dynamic WS-Addressing or WS-Security headers are not supported. To have these types of headers, enter them statically as part of the SOAP request in the input area. If your request contains items like a WS-Security signature token, the signed elements cannot be parameterized or the signature is no longer valid.



Note: This step is not limited to SOAP calls. You can also do XML or text POSTs.

To create a Raw SOAP Request:

Complete the following fields:

- **SOAP Server URL**

Enter the URL of the web service endpoint. The URL is converted into a single property instead of simply substituting the WSSERVER and PORT properties.

- **SOAP Action**

Complete the SOAP action as indicated in <soap: operation> tag in the WSDL for the method being called. This action is required for SOAP 1.1 and is typically required to be left blank for SOAP 1.2.

- **Content Type**

Select the Content Type. Use text/HTML for SOAP 1.1, application/SOAP+XML for SOAP 1.2.

- **Advanced button**

Click to add any custom HTTP headers.

- **Discard response**

Check to discard the response, replacing it with a small valid but static SOAP text. This feature is intended for load testing where processing a large response limits the scalability of the load generator computers.

Type or paste the SOAP Request into the editor, or click **Read Request From File** and browse to the file containing the SOAP Request.

Now you can parameterize the request with properties.

To execute the call, click **Test**.

To examine the response, click the **Results** tab.

You are now ready to add filters and 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.

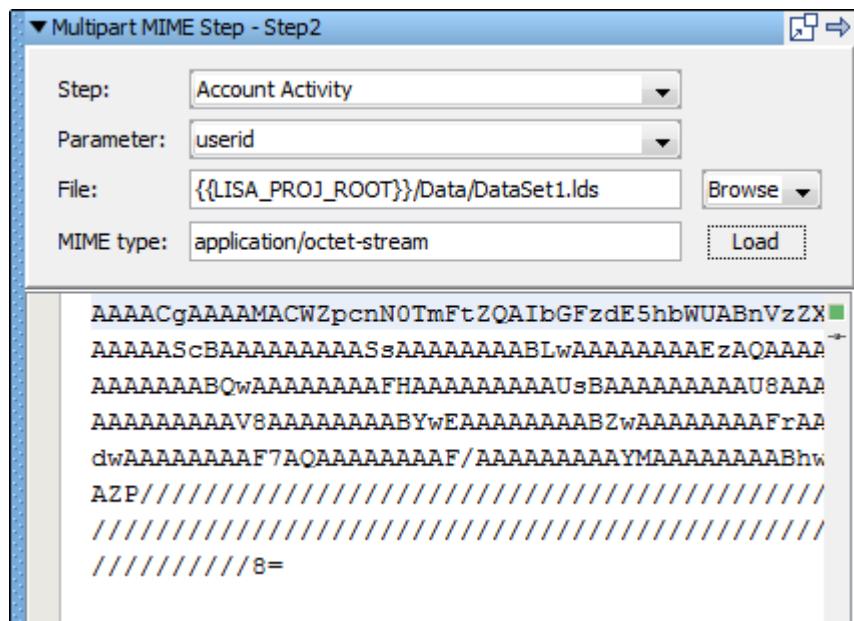
▪ **File**

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

Contents

- [Connection \(see page 1754\)](#)
- [Subject \(see page 1755\)](#)
- [Response \(deprecated\) \(see page 1755\)](#)
- [Query \(see page 1755\)](#)

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.

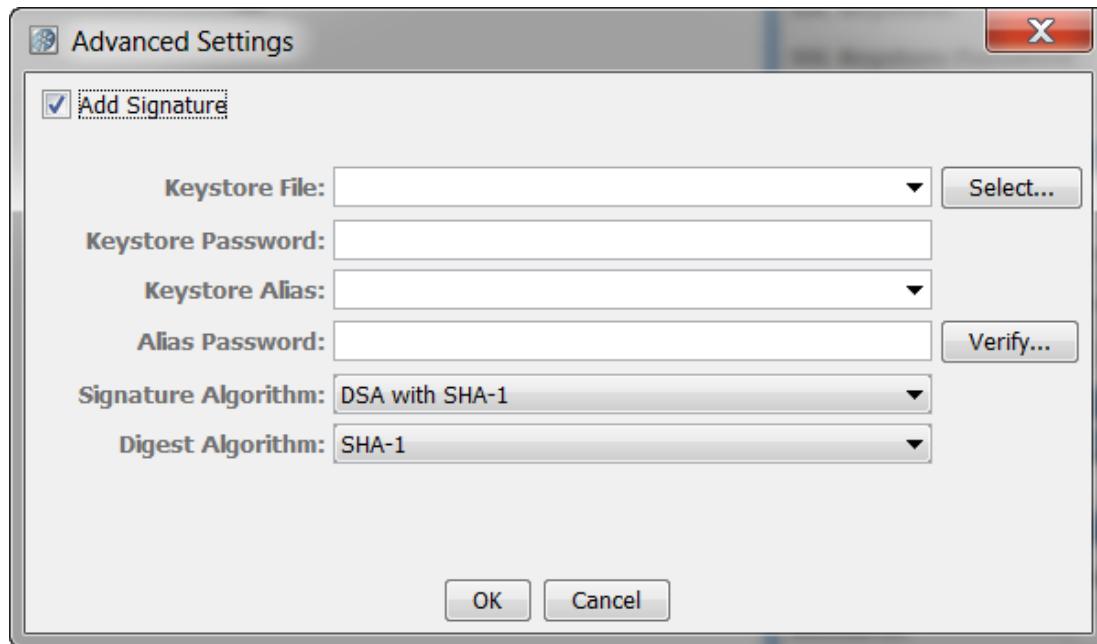
▼ SAML Assertion Query - SAML Assertion Query

Connection		
Endpoint:	http://wayf.internet2.edu:443/shibboleth-idp/AA	
SSL Keystore:	<input type="button" value="Select..."/>	
SSL Keystore Password:		
SAML Version:	<input type="radio"/> 1.0	<input checked="" type="radio"/> 1.1
<input type="button" value="Advanced..."/>		
Subject		
Name:	test-handle	
Name Qualifier:	urn:mace:enqueue:example:edu	
Format:	urn:mace:shibboleth:test:nameidentifier	
Confirmation Methods:	<input type="checkbox"/> Holder of Key <input type="checkbox"/> Sender Vouches <input type="checkbox"/> Bearer <input type="checkbox"/> Artifact	
Response (deprecated)		
Respond With:	Local Part	Namespace
Query		
Type:	<input checked="" type="radio"/> Attribute	<input type="radio"/> Authorization
<input type="button" value="Test"/>		
Resource:	<input type="button" value=""/>	
Attribute Designators:	Name	Namespace
<input type="button" value="Editor"/> <input type="button" value="Last Request"/> <input type="button" value="Raw Query Result"/> <input type="button" value="Last Response"/>		

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 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.

- **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

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:

- [Dynamic Java Execution \(see page 1757\)](#)
- [RMI Server Execution \(see page 1760\)](#)
- [Enterprise JavaBean Execution \(see page 1764\)](#)

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.



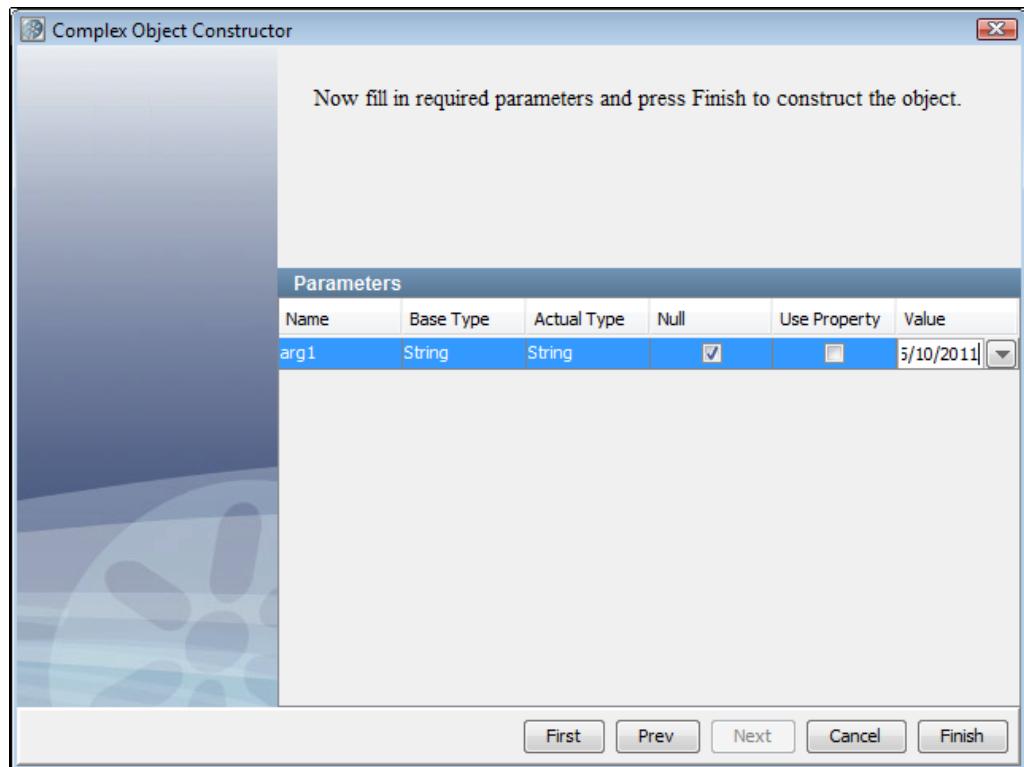
Note: If you require that the Java object be loaded by its own classloader, add the **Class Loader Sandbox Companion**.

2. Click **Construct/Load Object**.

The **Complex Object Constructor** window opens and lists the available constructors for your object.

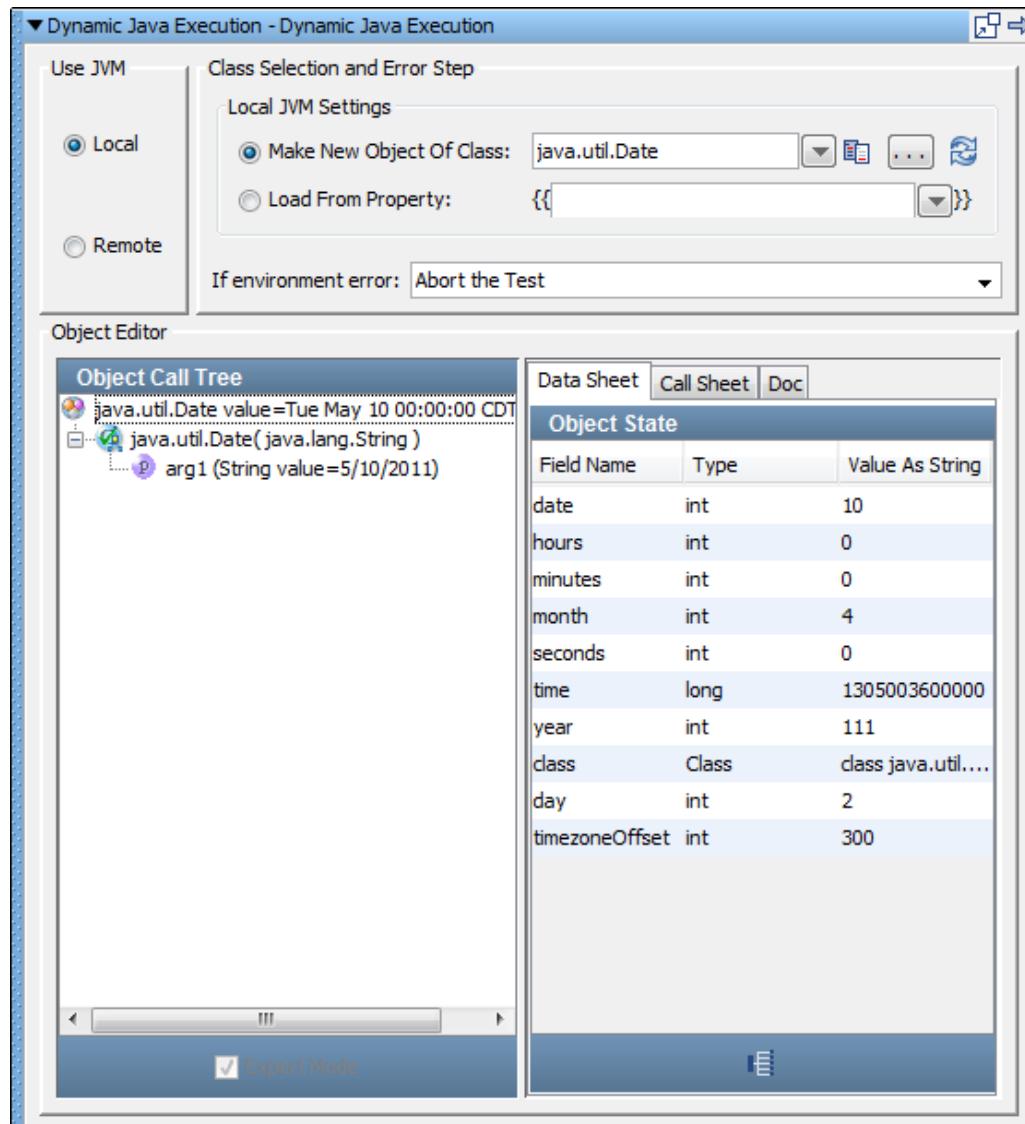
3. Select a constructor and click **Next**.

4. Enter any input parameters that the constructor needs.



Dynamic Java Execution step Complex Object Constructor

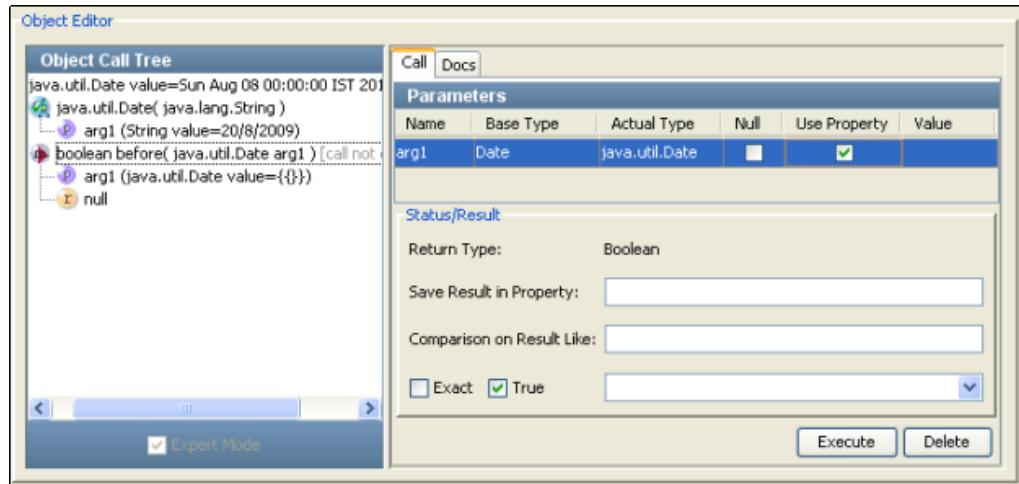
5. In this example, enter a string representation of a day (5/10/2011). You can enter a value, a property, or null. DevTest constructs the object and loads it into the Complex Object Editor.



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 \(COE\) \(see page 443\)](#).
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 complete the following actions:

- Acquire a reference to a remote Java object through RMI (Remote Method Invocation)
- 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.

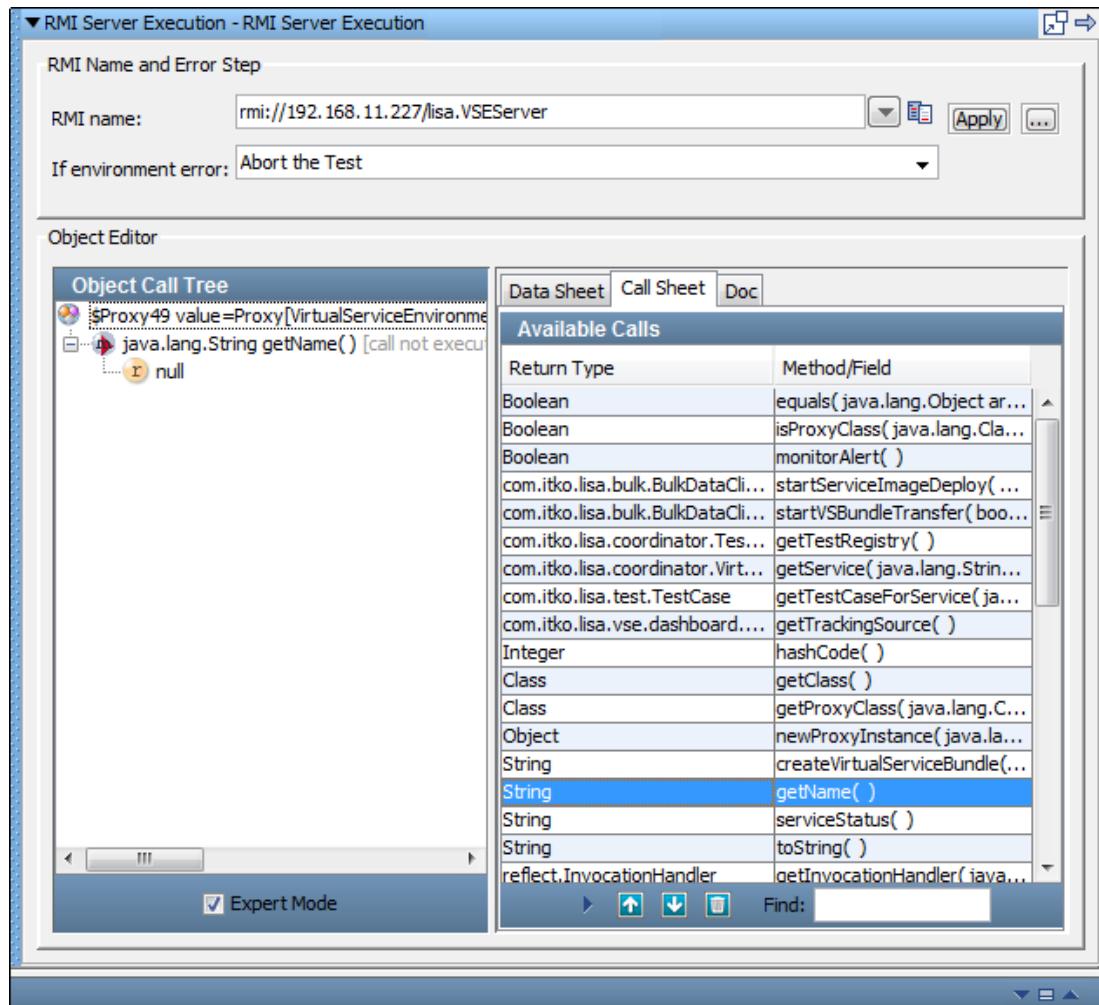
The RMI Server step editor lets you enter the following parameters:

▪ **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.

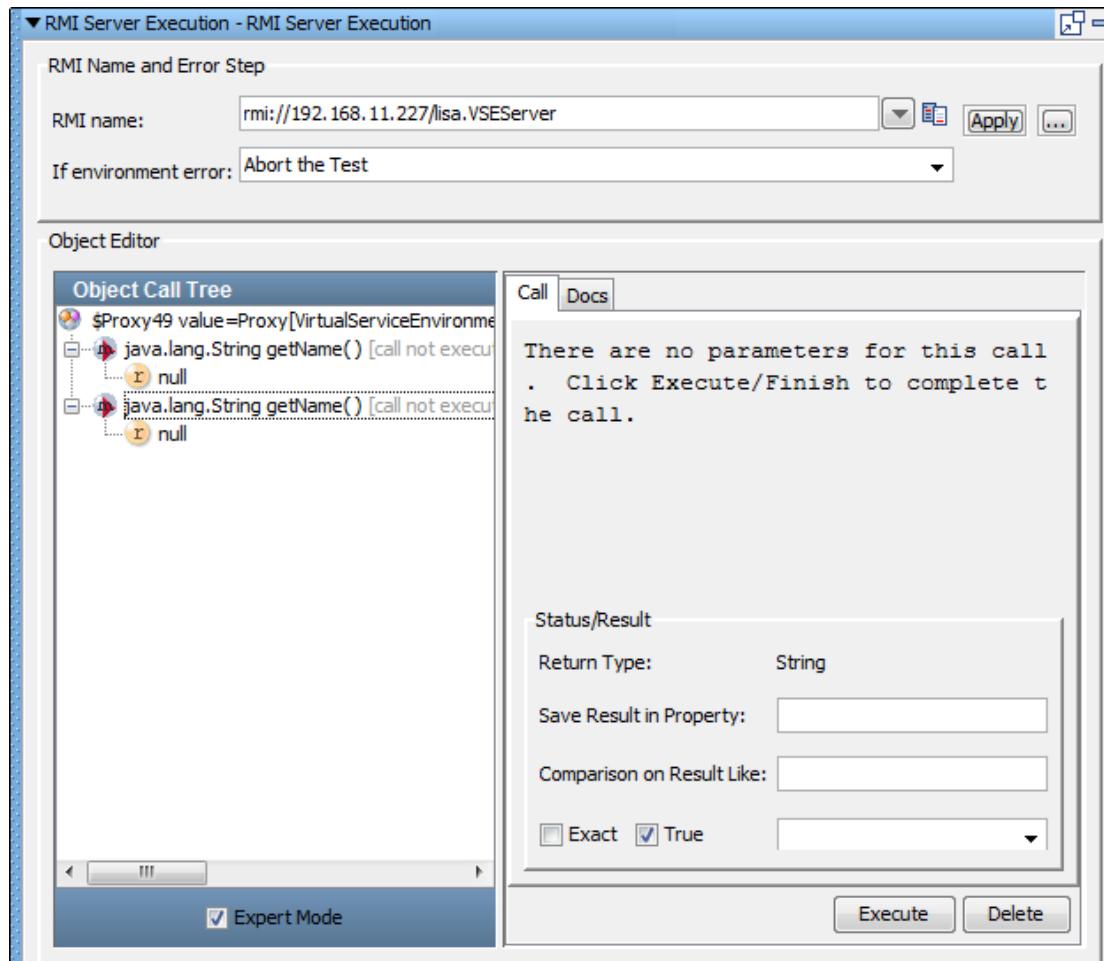
DevTest constructs the object and loads it into the Complex Object Editor.



COE with getName method selected

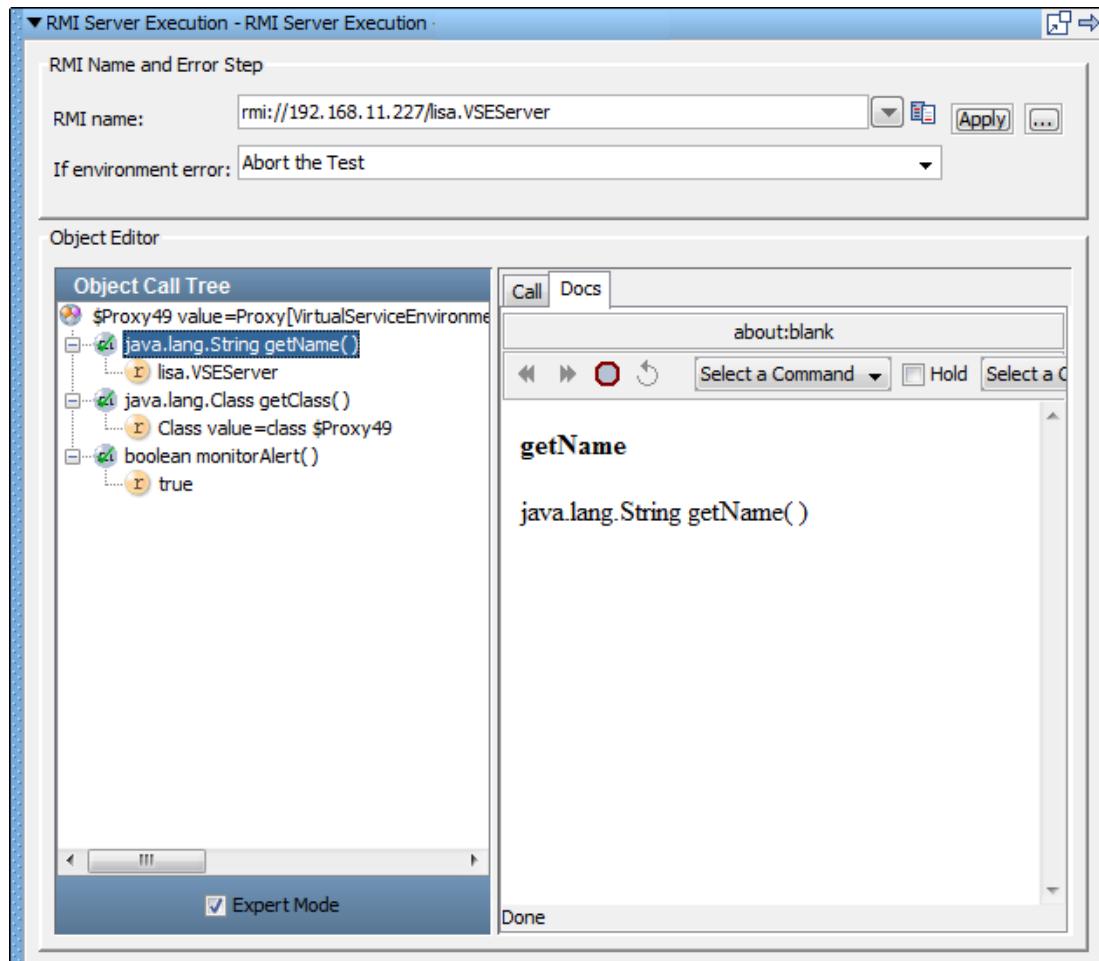
You can now manipulate the object, and execute methods, using the Complex Object Editor.

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

Contents

- [Connecting to WebSphere with DevTest Solutions using SIBC \(see page 1764\)](#)
- [Example \(see page 1765\)](#)

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.

Prerequisites: Knowledge of the Complex Object Editor is assumed. The application client JAR and client EJB JAR must be in the DevTest classpath. Both of these JAR files are copied into the hotDeploy directory. The hotDeploy directory contains the jboss-all-client.jar file for the JBoss application server and the examples JAR file so you can run the EJB examples immediately.

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.

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. The client is available [here](http://www-01.ibm.com/support/docview.wss?rs=0%26uid=swg24012804) (<http://www-01.ibm.com/support/docview.wss?rs=0%26uid=swg24012804>).

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

Example

This example uses the ITKO example server, a JBoss server. To use your local demo server, use localhost as the host name.

1. Enter the following parameters:

- **Choose App Server**

Select your application server from the list. If your application server is not on the list, select the **Other/You Specify** option.

The lower section of the editor changes, depending on your selection. The previous graphic shows the configuration panel for JBoss.

2. For the JBoss panel, 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.

3. Click **Next**.

For the Other/You Specify Window

1. 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**

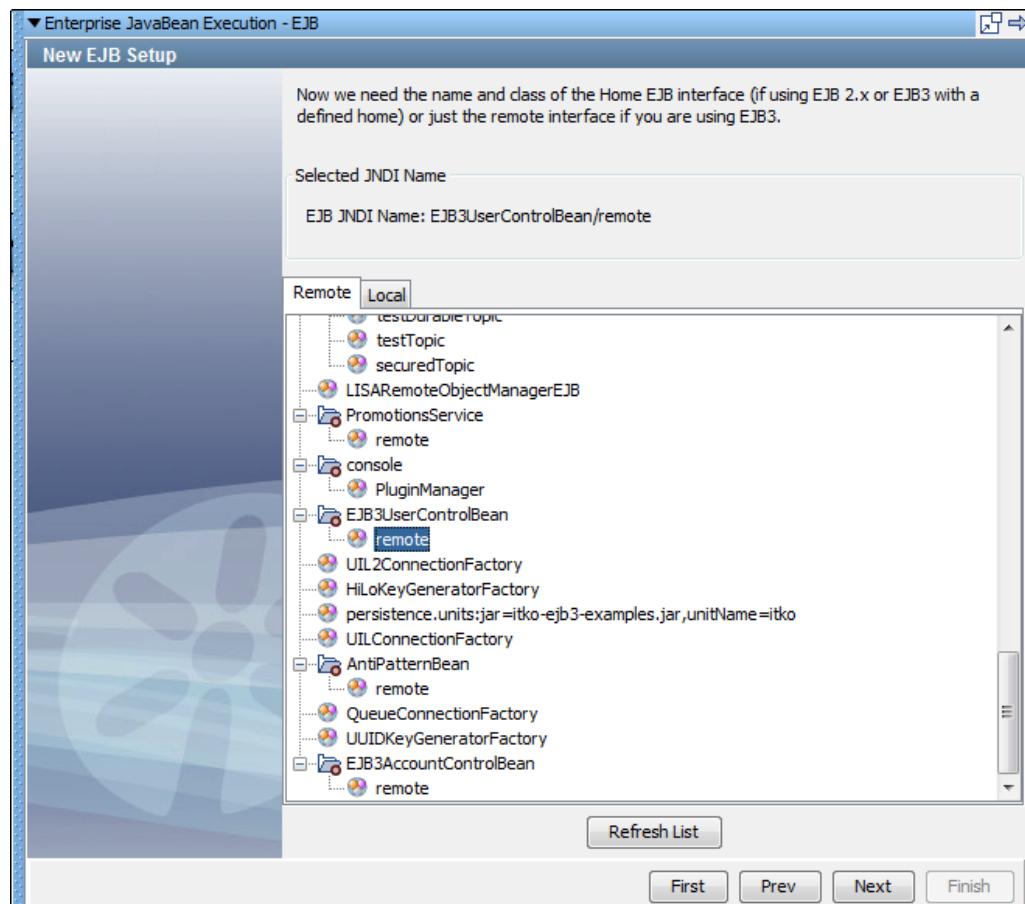
Enter if a user ID is required for the application server.

▪ **Password**

Enter if a password is required for the application server.

2. Click **Next**.

The **New EJB Setup** window opens and lists all the JNDI names that are registered with the application server.



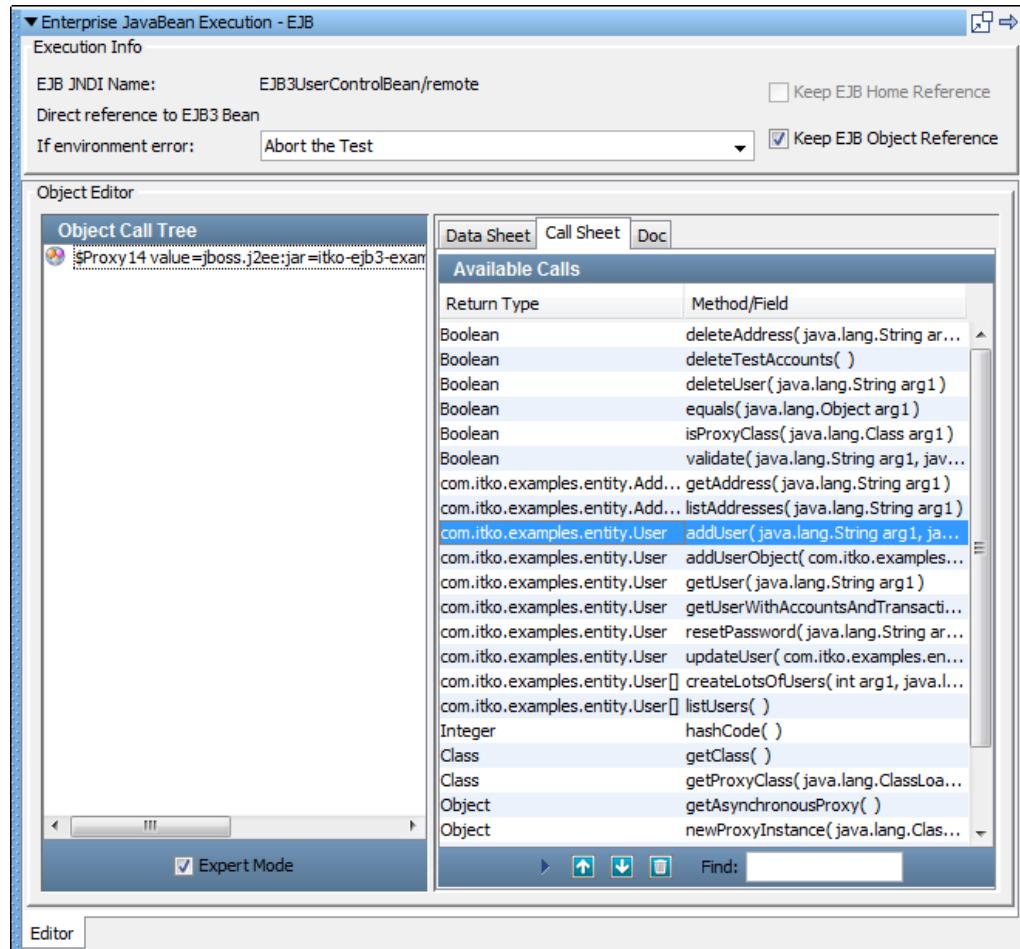
Enterprise JavaBean Execution step New EJB Setup window

3. Select the name of the appropriate EJB home interface.

In this example, the JNDI name is **com.itko.examples.ejb.UserControlBean**. The EJB3 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.

4. 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 following check boxes:

- **Keep EJB Home Reference**
- **Keep EJB Object Reference**

If the bean is an EJB 3 bean without a home interface, the **Keep EJB Home Reference** check box is disabled. Set the **If Exception** to the step to redirect to if an exception occurs.

The EJB step has a default name using this convention: *EJB javaMethod dynamic java execution*. Before the step is saved, is entered, as shown previously, 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.

5. 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 \(see page 1760\)](#) step.

Other Transaction Steps

The following steps are available:

- [SQL Database Execution \(JDBC\) \(see page 1768\)](#)
- [SQL Database Execution \(JDBC with Asset\) \(see page 1771\)](#)
- [CORBA Execution \(see page 1773\)](#)

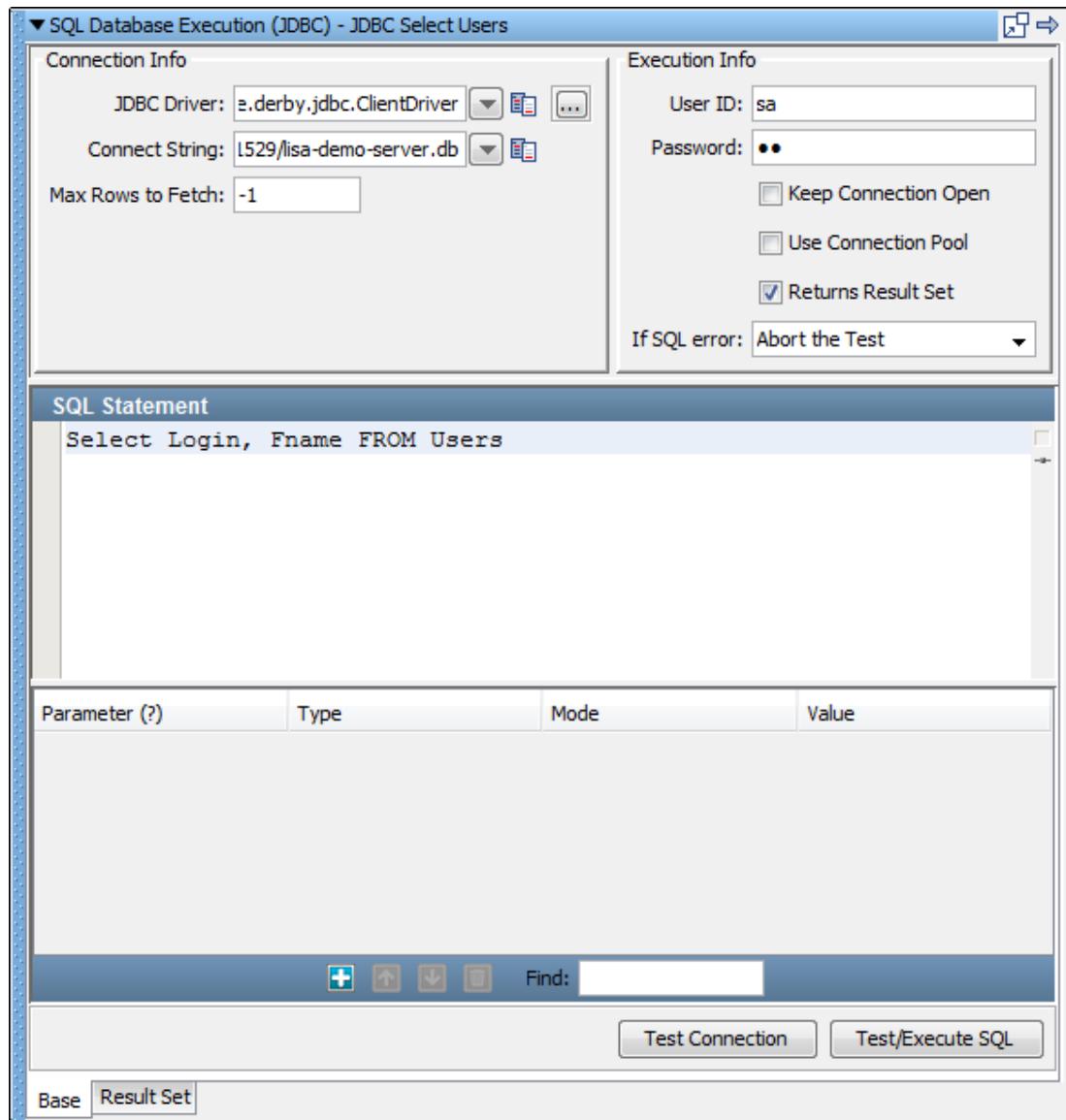
SQL Database Execution (JDBC)

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.

- **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 \(see page 1587\)](#) and [Assertion Descriptions \(see page 1470\)](#).

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.

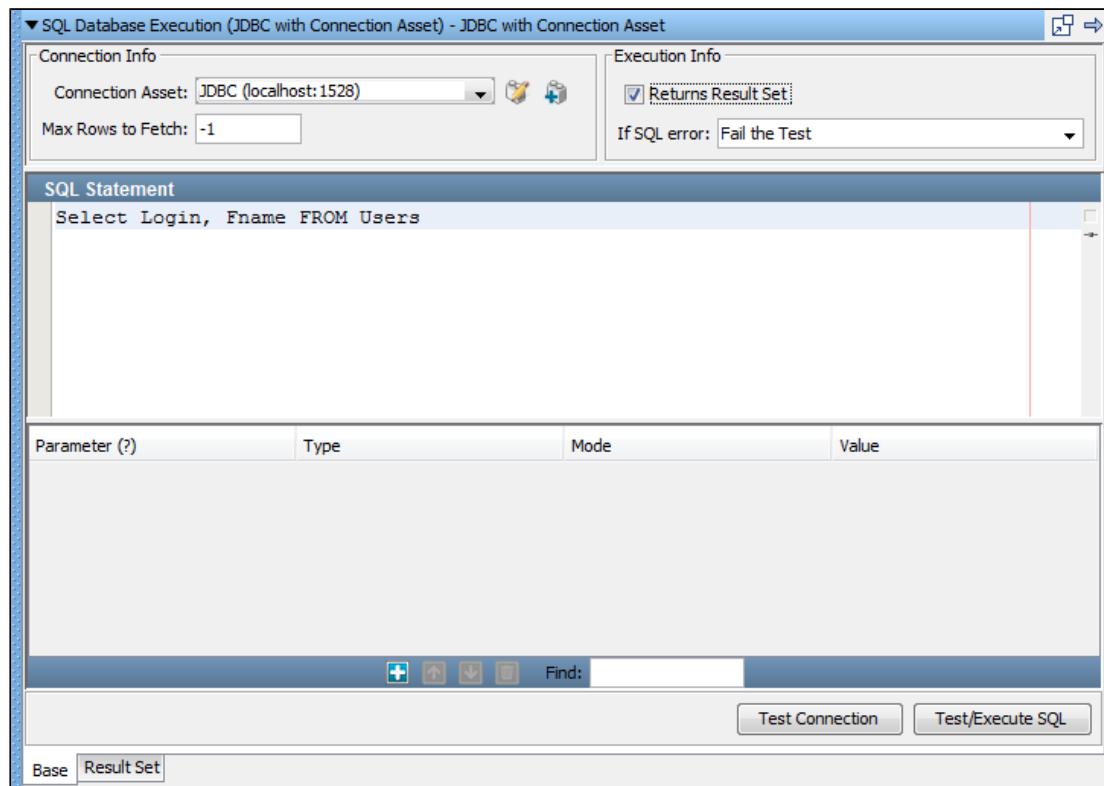
SQL Database Execution (JDBC with Asset)

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 \(see page 1514\)](#) 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 \(see page 403\)](#).

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](#) (see page 1514) 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 \(see page 1587\)](#) and [Assertion Descriptions \(see page 1470\)](#).

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:

- [Save Property as Last Response \(see page 1775\)](#)
- [Output Log Message \(see page 1775\)](#)
- [Write to Delimited File \(see page 1775\)](#)
- [Read Properties from a File \(see page 1776\)](#)
- [Do-Nothing Step \(see page 1777\)](#)
- [Parse Text as Response \(see page 1778\)](#)
- [Audit Step \(see page 1778\)](#)
- [Base64 Encoder Step \(see page 1779\)](#)
- [Checksum Step \(see page 1779\)](#)
- [Convert XML to Element Object \(see page 1780\)](#)
- [Compare Strings for Response Lookup \(see page 1781\)](#)
- [Compare Strings for Next Step Lookup \(see page 1783\)](#)
- [Send Email \(see page 1784\)](#)

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.
- **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.

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**

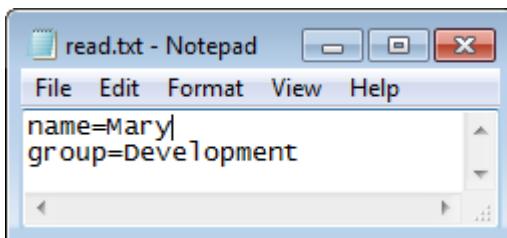
Enter the path name of the file, or use the **Browse** button to browse to the file.

▪ 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.

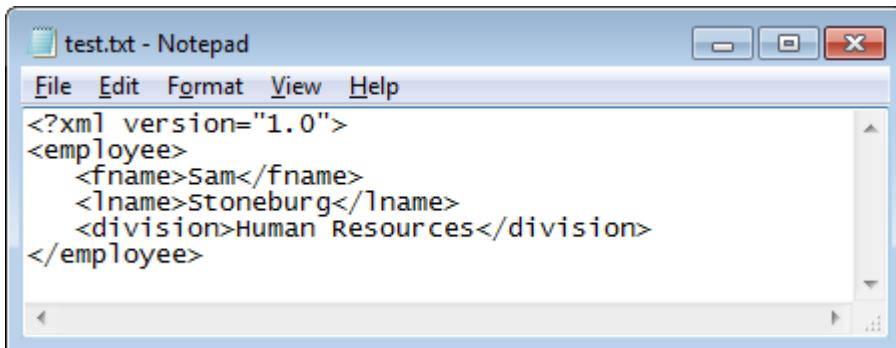
▪ Type of File

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 have any functionality by itself. However, this step is useful in specific 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 \(see page 510\)](#) 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

This 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 a 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.

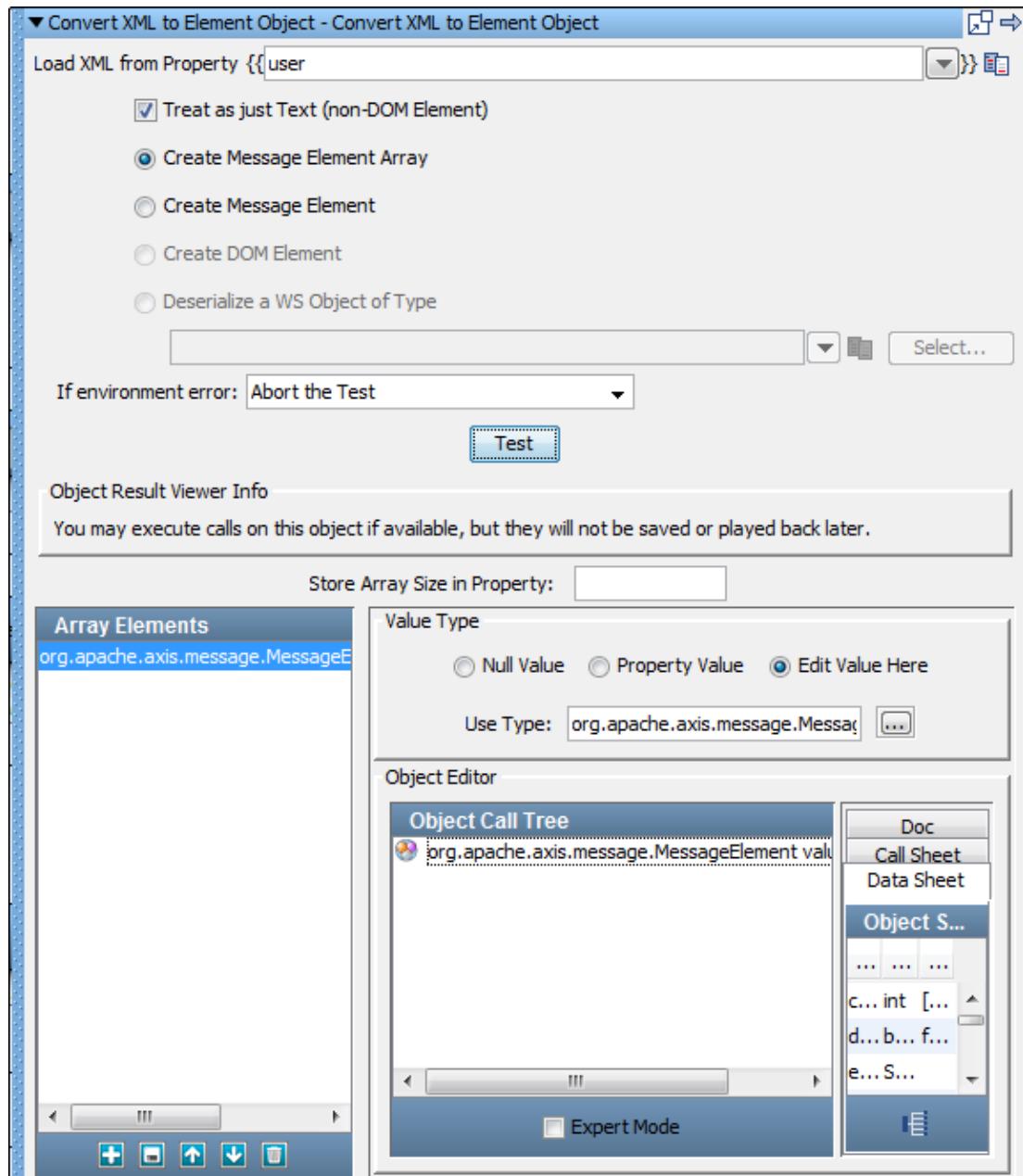
- **Treat as just Text**

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

This 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, among 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

To send an email notification from a test case, use the Send Email step.

Complete the following fields:

- **Connection**
Specifies the [asset \(see page 1511\)](#) 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:

- [Execute External Command \(see page 1786\)](#)
- [File System Snapshot \(see page 1788\)](#)
- [Execute Subprocess \(see page 1788\)](#)
- [JUnit Test Case-Suite \(see page 1789\)](#)
- [Read a File \(Disk URL or Classpath\) \(see page 1790\)](#)
- [External - FTP Step \(see page 1791\)](#)

Execute External Command

Use the Execute External Command step to 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 external program syntax depends on your operating system.

Enter the following parameters in the Execute External Command editor:

- **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.

- **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.

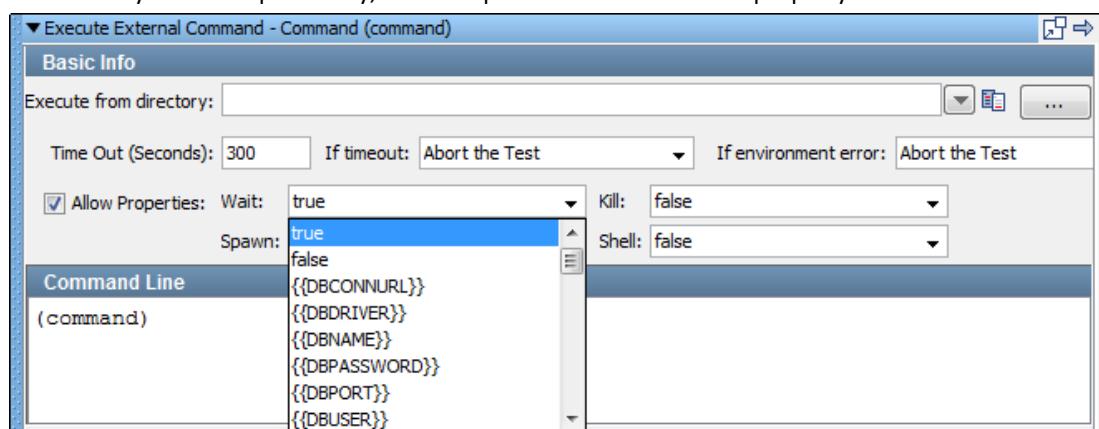
- **Spawn Process**

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 \(see page 489\)](#).

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.

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 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 <ldtF> select an encoding type for you.

- **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.

- **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 FTP protocol. After entering the FTP information, user name, and user password you can either upload a file or download a file.

If necessary, 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 and tutorial are available:

- [JMS Messaging \(JNDI\) \(see page 1792\)](#)
- [JMS Messaging - Message Consumer \(see page 1800\)](#)
- [JMS Send Receive Step \(see page 1803\)](#)
- [Tutorial - Send and Receive a JMS Message \(see page 1807\)](#)

JMS Messaging (JNDI)

Contents

- [Base Tab \(see page 1793\)](#)
- [Selector Query Tab \(see page 1798\)](#)
- [Send Message Data Tab \(see page 1798\)](#)
- [Response Message Tab \(see page 1799\)](#)

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 \(see page 1380\)](#).

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:

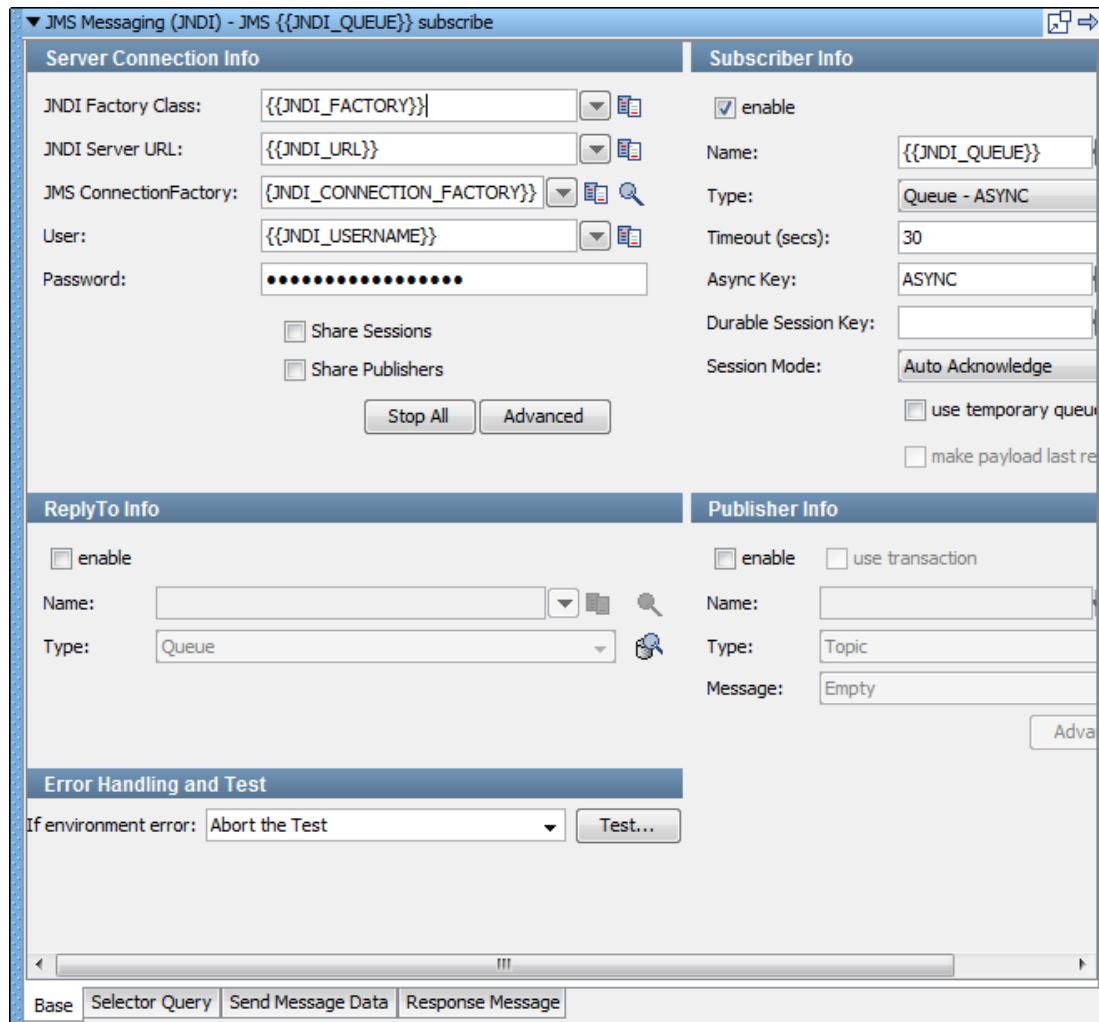
- The **Base** tab is where you define the connection and messaging parameters.
- The **Selector Query** tab lets you specify a selector query to be run when listening for a message on a queue.
- The **Send Message Data** tab is where you create the message content.
- The **Response Message** tab is where the response messages are posted.

Base Tab

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](https://support.ca.com/irj/portal/kbtech?docid=603964&searchID=TEC603964&fromKBResultsScreen=T) (<https://support.ca.com/irj/portal/kbtech?docid=603964&searchID=TEC603964&fromKBResultsScreen=T>) 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 **JMS 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.



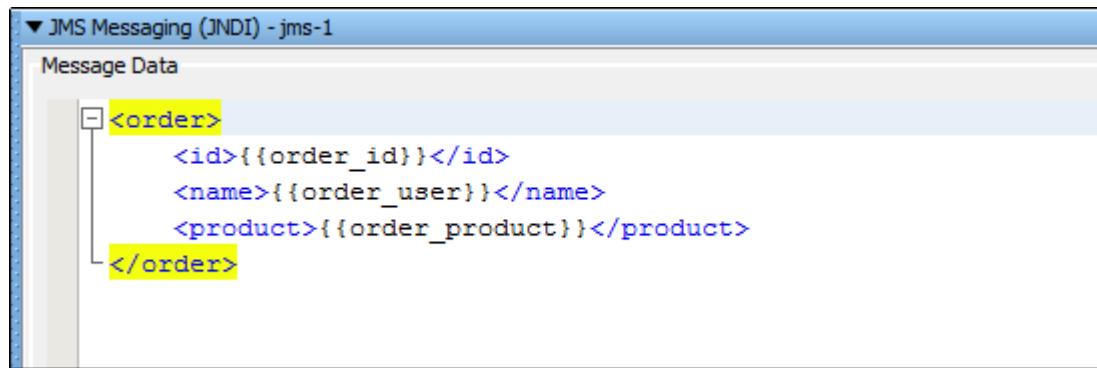
JMS Messaging (JNDI) step Selector Query Tab

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 \(see page 443\)](#) 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.

The screenshot shows the DevTest Object Editor interface. The title bar reads "JMS Messaging (JNDI) - JMS {{JNDI_QUEUE}} subscribe". The main area is titled "Object Call Tree" and shows a single entry: "org.jboss.mq.SpyMessage value=org.jboss.mq...". Below this is the "Object State" table, which lists various JMS message properties:

Field Name	Type	Value As String
JMSCorrelationID	String	(null)
JMSDeliveryMode	int	-1
JMSExpiration	long	0
JMSMessageID	String	(null)
JMSPriority	int	-1
JMSRedelivered	boolean	false
JMSTimestamp	long	0
JMSType	String	(null)
JMSCorrelationIDAsB... [B	[B	[Access Not Allowed]
JMSDestination	Destination	(null)
JMSReplyTo	Destination	(null)
class	Class	class org.jboss.mq.S...
outdated	boolean	false
propertyNames	Enumeration	java.util.Collections\$...

At the bottom of the editor, there is an "Expert Mode" checkbox checked, and tabs for "Editor", "Base", "Selector Query", "Send Message Data", and "Response Message".

JMS Messaging (JNDI) step Response Message Tab

JMS Messaging - Message Consumer

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.

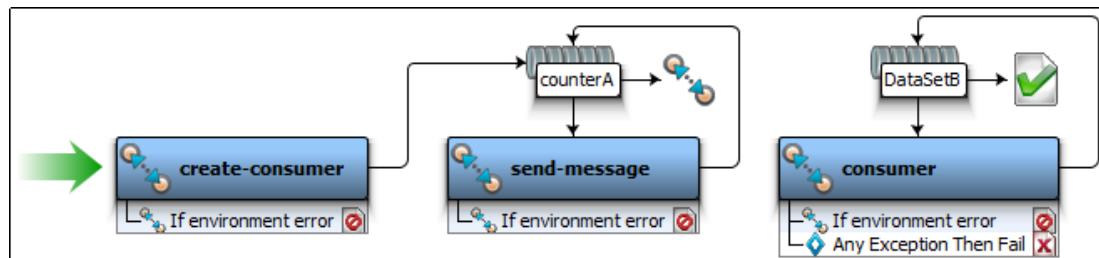
Prerequisite: You must have the demo server running before you execute the example test case.

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.

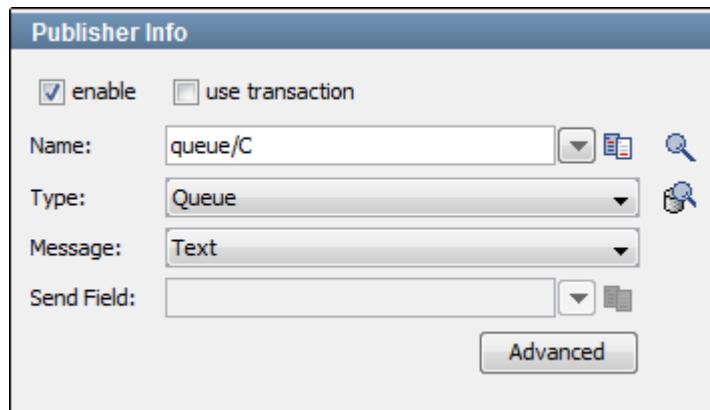
Subscriber Info	
<input checked="" type="checkbox"/> enable	
Name:	topic/testTopic
Type:	Topic - ASYNC
Timeout (secs):	30
Queue Model:	
Async Key:	EXAMPLE-ASYNC-WRAPPER
Durable Session Key:	
Session Mode:	Auto Acknowledge
<input type="checkbox"/> use temporary queue/topic	

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

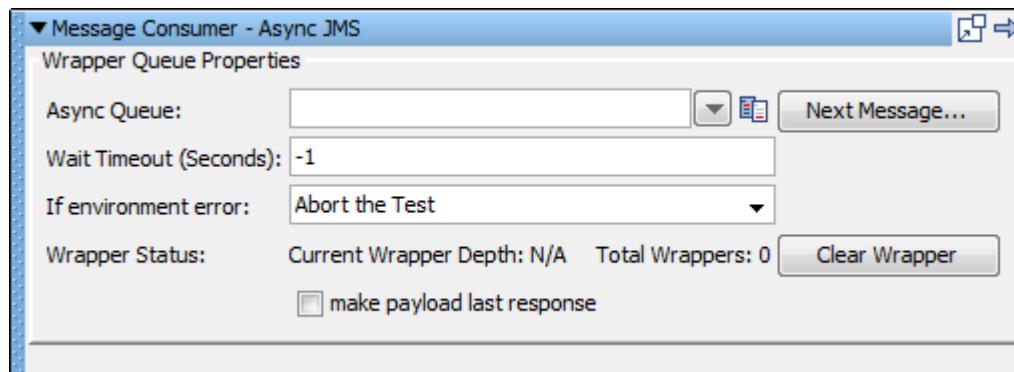
Select whether you are using a topic or queue. To see what messages are waiting to be consumed from a queue (only), use **Browse** to the right of this field.

▪ 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

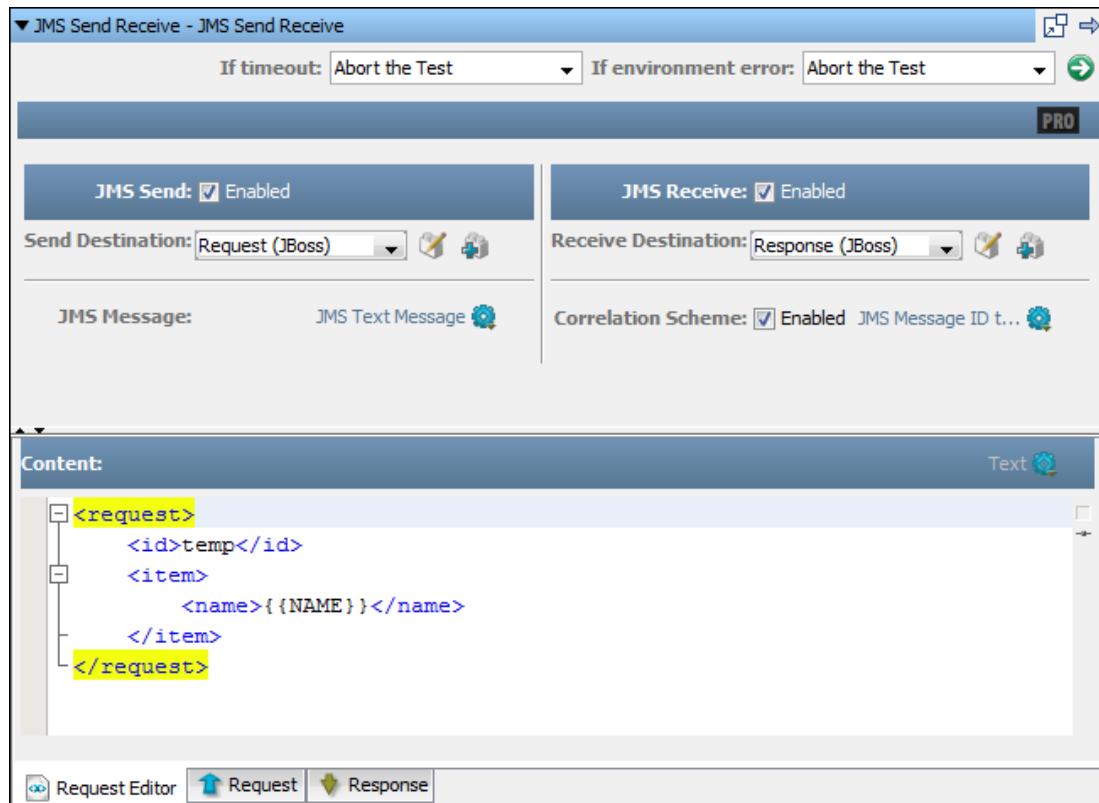
The **JMS Send Receive** step lets you connect to any JMS-compatible message server and do the following actions:

- 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.

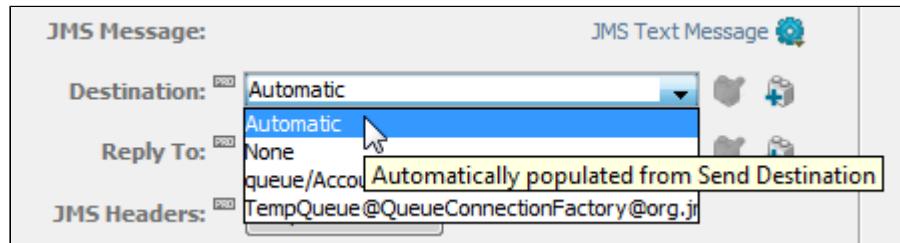


Screen capture of send receive step.

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

The **JMS Send Receive** step has separate areas for configuring the JMS send operation and the JMS receive operation.

Specify two [destinations \(see page 1516\)](#) 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 \(see page 1516\)](#). 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 \(see page 1555\)](#).

Each asset has a [runtime scope \(see page 402\)](#). 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](#) (see [page 1516](#)):

```
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.

Name	Type	Scope	Status
consumer:queue/B@ConnectionFactory@org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099	JMS Consumer	jms-demo	Yellow
ConnectionFactory@org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099	JMS Session	jms-demo	Green
ConnectionFactory@org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099	JMS Connection	jms-demo	Green
ConnectionFactory@org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099	JMS Generic Connection Factory	jms-demo	Yellow
org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099	JNDI Initial Context	jms-demo	Yellow
queue/B	JMS JNDI Queue	jms-demo	Yellow
producer:queue/A@ConnectionFactory@org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099	JMS Producer	jms-demo	Yellow
queue/A	JMS JNDI Queue	jms-demo	Yellow

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. Select 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:
 - a. **Name**: The name of the asset.
 - b. **Type**: The type of asset.
 - c. **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.
 - d. **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

Contents

- [Step 1 - Create JMS and JNDI Assets \(see page 1808\)](#)
- [Step 2 - Configure the JMS Send Receive Step \(see page 1808\)](#)
- [Step 3 - Examine JMS and JNDI Assets \(see page 1809\)](#)
- [Step 4 - Test the JMS Send Receive Step \(see page 1809\)](#)

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:

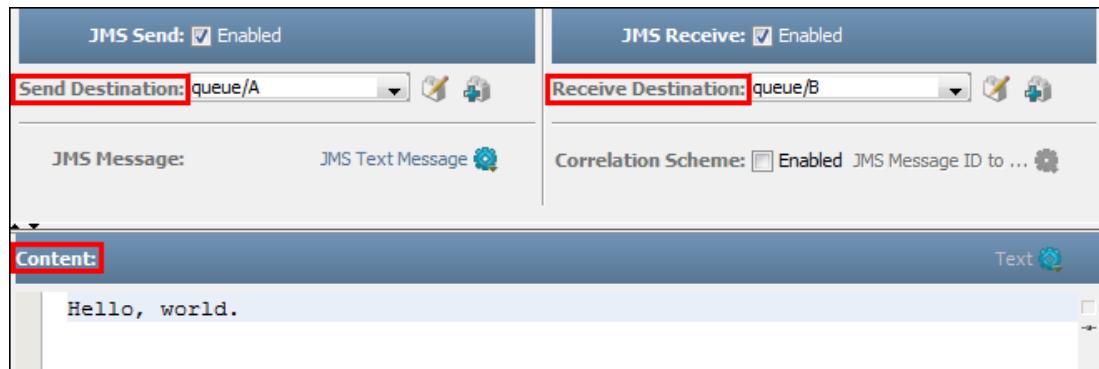
1. Go to DevTest Workstation and open the examples project.
2. In the **Tests** folder, open the **jms.tst** test case.
3. Select the first **JMS Messaging (JNDI)** step. This step has the name **jms-1**.
4. Click **Generate assets from the selected steps** in the model toolbar.
5. Select the second **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 **Edit Selected Asset** 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 **Edit Selected Asset** 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 step editor.
2. Confirm that the **Request** tab and the **Response** tab contain the same sentence.

3. Close the step editor.

BEA Steps

The following steps are available:

- [WebLogic JMS \(JNDI\) \(see page 1810\)](#)
- [Message Consumer \(see page 1815\)](#)
- [Read a File \(see page 1815\)](#)
- [Web Service Execution \(XML\) \(see page 1816\)](#)
- [Raw SOAP Request \(see page 1816\)](#)
- [FTP Step \(see page 1816\)](#)

[WebLogic JMS \(JNDI\)](#)

Contents

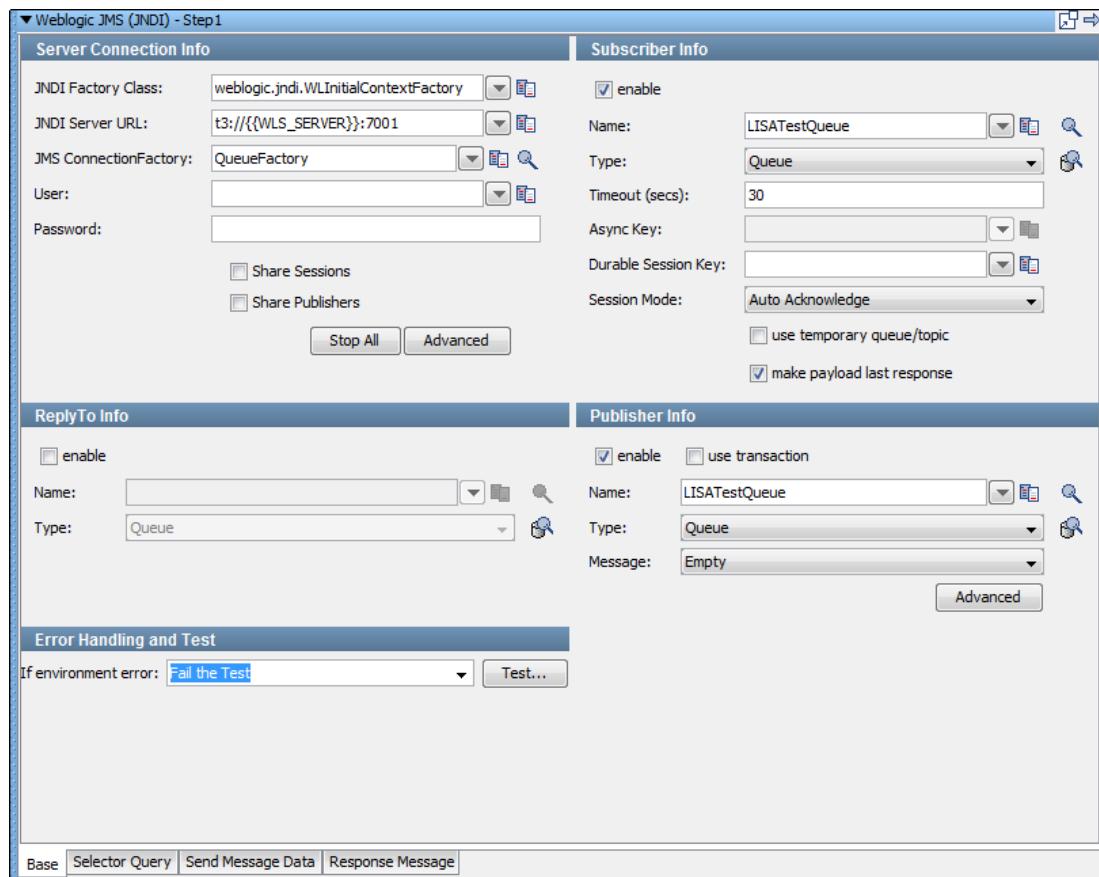
- [Base Tab \(see page 1811\)](#)
- [Selector Query Tab \(see page 1815\)](#)
- [Send Message Data Tab \(see page 1815\)](#)
- [Response Message Tab \(see page 1815\)](#)

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 \(see page 1380\)](#).

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.



WebLogic JMS (JNDI) step

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 properties simplify 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](https://support.ca.com/irj/portal/kbtech?docid=603964&searchID=TEC603964&fromKBResultsScreen=T) (<https://support.ca.com/irj/portal/kbtech?docid=603964&searchID=TEC603964&fromKBResultsScreen=T>) 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

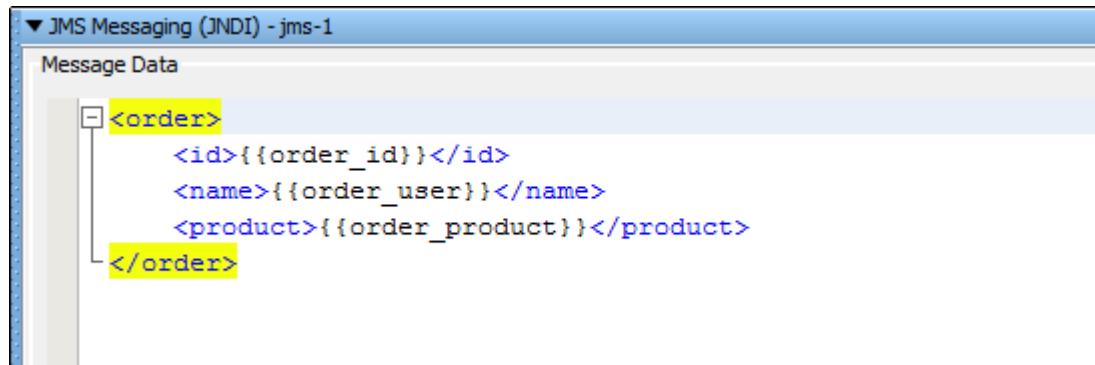
You can enter a JMS selector query in this editor. The syntax closely follows SQL and is a subset of SQL92. A JMS selector query can be specified when listening for a message on a queue that is a response to a published message. The previous graphic shows a specific query for a **JMSCorrelationID** that 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.



JMS Messaging (JNDI) step Send Message Data Tab

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 \(JNDI\) \(see page 1792\)](#).

Message Consumer

For detailed information about this step, see [JMS Messaging - Message Consumer \(see page 1800\)](#).

Read a File

For detailed information about this step, see [Read a File \(Disk, URL or Class Path\) \(see page 1790\)](#).

Web Service Execution (XML)

For detailed information about this step, see [Web Service Execution \(XML\) Step \(see page 1711\)](#).

Raw SOAP Request

For detailed information about this step, see [Web_Raw SOAP Request \(see page 1749\)](#).

FTP Step

For detailed information about this step, see [External - FTP Step \(see page 1791\)](#).

Sun JCAPS Steps

The following steps are available:

- [JCAPS Messaging \(Native\) \(see page 1816\)](#)
- [JCAPS Messaging \(JNDI\) \(see page 1818\)](#)



More Information:

- [Web Service Execution \(XML\) Step \(see page 1711\)](#)
- [SQL Database Execution \(JDBC\) \(see page 1768\)](#)
- [Read a File \(Disk URL or Classpath\) \(see page 1790\)](#)
- [External - FTP Step \(see page 1791\)](#)
- [Message Consumer \(see page 1815\)](#)
- [Raw SOAP Request \(see page 1816\)](#)

JCAPS Messaging (Native)

Contents

- [Base Tab \(see page 1818\)](#)

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.

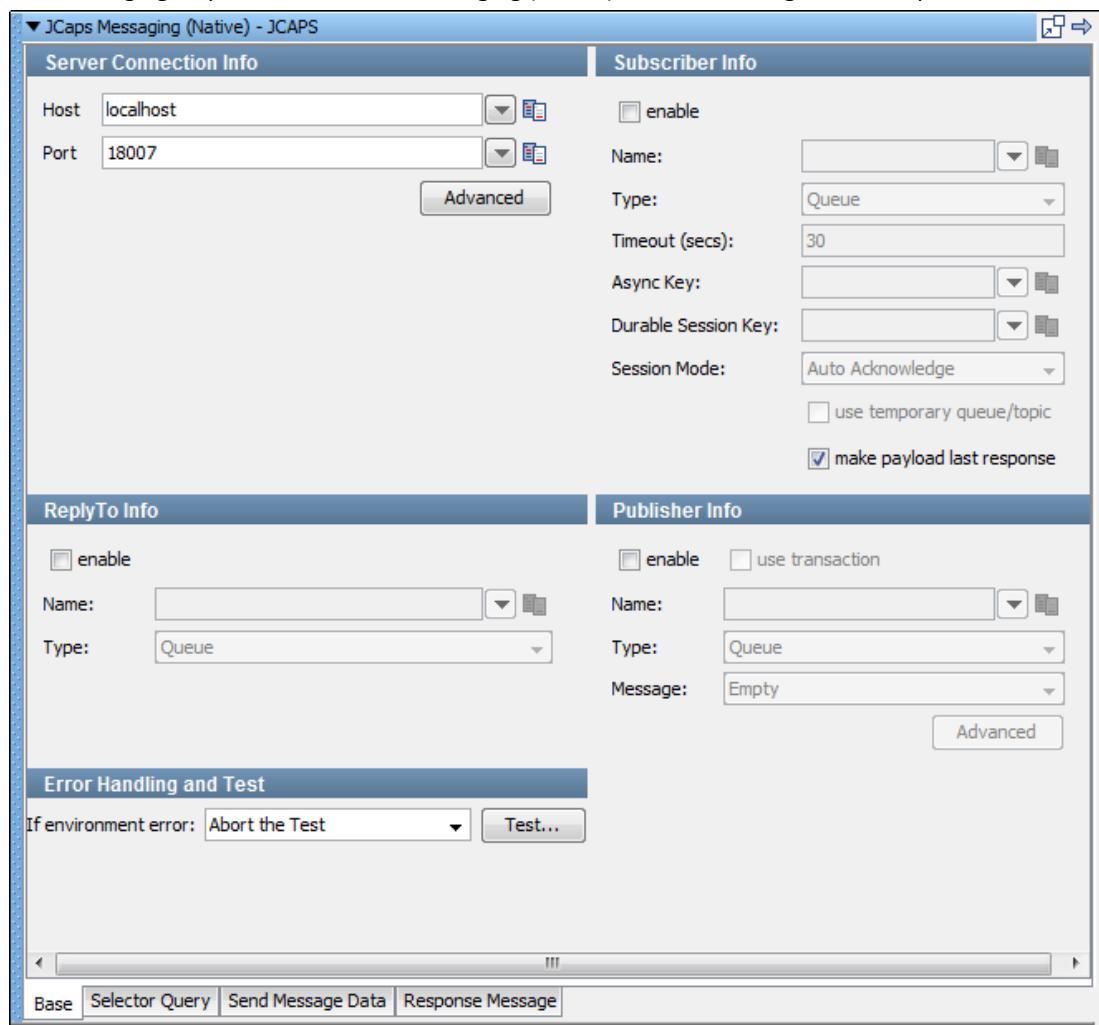
JCAPS Messaging (Native) supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

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 \(see page 1380\)](#).

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 \(JNDI\) \(see page 1792\)](#).

JCAPS Messaging (JNDI)

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 \(see page 1380\)](#).

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 \(JNDI\) \(see page 1792\)](#).

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:

- [Oracle OC4J \(JNDI\) \(see page 1819\)](#)
- [Oracle AQ Steps \(see page 1820\)](#)



More Information:

- [Web Service Execution \(XML\) Step \(see page 1711\)](#)
- [Web-Raw SOAP Request \(see page 1749\)](#)
- [SQL Database Execution \(JDBC\) \(see page 1768\)](#)
- [Read a File \(Disk URL or Classpath\) \(see page 1790\)](#)
- [External - FTP Step \(see page 1791\)](#)
- [Message Consumer \(see page 1815\)](#)

Oracle OC4J (JNDI)

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 \(see page 1380\)](#).

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 \(JNDI\) \(see page 1792\)](#).

Oracle AQ Steps

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 is added to a test case to allow sending messages, receiving messages, receiving messages asynchronously, or some combination of the three.

For AQ, there are two separate step types (JMS and JPUB). They have the same functionality but represent two different methods for communicating with the Oracle AQ messaging provider.

Both Oracle AQ steps have 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.

Two distinct ways to use AQ are:

- [JMS \(see page 1821\)](#)
- [JPUB \(see page 1825\)](#)

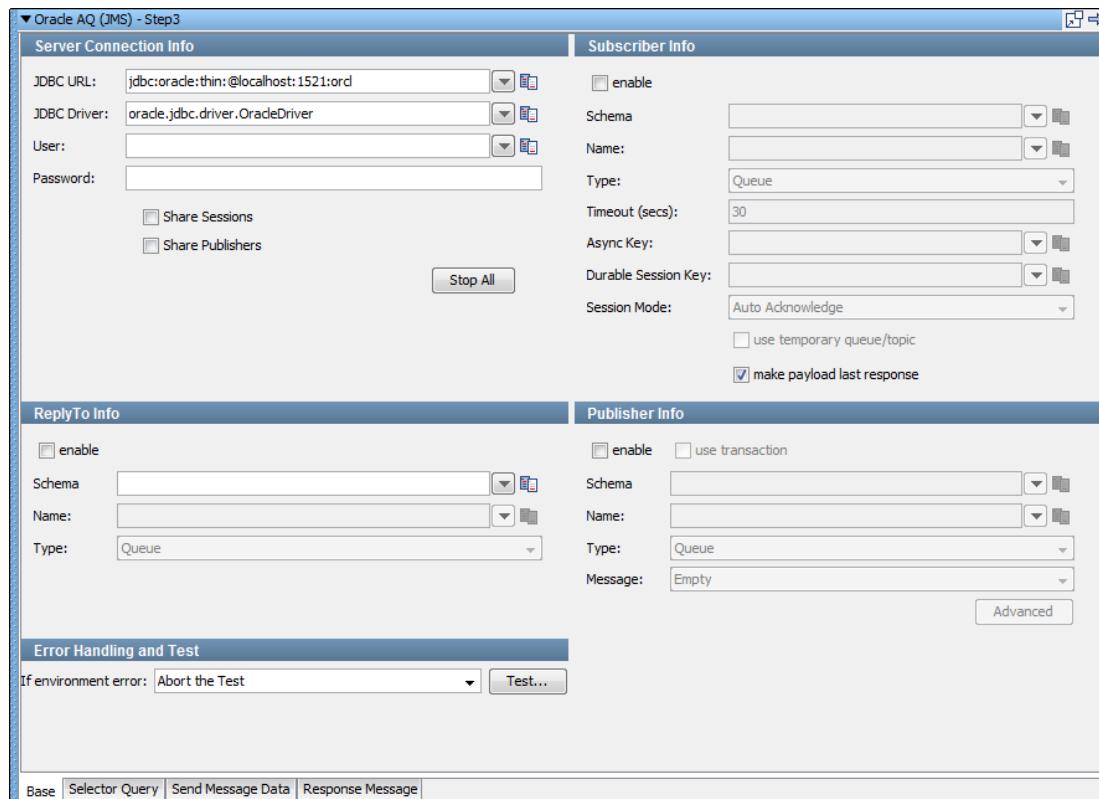
Oracle AQ (JMS)

Contents

- [Base Information Tab \(see page 1822\)](#)

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.

One of the two ways to use AQ is JMS.



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](https://support.ca.com/irj/portal/kbtech?docid=603964&searchID=TEC603964&fromKBResultsScreen=T) (<https://support.ca.com/irj/portal/kbtech?docid=603964&searchID=TEC603964&fromKBResultsScreen=T>) 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.

Oracle AQ (JPUB)

Contents

- [Base Information Tab \(see page 1826\)](#)

Two distinct ways to use AQ are:

- JMS
- JPUB

The Oracle AQ JMS API is a layer that is built on top of a lower-level AQ API. This lower-level API is much more difficult to deal with, and it acts almost nothing like JMS.

The principal distinction is the message format. The low-level AQ messages contain a payload, which can be any type that is defined in the database. The type could be a varchar or a clob, but usually it is a user-defined structured database type. Similar to AQ JMS Queues, each AQ low-level queue can only handle one payload type.

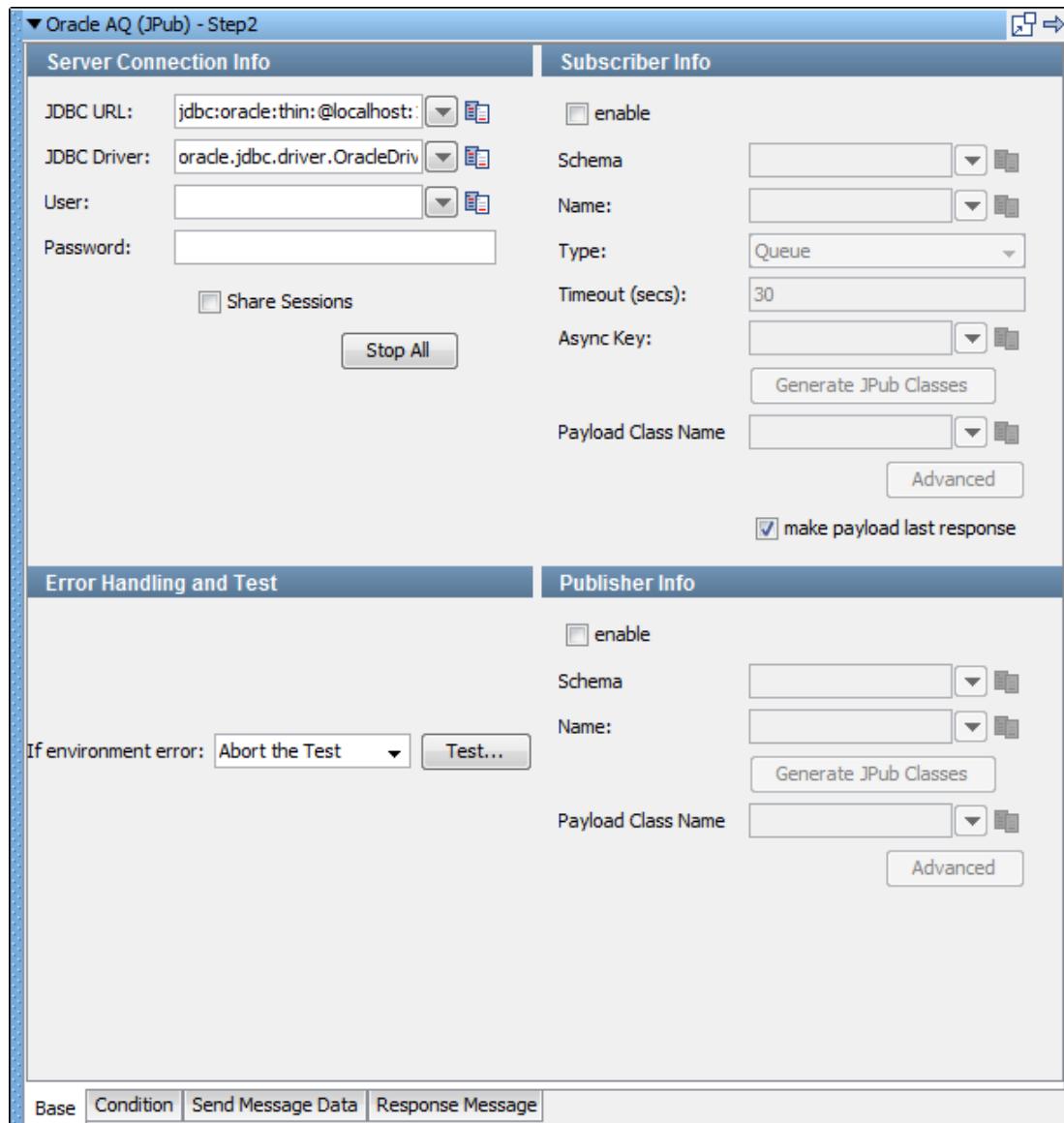
Oracle provides a utility with the name **JPUB** that can generate the Java objects that can deal with these user-defined structured types. JPUB works in the same way that Axis generates the Java objects that use web services. The low-level AQ step, **Oracle AQ JPUB**, can automatically use this utility to generate the client classes that are based on the queue information. The user then fills in their payload object using a standard COE.

A distinction between queues or topics does not exist. A client can:

- remove the next message from the AQ queue, making it essentially a queue, or
- read the next message from the AQ queue without removing it, making it essentially a topic.

Setting up the low-level AQ queues is again done through stored procedures. It may be necessary to create a specific user-defined structured type in the database before you create an AQ queue around it. Technically, you can interact with AQ JMS queues using the low-level API. The JMS queues simply have a specific payload type that is structured like a standard JMS message. However, you cannot use the AQ JMS API to interact with low-level AQ queues; that is, queues that do not use a JMS payload type.

To add an Oracle AQ (JPUB) step to a test case, click the step to open its editor.



Oracle AQ (JPUB) step

Four tabs are available at the bottom of the editor.

- The **Base** tab is where you define your connection and messaging parameters.
- The **Condition** tab lets you specify a condition to be run.
- 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 that the previous graphic shows is the default view and has the following sections:

- **Server Connection Info**

- **Subscriber Info**
- **Publisher Info**
- **Error Handling and Test**

The **Server Connection Info** and **Error Handling and Test** sections are always active. The **Subscriber Info** and **Publisher Info** sections can be enabled or disabled using the **enable** check box in the top left corner of each section. Using these check boxes, you can configure the step to publish a step, subscribe a step, or both.

When you have configured your test step, click **Test** in the **Error Handling and Test** section to test your configuration settings.

Server Connection Info

To simplify changing the application under test, parameterize the values with properties from your configuration. By default, the **oracle.jdbc.driver.OracleDriver** in the JDBC Driver location is used.

- **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** (<https://support.ca.com/irj/portal/kbtech?docid=603964&searchID=TEC603964&fromKBResultsScreen=T>) 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.

Enter the following parameters:

- **Schema**
Enter the name of the schema to use.

- **Name**

Enter the name of the topic or queue to use.

- **Generate JPub classes**

Click to generate the JPub classes.

- **Payload Class Name**

Enter the payload class name.

- **Advanced button**

To open the **Publisher Advanced** dialog, click **Advanced**, then enter or select the **Correlation** and click **OK**.

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.

- **Generate JPub classes**

Click to generate the JPub classes.

- **Payload Class Name**

Enter the payload class name.

- **Advanced**

Click to open the **Subscriber Advanced** dialog.

In the **Advanced** dialog, you can enter the **Consumer Name**, **Correlation**, and **Message ID**.

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:

- [TIBCO Rendezvous Messaging \(see page 1829\)](#)
- [TIBCO EMS Messaging \(see page 1834\)](#)
- [TIBCO Direct JMS \(see page 1835\)](#)



More Information:

- [Web Service Execution \(XML\) Step \(see page 1711\)](#)
- [Web-Raw SOAP Request \(see page 1749\)](#)
- [SQL Database Execution \(JDBC\) \(see page 1768\)](#)
- [Read a File \(Disk URL or Classpath\) \(see page 1790\)](#)
- [External - FTP Step \(see page 1791\)](#)
- [Message Consumer \(see page 1815\)](#)

TIBCO Rendezvous Messaging

Contents

- [Base Tab \(see page 1831\)](#)
- [Send Message Data \(see page 1833\)](#)
- [Response Message Tab \(see page 1834\)](#)

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 \(see page 1380\)](#).

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.

The screenshot shows the configuration interface for a TIBCO Rendezvous Messaging step. The interface is divided into several tabs:

- Server Connection Info:** Contains fields for Service (7474), Network, Daemon (192.168.10.221:7474), and Client Mode (Native Client). A "Stop All" button is located below these fields.
- Subscriber Info:** Contains checkboxes for "enable" and "queue.sample" (selected), and fields for Subject (queue.sample), Timeout (secs) (30), and Async Key (ASYNC).
- Certified Transport Info:** Contains checkboxes for "enable" and "Sender Name" (empty), Advisory Subject (_RV.INFO.RVCM.DELIVERY.CONFIRM,>), and Time Limit (empty).
- Publisher Info:** Contains checkboxes for "enable" and "Enable Inbox Type", and fields for Subject (empty), Message (Empty), and Send Field (empty). It also includes "Inbox Timeout" and "Enable sendReply" checkboxes.
- ReplyTo Info:** Contains checkboxes for "enable" and "Subject" (empty).
- Error Handling and Test:** Contains a dropdown for "If environment error" (set to "Abort the Test") and a "Test..." button.
- Bottom Tabs:** The "Base" tab is selected, while "Send Message Data" and "Response Message" are also visible.

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**

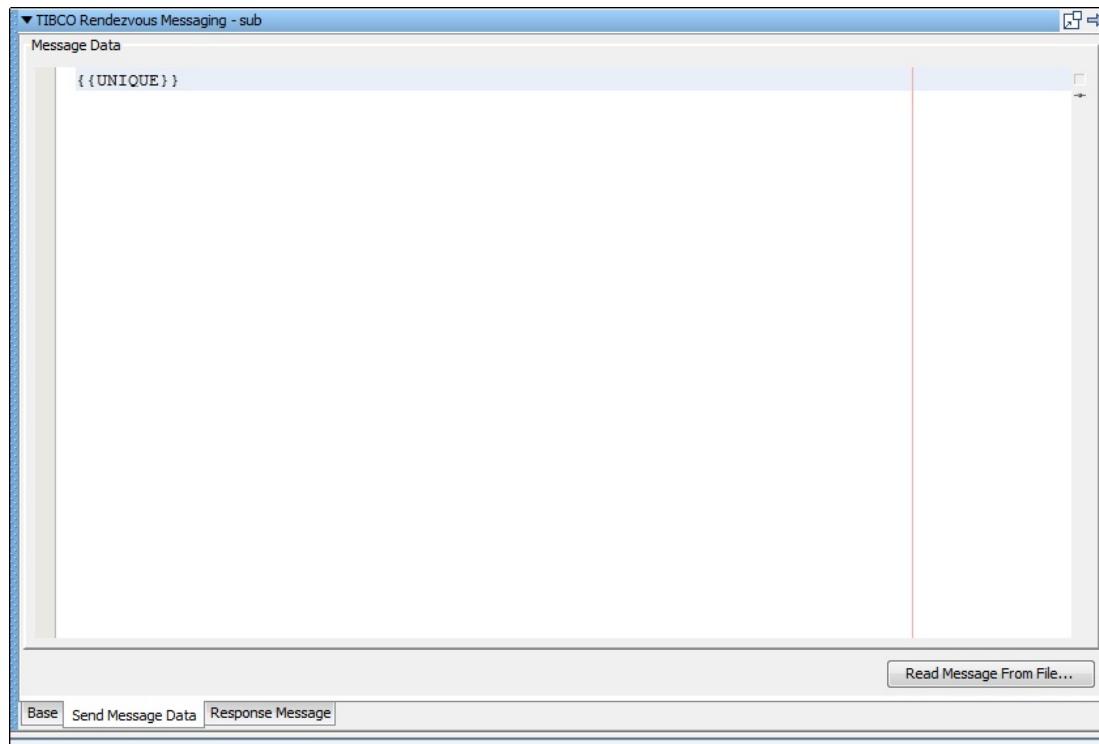
Specifies the action to take or the step to go to if the test fails because of an environment error.

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 \(JNDI\) \(see page 1792\)](#).

TIBCO EMS Messaging

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 \(see page 1380\)](#).

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 \(JNDI\) \(see page 1792\)](#)

TIBCO Direct JMS

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 \(see page 1380\)](#).

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 \(JNDI\) \(see page 1792\)](#)

Sonic Steps

The following steps are available:

- [SonicMQ Messaging \(Native\) \(see page 1836\)](#)
- [SonicMQ Messaging \(JNDI\) \(see page 1836\)](#)



More Information:

- [Web Service Execution \(XML\) Step \(see page 1711\)](#)
- [Web-Raw SOAP Request \(see page 1749\)](#)
- [SQL Database Execution \(JDBC\) \(see page 1768\)](#)

- [Read a File \(Disk URL or Classpath\) \(see page 1790\)](#)
- [External - FTP Step \(see page 1791\)](#)
- [Message Consumer \(see page 1815\)](#)

SonicMQ Messaging (Native)

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 \(see page 1380\)](#).

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 \(JNDI\) \(see page 1792\)](#)

SonicMQ Messaging (JNDI)

SonicMQ Messaging (JNDI) supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

The SonicMQ Messaging (JNDI) step lets you send and receive messages from topics and queues. You can also receive, modify, and forward an existing message.

The SonicMQ Messaging (JNDI) step is configured using a single 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 \(see page 1380\)](#).

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 \(JNDI\) \(see page 1792\)](#)

.

webMethods Steps

The following steps are available:

- [webMethods Broker \(see page 1837\)](#)
- [webMethods Integration Server Services \(see page 1841\)](#)

webMethods Broker

Contents

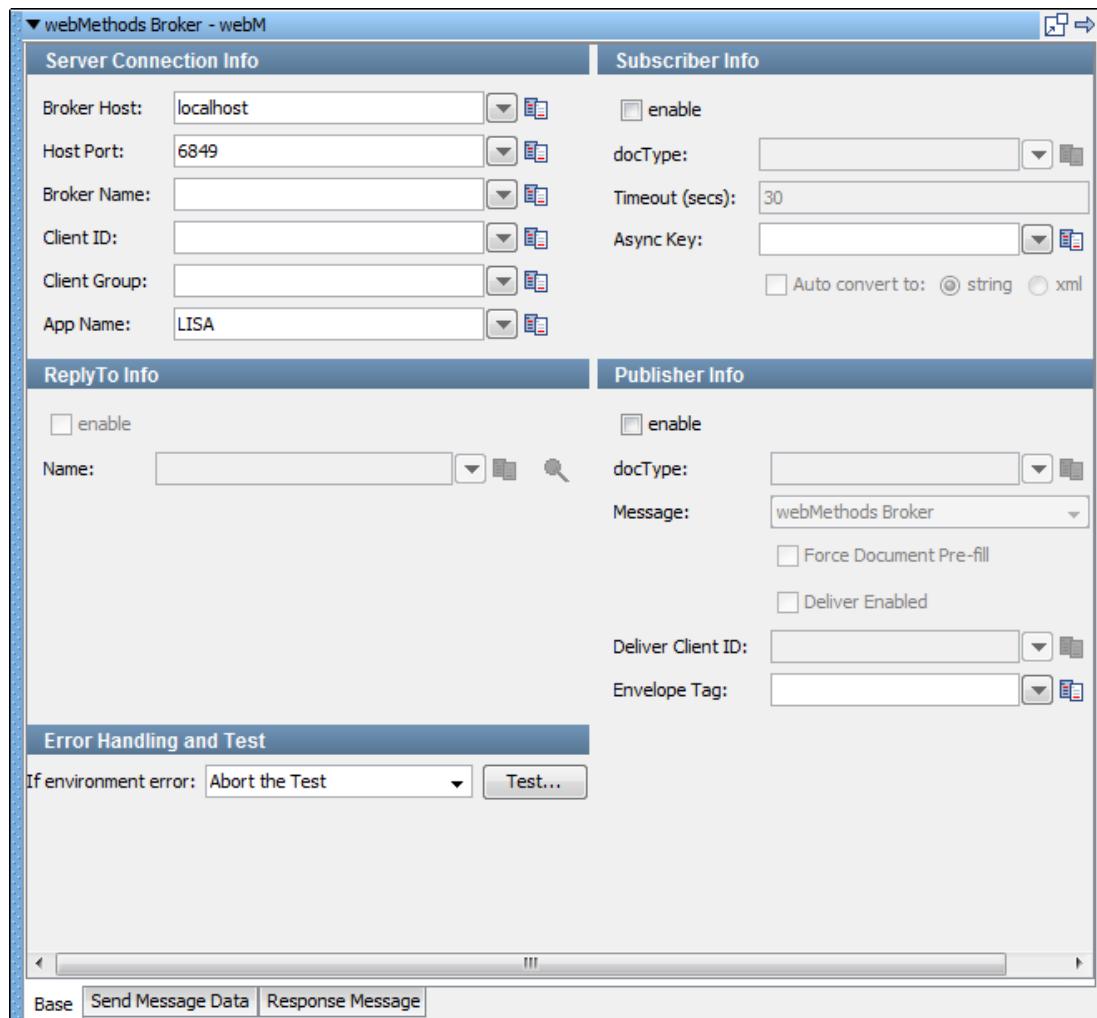
- [Base Tab \(see page 1838\)](#)
- [Send Message Data Tab \(see page 1841\)](#)
- [Response Message Tab \(see page 1841\)](#)
- [Default Step Names \(see page 1841\)](#)

webMethods Broker supports Mapped (Extended) messages that create Broker Events.

The webMethods Broker step lets you send and receive messages from the Broker. You can also receive, change, and forward existing Broker Events/ Messages.

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 \(see page 1380\)](#).

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 \(JNDI\) \(see page 1792\)](#).

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

Contents

- [Base Tab Server Connection Info \(see page 1841\)](#)
- [Pipeline Input Tab \(see page 1844\)](#)
- [Pipeline Output Tab \(see page 1845\)](#)
- [Default Step Names \(see page 1845\)](#)

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 \(see page 1380\)](#).

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.

▪ 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

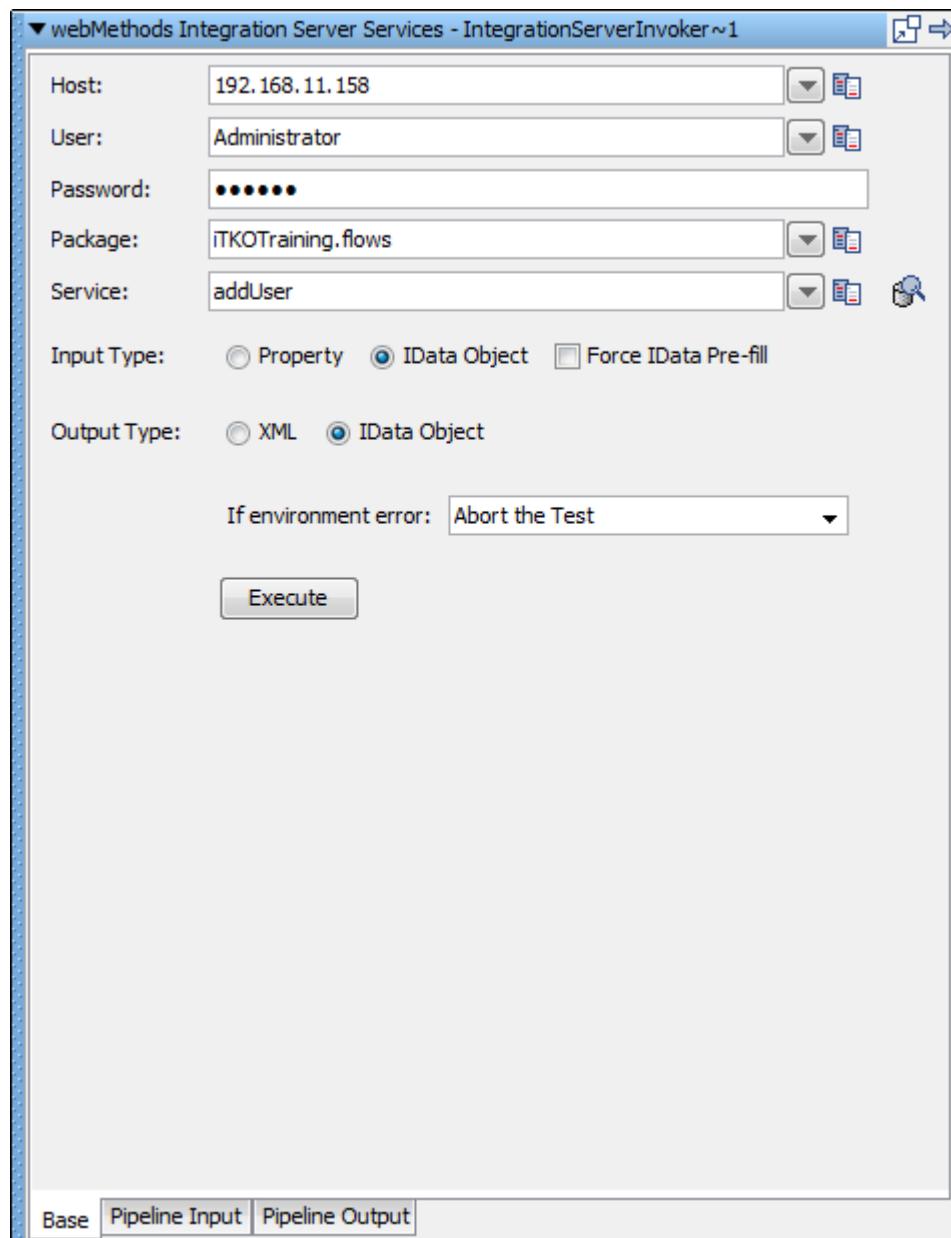
Select the output type from **XML** or **IData Object**.

▪ 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 Services - IntegrationServerInvoker~1

Input Pipeline View

Name	Value
EMAIL	testuser3@itko.com
FNAME	Test
LNAME	User
LOGIN	testuser3
PHONE	214-444-4444
PWD	pass
ROLE_NAME	QualityEngineer

Base Pipeline Input Pipeline Output

Load From File

webMethods Integration Server step Pipeline Input tab

Pipeline Output Tab

The screenshot shows the Pipeline Output tab interface. At the top, there are three tabs: Response, Properties, and Test Events. The Response tab is selected. Below the tabs is a title bar "Output Pipeline View". The main area is a table with two columns: "Name" and "Value". The table contains the following data:

Name	Value
abc fileName	E:\temp\TKOTraining_flows_ad...
abc EMAIL	testuser3@itko.com
abc FNAME	Test
abc LNAME	User
abc LOGIN	testuser3
abc PHONE	214-444-4444
abc PWD	newpass34
abc lisaFrameID	3ac03700-ba49-11e1-a856-02...
abc lisaFrameRoot	true
abc lisaFrameRemoteIP	192.168.168.152
abc encodePass	R.17klxvEc+S4SEEszvzTISI+1tE=
abc ROLE_NAME	QualityEngineer

At the bottom of the window, there are several icons: a magnifying glass, a plus sign, a minus sign, a refresh symbol, and a help symbol. Below these icons is a navigation bar with three tabs: Pipeline View (selected), XML View, and Pathfinder.

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:

- [IBM WebSphere MQ Step \(see page 1846\)](#)
- [IBM MQ Native Send Receive Step \(see page 1851\)](#)



More Information:

- [Message Consumer \(see page 1815\)](#)

IBM WebSphere MQ Step

Contents

- [IBM WebSphere MQ Base Tab \(see page 1847\)](#)

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 \(see page 1380\)](#).

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 \(JNDI\) \(see page 1792\)](#).

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**

The screenshot shows the 'IBM Websphere MQ - MQ' configuration dialog with the 'Base' tab selected. The dialog is divided into four main sections:

- Server Connection Info:** Contains fields for Host Name, TCP/IP Port, Channel, Queue Manager, CCID, User, Password, and Client Mode (set to JMS). It also includes 'Advanced' and 'Stop All' buttons, and a 'Share Sessions' checkbox.
- Subscriber Info:** Contains fields for enable, Name, Type (set to Queue), Timeout (secs) (set to 30), Queue Model, Async Key, Durable Session Key, Session Mode (set to Auto Acknowledge), and three checkboxes: 'use temporary queue/topic', 'make payload last response', and 'use correlation ID for subscri'. A 'Subscribe Properties' button is also present.
- ReplyTo Info:** Contains fields for enable, Name, Type (set to Queue), and Queue Manager.
- Publisher Info:** Contains fields for enable, Name, Type (set to Queue), Message (set to Empty), and Alt QManager. A 'Message Properties' button is also present.

At the bottom of the dialog, there are tabs for 'Base', 'Selector Query', 'Send Message Data', and 'Response Message'.

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 (<https://support.ca.com/irj/portal/kbtech?docid=603872&searchID=TEC603872&fromKBResultsScreen=T>).

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:

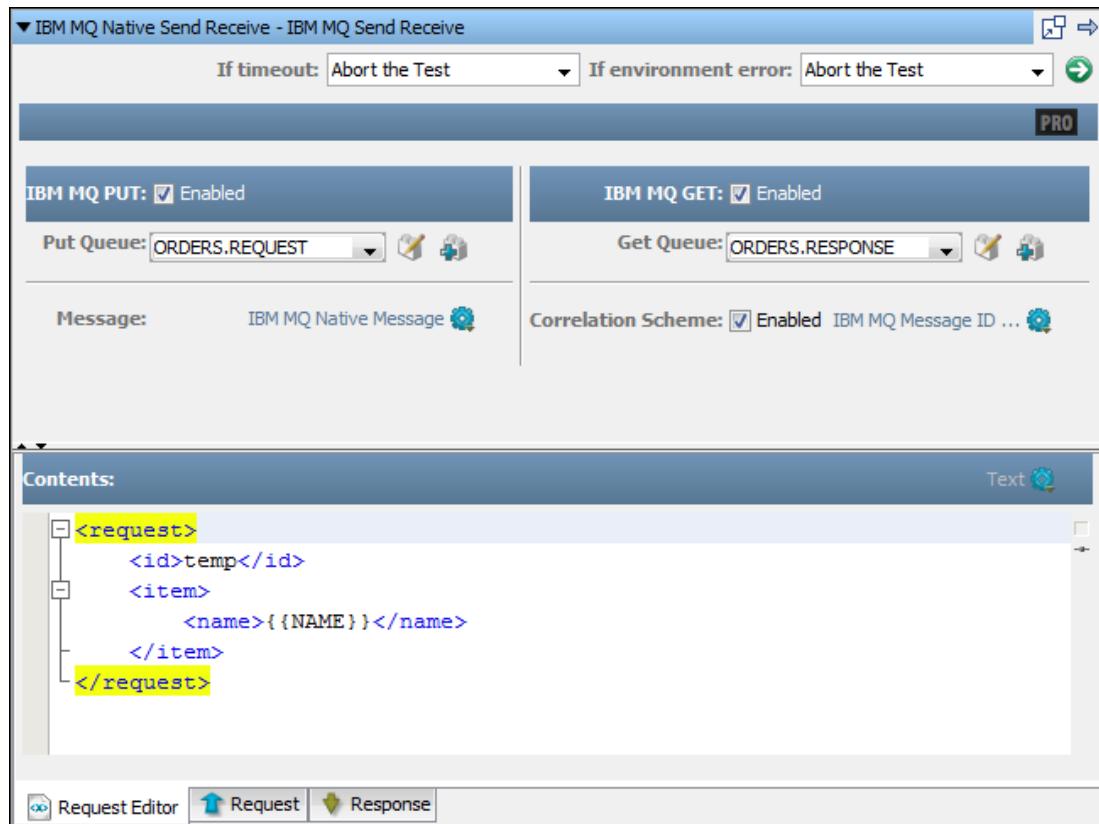
- Send a request message
- Receive a response message



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.

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

The **IBM MQ Native Send Receive** step has separate areas for configuring the put operation and the get operation.

Specify two [queues \(see page 1513\)](#) 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 \(see page 1555\)](#).

Each asset has a [runtime scope \(see page 402\)](#). 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:

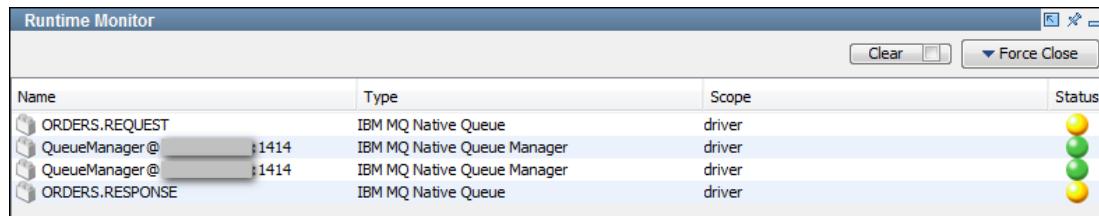
1. Click the green **Execute** button in the step editor.

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.



A screenshot of the 'Runtime Monitor' window. The window has a title bar 'Runtime Monitor' and a toolbar with 'Clear' and 'Force Close' buttons. The main area is a table with columns: Name, Type, Scope, and Status. There are four rows of data:

Name	Type	Scope	Status
ORDERS.REQUEST	IBM MQ Native Queue	driver	●
QueueManager @ 1414	IBM MQ Native Queue Manager	driver	●
QueueManager @ 1414	IBM MQ Native Queue Manager	driver	●
ORDERS.RESPONSE	IBM MQ Native Queue	driver	●

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. Select 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:

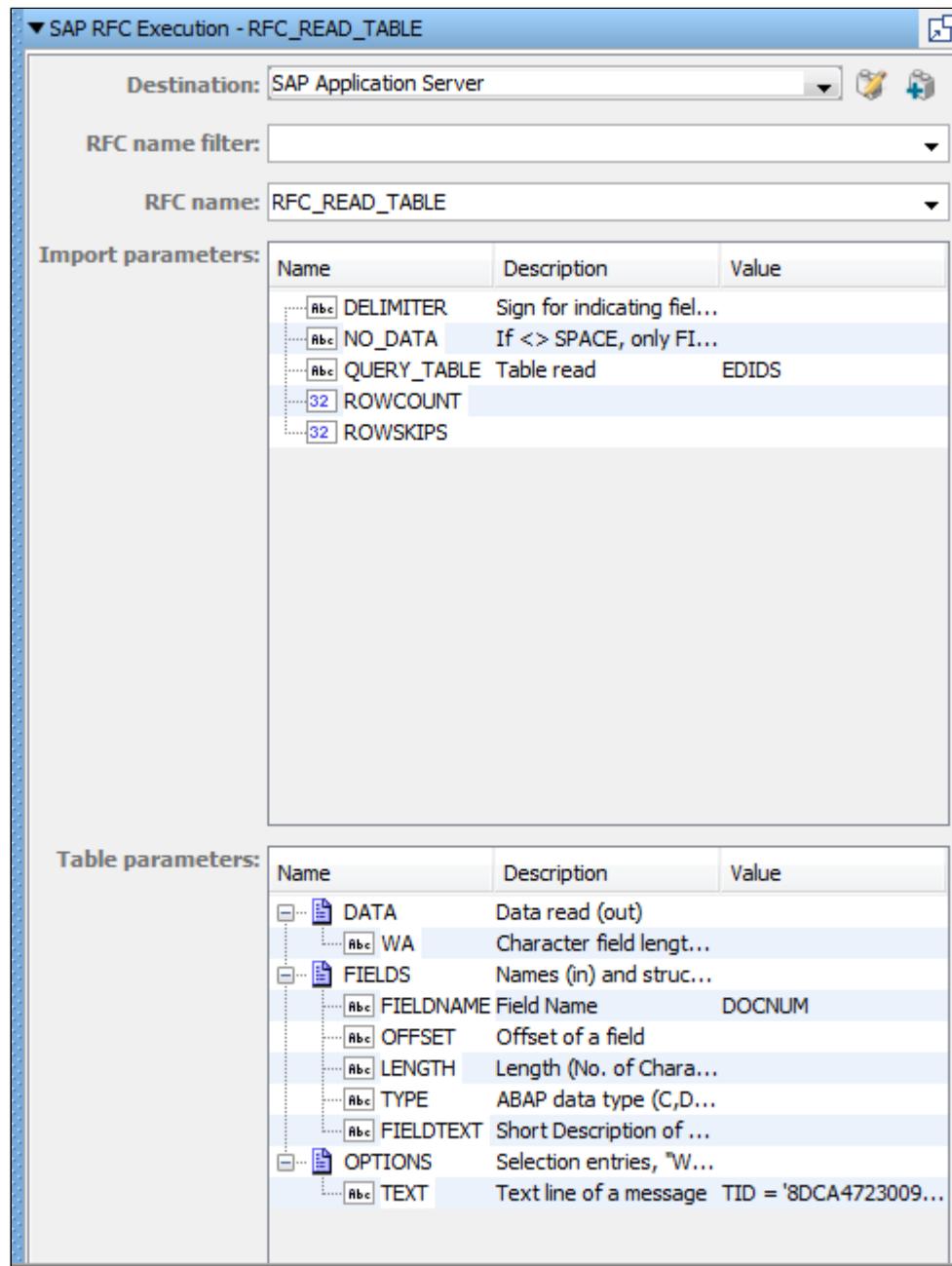
- [SAP RFC Execution \(see page 1855\)](#)
- [SAP IDoc Sender \(see page 1857\)](#)
- [SAP IDoc Status Retriever \(see page 1858\)](#)

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 \(see page 1380\)](#).

To create an SAP RFC Execution step:

1. Select a **Destination asset (see page 1526)** 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 \(see page 1380\)](#).

To create an SAP IDoc Sender step:

1. Select a **Destination asset** ([see page 1526](#)) 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 it 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 \(see page 1380\)](#).

To create an SAP IDoc Status Retriever step:

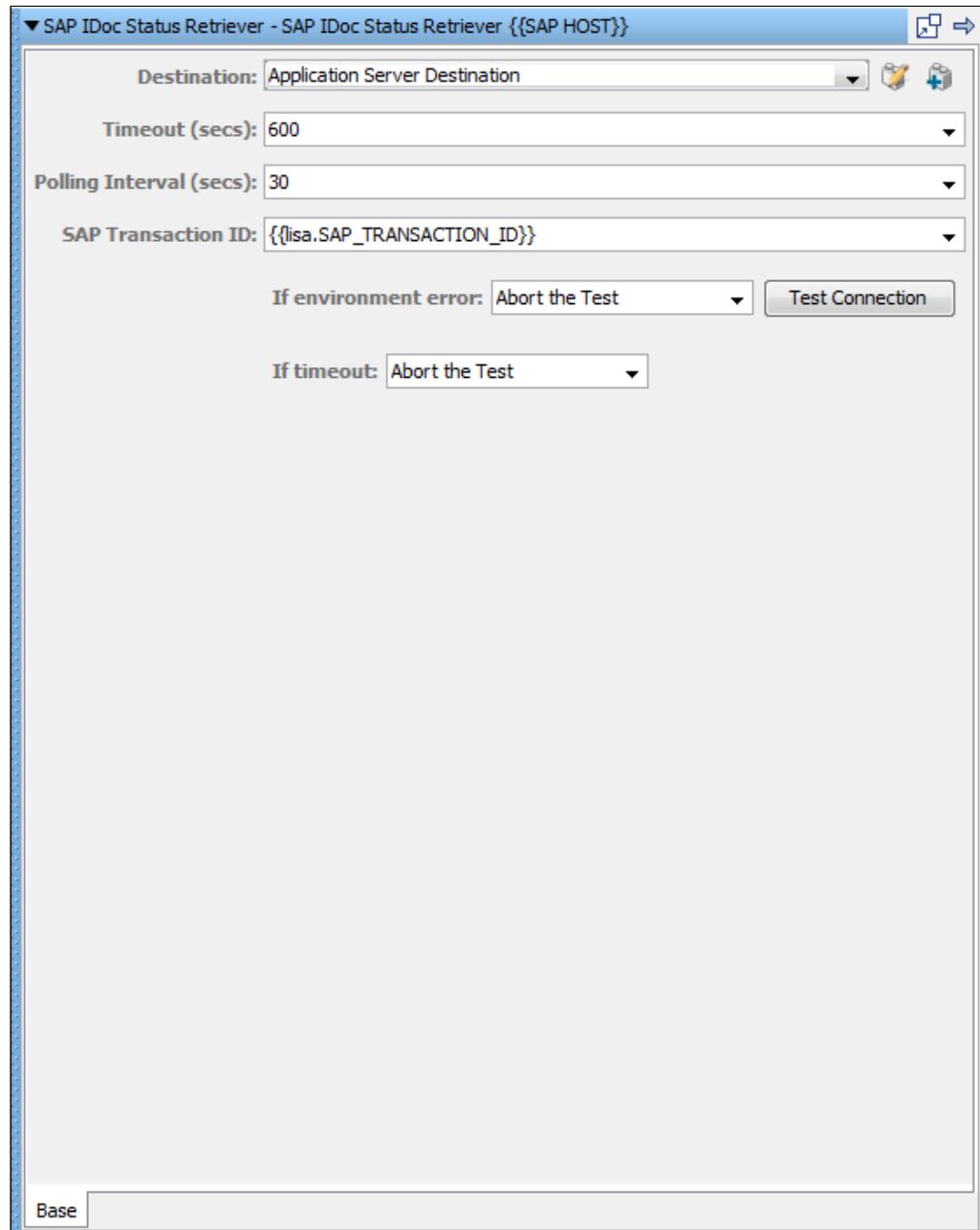
1. Select a **Destination asset** ([see page 1526](#)) that includes the connection information for either an application server or a message server.

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 create a Selenium step, complete the following tasks:

- [Create and Export a Selenium Builder Recording \(see page 1861\)](#)
- [Import the Selenium Builder JSON into CA Application Test \(see page 1862\)](#)



Note: Browser support for recording Selenium Integration tests is limited to Firefox. After you import these steps to CA Application Test, you can run the test cases on Google Chrome, Mozilla Firefox 24 or later, or Internet Explorer 8.0 or later.

You can also export a CA Application Test Selenium test case to a JSON script. For more information, see:

- [Export a Selenium Test Case to a JSON Script \(see page 1864\)](#)

Create and Export a Selenium Builder Recording

Selenium Builder is a Firefox add-on that lets you record actions in a web-based user interface and create Selenium tests. The first step in creating a Selenium test step in CA Application Test is to create a Selenium Builder test script. The following video is 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. Click **Firefox, Add-ons** on the main Firefox menu.
The Add-ons Manager opens.
2. Click **Extensions** and double-click **Selenium Builder**.
3. Select the Off option for Automatic Updates.

Follow these steps:

1. Open your Firefox browser.
2. Click **Firefox, Web Developer, Launch Selenium Builder** on the main Firefox menu.
The Selenium Builder page opens.



Note: If you have the Add-on Bar enabled, you can click **Launch Selenium Builder**  in the bottom right corner of the Firefox browser window.

3. Enter the URL of the web application you want to test in the **Start recording at** field.
4. Click **Selenium 2**.



Note: Selenium 1 does not support the required ability to export to JSON.

5. Navigate to the web application and perform the actions you want to test.
6. When you are done, return to the Selenium Builder page and click **Stop Recording**.



Note: For more information about creating Selenium Builder tests, see the Selenium Builder documentation at <http://seleniumbuilder.github.io/se-builder/>.

7. Click **File, Export**.
The **Choose Export Format** dialog opens.
8. Click **Save as JSON**.
9. Browse to the directory where you want to save the JSON file.
10. Enter a name for the JSON file and click **Save**.
11. To verify that the script is valid, run the script in Selenium Builder.
12. Click **Run, Run test locally**.
The results and any script errors appear.

Import a Selenium Builder JSON into CA Application Test

Use the Selenium Integration test steps to import a [JSON test script that you created \(see page 1861\)](#) in Selenium Builder to create a test step or script that is based on that JSON file.

Follow these steps:

1. Open an existing or create a new test case.
2. Perform one of the following actions:
 - Click **Selenium** .
 - Click **Add Step**  , **Selenium**, **Selenium Import/Export**.
- The **Import** tab of the Selenium Integration page opens.
3. Click **Browse** next to the **Input JSON Script** field and browse to the location of the JSON test script you want to import.
4. Select one of the following output options:
 - **Selenium Script**
Imports the complete JSON test script in one test step. This option also supports Selenium Builder suites.
 - **Selenium Step**
Divides the JSON script to create a separate test step for each action in the script. This option does not support Selenium Builder suites.

5. Click **Build**.

A new step (Selenium Step) or new steps (Selenium Script) appear in your test case. The Selenium Integration Import dialog opens. This page notifies you of any warnings or errors that occurred during the import. The page also shows you the number of steps that the import process generated.

6. Click **Close**.

7. Double-click each step to view the JSON script in the Selenium Script or Selenium Step tab of the Elements tree 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 app.

- **Alert Action (Optional)**

Defines the action to take in response to a modal alert dialog.

Values: Accept, Dismiss, Answer

- **Input Text (Optional)**

Defines text to input when the **Alert Action** 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 \(see page 479\)](#) and [Generate Warnings and Errors \(see page 480\)](#).



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\temp\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.

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 DevTest 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 only 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

To install the DevTest plugin:

1. Copy the entire **lisa** directory, and its contents, from [LISA_HOME]\addons\sebuilder-plugin on the DevTest Workstation to the [Firefox profile]\SeBuilder\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.

2. Copy the **lisa** directory to the SeBuilder\plugins directory.

To export a Selenium test case:

1. Open an existing test case.
2. Perform one of the following actions:
 - Click **Selenium** .
 - Click **Add Step**  , **Selenium**, **Selenium Import/Export**.
- The **Selenium Integration** page opens.
3. Click the **Export** tab.
4. Click **Browse** next to the **Output JSON Script** field and browse to the location of the JSON test script you want to export, or enter a path and file name.
5. Click **Save**.

Virtual Service Environment Steps

The CA Service Virtualization test steps are described in [Editing a VSM \(see page 934\)](#).

CAI Steps

The following step is available:

- [Execute Transaction Frame \(see page 1866\)](#)

Execute Transaction Frame

The Execute Transaction Frame step lets you run a [transaction frame \(see page 1011\)](#) 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 \(see page 1110\)](#).

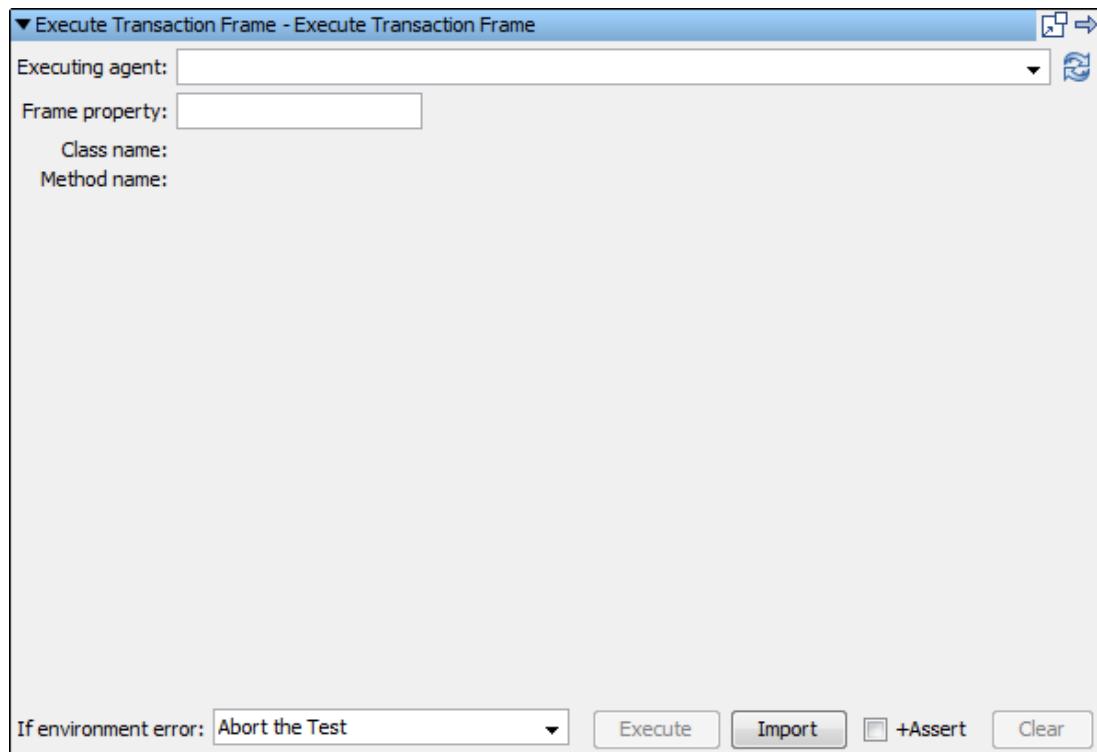
This step can be used on any type of transaction frame that the agent can capture.

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 \(see page 1056\)](#) from the 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.

Node	Type	Occurs	Nil	Nullable	Value
EJB					
java.naming.factory.initial					{{{JNDIFACTORY}}}
java.naming.provider.url					{{{JNDIURL}}}
jndi.name					{{{JNDINAME}}}
isEjb3					true
serializedFully					false

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 the 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 \(see page 1482\)](#) 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.

Mobile Steps

Mobile test steps are automatically created when you [record a mobile test case \(see page 636\)](#). The test steps represent the mobile actions or gestures that are captured during the recording. Each test step represents the actions that you performed on a specific screen in the application.

When the recorded test case is complete, you can modify the test case in a variety of ways by manually reordering the actions, inserting additional actions, or adding [assertions \(see page 430\)](#).

The following step is available:

- [Modify Mobile Test Steps \(see page 1868\)](#)

Modify Mobile Test Steps

Follow these steps:

1. Open the mobile test case that you want to update.
2. To view the details of each step:
 - a. Click the test step that you want to review.
 - b. Click **Mobile testing step** in the element tree on the right to expand the details for the step.
The Mobile Testing Step tab opens and shows a screenshot of the test application. The Actions section at the top of the tab shows the individual actions that are performed in the test step.
 - c. To view the screenshot associated with a specific action, click the action in the Actions section.
3. To manually add a new action, click **Add (+)** in the **Actions** section and enter the Key/Value pair for that action.
4. To rearrange the order of the actions, click the action you want to move and use the up and down arrows to move the action.
5. To delete an action, click the action and click **Delete**.
6. You can also add the following element or screen actions by right-clicking the screenshot for a recorded action:
 - **Change all assets to...**
Select an asset in the drop-down menu and all assets in the test are changed to that one.
 - **Back**
Inserts a Back command to return to the previous screen in the application. Android only.
 - **Get element**
Performs a Get on the selected element. The name of the selected element appears next to the Get element option.
 - **Tap element**
Performs a Tap action on the selected element. The name of the selected element appears next to the Tap element option.
 - **Send keys**
Brings up the keyboard and simulates pressing the specified keys to set the value of the text element.

Note: In some instances, the application does not display the typed values. For example, a password field might not display the entered values.

 - **Set value**
Sets the value for the selected element. The name of the selected element appears next to the Set value option.



Note: You can also use this action to set the value for a slider. For example, setting a value of 0.8 moves the slider 80% to the right. Use increments of 0.1 when setting the value for a slider. Setting a value of 0.25 is unrecognized and does not move the slider.

- **Long Tap screen**

Performs a long tap on the screen, as opposed to a specific element.

- **Long Tap element**

Performs a long tap on the selected element. The name of the selected element appears next to the Long Tap element option.

- **Wait**

Inserts a pause in the test. You can express the wait like ThinkTime at the step level. For example, **1s-10s** inserts a random pause between 1 and 10 seconds. **100z** inserts a pause of 100 milliseconds.

- **Comment**

Lets you insert a comment for a specific action. The comment performs no action, it is for documentation only.

- **Script**

Lets you insert an arbitrary beanshell script. Typically, this script does not interact with the device or simulator. This action provides the means of inserting a custom assertion.

- **Change Orientation**

Changes the orientation of the device.

- **Shake**

Performs a shake gesture on the device.

- **Go to background**

Performs the action of pressing the home button. This action forces the application to the background for 10 seconds and then brings it back. This action is useful for testing because it typically forces the application to release memory and stop any CPU-intensive work. iOS only.

- **Assertion**

For more information, see [Add an Assertion to a Mobile Test Step \(see page 430\)](#).

- **Execute Script**

Lets you write and run a script to perform a customized function or procedure. Right-click a step in the test pane and select Add step after..., Custom Extensions, Execute script (JSR-223). See [Execute Script \(JSR-223\) Step \(see page 1874\)](#) for more information.



Note: When scripting with Appium, use the _webDriver variable. This variable is a reference to the [RemoteWebDriver](https://selenium.googlecode.com/svn/trunk/docs/api/java/org/openqa/selenium/remote/RemoteWebDriver.html) (<https://selenium.googlecode.com/svn/trunk/docs/api/java/org/openqa/selenium/remote/RemoteWebDriver.html>) (public interface) object that is associated with the test when it is active in a running session (that is being recorded) or when it is played back. Because of this, the Test button at the bottom of the dialog results in an error when clicked.

7. Click **Save**.

Filters and Assertions in Mobile Testing

Adding a combination of a filter and an assertion to a mobile test is a useful way to include specific conditions in your test step that can specify a desired behavior.

The following example describes how to add a filter and assertion to a mobile step that creates a loop that is based on the state of a license agreement. The assertion ensures that the license agreement has been fully scrolled, and moves the test to a point where the user can tap the I Agree button.

Follow these steps:

1. Open an existing test case or create a new one.

In this example, the test case includes a step named Swipe Screen.



2. Add a filter to the Swipe Screen test step.

- a. In the model editor, select **Swipe Screen**.

- b. Open the **Filters** tab.

- c. Click **Add**.

- d. From the XML submenu, select **XML XPath Filter**.

The filter editor opens.

- e. In the **Filter in** field, enter:

```
pageSource.xml
```

- f. In the **Save To Property** field, enter:

```
is.agree.enabled
```

- g. In the **XPath Query** field, enter:

```
//android.widget.Button[@text='I Agree']/@enabled
```



- h. Click **Save**.

3. Add an assertion to the Swipe Screen test step.

- a. In the model editor, select **Swipe Screen**.

- b. Open the **Assertions** tab.

- c. Click **Add**.

- d. From the **Other** submenu, select **Ensure Property Matches Expression**.

The new assertion that is applied to Swipe Screen is added to the Assertions tab.

The assertion editor opens.

- e. In the **If** list, select **False**.

- f. In the **then** list, select **Go to Swipe Screen**.

- g. In the **Log** field, use the default setting.

- h. In the **Property Key** field, use **is.agree.enabled** from the previous step.

- i. In the **RegExpression** field, enter **true**.



- j. Click **Save**.



Custom Extension Steps

The following steps are available:

- [Custom Test Step Execution \(see page 1873\)](#)
- [Java Script Step \(deprecated\) \(see page 1873\)](#)
- [Execute Script \(JSR-223\) Step \(see page 1874\)](#)

Custom Test Step Execution

The Custom Test 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 \(see page 1212\)](#).

Java Script Step (deprecated)



Note: The Java Script step has been deprecated. Use the [Execute Script \(JSR-223\) \(see page 1874\)](#) 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



Note: The Java Script step has been deprecated. Use the Execute Script (JSR-223) step instead.

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 \(see page 1385\)](#)."

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 \(see page 353\)](#) project.

Set the default scripting language by using the [lisa.scripting.default.language \(see page 1671\)](#) property

RabbitMQ Steps

The following step is available:

- [RabbitMQ Send Receive Step \(see page 1875\)](#)

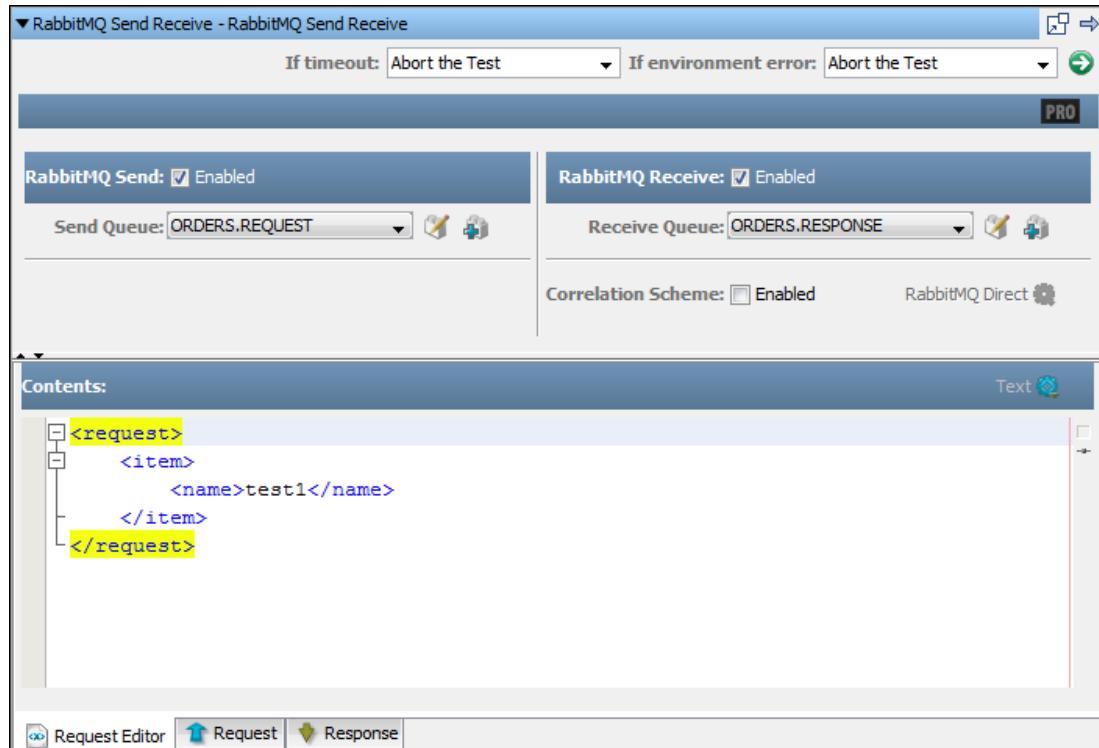
RabbitMQ Send Receive Step

This step lets you send and receive messages over RabbitMQ.



Note: This feature is in preview status. To learn how to access this feature and provide feedback, you must be a member of the DevTest Customer Validation Group. For information about becoming a member, see <https://communities.ca.com/community/canadian-test-community/blog/2015/02/18/thank-you-to-the-devtest-customer-validation-team>.

The following graphic shows the RabbitMQ Send Receive step.



Screen capture of RabbitMQ Send Receive step.

For more conceptual information about RabbitMQ and the corresponding assets, see [RabbitMQ Assets \(see page 1522\)](#).

Messages 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 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 it might contain something completely different.

Message Contents

Like WebSphere MQ, the contents of a RabbitMQ message are presented as an opaque byte array. It is up to the application to interpret the data. The RabbitMQ message properties provide some vague semantics that can help that process.

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, but 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 seen 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 both 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. They do almost exactly the same thing, with the exception of the default exchange name.

RabbitMQ Headers

In this correlation scheme, the request message contains exactly one property used for correlation. It is assumed that the service will copy that 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 works exactly like it does with the other providers.

Connect

User Community

Support

- [CA Application Test](http://www.ca.com/us/support/ca-support-online/support-by-product/ca-application-test.aspx) (<http://www.ca.com/us/support/ca-support-online/support-by-product/ca-application-test.aspx>)
- [CA Service Virtualization](http://www.ca.com/us/support/ca-support-online/support-by-product/ca-service-virtualization.aspx) (<http://www.ca.com/us/support/ca-support-online/support-by-product/ca-service-virtualization.aspx>)
- [CA Continuous Application Insight](http://www.ca.com/us/support/ca-support-online/support-by-product/ca-continuous-application-insight.aspx) (<http://www.ca.com/us/support/ca-support-online/support-by-product/ca-continuous-application-insight.aspx>)

Webcasts, Success Stories, White Papers, Demos, and More

- [CA Application Test](http://www.ca.com/us/devcenter/ca-application-test.aspx) (<http://www.ca.com/us/devcenter/ca-application-test.aspx>)
- [CA Service Virtualization](http://www.ca.com/us/devcenter/ca-service-virtualization.aspx) (<http://www.ca.com/us/devcenter/ca-service-virtualization.aspx>)

CA Education

CA Education DevTest Solutions Resources:

- [Course Search](http://education.ca.com/) (<http://education.ca.com/>)
- [CA Service Virtualization YouTube Playlist](https://www.youtube.com/playlist?list=PLynEdQRJawmzre0WtxEkIH2Nx3q4FUVZO) (<https://www.youtube.com/playlist?list=PLynEdQRJawmzre0WtxEkIH2Nx3q4FUVZO>)
- [CA Continuous Application Insight 8 \(CA CAI\) YouTube Playlist](https://www.youtube.com/playlist?list=PLynEdQRJawmwiGvI9t-2NAfI50Az_xeHz) (https://www.youtube.com/playlist?list=PLynEdQRJawmwiGvI9t-2NAfI50Az_xeHz)
- [Learning Path](http://www.ca.com/us/~/media/Files/LearningPaths/clp_devtest-solutions.pdf) (http://www.ca.com/us/~/media/Files/LearningPaths/clp_devtest-solutions.pdf)

Extensions

This section contains extensions to DevTest Solutions.

[OData Dynamic Virtual Services \(DVS\) \(see page 1880\)](#)

OData Dynamic Virtual Services (DVS)

OData Dynamic Virtual Services (DVS) is an extension to CA Service Virtualization and CA Application Test. DVS allows users to create OData service virtualizations that are hosted in a virtual service environment (VSE) that behave as real OData services.

You use POST, PUT, and DELETE to create, modify, or delete OData Entity Instances (records). Subsequent data retrieval through GET reflects these updates. With limited exceptions imposed by DevTest full OData v4, [OData Minimal Conformance Level](http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html#_Toc406398370) (http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html#_Toc406398370) for an update service is provided with many intermediate and advanced features.

You manage the data that you create through a REST API to save and load useful data sets. Development and QA use DVS in creating repeatable tests for applications using OData services without requiring an instance of a real service.

DVS consists of the following two components:

- Eclipse plug-in that allows designers to create the MAR file.
The MAR file instantiates the OData service from either a RAML or a configuration file (AEDM).
This file is similar to an Entity Data Model file.
- WAR file that can be deployed into a web server instance such as Tomcat, that supports a REST API.
This file allows direct creation and deployment of an OData virtual service from a RAML.

DVS uses an extension to CA Service Virtualization named the OData Virtual Services Extension which provides the OData emulation.

Click any of the following images for additional information:

DVS Features (see page 1881)	Building DVS (see page 1904)	Installing DVS (see page 1880)	Designing Your Virtual Service (see page 1913)	Deploying Your Virtual Service (see page 1922)	Managing Your Virtual Service (see page 1922)
 (see page 1881)	 (see page 1904)		 (see page 1913)	 (see page 1922)	 (see page 1922)

DVS Features (see page 1881)	Building DVS (see page 1904)	Installing DVS (see page 1880)	Designing Your Virtual Service (see page 1913)	Deploying Your Virtual Service (see page 1922)	Managing Your Virtual Service (see page 1922)
---------------------------------	------------------------------	-----------------------------------	---	---	--



(see page 1908)

OData Dynamic Virtual Service Features

DVS provides support at the [OData Minimal Conformance Level](http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html#_Toc406398370) (http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html#_Toc406398370) for an updatable OData v4 service. Limited exceptions are listed in the [\(see page 1881\) DVS Limitations](#) ([see page 1903](#)) section. [\(see page 1903\)](#) DVS also supports many intermediate and advanced features. Legacy OData v3 compliance to the same level is provided with exceptions, but is not being actively developed. See [DVS Limitations](#) ([see page 1903](#)) for a list of limitations in the DVS implementation.

Most of the example URLs given in this document use the **Bookshop** example service. We recommend that the Bookshop example service is installed in one of your VSE. Try out the examples and play with the virtual service (VS) using an interactive REST client such as *PostMan* or *Advanced Rest Client*. Remember, you can restore the example data to its original state. Restore the data by either stopping and restarting the VS or by reloading the data using the Administration API from seeddata.sql. In most cases, the example URLs have the protocol, server, port, and base path to the service elided for brevity. If the full URL is <http://myVSEserver:8844/odata/v4/Bookshop/Authors>, the example is written as .../Authors. When trying the example, replace the ... with the elided portions of the URL. Ensure accuracy when specifying your server name in place of myVSEserver. All examples use GET as the HTTP method unless explicitly noted. Most examples are based on the Bookshop example project that is provided with DVS.

You should be familiar with OData terminology and the meaning of the following terms:

- Entity set
- Entity type
- Property
- Navigation property

See www.odata.org (<http://www.odata.org>) for current **external** documentation for OData related topics, example service implementations, and other useful resources.

Method Support

HTTP methods POST, GET, PUT, and DELETE are used to create, read, update, and delete OData Entity Instances (records). For all requests that include a request body, specify **Content-Type:application/json** in the request header.

POST

POST is used when creating records. The resource URL must be an entity set URL. The resource can also be a URL whose final segment is a navigation property that points to a collection of entity types. In the second case, an implicit linkage is made between the entity instance (record) whose navigation property was used, and the new record. In each case, the call must include a request body in JSON format that contains at least one property for the entity type the resource path resolves. The key and required properties must be present, except for those key and required properties the model configuration (AEDM file) for the VS defines as automatically generated. Property names must be in double quotes and are case-sensitive. Property values for string and date time types must be in double quotes. Numeric and Boolean types can either be quoted or unquoted. The presence of an invalid property name causes the request to fail with status **400**.

You can perform a **deep insert** by optionally including in the request body the additional record data for other new records to be created simultaneously as the **parent** record. Each of these records is implicitly linked through a navigation property in the **parent** record.

By default, result of POST is echoed back in the response body. This result is useful when some properties have values that are generated or have defaults.

A successful POST creates a new record and returns status **201**.

Example 1 – Create a new author record by posting directly to the entity set.

POST/Authors

```
{
  "Name": "Anonymous",
  "DOB": null,
  "DOD": null,
  "Bio" : "Formerly the most prolific author of all."
}
```

POST is the http method,/Authors is the abbreviated URL which would be <http://myVSEserver:8844/odata/v4/Bookshop/Authors> in actual use and the block that is delimited by the curly braces the request body.

Example 2 – Create a book record and create an implicit link between that book and author by posting through a navigation property in the parent record.

POST/Authors('101')/Books

```
{
  "Id": "FIC-101-005",
  "AuthorId": "101",
  "Title": "Pickwick Papers",
  "Synopsis": "Still funny after all these years",
  "Weight": 1.0,
  "Size": {
    "Height": 9.0,
    "Width": 5.5,
```

```

        "Thickness": 1.0
    }
}

```

Example 3 – Create a book record and create a comment on the book simultaneously. This example is using **deep insert** to create multiple records at a time.



Note: In this example, because the **Inventory** navigation property is **one to one** and can only reference at most one record, the navigation property value is a single JSON object with the record value **e**. Where the navigation property is **one to many**, or **many to many**, the navigation property points to a collection of records. The value of the navigation property is a set of zero or more comma separated record instances as JSON objects, all within square brackets braces.

POST/Books

```
{
    "Id": "FIC-101-006",
    "AuthorId": "101",
    "Title": "The Old Curiosity Shop",
    "Synopsis": "Schmaltz sells..",
    "ListPrice": 19.95,
    "Weight": 1,
    "Size": {
        "Height": 9,
        "Width": 5.5,
        "Thickness": 1
    },
    "Inventory": {
        "odata.type": "Inventory",
        "Id": "FIC-101-006",
        "Price": 14.99,
        "InStock": 6
    }
}
```



Note : DVS has a defect where the response body for a deep insert only contains the **parent** record.

POST is used to create references (linkages) between records. See **\$ref** for this usage of POST.

GET

Used to return either single Entity Instances (records), or Collections of records, and in concert with **\$expand** can return records or Collections of records from multiple resource paths. Data that reaches a resource path is returned in JSON format, with scope and presentation of the data returned can be further controlled through System Query Options **\$filter**, **\$select**, **\$format**, **\$top**, **\$skip**, **\$count** and **\$orderby**, and also by certain request headers.

GET can also return a single property when the resource path resolves to one specific property instance. If this property is not a collection, or a complex property, **\$value** can be used in the URL to limit response to only the value data. (see **\$value**)

GET is also used to return a service document, or metadata about the OData service. See Service Document and **\$metadata** for more information.

A successful GET returns status **200 OK**.

For examples of using GET, see the information under System Query Options, Filter Operations, and Filter Functions.

PUT

Used to perform a full update in-place update of a single record. Request body is in JSON format, and all required properties **must** be present. Key properties can be omitted and always retain their original value. Any non-key properties absent from the request body are set to **null**. A successful PUT, by default returns no response body and have a return code of **204 No Content**. If a Prefer request header is specified and contains the value **return=representational**, then the updated record is returned, and the return code is **200 OK**.

Example 1 – Update author with Id '103'. Bio is set to a new value. DOB is set to the value previously held. Id retains the value of 103 and would not be changed even if specified in the request body. DOD is set to null because it is not specified and is not a key property.

```
PUT ..../Authors('103')
```

```
{
  "Name": "Julius G. A. Payder IV",
  "Bio": "Unpublished aspiring author",
  "DOB": "1956-07-04",
}
```

PUT can also be used to update a single property. This property can be either a simple property, a collection, or a complex property. DVS has a limitation in that complex properties that are included within another complex property cannot be updated. Nor can any property under such a complex property be updated.

Example 2 – Update a Complex Property within the Book instance with key 'FIC-101-001'.

```
PUT ..../Books('FIC-101-001')/Size
{ "value": { "Height": 9, "Width": 5.5, "Thickness": 0.2} }
```

Example 3 – Example of a syntactically valid PUT that fails due a DVS implementation limitation.

```
PUT ..../Books('FIC-101-001')/Size/Thickness
{ "value": 0.2 }
```

The message, **Multiple level of complex type property is not supported yet** with status **501 Not yet implemented** is returned.

DELETE

DELETE is used to delete a single record. DVS prevents an entity being deleted if it has linkages with one or more other entities. Until the links to those entities are removed to assure referential integrity is maintained. DELETE can manage linkages between records. See [\\$ref](#) for this usage of POST. DELETE can also be used to set individual properties to **null** or collection properties to an empty set. Required or key properties cannot be set to null. The same limitations that apply to single property PUT updates apply to DELETE.

A successful DELETE returns status **204 No Content**.

Example 1 – Delete the Book with Id 'FIC-101-001'

```
DELETE ..../Books('FIC-101-001')
```

Example 2 – Set the SkillSet of employee 'EMP00127' to the empty set.

```
DELETE ..../Personnel('EMP00127')/SkillSet
```

Resource Paths Supported

For any OData service virtualized by DVS, **all** resource paths that originate from an entity set and pass through the navigation properties defined by the developer for the OData VS are supported. The resource path does not need to be defined in the DevTest VSI to be processed. DVS calculates the valid paths using the EDM defined in the EDMtoDB.xml configuration file for the VS. Calls against the VS can use URLs that pass through any number of Navigation Properties (NPs) to any depth, as long as the NPs are valid for the type containing the NP, **and** the path is syntactically valid per OData standards. For example, OData only permits navigation properties to collections that are not qualified by a key as the last resource segment of a URL. DVS correctly returns an error when processing a URL that violates that rule.

Navigation properties can be defined as either one way or bidirectional. Properties can map one to one, one to zero or one, one to many, many to one, or many to many relationships between entity instances (records) that can be in the same or different entity sets.

Resource path can terminate in a resource that resolves to either an entity set, an entity type instance (record), a single entity type instance property, or a collection property.

Certain keywords can also appear in resource paths that resolve to user-defined resources. Only one of these keywords can be present in any (valid) URL.

Specific instances within an entity set or a navigation property that resolves to an entity set is selected by appending *(keyvalue)* or *(keyprop=keyvalue)* to the resource segment. The second syntax is required if there are multiple key properties with each *keyprop=keyvalue* pair that is separated by a comma. Non-numeric key values must be within single quotes.

\$value

\$value is used with GET and must be the last segment in the resource path. Like all Single Property requests, the Resource path (URL) must otherwise resolve to a single Property instance. For example:

`h http://myVSEserver:8769/mybaseresourcepath/myEntitySet('A_Key_Value')/anEntityProperty /
$value (http://devjo01w7a:8066/odata/vPub/v4/Beers\(\))`

Causes response body to be simply the value of the requested Property, in text/plain form without any surrounding JSON formatting.

\$count (resource path keyword)

\$count (resource path keyword) can only be used with GET and must be the last segment in the resource path. The resource path must otherwise resolve to a collection (a set of zero or more records).

For example:

`http://myVSEserver:8769/mybaseresourcepath/myEntitySet /$count (http://devjo01w7a:8066/odata/vPub/v4/Beers\(\))`

Causes response to be the integer count of the number of matching items in text/plain form without any surrounding JSON formatting. The value that is returned reflects the effect of any **\$filter**, **\$skip**, and **\$top** System Query Options present as parameters.

\$ref

Available for OData version **4.0** compatible virtual services only. **\$ref** must be the last segment in the resource path. **\$ref** changes the meaning of the URL from referring to the object to a reference of the object.

\$ref is used with POST, PUT, and DELETE, and is used to manage navigation links between entity instances.

The OData standard requires that a navigation property resource path segment to immediately precede the **\$ref** keyword because the navigation property specifies what association is modified. It is valid for multiple navigation properties to exist between the same two entity sets. For example, an employees entity set can contain records of entity type employee where the employee has both a supervisor and staff navigation property pointing back to the employees entity set. While both would link records of the same type, they would represent different semantics.

If the navigation property was defined with a partner, both the navigation property that is specified, and its partner are affected. For example, if the supervisor represented an employees direct manager, it would be defined as the partner navigation to staff. The staff navigation property be a one to many relationship. An employee can have zero or more employees reporting to them, but an employee would only have one direct supervisor.

Examples:

`POS T .../WorkTeams('Associates') / (http://devjo01w7a:8066/odata/vPub/v4/Beers\(\))Members/$ref`

With a request body of the following:

```
{
  "url" : "Personnel('EMP00133')"
}
```

Would explicitly associate the Personnel record with Id 'EMP00133' with the WorkTeam record with key 'Associates' .

DEL ETE .../WorkTeams('Associates') / ([http://devjo01w7a:8066/odata/vPub/v4/Beers\(\)Members/\\$ref](http://devjo01w7a:8066/odata/vPub/v4/Beers()Members/$ref)) is the reverse of the POST example and removes any association that had been established between the two records through the navigation property.

Both will return a Status 204 "No Content" on success.



Note: DVS does not support the **\$id** keyword. Resource URLs containing **\$id** as a parameter are not supported.

\$link

\$link is available for OData version **3.0** compatible virtual services only. OData version **3** is the version 3 equivalent to \$ref, and functions in a similar way. One notable difference is that the **\$link** keyword appears **before** the last resource path segment, which **must** be a navigation property.

Two Examples:

POST .../myEntitySet('EndpointOneKey')/\$links/NPName

With a request body of the following:

```
{
  "url" : "TargetEntitySet('keyvalue')"
}
```

Would explicitly associate the record in Entity Set myEntitySet with key 'EndpointOneKey' to the record in TargetEntitySet with key 'keyvalue'.

DELETE .../myEntitySet('EndpointOneKey') / ([http://devjo01w7a:8066/odata/vPub/v4/Beers\(\)\\$links/NPName\('keyvalue'\)](http://devjo01w7a:8066/odata/vPub/v4/Beers()$links/NPName('keyvalue'))) is the reverse of the POST example that is given previously, and removes any association that had been established between the two records through the navigation property.

Both return a Status 204 "No Content" on success.

Special Resource Paths

In addition to resource paths that resolve to user-defined resources, two special resource paths are provided.

Service Document

A GET to the base URL of the OData VS returns a **Service Document** in JSON format, which is a list of the entity sets defined in the VS.

For example: GET <http://myVSEserver:8844/odata/v4/Bookshop> returns 200 OK with a response body of

```
{
  "@odata.context": "http://myVSEserver:8844/odata/v4/Bookshop/$metadata
  "value": [
    {
      "name": "Customers",
      "kind": "EntityType",
      "url": "Customers"
    },
    {
      "name": "Comments",
      "kind": "EntityType",
      "url": "Comments"
    }
  ]
}
```

\$metadata

A GET to the base URL of the OData VS followed by the key word **\$metadata** returns an XML representation of the Entity Data Model schema for the VS.

See [OData Version 4.0 Part 3: Common Schema Definition Language](http://docs.oasis-open.org/odata/v4.0/odata-v4.0-part3-csdl.html) (<http://docs.oasis-open.org/odata/v4.0/odata-v4.0-part3-csdl.html>) for a full description of the elements that can be returned from a \$metadata.

In its **\$metadata** response, DVS returns

Element	Child Elements
EnumType	Member
ComplexType	Property
EntityType	Key, Property, NavigationProperty
EntitySet	NavigationPropertyBinding

For example: GET [http://myVSEserver:8769/mybaseresourcepath/\\$metadata](http://myVSEserver:8769/mybaseresourcepath/$metadata) can return a response similar to the following response:

```
<SchemaName version=1.0>

<EnumType Name="SomeEnumeratedType" UnderlyingType="Edm.Int32">

  <Member Name="4", Value="0" />

  ... more Member definitions ..

</EnumType>

... more EnumType definitions ...

<ComplexType Name="SomeComplexType1">

  <Property Name="APropertyForThisType" Nullable="false" Type="Edm.String",>

    <Property Name="AnotherPropertyForThisType" Nullable="true" Type="Edm.Int32",>
```

... more Property definitions ...

```
</ComplexType>
```

... more ComplexType definitions ...

```
<EntityType Name="SomeEntityType1">
```

```
<Key>
```

```
<PropertyRef Name="AKeyField">
```

```
</Key>
```

```
<Property Name="AKeyField" Nullable="false" MaxLength=16 Type="Edm.String" />
```

```
<Property Name="AFieldwithDefault" Nullable="false" DefaultValue="something" Type="Edm.String" />
```

... more Property definitions ...

```
<NavigationProperty Name="NPName" Partner="NameOfReverseNavigationInTarget" Type="Collection(SchemaName.TargetEntityType)" />
```

```
<NavigationProperty Name="NPName2" Type="SchemaName.YetAnotherTargetEntityType" />
```

... more Navigation Property definitions ...

```
</EntityType>
```

... more EntityType definitions ...

```
<EntityContainer>
```

```
<EntitySet Name="SomeEntitySet1" EntityType="SchemaName.SomeEntityType1" >
```

```
<NavigationPropertyBinding Path="NPName", Target="TargetEntitySet" />
```

```
<NavigationPropertyBinding Path="NPName2", Target="YetAnotherTargetEntitySet" />
```

... more Navigation Property Binding declarations ...

```
</EntitySet>
```

... more EntitySet declarations ...

```
</SchemaName>
```

System Query Options

Query Options (SQOs) are parameters added to an OData URI to control aspects of the data. Most parameters are only valid with GET. SQOs are keywords that always appear after the ? that terminated the resource path portion of the OData URI from the parameter portion. Each SQO has an argument which is separated from the keyword by '='. Multiple SQOs can be specified in a URI by separating them with &. Each SQO can only appear once for a given resource path.

\$format

\$format controls the representation of the data in the response body. In general, DVS returns response bodies in JSON format with the exception that \$metadata is always returned as XML. URLs containing the \$value, and \$count keywords always return data as text/plain. DVS can use a valid default for each request. Setting the data format is never required. Format can also be specified in the request header. OData v4, and OData v3 support different options for JSON responses.

\$format=json is the default for all calls that would return a JSON response.

\$format=verbosejson can be used with OData version 3.0 services only and causes more meta data to be included in the response body.

\$format=text/plain must be used with \$count, and \$value.

\$format=xml must be used for \$metadata requests.

With OData version 4.0, you can explicitly control the amount of meta data returned.

\$format=application/json;odata.metadata=minimal is the default metadata setting, and causes the "@odata.context" to be included in the response body.

\$format=application/json;odata.metadata=None suppresses all metadata in the response body.

\$format=application/json;odata.metadata=full causes all available metadata to be returned. For DVS, this includes @odata.context and @odata.type", @ data.id (<http://data.id/>), and @data.editlink for each record, and NavigationPropertyName@odata.associationLink, and N navigationPropertyName@odata.navigationLink for each Navigation Property in each record.

With either OData version, you can specify the format in the request header.

- **Accept: application/json** - for the default less verbose JSONOutput
- **Accept: application/json;odata=verbose** - for verbose JSON output (Odata v3)
- **Accept: application/json;odata.metadata=minimal** - same as parameter equivalent.
- **Accept: application/json;odata.metadata=None** - same as parameter equivalent.
- **Accept: application/json;odata.metadata=full** - same as parameter equivalent.
- **Accept: application/xml** – for \$metadata requests
- **Accept: text/plain** – for \$count and \$value requests

A \$format specification in the URL overrides the Accept settings in the request header.

\$filter

\$filter is used to restrict the data returned to only those records which meet certain criteria. It is only used with GET. The argument to **\$filter** is a logical expression that is comprised of references to properties in the entity pointed to by the current resource path context, or that can be reached through navigation properties from the current resource path context, filter operators, filter functions, and/or literals. **\$filter** SQOs only affect the resource for the resource path context in which they are declared. Requests that reference multiple entities can have separate **\$filter** SQOs applied to each resource (see **\$expand**).

See the sections for filter operators, and filter functions for the list of supported operators and functions. DVS does not support **\$count** in **\$filter** arguments.

Example 1 - Return only those books that fit on a 10 inch high shelf.

```
.../Books?$filter=Size/Height lt 10
```

Example 2 - Return only those books which have a cost less than 20.00 dollars and written by Dickens.

```
.../Books?$filter=ListPrice lt 20.00 and Author/Name eq 'Charles Dickens'
```

Example 3 - Return only those books which cost less than 20.00 dollars and were written by Dickens, OR has a name that contains the string **Gulden**.

```
.../Books?$filter=(ListPrice lt 20.00 and Author/Name eq 'Charles Dickens') or contains>Title,'London')
```

\$select

\$select controls which properties are returned in the response body. **\$select** is only valid for GET and can appear as a subsidiary parameter for **\$expand**.

Argument to **\$select** is either a comma-separated list of property names, or the asterisk (*) character. **\$select** is a shortcut equivalent to a list of all non-navigation properties. The property names must be valid for the entity type corresponding to the resource path against which **\$select** is applied, and DVS does not support selecting only some properties from within a complex property that is part of that Entity, so for DVS, only property names that are directly in the object can be arguments. ... /myEntitySet?\$select=APropertyName,AComplexPropertyName is valid, assuming the two property names that are listed are part of the entity type for entity Set *myEntitySet* , ... /myEntitySet?\$select=APropertyName,AComplexPropertyName/AComplexPropertyProperty is not supported by DVS.

Example 1 - Return the title and synopsis for each book

```
.../Books?$select=Title,Synopsis
```

Example 2 - Return the name title and synopsis for each book which weighs over two pounds. (Demonstrates multiple SQOs in a request)

```
.../Books?$select=Title,Synopsis&$filter=Weight gt 2.0
```

Example 3 - Return all the property values for a specific Author.



Note: Returning all properties is the default behavior for an OData service. Explicitly specifying \$select=* is typically not required.

.../Authors('100')?\$select=*



Note: For OData version 3.0 only, you can include properties reached through navigation properties in the argument list for \$select . This is because the limited \$expand syntax supported by OData version 3.0 had no provision for associating SROs with specific branches of an expand .

\$orderby

\$orderby allows control of the order in which records are returned. Any simple property in the set of entities and by the resource path be used in the argument list and simple properties reachable using navigation paths from the current context. These navigation paths must all resolve to a single instance. One to many or many to many navigation paths are not permitted. Keyword **desc** can be used to override the default sort order of low to high. Keyword **asc** is also available if you want to state the sort order.

Example 1 - Return the title, synopsis, and list price for each book that is ordered from the lowest list price to the highest list price.



Note: Null values sort low.

.../Books?\$select=Title,Synopsis,&\$orderby=ListPrice

Example 2 - Return the title, synopsis and list price for each book ordered from the highest list price to the lowest list price, and sort ties by title in ascending order. The **asc** keyword is optional.

.../Books?\$select=Title,Synopsis,ListPrice&\$orderby=ListPrice desc,Title asc

Example 3 - Return the title, synopsis and list price for each book ordered by the actual price asked.

.../Books?\$select=Title,Synopsis,ListPrice&\$orderby=Inventory/Price&\$expand=Inventory
(\$select=Price,InStock)



Note: Currently, DVS does not enforce the restriction prohibiting one to many or many to many (<http://manymany/>) navigation paths to properties used in an \$orderby and will return reasonable results. This behavior is not dependable as the presence of one to many or

many to many (<http://manymany/>) navigation paths means that the path does not necessarily resolve to one single value. The results can be ambiguous or undefined. Defect [218056](https://jazz01.ca.com/ccm04/resource/itemName/com.ibm.team.workitem.WorkItem/218056) (<https://jazz01.ca.com/ccm04/resource/itemName/com.ibm.team.workitem.WorkItem/218056>) is opened to resolve this. The correct behavior is to return a 400 Bad Request status.

\$skip

\$skip takes as an argument an integer which is the number of otherwise qualifying records to pass over and omit from the response body. **\$filter** is applied to the result set before **\$skip** when determining the result set in the response. When counting records for **\$skip**, the record at the same level as **\$skip** and any records nested within that record due **\$expand** are counted as one record. Together with **\$top** this is used by an application to page though data where it would be inconvenient to retrieve all of it at one time.

Example - Fetch the name and description for all authors, but skip the first three authors.

```
.../Authors?$select=Name,Bio&$skip=3
```

\$top

\$top allows the user to restrict the number of records returned by specifying the maximum number of records to return as an argument. The effect of **\$filter** and then **\$skip** are applied before **\$top**. Like **\$skip**, when counting records for **\$top**, the record at the same level as **\$top** and any records nested within that record due **\$expand** are counted as one record. Together with **\$skip** this is used by an application to page though data where it would be inconvenient to retrieve all of it at one time.

Example 1 - Fetch the title and list price for all books whose list price is less than 50.00 but skip the first three books, and only return a maximum of five books.

```
.../Books?$select=Title,ListPrice&$filter=ListPrice lt 50.00&$skip=3&$top=5
```

Example 2 - Same as previous example, but include the comments on the books. The same number of books is returned with selected properties from the comment records on these books.

```
.../Books?$select=Title,ListPrice&$filter=ListPrice lt 50.00&$skip=3&$top=5&$expand=Comments ($select=CriticName,Comment)
```

\$expand

\$expand allows a user to request related data from multiple Entity Sets. **\$expand** accepts a comma-separated list of resource paths composed of navigation properties originating from context in which the **\$expand** is specified. Each top-level record is returned with child record data nested within it. If an expand resource path traverses several entities, all entities traversed are expanded, and data for each entity at a deeper level is nested within the parent record in the level above. DVS will ignore a **\$ref** keyword that is the last segment of an **\$expand** resource path argument.

OData version 4.0 compatible services allow each of the resource paths arguments to expand to specify System Query Options to apply only to that resource path. These options appear within parenthesis directly after the resource path, with multiple System Query Options separated by a semi-colon (";"). The System Query Options supported DVS within **\$expand** are **\$filter**, **\$select**, **\$top**, **\$skip**, and **\$orderby**. Advanced feature **\$levels** is not supported.

Some examples using the vPub schema (omitting the base path to the OData service)

Example 1 - Return all the author records with book records nested under each author. A **authors** is an entity set, and books is a navigation property in entity type author.

```
.../Authors?$expand=Books
```

Example 2 - Return all the book records for author('101') with the comment records for each book nested under each author. A **authors** is an entity set. Books are a navigation property in entity type author. Comments is a navigation property in entity type book.

```
.../Authors('101')/Books?$expand=Comments
```

Example 3 - Both these URLs returns all author records with book records nested under each author and comments on each book nested each book record. The second form is only supported under OData version 4.0.

```
.../Authors?$expand=Books/Comments
```

```
.../Authors?$expand=Books($expand=Comments)
```

Example 4 - Return all author records with book records nested under each author in title order, but only for those book records whose **ListPrice** Property value is over 10.00, and only the **Title** and **Synopsis**. Comments on each book are nested under the book record, but only return the **CriticName**, and **Comment** Property values.

```
.../Authors?$expand=Books($filter=ListPrice gt 10.0;$orderby=Title;$select=Title,Synopsis;$expand=Comments($select=CriticName,Comment))
```

Example 5 - As Example 4, except that **Authors** without books whose list price is greater than 10 dollars are excluded. Remember comparisons to null evaluate to false and System Query Options are only applied to the resources at the same level they appear.

```
.../Authors?$expand=Books($filter=ListPrice gt 10.0;$orderby=Title;$select=Title,Synopsis;$expand=Comments($select=CriticName,Comment))&$filter=Books/ListPrice gt 10
```

\$count (System Query Option)

The **\$count** keyword is used in two ways within OData. It can be used as part of a resource path, or starting with OData version **4.0** as a System Query Option. As a System Query Option it allows the user to control whether **@odata.count** is part of the response body. If the argument to **\$count** is **true**, then **@odata.count** is returned and has a value of the integer count of the qualifying records. The count is affected by **\$filter** but ignores the effect of **\$skip** and **\$top**. This number does not reflect the actual number of records in the response body.

If **\$count** is absent or has an argument of **false**, then **@odata.count** is omitted from the response.

Example 1 - Fetch the title and list price for all books whose list price is less than 50.00 but skip the first three books, and only return a maximum of five books. Include the count of all qualifying records.

```
.../Books?$select=Title,ListPrice&$filter=ListPrice lt 50.00&$skip=3&$top=5&$count=true
```

While a maximum of five records are returned due to `$top`, the `@odata.count` value reflects the full number of records that match the `$filter` restriction, and that would have been returned if both `$top`, and `$skip` were absent.



Note: `$count` is not a valid System Query Option in OData version 3.0. Instead `$inlinecount` is used with an argument of either `allpages` or `none`.

Filter Operators

DVS supports all `$filter` Operators that are defined in OData version 3.0. DVS does not support the `has` operator that is introduced with OData version 4.0.

Sub-expressions can be grouped using parenthesis to override precedence. Operators are processed in precedence level order (level 1 = highest priority). Operators within an expression with equal precedence are evaluated left to right.

Operator	Description	Precedence Level
<code>mul</code>	Arithmetic multiplication . Arguments must be of numeric types.	1
<code>div</code>	Arithmetic division . Arguments must be of numeric types. Right Argument of zero will fail query and return a status 500	1
<code>mod</code>	Arithmetic modulo . Arguments must be of numeric types. Right Argument of zero will fail query and return a status 500	1
<code>add</code>	Arithmetic addition . Arguments must be of numeric types	2
<code>sub</code>	Arithmetic subtraction . Arguments must be of numeric types.	2
<code>eq</code>	Compare arguments and evaluate to true if arguments are equal	3
<code>ne</code>	Compare arguments and evaluate to true if arguments are not equal	3
<code>gt</code>	Compare arguments and evaluate to true if Left argument Greater than Right Argument	3
<code>ge</code>	Compare arguments and evaluate to true if Left argument Greater than or equal to Right Argument	3
<code>lt</code>	Compare arguments and evaluate to true if Left argument Less than Right Argument	3
<code>le</code>	Compare arguments and evaluate to true if Left argument Less than or Equal to Right Argument	3
<code>not</code>	Logical not . Evaluates to true if right argument is false, evaluates to false if right argument is true	4
<code>and</code>	Logical and . Evaluates to true if both arguments are true	5
<code>or</code>	Logical or . Evaluates to true if either argument is true	6



Note: Comparisons must be made between expressions of compatible types. For example, an expression that resolves to any numeric types may be compared against other expression that resolves to a numeric type, but may not be compared to an expression whose base type is a string.

Any property which is not a key, and was not defined as being required may have a null value to indicate its value is undefined. Any comparison where one or both of the arguments are **null** will evaluate to **false**, except comparisons using **eq** or **ne** against the keyword literal **null**. Both `$filter=APropertyWithNullValue le SomeLiteral` and `$filter=APropertyWithNullValue ge SomeLiteral` will evaluate to **false** when `APropertyWithNullValue` contains a value of **null** even if `SomeLiteral` is the literal **null**.

Built-in Filter Functions

DVS supports nearly the full set of filter functions. The exception being those functions such as **isof**, which allow tests for specific entity types.

String functions

contains(string,substring) - Returns **true** if second string argument is the same as, or is found within, the first string argument, or if `substring` is **null**. Will return **false** if `string` is **null**. Introduced as of OData version 4.0.

```
.../Books?$filter=contains>Title,'London'
```

substringof(substring,string) - Returns true if first string argument is the same or is found within the second string argument, or if `substring` is **null**. Will return false if `string` is **null**. This is the OData version 3.0 equivalent of **contains**. This is not supported by Odata version 4.0, and also notes the reversal of the parameters as compared to other string functions.

```
.../Books?$filter=substringof('London',Title)
```

startswith(string,substring) - Returns true if second string argument matches the first string argument starting from the beginning of the first string argument, or if `substring` is **null**. Will return false if `string` is **null**.

```
.../Books?$filter= startswith ( Title,'T' )
```

endswith(string,substring) - Returns true if second string argument matches the end of the first string argument, or if `substring` is **null**. Will return false if `string` is **null**.

```
.../Books?$filter= endswit h( Title,'Cities' )
```

length(string) - Returns an integer with value equal to the number of characters in the string. If the value of `string` is **null**, **null** is returned.

```
.../Comments?$filter= length ( Comment) gt 40
```

indexof(string,substring) - Returns an integer with value equal to the position of `substring` within `string`, with 1 being the first position. If `substring` is not in `string`, 0 is returned. If either argument has a value of **null**, **null** is returned.

.../Comments?\$filter=indexof(Comment,'read') ne 0

substring(string,position) - Returns a String whose value is the value of *string* starting at *position*, where a position of 1 is the first character of the string argument. If either argument has a value of **null**, **null** is returned.

.../Books?\$filter=substring>Title,7) eq 'H ouse'

substring(string,position,length) - Returns a String whose value is the value of *string* starting at *position* up to a maximum of *length* characters, where the first character of the string has position 1. If any argument has a value of **null**, **null** is returned.

.../Authors?\$filter=substring(Bio,14,5) eq 'Irish'

tolower(string) - Returns a String whose value is the original value with all uppercase characters converted to lowercase. If the value of *string* is **null**, **null** is returned.

.../Comments?\$filter=indexof(tolower(Comment),'dark') ne 0

toupper(string) - Returns a String whose value is the original value with all lowercase characters converted to uppercase. If the value of *string* is **null**, **null** is returned.

.../Comments?\$filter=indexof(toupper(Comment),'DARK') ne 0

trim(string) - Returns a String whose value is the original value with all leading and trailing white space (space, tab, etc. characters) removed. If the value of *string* is **null**, **null** is returned.

.../Authors?\$filter= trim (substring(Name,1,8)) eq trim (' Charles ')

concat(string1,string2) - Returns a String whose value is the *string2* appended to *string1*. If either argument has a null value the value of the non-null argument is returned, and if both arguments have null values, **null** is returned.

.../Comments?\$filter=contains(Comment,concat('child','ren'))

replace(string,substring1,substring2) - OData version **3.0** only. Returns a string whose value is the value of *string*, except that each occurrence of *substring1* is replaced by *substring2*. If the value of *string* is **null**, **null** is returned. If *substring1* is null, the value of *string* is returned unchanged, and if *substring2* is null, all occurrences of *substring1* are removed from the result string. In the example, all space characters are being removed from name before the comparison to the literal.

.../SomeEntitySet?\$filter=contains(replace(SomeProperty,' ',null), 'SomeStringLiteral')

Date and Time Functions

year(DateTimeOffset) - Return the year portion of a DateTimeOffset as an integer. If *DateTimeOffset* is null, returned value is **null**.

.../Comments?\$filter=year(Posted) eq 2015

month(DateTimeOffset) - Return the month portion of a DateTimeOffset as an integer (January equals 1, December equals 12). If *DateTimeOffset* is null, returned value is **null**.

.../Comments?\$filter=month(Posted) eq 5

day(DateTimeOffset) - Return the day of the month portion of a DateTimeOffset as an integer. If *DateTimeOffset* is null, returned value is **null**.

.../Comments?\$filter=day(Posted) eq 2

hour(DateTimeOffset) - Return the hour portion of a DateTimeOffset as an integer. If *DateTimeOffset* is null, returned value is **null**.

.../Comments?\$filter=hour(Posted) lt 11

minute(DateTimeOffset) - Return the minute portion of a DateTimeOffset as an integer. If *DateTimeOffset* is null, returned value is **null**.

.../Comments?\$filter=minute(Posted) ge 30

second(DateTimeOffset) - Return the second portion of a DateTimeOffset as an integer. If *DateTimeOffset* is null, returned value is **null**.

.../Comments?\$filter=second(CommentTimestamp) lt 30

fractionalseconds(DateTimeOffset) - Return the fractional seconds portion of a DateTimeOffset as a float. If *DateTimeOffset* is null, returned value is **null**.

.../Comments?\$filter=fractionalseconds(CommentTimestamp) eq 0.5

now() - Return the current DateTimeOffset. Timezone is determined by the timezone in effect on the server hosting CA Service Virtualization Virtual Service Environment hosting the OData VS.

.../Comments?\$filter=Posted gt now()

mindatetime() - Return the earliest date that is supported by the OData VS. DVS returns January 1st 1753, which is the date many nations that are switched to the Gregorian Calendar.

.../Comments?\$filter=year(Posted) eq 2015 and year(mindatetime()) eq 1753 and month(mindatetime()) eq 1 and day(mindatetime()) eq 1

maxdatetime() - Return the most distant date that is supported by the OData VS. DVS returns Dec 31st, 9999. However this is not actually true, as DVS with the H2 database will support dates millions of years in the future.

.../Comments?\$filter=year(Posted) eq 2015 and year(maxdatetime()) eq 9999 and month(maxdatetime()) eq 12 and day(maxdatetime()) eq 31

Mathematical Functions

round(decimalValue) - Returns the input numeric value that is rounded to the nearest integer value. If the decimal portion of the input is less than 0.5, the next lowest integer value is returned. Otherwise, the value is increased to the next integer. If the input value resolves to null, **null** is returned. Both following examples returns the same result set.

.../Books?\$filter=round(Weight) eq 1.0

```
./Books?$filter=round(Weight) eq 1
```

floor(decimalValue) - Returns the input numeric value that is reduced to the integer value. That is if the decimal portion of the input is set to zero. If the input value resolves to null, **null** is returned.

```
./Books?$filter=floor(Weight) eq 0
```

ceiling(decimalValue) - Returns the input numeric value that is increased to the next highest integer value if the decimal portion of the input is not zero. If the input value resolves to null, **null** is returned.

```
./Books?$filter=ceiling(Weight) eq 2
```

Request and Response Headers

In addition to the headers described under **\$format**, DVS supports or generates the following headers.

Odata-Version

OData-Version can be provided as a request header. Value should resolve to a number. DVS checks this numeric value against the OData version the target VS is configured to emulate, and if versions are not equal the request is rejected with status 400, and a message indicating what OData version is supported. At present a DVS configured for OData version 4.0 does not support OData version 3.0 requests, and the other way around, and no other version of OData is supported.

Odata-Version is always returned as a response header whose value is either 3.0, or 4.0 depending on the OData version the target VS is configured to emulate.

OData-MaxVersion

OData-MaxVersion may be provided as a request header. Its value is ignored if OData-Version is also present. If OData-MaxVersion has a value that resolves to a number equal to or greater than the OData version the target VS is configured to emulate, then the request passes, but is otherwise rejected with status 400. A message indicating the OData version the target VS is configured to emulate.

Prefer

By default POST returns a response body containing the newly created entity in JSON format, while PUT and by default, PATCH responds with a **204 No Content**. The following prefer headers allow explicit control of this behavior.

return=minimal

For successful calls to methods POST, PUT, (and if supported PATCH) with a VS configured for OData version **4.0**, **return=minimal** suppresses the response body. For a VS configured for OData version, **3.0,return-no-content** has the same effect.

[return=representational](#)

For successful calls to methods POST, PUT, (and if supported PATCH) with a VS configured for OData version **4.0**, **return=representational** forces a return of a response body containing the newly created or modified values in JSON format. For a VS configured for OData version **3.0**, **return-content** has the same effect.

[Location](#)

A response header, only returned when performing a POST to a VS configured for OData version **4.0**. Value of header is the edit URL of the newly created entity.

[OData-EntityId](#)

A response header, only returned when performing a POST to a VS configured for OData version **4.0** where a Prefer header with a value of return=minimal was in effect. Value of header is the entity ID (key or keys), of the newly created entity.

Extensions Outside of OData Standard

The following features support DVS, but are not part of the OData standard.

Automatic Generation of Property Values

You define the entities in your DVS VS to have one or more of the property values automatically generated.

Property values can be configured to be generated once when the entity instance is first created using POST, or for every update. The first case is useful for automatically generating key values, or a creation timestamp. The second case can be used to generate the value for a property whose value is the last modification timestamp.

See Property in [Designing Your Virtual Service \(see page 1913\)](#) about configuring a property to use automatic value generation.

REST API to Manage Data

Any DVS VS also supports a REST API that allows saving the current set of entity data out to a file or loading of the entity data from a file. This feature is useful when there is a need to reset data to a known state for testing.

See [Managing Your Virtual Service Data \(see page 1926\)](#) for details.

Custom Properties Defined at Run-Time

DVS supports adding custom properties (CPs) at runtime for entity sets whose entity type has the feature enabled. If an entity type is enabled, there is an implicit entity type and entity set created with the names `x` `Property` , and `x` `Properties` . Where `x` is the name of the entity type having the custom properties feature enabled. Users should avoid defining names that would conflict with these generated names.

For example, entity type order in the Bookshop example is defined with custom properties enabled. Therefore

```
<EntityType Name="OrderProperty">
<Key>
<PropertyRef Name="name"/>
</Key>
<Property MaxLength="256" Name="name" Nullable="false" Type="Edm.String"/>
<Property MaxLength="256" Name="dataType" Type="Edm.String"/>
<Property MaxLength="4096" Name="description" Type="Edm.String"/>
<Property MaxLength="256" Name="defaultValue" Type="Edm.String"/>
<Property MaxLength="4096" Name="possibleValues" Type="Edm.String"/>
<Property MaxLength="256" Name="producerMappingProperty" Type="Edm.String"/>
<Property MaxLength="256" Name="entityName" Type="Edm.String"/>
</EntityType>
```

and

```
<EntitySet EntityType="Bookshop.OrderProperty" Name="OrderProperties"/>
```

appear in the \$metadata for the Bookshop VS.

Defining a Custom Property

To create or modify custom properties, standard OData operations are performed against this associated entity set. For example, to create a new custom property for order:

```
POST .../OrderProperties
{
  "name": "ANewCustomProperty",
  "dataType": "edm.String",
  "description": "free form text description of property"
}
```

With the following response:

```
{
  "@odata.context": "http://\\MyVSEServer:8844\\odata\\v4\\Bookshop\\$metadata#OrderProperties",
  "value": [
    {
      "name": "ANewCustomProperty",
      "dataType": "Edm.String",
```

```

"description": "free form text description of property",

"defaultValue": null,

"possibleValues": null,

"producerMappingProperty": null,

"entityName": null

}

]

}

```

The following properties that can be specified:

- ▪ **name**: The identifier of the CP and must conform to the syntax of a property name. Name must be unique within all the properties of the entity type containing the CP.
- **dataType**: The datatype to be used for the CP. Only simple types such as Edm.String, and Edm.Int32 are supported.
- **description**: Intended to contain free form text which the designer can use to describe the CP.
- **defaultValue**: The value to give the CP if CP is not specified in a POST or PUT action.
- **possibleValues**: A comma delimited list of values which is acceptable as valid for the CP.
- **producerMappingProperty**: For future use, do not specify.
- **entityName**: For future use, do not specify.

You can modify characteristics of a previously defined custom property by using PUT. For example, the following changes the description and add a default value to the CP defined in the previous example:

```

PUT .../OrderProperties('ANewCustomProperty')
{
  "dataType": "edm.String",
  "description": "updated free form text description of property",
  "defaultValue": "SomeDefaultStringValue"
}

```

Some limitations are imposed by the underlying database. For example, changing the datatype can fail if values stored for the CP are incompatible with the new type.

Working with Custom Properties

Depending on how the EntityType is defined, the new custom property can either be accessible as a simple property in the entity itself or as one of a set of name value pairs in a collection (NVP CP). See WorkTeam in the Bookshop example for an entity with CPs used this way. The name of the collection is specified as part of the configuration, and the collection consists of instances of an implicit complex type with properties **propertyName**, and **propertyValue**. See the following example and order in the Bookshop example for an entity with CPs used this way.

```
POST ..../Orders
{
  "CustomerId": "CUS00137",
```

Other regular properties:

```
"Special": [
  {
    "propertyName": "ANewCustomProperty",
    "propertyValue": "MyValue"
  }
]
```

See EntityType in [Designing Your Virtual Service \(see page 1913\)](#) for defining an entity type with this feature enabled.

DVS Limitations

DVS does not support any feature that is only required for intermediate or advanced OData v4 conformance, unless indicated. Features that are part of the [OData Minimal Conformance Level](#) (http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html#_Toc406398370) for an updatable service are typically implemented with exceptions.

Omissions to OData Minimal Conformance Level

PATCH method for differential update. This method should be available for a compliant updatable OData service. CA Service Virtualization does not allow HTTP methods apart from POST/GET/PUT /DELETE. Until HTTP methods are supported, PATCH cannot be implemented.

Other omissions

These omissions are not required for minimal conformance. DVS returns a **501 Not Supported** message for unsupported features.

- DVS does not support stream, media, geometric, and geographic types.
- DVS does not support **\$filter** functions related to class. For example, the **isof** functions are not supported. Due to limitations in the Olingo 4.0.0-Beta-02, parser **\$ref** cannot be used with **\$filter** or **\$orderby**.
 - bool IsOf(type value), for example, **isof('NorthwindModel.Order')**
 - bool IsOf(expression value, type targetType), for example, **isof(ShipCountry,'Edm.String')**
- No support for **\$count** in **\$filter** arguments.
- **\$filter** does not support the **has** operator.
- **\$expand** does not support **\$format**, **\$count**, or **\$levels**.
- **\$select** does not support selecting only some properties from within a complex property that is part of that entity against which **\$select** is applied.
- **\$id** URL keyword is not supported.
 - Deep insert where nested records have generated keys does not work.
 - Cannot POST or PUT to properties within complex properties.
 - DELETE of a **custom** property is not supported.

Known Issues

See README.txt in the distribution to see a list of known defects at the time of posting.

Building Dynamic Virtual Service

DVS is not distributed as binary files that can be directly used for installation. The download contains the necessary source files and build artifacts to create the install binaries. Third-party components are loaded from public repositories as needed. The build process is simple. Only commonly available, free open source tools are required to perform the build.

The DVS sources builds two primary artifacts for the OData Dynamic Virtualization Services project. The artifacts are the DVS servlet and the OData Management Extension Assistant (an Eclipse plug-in). The DVS sources packages for deployment the OData DVS extension. The extension is deployed into DevTest Virtual Service Environments to enable the OData emulation.

Prerequisites

To build the DVS, the DVS must be downloaded and the following tools must be installed and configured:

- Java SE Development Kit (JDK) to compile the source code
- Apache Ant to orchestrate the build

- A small set of Ant extensions (installed in lib directory of Ant)
- Maven to build the Eclipse Plug-in



Note: Line breaks do not display correctly in certain editors when DVS is downloaded as a zip and expanded into a folder. This limitation is caused by UNIX style line terminators in the text files. The build is not effected but can be difficult examining or modifying the files under Windows. Avoid this issue by cloning the dvs_ce repository to your desktop instead of downloading the zip.

Java SE Development Kit (JDK)

A Java SE Development Kit must be installed on the system where DVS is built.



Note: If you want to run DVS components in Java 7 and Java 8 runtime environments, JDK 7 must be used. If you are currently running your Java applications (Eclipse and Tomcat7) on Java 8, use JDK 8 to build.

1. If not already present, download the JDK from the [Oracle download site](http://www.oracle.com/technetwork/java/javase/downloads/index.html) (<http://www.oracle.com/technetwork/java/javase/downloads/index.html>). Ensure that you select a Java Platform (JDK) installer most suitable for your host platform (x86 or x64).
2. Follow Oracles instructions for installing the JDK.
3. Set the environment variable **JAVA_HOME** to the folder containing the newly installed Java SDK (for example, C:\Program Files\Java\jdk1.7.0_79).
4. Add %JAVA_HOME%\bin to your user PATH (for example set PATH=%JAVA_HOME%\bin;%PATH%).
5. Open a new console (cmd.exe) window and verify that JAVA_HOME is set and the JDK is in your search path.
6. Run **java -version** and examine the output to make you are seeing the expected version of Java.

Apache Ant

Any current version of Apache Ant is expected to work. Version 1.9.4 was used in testing this procedure. The Apache documentation, [Installing Apache Ant](http://ant.apache.org/manual/install.html) (<http://ant.apache.org/manual/install.html>) can be used as a reference when installing Ant.

Follow these steps:

1. Download Ant from the [Apache download site](http://ant.apache.org/bindownload.cgi) (<http://ant.apache.org/bindownload.cgi>), selecting the .zip of the Ant distribution.



Note: You can elect to use the tar distribution. Long file names in the tar archive can require gnu tar to perform the extraction.

2. Unzip the Ant distribution into your tools folder (that is, C:/tools). A folder reflecting the version of Ant chosen (for example, **apache-ant-1.9.4** is created under your tools folder).
3. Set environment variable ANT_HOME to the Ant directory path. (for example, set **ANT_HOME=c:\tools\apache-ant-1.9.4**).
4. Add %ANT_HOME%\bin to your user PATH (for example, set **PATH=%PATH%;%ANT_HOME%\bin**).
5. Open a new console (cmd.exe) window to verify that ANT_HOME is set and running **ant -version** returns the expected version of Ant.

Install Required Ant Extensions

The download for DVS includes the Ant build target to deploy the required extensions to Ant. These extensions include Ivy (dependency management), ant-contrib (if, else, and so on) & jacoco (code coverage reporting).

1. Download DVS into a folder.
2. Make the folder where the download has been extracted the current folder.
For example, if the DVS download was extracted into the **C:\proj\dvs_ce** cd /D **C:\proj\dvs_ce**
3. Run Ant specifying setup.xml as the build file.
ant -f setup.xml
The script takes a few seconds to run.
4. Verify that the output is similar to the following output:

```
_init:
[get] Getting: http://search.maven.org/remotecontent?filepath=ant-contrib/ant-
contrib/1.0b3/ant-contrib-1.0b3.jar
[get] To: C:\tools\apache-ant-1.9.4\lib\ant-contrib-1.0b3.jar
[get] http://search.maven.org/remotecontent?filepath=ant-contrib/ant-contrib/1.
0b3/ant-contrib-1.0b3.jar moved to https://repo1.maven.org/maven2/ant-contrib/ant-
contrib/1.0b3/ant-contrib-1.0b3.jar
[get] Getting: http://search.maven.org/remotecontent?filepath=org/apache/ivy/
/ivy/2.4.0/ivy-2.4.0.jar
[get] To: C:\tools\apache-ant-1.9.4\lib\ivy-2.4.0.jar
[get] http://search.maven.org/remotecontent?filepath=org/apache/ivy/ivy/2.4.0
/ivy-2.4.0.jar moved to https://repo1.maven.org/maven2/org/apache/ivy/ivy/2.4.0
/iv
y-2.4.0.jar

install-antlibs:
[ivy:resolve] :: Apache Ivy 2.4.0 - 20141213170938 :: http://ant.apache.org/ivy/
::
[ivy:resolve] :: loading settings :: file = C:\proj\dvs_ce\build-
essentials\ivysettings.xml
```

```
[ivy:resolve] :: resolving dependencies :: com.ca (http://com.ca).dvs#dvs.build;
working@artal03-m
[ivy:resolve] confs: [build]
[ivy:resolve] found ant-contrib#ant-contrib;1.0b3 in mvnrepository
[ivy:resolve] found org.jacoco#org.jacoco.ant;0.7.4.201502262128 in
mvnrepository
[ivy:resolve] found org.jacoco#org.jacoco.agent;0.7.4.201502262128 in
mvnrepository
[ivy:resolve] found org.jacoco#org.jacoco.report;0.7.4.201502262128 in
mvnrepository
[ivy:resolve] found org.jacoco#org.jacoco.core;0.7.4.201502262128 in
mvnrepository
[ivy:resolve] :: resolution report :: resolve 218ms :: artifacts dl 16ms
-----
| | modules || artifacts |
| conf | number| search|dwnlded|evicted|| number|dwnlded|
-----
| build | 5 | 0 | 0 | 0 || 5 | 0 |
-----
[ivy:retrieve] :: retrieving :: com.ca.dvs#dvs.build
[ivy:retrieve] confs: [build]
[ivy:retrieve] 5 artifacts copied, 0 already retrieved (778kB/31ms)

all:
BUILD SUCCESSFUL
Total time: 6 seconds
```

Install Apache Maven

1. Download Maven from Apache download [site \(http://maven.apache.org/download.cgi\)](http://maven.apache.org/download.cgi). Any current version of Apache Maven works. Version 3.3.1 was used in testing this procedure.
2. Follow the Apache provided [instructions \(http://maven.apache.org/download.html#Installation\)](http://maven.apache.org/download.html#Installation)for installing Maven appropriate for your platform.

Build the DVS Components

1. Open a command window.
2. Set the current directory to your project root folder. For example, if the DVS download was extracted into C:\proj\dvs_c cd /D C:\proj\dvs_c
3. Execute Ant.

ant

When the build completes, the content in your console window displays similar to the following screen:

```

C:\Administrator: Command Prompt
publish:
all:
[ivy:retrieve] :: loading settings :: file = C:\proj\dvs_ce\build-essentials\ivysettings.xml
[ivy:retrieve] :: resolving dependencies :: com.ca.dvs#dvs.build;working@artal03-U
[ivy:retrieve]   confs: [install]
[ivy:retrieve]   found com.ca.dvs#dvs.app.dvs_servlet;1.0.99999-DEU-SNAPSHOT in local-ivy
[ivy:retrieve]   found com.ca.dvs#dvs.lisa-plugins;1.0.99999-DEU-SNAPSHOT in local-ivy
[ivy:retrieve] :: resolution report :: resolve 47ms :: artifacts dl 0ms
+-----+-----+-----+-----+
|       |       modules      |       artifacts    | | | | | | |
|       |       | number | search|dwnlded|evicted| number|dwnlded|
|       |       | install|  2   |  2   |     0  |     0  |     3  |     0  |
+-----+-----+-----+-----+
[ivy:retrieve] :: retrieving :: com.ca.dvs#dvs.build
[ivy:retrieve]   confs: [install]
[ivy:retrieve] 2 artifacts copied, 0 already retrieved (29941kB/93ms)
BUILD SUCCESSFUL
Total time: 3 minutes 34 seconds
C:\proj\dvs_ce>

```

Screen capture of command prompt screen

The initial build can take 15 minutes or more depending on the capability of your system and network latency. Subsequent builds are faster. The subsequent build takes advantage of local copies of third-party jars loading into an Ivy cache during the first build.

The following output build artifacts are located in the build directory.

- **dvs.app.dvs_servlet** contains **dvs.app.dvs_servlet.war** which can be deployed under Tomcat to expose DVS features through a REST API.
- **dvs.lisa-plugins** contains **dvs.lisa-plugins.zip**, which is the site assembly for installing the Eclipse plug-in in archive form.
- **dvs.lisa-extensions.odata** contains **dvs.lisa-extensions.odata.zip** which contains the OData DVS extension. The OData DVS extension must be deployed into your VSE to support the DVS OData virtualizations.

Next Steps

See [Installing DVS \(see page 1908\)](#) for instructions about how to deploy the newly created build artifacts.

Installing Dynamic Virtual Service

DVS software contains two separate components and has a dependency on a third component. The third component is an extension that is named OData Dynamic Virtualization Extension.

The two components that are part of this project are:

- **ODME Assistant - a n Eclipse plug-in**

The ODME Assistant is used to create MAR files containing a DevTest project.

The project implements an OData DVS that is based on a definition you provide . ODME Assistant is not required to be deployed on the same system as the CA Virtual Service Environment (VSE) that the OData Dynamic Virtualization is deployed. DevTest is not required to be installed .

- **A war file**

War file which can be deployed into a Web Server instance such as Tomcat. The war supports a REST API which allows direct creation and deployment of an OData Virtual Service from a RAML. The war can reside on a separate system but must be configured with connection information to a VSE. Using this component is optional.

The third dependent component contains jars that are provided by CA Technologies with other vendors. The jars must be deployed into every DevTest VSE that the OData Dynamic Virtualization is using.

Before installing these components for the first time, they must be built. See Building Dynamic Virtual Services ([see page 1904](#)) for the required steps to build the components.

Prerequisites for Installing the DVS Eclipse Plug-in

The developer portion of DVS is implemented as an Eclipse plug-in. The install process was tested with Kepler Service Release 2. Any compatible version of the Eclipse IDE can be used. See <https://eclipse.org> (<https://eclipse.org>) for access to Eclipse downloads and instructions for installing Eclipse.



I nstalling Eclipse or the DVS Eclipse plug-in is not required for the following situations:

- Only defining the OData virtual services using the RAML format
- Only deploying the virtual services through the REST API supported by the DVS servlet

Installation

1. From the Eclipse main menu bar, select **Help, Install New Software....**
Install dialog displays.
2. Click **Add** to add a new site and c lick **Archive** from the **Add Repository** dialog.
3. Browse to, or enter the location where the site assembly for the ODME Assistant was built or copied to.
4. Verify that the name **DVS DevTest Eclipse Plugin** populates in the **Name** column and click **OK**.
5. Check the **DVS DevTest Eclipse Plugin** checkbox and click **Next** .
6. If you installed the plug-in previously, select **Update my installation to be compatible with the items being installed** and click **Next** .

7. Verify the **Install Details** and click **Next**.
8. Accept the license and click **Finish**.
Because a trusted certificate was not used to sign the code, a warning displays.
9. Click **OK** to proceed.
10. (Optional) Click **Run in Background** to run the install in the background.
11. When the installation is complete, click **Yes** to restart Eclipse and finish the installation.

Configuring the Eclipse Plug-in

Typically there is no reason to configure the plug-in. One exception is if you have an OData virtual service definition with many entities and many interconnecting navigation properties. The number of transactions added to the LISA Virtual Service Image (VSI) file can increase to thousands even with the default **Maximum depth of the resource path** of 4. These transaction are only added to the VSI for documentation purposes. The OData virtual service extension does not actually use these specific paths. Instead, the extension resolves the resource paths of any depth solely using the meta model that it reads from the EDMtoDB.xml (AEDM file).

If you do not need to review the transactions, you can perform the following options:

- Reduce the **Maximum depth of the resource path**.
- Turn off non-generic transaction generation.

Follow these steps:

1. In Eclipse, select the **Windows, Preferences**, and click **DVS Assistant** in the left panel of the dialog.
2. To have DVS only generate the minimum number of generic transactions, clear the **Enable to Populate Transactions** checkbox.
3. To control the number of transactions added to the VSI file, enter the integer value (1-10) for the **Maximum depth of the resource path**.

Installing the DVS Servlet

This section provides information and instructions to install the DVS servlet.



Note: Installing DVS servlet is optional. Installation is only required if you intend to use the DVS REST API.

Prerequisites

The war file containing the DVS servlet was tested with Apache Tomcat 7.0.57. Other versions of Tomcat or other web servers are supported.

A Java 7 or Java 8 run-time environment must be installed on the system that hosts the web server. To obtain the download and instructions from www.oracle.com. (<http://www.oracle.com>) If Apache Tomcat 7.0.57 is not installed, go to <https://tomcat.apache.org> for the download and instructions.

Installing the DVS Servlet

1. Stop your Tomcat installation by running the shutdown.bat (Windows), or shutdown.sh (Unix) batch files from Tomcat bin folder.
2. Copy the war file that is created by the build process to the webapps folder of your Tomcat instance.
3. (Recommended) Shorten any long war file names. The name of the war file determines the base portion of the URL resource path that is used to access the DVS REST API. In the examples, instructions and testing the war file is renamed to **dvs.war**.
4. Start your Tomcat instance.
5. The batch file startup.bat (Windows) or startup.sh (Unix) is a recommended. Tomcat expands the war file into a folder with the same name as the war file. If your DevTest VSE is installed on the same system as the Tomcat instance, no further configuration is necessary.

Configuring the DVS Servlet

The following configuration items can be changed:

- **vseServerUrl** - URI for the DevTest VSE that hosts the virtual service instances. Default is <http://localhost:1505/api/Dcm/VSEs/VSE>.
- **vseServicePortRange** Range from which the listen port for deployed VS instances are allocated. Multiple ranges are comma-separated. A range can be either a single integer or min-max. The default is 46001-46999.
- **vseServiceReadyWaitSeconds** - Delay between when service is deployed and servlet **pings** the service to make certain it is still up.
- This delay is the major part of the time it takes for the **PUT /raml/vs** call to execute. The delay and a ping guard against reporting success because the VS was successfully deployed but having the VS die during initialization. This result can happen with hand-crafted AEDM files, but has not yet been seen with AEDMs generated from a RAML. (Generating a VS from a RAML can fail if the RAML is malformed or if the RAML contains incomplete or inconsistent data. The failure is detected before deployment). Delay was made configurable because VS initialization failures have always been observed to occur within 7-9 seconds. You can reduce the value if your VSE consistently responds quickly. Increase the value if your VSE is unusually slow. The default is 15.

To change the configuration:

1. Stop your Tomcat installation.

2. Modify the contents of context.xml using any text/XML editor. The context.xml file is located in the webapps/dvs/META-INF folder of the Tomcat instance. Change the value of the **value** attribute of the **Environment** XML element whose **name** attribute value matches the configuration item of interest.
3. Save file.
4. Restart Tomcat.
5. Confirm that servlet has restarted and the new configuration values are in place. Follow the step in [Verifying DVS Servlet Operation \(see page 1908\)](#) in the next section.

Verifying DVS Servlet Operation

The simplest way to verify the operation is to query the DVS servlet for its configuration. Place the URL **Protocol:hostName:Port/dvs/rest/config** into a web browser. You can also use the same URL with a GET in a REST client. **Protocol** is either http or https depending on how your Tomcat is configured, **HostName** is the system where Tomcat is running. **Port** is the Tomcat listen port. The port number is typically 80 or 8080. The name **dvs** is used to the copy of the war file that is deployed into the webapps folder.

If Tomcat and the servlet is running, a similar output follows:

```
{
  "vseServerUrl": "http://MyHostName:1505/api/Dcm/VSEs/VSE",
  "vseServicePortRange": "46001-46999",
  "vseServiceReadyWaitSeconds": 10
}
```

Installing the OData Dynamic Virtualization Services Extension

1. Obtain the **dvs.lisa-extensions.odata-install.zip** file from the **dvs.lisa-extensions.odata** folder. This folder is located under the build folder of your DVS build on the system that is used to build DVS.
2. If the OData Dynamic Virtualization Extensions were previously deployed, back up the files and folders in your DevTest **hotDeploy** folder listed in step 5.
3. If the OData Dynamic Virtualization Extensions were previously deployed, stop any running DevTest services on the system hosting your VSE before deploying the new jar files. When these instructions were written, there was an open defect in DevTest where the code managing the hotDeploy jars was not releasing dependent jars. If the jars that are to be replaced are still in use, you will not be able to copy over them.
4. Extract the full contents of the zip archive to the **hotDeploy** folder of the DevTest instance that the VSE hosts the OData DVS. For a Windows system, this is typically **C:\Program Files\CA\Lisa\hotDeploy**.
5. Confirm that the target **hotDeploy** folder now includes the following file and folder:
 - dvs.lisa-extensions.odata.jar

- Folder dvs.lisa-extensions.odata
6. Restart any DevTest services that are stopped in step 3.

Installing the Example Bookshop DevTest Project

Part of the DVS download is an example DevTest project named Bookshop. The Bookshop example implements an OData version 4 virtual service which illustrates the features supported by OData and DVS.

To explore and use this project to perform the following steps:

1. Copy the examples\Bookshop folder from your DVS download to a location that can be accessed from your DevTest Workstation.
Even if a download is already on the system hosting your DevTest Workstation, make a copy of the Bookshop folder. You have a local backup to revert any changes that are introduced when using the project.
2. From DevTest Workstation, use the **File, Open ,Project, File System** menu path or click the **Open project** icon.
3. From the **Open Project** dialog, browse to the **Bookshop** folder created in step 1 and click **Open**.
A DevTest project displays.
4. To deploy the virtual service to the default VSE for this Workstation, right-click the bookshop.mari file located under MARInfos and select **Deploy/Redeploy to VSE@Default**
Alternately, you can right-click the bookshop.mari file under MARInfos, and select **Build Model Archive....** This allows you to create a Model Archive (MAR file) for this project. This MAR file can then be deployed into any VSE using any of the methods supported by DevTest.
5. Once deployed, verify operation by performing a GET to <https://ServerHostingVSE:8844/odata/v4/Bookshop> using a REST client, or type the URL into a browser.
In either case, a response listing the Entities that are supported by the Bookshop virtual service is returned.

Next Steps

To define your own virtual service see [Designing Your Virtual Service \(see page 1913\)](#).

Designing Your Virtual Service

While the mechanics of generating a DVS can be performed from start to the final deployment within minutes, preparing the design specification for your VS should be done carefully. Designers specify what entities the virtual service supports, how the data looks, and how the entities relate to each other. DVS supports two formats for specifying this information. The first is the Augmented Entity Definition Model (AEDM) file format. The AEDM file format is similar to a standard EDM and has extensions specific to DVS. The second is a REST API Modeling Language (RAML) format, which is not

tied to any product. The AEDM format provides the most power. All supported DVS features can be expressed with an AEDM definition. While RAML has the advantage that it is not specific to DVS, RAML can be used with the DVS REST API. This advantage also allows the specification of the data to populate the virtual service.

Defining a VS in AEDM Format

The AEDM file format is the internal configuration format used to describe the metadata of all the entities exposed by the VS. Even DVS virtual services defined through a RAML file, contain an EDMtoDB.xml file in their data folder. The EDMtoDB.xml file was generated from the information in the RAML. Because the configuration information required is similar to the information exposed through an OData GET \$metadata, it made sense that this configuration information is in an XML format similar to the EDM format used by OData. Since the underlying database should have additional implementation-specific information, the format has been modified to accommodate this. DVS provides no tooling for creating an AEDM, but any XML or text editor can be used.

The EDMtoDB.xml file in the **data** folder of the Bookshop example project was designed to illustrate many of the features. Using a copy of this file as the basis for your own designs is a good approach when getting started. All the example references in this section refer to this file. The **schema.sql** file in the **data\sql** folder should also be examined to help you understand how the AEDM configuration maps to the generated database schema.

Element Names

Many of the elements in an AEDM have a name attribute. The values for these attributes must all conform to the requirements for a JSON identifier. Typically, each element type has a different name space. It is possible for the same name to be used for an entity set, entity type, and a property. For example, see **Personnel**. Property and navigationproperty share the name space within the scope of the same parent element. You cannot have a property and navigationproperty with the same name in the same entity type. You can have the same property name in multiple complextype, and entity type declarations.

Elements of an AEDM

The primary elements in an AEDM are **EntitySet**, **EntityType**, and **Association**. **Property** and **NavigationProperty** elements help define the EntityType. The **schema** element provides information that is relevant to your entire VS. **ComplexType** allows other properties to be grouped for reference and **EnumType** allows definition of a property with values restricted to members of a set of valid values.

Schema

The **schema** element is the parent element for the entire configuration and holds information for the configuration as a whole.

- **Namespace** specifies the name for the scope of the EntityType and EntitySet definitions in a reversed internet domain style similar to Java packages. For example, **com.ca.example.Bookshop**.

- The optional attribute **Alias** is a shorthand version of namespace, and generally is equal to the last segment of the namespace value. In our example is **Bookshop**. When there is a need to qualify a name with a namespace, such as when specifying the value for the type attribute within a property element, or the value for **odata.type** in a POST, the value of Alias is used to refer to items declared in this model. If absent, DVS assumes that the last segment of the namespace value is the alias. As a best practice is to explicitly include this.
- **Version** should be set to a value of the form **n.m**. Where n is an integer major version and m is an integer minor version. This is currently not inspected, but it is strongly suggested that the major version is kept at 1. The designer should increment this whenever there is a significant change in the format of the AEDM itself. While the minor versions be used to track changes to your specific version of the namespace being defined.

EntityType

An **EntityType** defines one of the resources available from the VS. Each **EntityType** references one **EntitySet**.

- The **Name** attribute value will be the name of the collection sets that are the first URL segment after the base path. By convention the name that is used is the plural of the Entity Type Name. If you want to define a collection of books, give the Entity Type the Name of *Book* and the Entity Set the Name of *Books*. However, the name space for Entity Sets and Entity Types are separate and can share the same name. It is often the case where the plural of an Entity Type name is the same as the singular. For example, *Personnel*.
- The **EntityType** attribute is the Name of the EntityType for this EntitySet.

EntitySet

An **EntitySet** element and its nested elements specify the details about the structure of a resource and maps to a TABLE in the underlying database. In most cases, a **EntityType** references a single **EntitySet**. However it does not have to be. Having a **Hidden** EntityType is useful in the specific case of supporting many to many relationships. Where the most efficient representation is to have an internal table that contains the keys for the two Entities being associated.

- The **Name** attribute value is the identifier that is used when referring to the EntityType . By convention it is a non-plural noun describing the content. For example, *Author*.
- The optional **DBTable** attribute allows specification of the name of the TABLE supporting the EntityType. If omitted, the TABLE name is an upper case version of the Name value. See the *Order* EntityType for an example where the table name does not use the default.
- The optional **CustomPropertiesEnabled** attribute if set to the value **true** enables the run-time addition of custom properties to this Entity Type in a way that is not part of the OData standard, but which is part of the feature set for DVS.
- The optional **CustomPropertiesName** attribute allows specification of the name of a special collection property that contains a set of name value pairs for the custom properties if CustomPropertiesEnabled is set to true. This name must not be the same as the name of another Property or Navigation Property for the Entity Type. See the *Order* EntityType for an example of this usage. If CustomPropertiesEnabled is set to true, and CustomPropertiesName is not part of the EntityType definition, then the custom properties appear inline with the statically defined

Properties for the **EntityType**. See the *WorkTeam* **EntityType** for an example of this usage. See **Custom Properties** in OData Dynamic Virtual Service Features for ([see page 1881](#)) more information about custom properties.

An **EntityType** contains one or more nested **property** elements, and zero or more **NavigationProperty** elements.

See the definition for the *Membership* **EntityType** for an example of a hidden Entity Type being used to generate an internal table for use in a many to many navigation.



Note: Management of the content of these tables is handled implicitly by DVS. Do not modify the contents directly.

Property

The **Property** element describes the individual data items that comprise the resource and map to a database COLUMN within the TABLE.

- The mandatory **Name** attribute is simply the name for the data item. It must be unique with the names of the Properties, and **Navigation properties** for this **EntityType**.
- The mandatory **Type** attribute specifies the format of the data. Types can be either base Types, Complex Types, EnumTypes or Collections of base Types, Complex Types, or EnumTypes.

The following supported base types include:

- "Edm.String" a text string.Edm.Boolean" a Boolean value"Edm.Date" a date value. Input format is 'YYYY-MM-DD', for example: '2014-09-15'.
- "Edm.DateTimeOffset" a date-time value. Full Input format is 'YYYY-MM-DD[T]HH:MM:SS.sss[+/-ZZZ]', for example, '2013-10-21T12:10:43.123' format, but *sss* (milliseconds), and *SS.sss* (seconds and milliseconds). The timezone offset can be omitted. Either a **T**, or a space (" ") can be used to separate the date portion from the time portion. All other values are mandatory, but Months, Days, Hours can omit leading zeros.
- "Edm.Decimal" a decimal numeric value (for example, 3.14). When in putting a decimal literal, you can add a **d** suffix to indicate clearly this is a decimal and not a double (for example, 3.14d).
- "Edm.Double" a double (floating-point) numeric value.
- "Edm.Int32" an integer numeric value.

To indicate that a Property is a collection, set the value of Type to **Collection(type)**. Where *type* is the qualified name of the type of the object which composes the collection. For example, the SkillSet Property in Bookshop.Personnel is a collection of strings as has Type="Collection(Edm.String)".

The value of Type can also be the name of a Complex Type or an Enumerated Type that is defined in your schema. Both the Schema and the Name of the Complex Type or EnumType must be specified using the format "*Alias.TypeName*". For example, the ContactInfo Property in Customer has Type="Bookshop.ContactInfo".

- The optional **MaxLength** attribute is used with the **Edm.String** type to specify the maximum length that is allowed for the string.
- The optional **DBColumn** attribute can specify the name of the COLUMN in the database. If absent, COLUMN name is the property name in upper case.
- The optional **Key** attribute indicates which Properties can be specified as a key predicate in a collection navigation. For example, the property that would be compared against the value *x* in the following URL, <http://myserver:9999/basepath/entityset> ([http://myserver:9999/basepath/entityset\(\)x](http://myserver:9999/basepath/entityset()x)). At least one property MUST have the Key attribute set to true. These properties form the PRIMARY KEY for the TABLE in the database.
- The optional **DefaultValue** attribute if present holds the value that is used to set the value of the property if the property is not explicitly given a value in a POST, or PUT request body. The value that is specified is interpreted as a SQL DEFAULT expression. A string literal default must be specified within single quotes. See the ShipVia Property in the Order Entity Type, for an example of a string literal default. This convention also allows you to set the value to an expression, or SQL function, such as Current_Timestamp(). This convention is often useful for automatically generating values. See the Posted Property in the Comment Entity Type for an example.
- The optional **Nullable** attribute if set to **false** indicates that the property is required. If absent or set to **true**, the property can be omitted from POST and PUT request bodies. The value is to null or the value that is specified for DefaultValue.
- The optional **Generated** attribute if set to **once** it indicates that the value of this Property is automatically generated by DVS when the record is first created, and any value that is passed by the user is ignored. If set to **always**, then the value generation happens each time that the record is updated.
- The optional **GenMethod** attribute if set controls how a **generated** property generates its value. Valid values for this attribute are **Init**, or **Sequence**, with default **Init**. If **Init** then the value from the DefaultValue attribute is used. If **Sequence** then the value is created using a unique monotonically increasing integer which starts at 100.



Note: All properties in all entities share the same sequence generator. There is no guarantee that records added to the same entity will be given sequential numbers.

- The optional **GenPattern** attribute if set allows the user apply a format specifier to the sequence number to create a string key. The value of GenPattern is a format string using the same syntax as the Java String.format method. The format value **must** generate a unique value from a unique integer input. In practice this means that the pattern should include a "%d", or "%x" format as part of the specification. GenPattern is only used if the Property is of type String, and GenMethod is "Sequence", it is otherwise ignored. See the Id Property of Personnel for an example of how Generated, GenMethod, and GenPattern work together.

NavigationProperty

The **NavigationProperty** element describes the relationships between EntityTypes and defines the allowable navigation from this EntityType.

- The **Name** attribute is the identifier of the NavigationProperty. It must be unique with the scope of names that are used by any of the properties and navigation properties for this entity type. This also determines the string used for the resource segment in a URL that for this navigation.
- The **Type** attribute specifies the name of the destination EntityType in the navigation.
- The **Relationship** attribute specifies the name of the association that contains the database implementation details for the navigation. In cases where two-way navigation is allowed between two entity types, each EntityType with have a NavigationProperty containing the same relationship value.
- The **Multiplicity** attribute specifies if there can be multiple entity instances of the targeted entity type reachable through this navigation (""), or zero or one instances ("0..1"). For instance the **Books** NavigationProperty in the **Author** EntityType has a "{}" Multiplicity value, because any single **Author** can have written multiple books. On the other hand, the **Inventory** NavigationProperty in the **Book** EntityType has a Multiplicity value of "0..1", because in the model that is used for Bookshop, a **Book** record is associated with only one **Inventory** record.

Association

An **Association** element specifies the implementation details of one (for unidirectional navigations), or two (for bidirectional navigations) Navigation Properties.

- The **Name** attribute must be unique within Association elements, and must match the **Relationship** value for the affiliated navigation properties.
- The **DBLinkTable** attribute MUST be the name of one of the EntityType TABLE names when the navigation is one to one or one to many. If the navigation is many to many, then this value MUST be the TABLE name of the intermediate table which holds the keys of the two Entities that are the end points of the navigation. See the **Worker_Team** Association element definition for an example of this usage. If not many to many, and Properties exist in both tables that can be used to associate records of the two Entity Types, this value MUST be the name of the table on which the database referential integrity constraint is placed. See the **Author_Book** Association element definition for an example of this case. Finally, in an association is for a navigation to and from the same Entity Type (and Table). The value is the table name of the one table involved. See the **Personnel_Supervisor** or **Personnel_Staff** Association element definitions for an example.
- The **DBJoinOn** attribute MUST be a list of one or more JOIN pairs of the form **TABLE1.COLUMN1=TABLE2.COLUMN2**.

The simplest case is where the association is between two different entities and a property exists in both Entity types that can be compared for equality. For example, see the **Author_Book** Association element definition for an example of this case.

If multiple properties are needed to connect two entities that both contain the properties that can be compared for equality, then each JOIN pair is a separate entry into the comma-separated list.

In the case where an association is for a navigation property that points back to the same Entity Type, the JOIN pair looks like **TABLE1.COLUMN1=TABLE1.COLUMN2** and which Property COLUMN has the role of **COLUMN1** and which has the role of **COLUMN2** effects the results that are returned from the navigation property. The **COLUMN1** role should be the Property COLUMN that should be matched for the Parent record. For example, the Navigation Property **Supervisor** in **Personnel** has the semantics of pointing to the **Personnel** record of the supervisor for a given **Personnel** record, while the Navigation Property **Staff** has the opposite meaning. As such each of the navigation properties required a

separate Association element (*Personnel_Supervisor* and *Personnel_Staff*). *Personnel_Supervisor* has a DBJoinOn value of "PERSONNEL.SUPERVISORID=PERSONNEL.ID" because SUPERVISORID has the key for the supervisor's *Personnel* record, so DVS processed from the employees record to the supervisors record. Conversely, *Personnel_Staff* has a DBJoinOn value of "PERSONNEL.ID=PERSONNEL.SUPERVISORID" because the intent is to match all records whose SUPERVISORID matches the supervisor's Personnel record.

Finally in the case of many to many navigation properties, or navigation properties where there do not exist properties in each entity type that can be matched together, the JOIN pair list must contain equijoins from the TABLE for the source entity type to a hidden internal TABLE, and from the hidden internal TABLE to the destination TABLE. See the *Membership* entity type definition, the *Personnel*, and *Teams* navigation properties in the *WorkTeam* and *Personnel Entity Types*, and the *Worker_Team* Association for an example of how this all hangs together.

EnumType

The **EnumType** is a way of declaring a data type that only accepts certain values.

- The **Name** attribute when combined with the namespace Alias is used as the type value for those properties which are using this EnumType. For example, see the status enumtype and the status property in order .
- The optional attribute **UnderlyingType** which can have the value of any integer type or **Edm.String** . If absent the default is **Edm.String**. This controls the type of the database column that is created for any properties declared to use the EnumType.
- In addition, an EnumType declaration **must** include one or more member elements. Each **member** element must include a **name** attribute, which is used as one of the permissible values for Properties declared to use the EnumType, and an optional **value** attribute, whose value is the numeric sort weight that is given to this member. If absent, the sort weight is one higher than the sort weight of the previous member. The sort weight starts at zero where there is no previous explicit declaration.

ComplexType

A **ComplexType** is a means by which related properties can be grouped and referenced as a whole.

- The **name** attribute when combined with the namespace Alias is used as the type value for those properties which are using this ComplexType.
- In addition, an ComplexType declaration **must** include one or more property elements. These property declarations cannot include generated values.

Defining a VS in RAML Format

In general, the RAML format does not provide as fine a degree of control over your VS model. It does have the advantage of allowing the initial data to be included directly in the RAML file, and can also be processed into a virtual service without having to use the eclipse plug-in.

The library.raml file contains an example of a simple model with authors and books which includes initial data.

Metadata declaration

You must define top-level schemas containing type definitions for all EntityTypes and EntitySets. Schemas must conform to the schema format in {+} (<http://json-schema.org/draft-04/schema>)<http://json-schema.org/draft-04/schema+>. In order to identify key fields, each schema definition must have an additional member (primaryKeys), containing a list of primary key property names.

For example:

```
- book: |
  { "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "description": "A single book",
    "properties": {
      "id": { "type": "integer" },
      "name": { "type": "string" },
      "isbn": { "type": "string" },
      "author_id" : { "type": "integer" }
    },
    "required": [ "id", "name", "isbn" ],
    "primaryKeys": [ "id" ]
  }

- books: |
  { "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "description": "a collection of books",
    "properties": {
      "size": { "type": "integer" },
      "books": {
        "type": "array",
        "items": { "$ref": "book" }
      }
    },
    "required": [ "size" ]
  }
```

All resource paths must be defined contiguously, according to standard OData format. Segmented path definitions are **not** handled.

For example:

```
/books:
  get:
    description: Get a list of all of the books in the system
    responses:
      200:
        Body
        :
/books({id}):
  get:
    description: Get a specific book
    responses:
      200:
        Body
        :
```

Do NOT specify like the following information:

```
/books:
  get:
    description: Get a list of all of the books in the system
    responses:
      200:
        Body
```

```

        :
  {id}):
  get:
    description: Get a list of all of the books in the system
    responses:
      200:
        Body
        :

```

Sample Data

Sample data must be inserted as JSON and is only considered when associated with a GET method for a resource. The associated schema name is specified within the application/json body type section.

For example, data that are associated with an entity set should look like the following information:

```

/books:
get:
  description: Get a list of all of the books in the system
  responses:
    200:
      body:
        application/json:
          schema: books
          example: |
            {
              "size": 10,
              "books": [
                { "id": 1, "name": "The Hobbit; or, There and
Back Again", "isbn": "054792822X", "author_id": 1 },
                { "id": 2, "name": "The Fellowship of the
Ring", "isbn": "B007978NPG", "author_id": 1 },
                { "id": 3, "name": "The Two Towers", "isbn":
"B007978PKY", "author_id": 1 },
                { "id": 4, "name": "The Return of the King",
"isbn": "054792819X", "author_id": 1 },
                { "id": 5, "name": "1984", "isbn":
"0451524934", "author_id": 2 },
                { "id": 6, "name": "Animal Farm: A Fairy
Story", "isbn": "B003K16PUU", "author_id": 2 },
                { "id": 7, "name": "The Sun Also Rises",
"isbn": "0743297334", "author_id": 3 },
                { "id": 8, "name": "For Whom the Bell Tolls",
"isbn": "0684803356", "author_id": 3 },
                { "id": 9, "name": "The Old Man and the Sea",
"isbn": "B000FC0SH8", "author_id": 3 },
                { "id": 10, "name": "A Farewell To Arms",
"isbn": "0684801469", "author_id": 3 }
              ]
            }

```

Sample data that are associated with a single entity should look like the following information:

```

/books({id}):
uriParameters:
  id:
    description: Book ID
    type: integer
get:
  description: Get a book
  responses:
    200:
      body:
        application/json:
          schema: book
          example: |
            {
              "id": 1, "name": "The Hobbit; or, There and Back

```

```
Again", "isbn": "054792822X", "author_id": 1
    }
```

Deploying Your Virtual Service

This page provides information about deploying your virtual service.

Prerequisites for Deploying When Using the DVS Assistant (Eclipse Plug-in) and DevTest Solutions

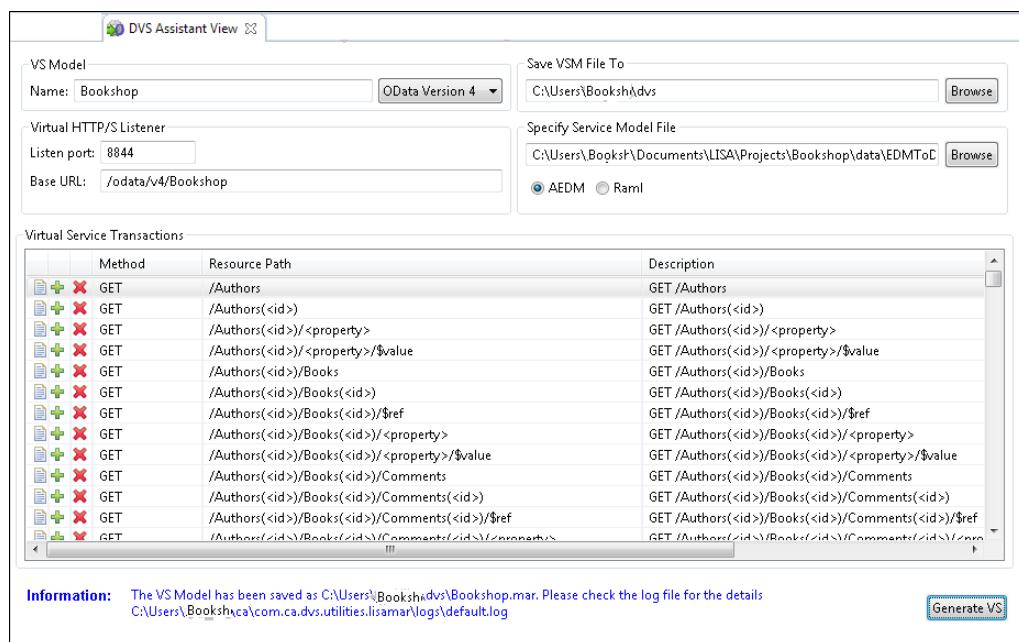
1. DVS Assistant Eclipse plug-in is installed.
2. OData DVS is deployed into the hotDeploy folder of your target VSE.
3. You have access to or have created an AEDM or RAML configuration file.
The configuration file describes the entities and navigation properties that your VS uses. See [Designing Your Virtual Service \(see page 1913\)](#) for more information.

If any of these prerequisites are not met, see [Installing Dynamic Virtual Service \(see page 1908\)](#) for more information about installing DVS.

Creating the Virtual Service MAR File Through Eclipse

1. From the main Eclipse menu, select **DVS Assistant, DVS Assistant View**.

The **DVS Assistant View** window displays:



2. Enter the following information in the fields:

- **VS Model:** The name of the MAR file that is created and the project contain in the zip file.

- **OData Version selector:** Select either OData Version 4 or OData Version 3.
- **Save VSM File To:** The folder where the MAR file is created.



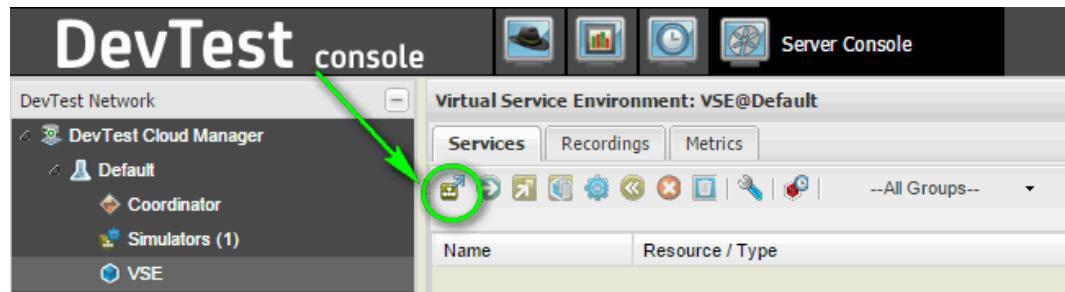
Note : If you extract the contents of your MAR file to examine it through the DevTest Workstation, do **not** extract it into the same folder where the MAR resides. The DVS Assistant uses this location to build a temporary image of the MAR file which it cleans up afterwards. If your project is located there, your project is removed as part of the clean-up.

- **Listen port:** Integer port value. Specify a value that is not used on the systems where your VS is deployed.
 - **Base URL:** Base part of the URL for your VS.
 - **Specify Service Model File:** Full file specification for the configuration file which describes the VSE. The AEDM or RAML buttons display which files are shown when browsing to select the Service Model file. The selected buttons also determine how that file is processed.
3. If the plug-in has been configured to populate transactions, you preview your valid resource paths by right-clicking any of the column headers of the **Virtual Service Transactions** table, and selecting **Populate All Transactions**.
 4. Click **Generate VS** to create your MAR file.
If successful, a message displays at the bottom of the screen with information about the MAR file.
 5. If a warning or an error status displays, do the following:
 - a. Review the log.
 - b. Correct the problem in your configuration definition and repeat step 4.

Deploying the MAR File

Deploy the newly created MAR file into any VSE with the OData DVS extension that is installed the hotDeploy folder. Use any of the following methods supported by DevTest Solutions:

- Uploading from the DevTest Server Console by clicking **Deploy/Redeploy a virtual service to the environment** :



Screen capture of the Server Console

- Using the `deployMar` method of the Invoke 2.0 REST API for DevTest Solutions. See [CA Support Documentation](https://support.ca.com/cadocs/0/CA%20DevTest%20Solutions%208%200-ENU/Bookshelf_Files/Invoke/index.html) (https://support.ca.com/cadocs/0/CA%20DevTest%20Solutions%208%200-ENU/Bookshelf_Files/Invoke/index.html) for the log in credentials that are required.

Once deployed, the virtual service is managed using any of the facilities that are provided by DevTest. See DevTest Solutions documentation for more information.

Prerequisites for Deploying When Using the DVS Servlet

- DVS Servlet into a Tomcat instance is installed.
- OData Dynamic VS is deployed into the **hotDeploy** folder of the target VSE.
- You can access and have created a RAML configuration file describing the entities and navigation properties your VS uses.
See [Designing Your Virtual Service \(see page 1913\)](#) for more information.

If any of these prerequisites are not met, see [Installing Dynamic Virtual Service \(see page 1908\)](#) for more information.

Creating and Deploying

The entire operation is done using on REST call to the **Transform RAML to OData Virtual Service** service. See [OData Dynamic Virtual Service Features \(see page 1881\)](#) for parameter details.

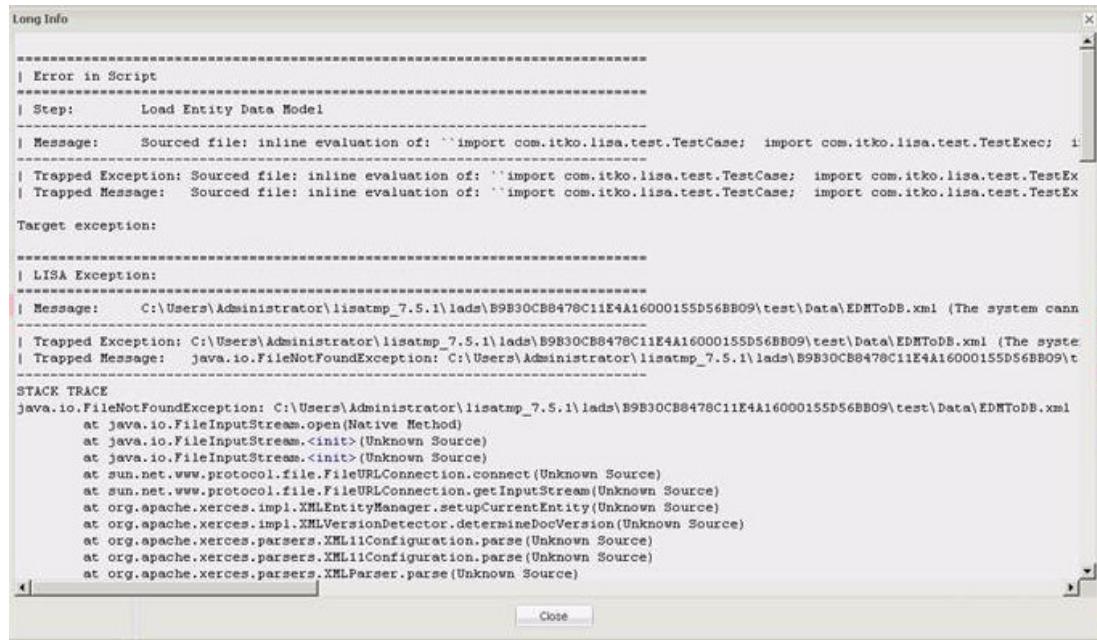
After, you can download the MAR file from your VSE using the DevTest Server Console. Then use the contents of the MAR file as a DevTest project as described in the following section.

Working with the MAR as a DevTestProject

- Using an archive tool such as zip extract the contents of the MAR file into any folder.
- From the DevTest Workstation, open the project.
- You can perform any operations that you could typically perform on a project.
- To redeploy the modified project, right-click the **MAR Info file** and select **Deploy/Redeploy to VSE@Default....** Or select **Build Model Archive...**, then deploy the new MAR file using any of the methods described under **Deploying the MAR File** section.

Note: Do NOT deploy the VSM file. Deploying the VSM file will only deploy the VSM and not the supporting data file. As a result of deploying the VSM file, the virtual service fails. Typically you see an exception similar to the one below, but other errors are possible.

The following graphic displays an exception:



```

Long Info

-----| Error in Script
-----| Step: Load Entity Data Model
-----| Message: Sourced file: inline evaluation of: ``import com.itko.lisa.test.TestCase; import com.itko.lisa.test.TestExec; i
-----| Trapped Exception: Sourced file: inline evaluation of: ``import com.itko.lisa.test.TestCase; import com.itko.lisa.test.TestEx
-----| Trapped Message: Sourced file: inline evaluation of: ``import com.itko.lisa.test.TestCase; import com.itko.lisa.test.TestEx

Target exception:

-----| LISA Exception:
-----| Message: C:\Users\Administrator\lisatmp_7.5.1\lads\B9B30CB8478C11E4A16000155D56BB09\test\Data\EDMToDB.xml (The system cann
-----| Trapped Exception: C:\Users\Administrator\lisatmp_7.5.1\lads\B9B30CB8478C11E4A16000155D56BB09\test\Data\EDMToDB.xml (The syste
-----| Trapped Message: java.io.FileNotFoundException: C:\Users\Administrator\lisatmp_7.5.1\lads\B9B30CB8478C11E4A16000155D56BB09\t

STACK TRACE
java.io.FileNotFoundException: C:\Users\Administrator\lisatmp_7.5.1\lads\B9B30CB8478C11E4A16000155D56BB09\test\Data\EDMToDB.xml
    at java.io.FileInputStream.open(Native Method)
    at java.io.FileInputStream.<init>(Unknown Source)
    at java.io.FileInputStream.<init>(Unknown Source)
    at sun.net.www.protocol.file.FileURLConnection.connect(Unknown Source)
    at sun.net.www.protocol.file.FileURLConnection.getInputStream(Unknown Source)
    at org.apache.xerces.impl.XMLEntityManager.setupCurrentEntity(Unknown Source)
    at org.apache.xerces.impl.XMLVersionDetector.determineDocVersion(Unknown Source)
    at org.apache.xerces.parsers.XML11Configuration.parse(Unknown Source)
    at org.apache.xerces.parsers.XML11Configuration.parse(Unknown Source)
    at org.apache.xerces.parsers.XMLParser.parse(Unknown Source)

```

Removing a Virtual Service

To completely and cleanly remove an OData VS from your environment, the following two steps are required:

1. Remove the virtual service from DevTest Server Console by right-clicking the VS and selecting **Remove from the environment**.
2. Remove the SQL script folder *LISA_HOME\dvs.dataproject_name* for example, *C:\Program Files\CA\Lisa\dvs.dataproject_name*.
Because DevTest is unaware of these additional files. The folder and its contents are not cleaned up as part of the first step. The files **may** be reused if a new service with the same name is deployed.

Next Steps

You can now access the virtual service using any RESTFUL client through URL **http://server_hosting_LISA_VSE:interger_port_specified/base_path_specified**.
See [OData Virtual Service Features \(see page 1881\)](#) for a description of the supported OData feature.
See [Managing Your Virtual Service Data \(see page 1926\)](#) for more information about how to manage data.

Managing Your Virtual Service Data

While a DVS is running, entity data is held in an in-memory H2 SQL Database instance. This process allows quick response and simple deployment because there is no need to have access to or an installation of RDBMS.

The database is resident in memory only. Each time DVS starts, data is re-initialized from the SQL script **current.sql** in the DVS data folder for the virtual service. The feature is beneficial because it allows reset to a known state. If modifications to the data have been made and retained, each time a VS is restarted, **all** updates are reset upon restart. There is a way to retain all the updates and inserts. Upon the shutdown, contents of the database are written out to **new.sql** in the DVS data folder for the virtual service. You copy this file over **current.sql** before the virtual service is restarted. All the updates made up to the previous shutdown are preserved. As a best practice, use the Runtime Management of Virtualization Data features. These features create named snapshots of the data and manage your data through this REST API.

The DVS data folder is under LISA_HOME (default value **C:\Program Files\CA\Lisa** on windows) with the rest of the folder path that is taken from the DDME_DATA_DIR LISA variable in projects.cfg. By default, this is **dvs.data/project_name**. The SQL initialization files for a DVS DevTest virtual service named **Nightsky**, by default is located in the folder **C:\Program Files\CA\Lisa\dvs.data\Nightsky**.



Note: Older projects can have data stored in a folder named casd.data. The folder that is used can be determined by examining the DDME_DATA_DIR LISA variable in projects.cfg.

Runtime Management of Virtualization Data

The DVS Admin API provides a means of servicing internal operations on any service that is based on the Dynamic Virtualization model. Currently, this API only provides for the management of SQL initialization scripts for the Dynamic Virtualization Data Store.

Web Client Requests

Obtain a List of Existing DataStore Checkpoints

Method: GET

Path: /dynvs_admin/Checkpoints

Keys: (none)

Results:

Result	HTTP-Status-Code	HTTP-Status-Text	Content-Type	Body	Description
SUCCESS	200	OK	application/json	List of JSON objects describing the scripts found	Method succeeded

Load a New Data Store From Specified Checkpoint

Method: GET

Path: /dynvs_admin/Checkpoints({id})

Keys: (none)

Params:

N	T	Description	Example
a	y		
m	p		
e	e		
id	St	The name of the script used to initialize a new datastore. The id key may not be specified. The associated value is a quoted string value, indicating the name of an existing DataStore initialization script.	GET /casd_vs_admin/Checkpoints('new.sql')
	ri		
	n		
	g		

Results:

Res	HTTP- ult	HTTP- Status-	Conte nt-	Body	Description
	Code	Status- Text	Type		
SU	200	OK	appli	JSON objects describing	Method succeeded. Existing DataStore is
CC			cation	script used to initialize	replaced by a new one, initialized by the
ESS			/json	datastore	specified script.
FAI	404	Not Found	text	Message describing	Method failed because the specified script
LU			/plain	failure	could not be located.
RE					
FAI	500	Internal Server Error	text	Message describing	Method failed (see body for more
LU			/plain	failure	information).
RE					

Save Active Data Store to Script

Method: PUT

Path: /dynvs_admin/Checkpoints({id})

Keys: (none)

Params:

N	T	Description	Example
a	y		
m	p		
e	e		
id	St	The name of the script that is used to initialize a new datastore. The id key may not be specified. The associated value is a quoted string value, indicating the name of an existing DataStore initialization script.	PUT /casd_vs_admin/Checkpoints('new.sql')
	ri		
	n		
	g		

Results:

Res	HTTP- ult	HTTP- Status- Code	Content Body -Text	Description
SUC	204	No Content	(none)	Method succeeded. Existing DataStore was saved to the specified initialization script
FAIL	500	Internal Server Error	text /plain	Message describing failure

Delete a Data Store Initialization Script

Method: DELETE
 Path: /dynvs_admin/Checkpoints({id})
 Keys: (none)

Params:

N	T	Description	Example
a	y		
m	p		
e	e		
id	S	The name of the script that is used to initialize a new datastore. The id key can be specified or cannot be specified. The associated value is a quoted string value, indicating the name of an existing DataStore initialization script.	DELETE /casd_vs_admin/Checkpoints('new.sql')
	g		

Results:

Res	HTTP-S	tatu	HTTP- Status- Text	Content- Body Type	Description
SUCC	204	No Content	(none)	None	Method succeeded. Specified DataStore initialization script was deleted.
FAIL	403	Forbidden	text /plain	Message describing failure	Method failed because the specified script could not be deleted.
FAIL	404	Not Found	text /plain	Message describing failure	Method failed because the specified script could not be located.

Unsupported Requests

*** Any requests other than the requests documented previously return a HTTP-Status-Code of 405 (Method Not Allowed)**

Backing up Data

To back up DVS data for a DVS DevTest VS, make a copy of the DVS data folder for that service.



Note: If the service is running, any updates that are not yet written to an SQL script are not preserved.

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
- AlivePDF 0.1.5RC
- amoebacode
- Angular-chosen-localytics 1.0.6
- Angular-dashboard-framework 0.8.0
- Angular-dynamic-locale 0.1.27
- Angular-elastic 2.3.5
- Angular-file-upload 1.6.12
- Angular-http-auth 1.2.1
- Angular-jquery-dialog-service 2013-10-25 “fix-remove-dialog-from-dom-on-close”
- Angular-router-extras 0.0.10
- Angular-selection-model 0.7.0
- Angular-translate 2.2.0
- Angular-tree-control 0.2.8
- Angular-typeahead 0.2.0
- Angular-ui-grid 3.0.0
- Angular-ui-layout 0.0.1
- Angular-ui-codemirror 0.1.6
- Angular-ui-router 0.2.10
- Angular-ui-sortable 0.12.8
- Angular-utf8-base64 0.0.5
- Angular-xeditable 0.1.8

- AngularJS 1.2.20
- annotation-detector
- Antlr 2.7.6
- Apache Commons FileUpload 1.3.1
- Apache Wink 1.1.1
- Appium 1.4
- Axiom 1.2.8
- Axis 1.4
- Balloon Tip for Java 1.2.1
- Bean shell (bsh) 2.1.8
- Beanshell v.2.0b4
- BlazeDS 4.0.0.14931
- Bootstrap 3.2.0
- Bouncy Castle 1.47
- Bouncy Castle Crypto Provider for JDK 1.5
- c3po 0.9.2.1
- cb2xml 0.94
- Codemirror 4.3.0
- Commons beanutils 1.7
- Commons Cli 1.0
- Commons Codec 1.3
- Commons Collections 3.2
- Commons DBCP 1.2.2
- Commons Digester 1.8
- Commons Discovery 0.2
- Commons Email 1.3.1
- Commons HttpClient 3.1

- Commons IO 1.1
- Commons Lang 2.3
- Commons Logging 1.2
- Commons Pool 1.3
- Commons Launcher 1.1
- Concurrent Utilities 1.3.4
- CyberNeko HTML Parser 0.9.5
- d3 3.4.8
- d3-tip 0.6.4
- Derby 10.5.3.0
- DiffMatchPatch
- Dk.brics.automation 1.11-8
- dom4j 1.6.1
- Dumbster 1.6
- EclipseLink 2.1.1
- elasticsearch 1.3.5
- Expression Layout 1.0
- EZMorph 1.0.6
- fastclick 1.0.2
- Flexbuilder Pro 3.0
- FlexLib 2.5
- flywaydb 2.2.1
- Font Awesome 4.2.0
- 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 mvp4g 1.2.0
- Google opensans version 8
- Google Web Toolkit 2.3.0
- grails-ziplet .3
- The Grinder - J2EE Performance Testing
- Guava r09
- guice 2.0
- GWT Hashcode Equals 0.1.0
- GWT Server Library SL 1.1
- Hamcrest 1.3
- highlight.js 8.2
- HTMLDecoder 1.6.1
- HttpAsyncClient 4.0.2
- HttpClient 4.3.6
- HttpComponents 4.0.1
- IBM DB2 Driver for JDBC and SQLJ 10.5
- JACOB - Java COM Bridge 1.9.1
- Jakarta Byte Code Engineering Library 5
- Jakarta Regexp 1.2
- Java Platform Standard Edition (Java SE) 6.0
- java tablelayout 1.5
- Javassist 3.18.0-GA
- javax.ejb-api 3.2
- javax.servlet 3.0
- Java Service Wrapper 2.2.9
- JBoss 4.2.3

- JDBC 11.2.0.3
- jdbi 2.51
- JDOM 2.0.6
- Jetty 7.3.1
- JFreeChart 1.0.13
- JGoodies Forms 1.0.7
- Jit 2.0.1
- Jode
- jQuery 2.1.1
- jQuery qTip2 Plugin 2.2.1
- jQuery UI 1.10.4
- JS Beautifier 1.5.5
- JSON 1.0
- JSON Assert 1.2.3
- JSON Diff Patch 0.1.22
- JSON Path 1.2.0
- JSON Smart 1.2
- JSON-lib 2.4
- JSON-schema-validator 2.2.3
- Jsoup 1.8.1
- JSQL Parser 0.9
- JUnit 4.5
- Log4J 1.2.13
- Lunr 0.5.4
- MC4J 1.2.2
- Mockrunner 0.4
- NETBeans IDE 6.5 subset

- Ng-file-upload 1.6.5
- Ng-grid 2.0.13
- Ng-idle 0.3.5
- Ng-table 0.3.2
- Ng-table-resizable-columns alpha
- node.js 0.10.25
- oclazyLoad 0.3.8
- okhttp 2.4.0
- Open SAML - Java 1.1b
- Open SAML 2.2.2.3
- OpenCover 4.5.1403
- OpenNLP 1.5.3
- Oracle Java Runtime Environment (JRE) 1.5.0
- ORO 2.0.6
- p6spy 1.1
- POI 3.5
- pygments 1.6.1
- qdox 1.6
- Quartz 1.6.5
- RAML Java Parser version 0.8
- RAML_Parser 0.8.4
- retrofit 1.9.0
- RSyntaxTextArea 2.0.6
- SAAJ 1.3
- ScintillaNET 2.2
- Scrollable Bar
- Select2 3.4.8

- Selenium 2.45.0
- 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
- SNMP4J 1.11.3
- Solr 4.9
- Spin.js 2.0.1
- Splunk Java SDK
- Spring Framework 4.0.5
- Spring Security 4.0.1
- StringSearch 1.2
- Struts 1.2.9
- Sun JAVA JRE 1.4.2_19
- swfobject 2.1
- swing-layout 1.0.3
- swingx 1.6
- tidy 4aug00
- Trilead SSH2
- UDDI4j 2.0.5
- ui-ace 0.2.3
- UI Bootstrap 0.11.0
- Velocity 1.7
- W3C Schemas
- WADL Java Parser 1.0
- WIFE 7.3
- WSDL4J 1.6.2

- wss4j 1.5.4
- Xalan-J 2.7.1
- Xerces-J 2.9.1
- XML Beans 2.4.0
- XML Schema 1.4.6
- XML Unit 1
- XOM 1.2.10
- XStream 1.3.1
- XULRunner 1.9.0.11
- Zip.js 4c93974