DevTest Solutions

Using CA Application Test

Version 8.0

ca technologies

This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the “Documentation”) is for your informational purposes only and is subject to change or withdrawal by CA at any time.

This Documentation may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Documentation is confidential and proprietary information of CA and may not be disclosed by you or used for any purpose other than as may be permitted in (i) a separate agreement between you and CA governing your use of the CA software to which the Documentation relates; or (ii) a separate confidentiality agreement between you and CA.

Notwithstanding the foregoing, 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.

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: Getting Started with Application Test 11

Registry 11

Start the Registry 12

Create a Named Registry 12

Change the Registry 13

Coordinator Server 13

Create a Coordinator Server 14

Monitor Coordinator Servers 14

Simulator Server 15

Create a Simulator Server 16

Monitor Simulator Servers 17

Command-Line Utilities 18

Chapter 2: Using the DevTest Portal with CA Application Test 19

Open the DevTest Portal 19

Chapter 3: Create an API Test 21

Create a Test by Importing R/R Pairs from a Test 21

Create a Test by Importing R/R Pairs from a File 22

Create a Test Manually 23

Edit an API Test 24

Chapter 4: Monitor Tests 25

Filter Tests and Suites 26

Monitor Tests and Suites 28

Chapter 5: Manage Test Artifacts 35

Chapter 6: Manage API Tests 37

Chapter 7: Manage Tests 39

Chapter 8: Manage Test Suites 41

Chapter 9: Using the Workstation and Console with CA Application Test 43

Workstation and Console Overview 43
- DevTest Workstation 44
- DevTest Console 70

Chapter 10: Building Test Cases 73

- Anatomy of a Test Case 73
- Properties 86
- Configurations 100
- Assets 106
- Filters 114
- Assertions 129
- Data Sets 145
- Companions 155
- Complex Object Editor (COE) 158
- Building Test Steps 188
- Creating Test Cases 199
- Building Subprocesses 209

Chapter 11: Building Documents 215

- Building Staging Documents 215
- Building Audit Documents 237
- Understanding Events 239
- Generating Metrics 243
- Building Test Suites 243

Chapter 12: Working with Model Archives (MARs) 255

- Model Archive (MAR) Overview 256
- Explicit and Implicit MAR Creation 258
- Create MAR Info Files 260

Create Monitor MAR Info Files 262
Edit MAR Info Files 265
Build MARs 266
Deploy to CVS 266
Make Mar Utility 267

Chapter 13: Running Test Cases and Suites 269

Using the Interactive Test Run (ITR) Utility 270
Stage a Quick Test 281
Stage a Test Case 289
Run a Test Suite 297
Run a Selenium Integration Test Case 304
Assumption of Load Testing 308
Test Runner 310
LISA Invoke 315
The REST API 321
Using the HP ALM - Quality Center Plug-in 322
HTTP and SSL Debug Viewer 329

Chapter 14: Cloud DevTest Labs 333

Labs and Lab Members 334
Virtual Lab Manager (VLM) 335
DevTest Cloud Manager 336
Configure DCM Properties 337
Configure vCloud Director 342
Dynamic Expansion of Test Labs 343
List the Available Labs 344
Start a Lab 346
Obtain Information About a Lab 349
Deploy a MAR to a Lab 350
Stop a Lab 351

Chapter 15: Continuous Validation Service (CVS) 353

Open the CVS Dashboard 354
CVS Dashboard Overview 355
Deploy a Monitor to CVS 363
Run a Monitor Immediately 365
View Test Details 365
Email Notification Settings 366
CVS Manager 367

Chapter 16: Reports 369

Report Generator Types....369

Chapter 17: Load Test Report Generator 371

Open the Reporting Portal....372
Reporting Portal Layout....372
Filtering Reports....375
Viewing Reports....376
Exporting Reports....406
Changing Reporting Databases....407
Troubleshooting Reporting Performance....408

Chapter 18: Recorders and Test Generators 409

Generate a Web Service....409
Record a Website....411
Record a Mobile Test Case....415

Chapter 19: Mobile Testing 419

Getting Started with Mobile Testing....419
How to Create and Replay a Mobile Test Case....422
Remote Testing....422
Web Browser Testing....425
The Mobile Lab....426
Automated Tests by Voyager....428
Troubleshooting Mobile Test Cases....431

Chapter 20: Advanced Features 433

Using BeanShell in DevTest....433
Class Loader Sandbox Example....438
In-Container Testing (ICT)....439
Generating DDLs....444

Appendix A: Appendix A - DevTest Property File (lisa.properties) 445

Comma-Separated List of Paths for Javadoc and Source Code....446
System Properties....447
Server Properties....447
OS X Properties....448
Update Notifications....448

Basic Defaults 449
HTTP Header Keys Properties 451
HTTP Field Editor Properties 452
Test Case Execution Parameters 453
TestEvent Handling Customizations 454
Test Manager/Editor Properties 455
J2EE Server Parameters 456
Native Browser Information to Use for Internal Rendering 458
Test Manager/Monitor Properties 458
Built-In String-Generator Patterns 458
JMX Information 459
Test Manager/ITR Properties 461
External Command Shells 462
Testing Parameters 462
Properties for Use by StdSchedulerFactory to Create a Quartz Scheduler Instance 463
VSE Properties 466
Network Port Properties 476
CA Continuous Application Insight Properties 478
Database Properties 483
Mainframe Properties 485
Selenium Integration Properties 487
VSEasy Properties 488

Appendix B: Custom Property Files 489

Local Properties File 490
- License Properties 490
- VSE Properties 491
- Enterprise Dashboard Properties 496
- Interconnectivity Properties 499
- License Properties if Using an HTTP Proxy Server 500
- SDK Properties 501
- SSL Properties 502
- HTTP Authorization Properties 503
- Kerberos Authentication Properties 503
- HTTP Proxy Server Properties 504
- XML Serialization Settings 505
- Workstation Management 506
- IP Spoofing 507
- Quick Start Recent Items Properties 507
- lisa-invoke Properties 507
- Server Host Name Properties 508

DevTest to DevTest Communication Encryption Properties 509
DevTest Agent and CAI Properties 510
IBM WebSphere MQ Properties 510
JMS Properties 511
Miscellaneous Properties 512
User Session Lifetime Properties 514
Site Properties File 515
DCM Settings 518
logging.properties 519

Glossary 525

Chapter 1: Getting Started with Application Test

This section contains the following topics:

Registry (see page 11)
Coordinator Server (see page 13)
Simulator Server (see page 15)
Command-Line Utilities (see page 18)

Registry

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, 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 334), 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

Coordinator servers receive the test run information as documents, and coordinate the tests that run on one or more simulator servers (see page 15). 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 334) 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 in *Administering.*

Monitor Coordinator Servers

If DevTest Workstation is attached to the associated registry, then a running coordinator server appears in the Registry Monitor.

To monitor coordinator servers from the Registry Monitor:

1. Click the Toggle Registry icon in DevTest Workstation.
2. Click the Coord Servers tab.

The coordinator servers also appear in the network graph of the Server Console.

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

Simulator servers run the tests under the supervision of the coordinator server (see page 13).

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 334) 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 in *Administering.*

For information about running the simulator as a service, see Running Server Components as Services in *Administering.*

Monitor Simulator Servers

If DevTest Workstation is attached to the associated registry, then a running coordinator server appears in the Registry Monitor.

To monitor simulator servers from the Registry Monitor:

1. Click the Toggle Registry icon in DevTest Workstation.
2. Click the Simulators tab.

Simulator servers also appear in the network graph of the Server Console.

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 in *Using CA Continuous Application Insight*.

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 367) in *Using CA Application Test.*

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 267) in *Using CA Application Test.*

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 in *Using CA Service Virtualization.*

Service Manager

Service Manager is used to verify the status of, reset, or stop a running server process. For more information, see Service Manager in *Administering.*

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 310) in *Using CA Application Test.*

VSE Manager

VSE Manager is used for managing virtual service environments. For more information, see VSE Manager Commands in *Using CA Service Virtualization.*

Chapter 2: Using the DevTest Portal with CA Application Test

This section contains the following topics:

Open the DevTest Portal (see page 19)
Create an API Test (see page 21)
Monitor Tests (see page 25)
Manage Test Artifacts (see page 35)

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 in *Installing*.

Follow these steps:

1. Complete one of the following actions:
   - Enter **http://localhost:1507/devtest** in a web browser. If the registry is on a remote computer, replace **localhost** with the name or IP address of the computer.
   - Select View, DevTest Portal from DevTest Workstation.
2. Enter your user name and password.
3. Click Log in.

Chapter 3: Create an API Test

The Create 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 Test 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.

This section contains the following topics:

Create a Test by Importing R/R Pairs from a Test (see page 21)
Create a Test by Importing R/R Pairs from a File (see page 22)
Create a Test Manually (see page 23)
Edit an API Test (see page 24)

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

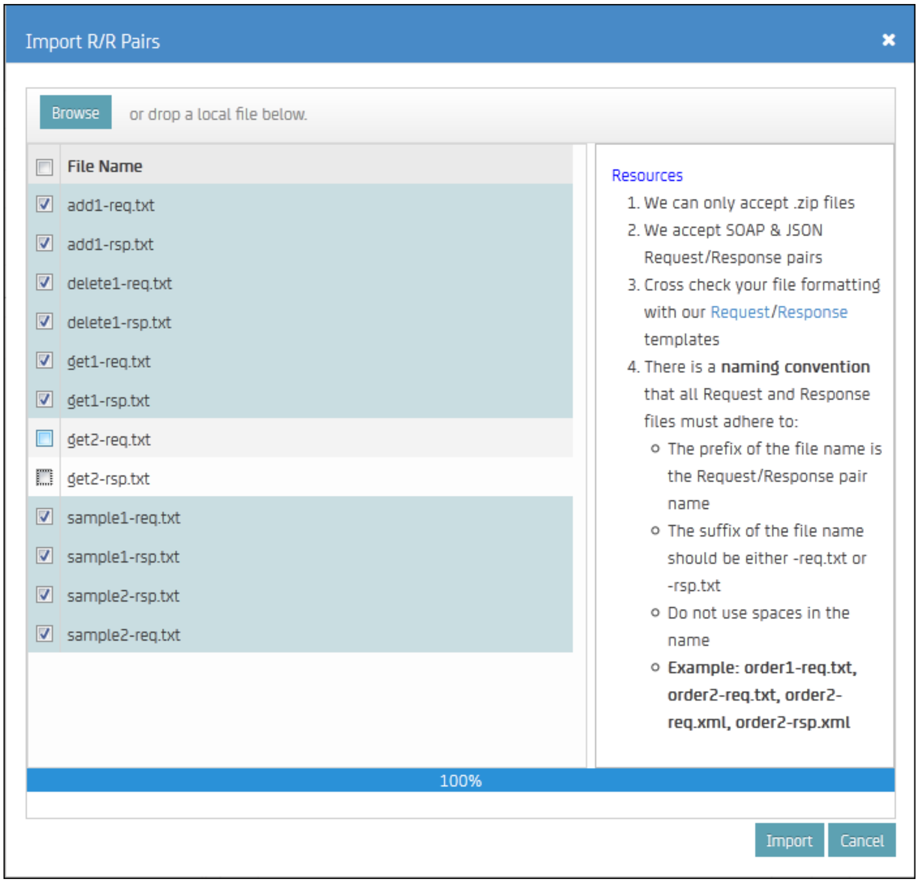
You can create a test by loading request/response pairs from a zip file into the Test Editor.

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.



or drop a local file below.

□ File Name
☑ add1-req.txt
☑ add1-rsp.txt
☑ delete1-req.txt
☑ delete1-rsp.txt
☑ get1-req.txt
☑ get1-rsp.txt
□ get2-req.txt
□ get2-rsp.txt
☑ sample1-req.txt
☑ sample1-rsp.txt
☑ sample2-rsp.txt
☑ sample2-req.txt

Resources

1. We can only accept .zip files
2. We accept SOAP & JSON Request/Response pairs
3. Cross check your file formatting with our Request/Response templates
4. There is a **naming convention** that all Request and Response files must adhere to:
   - The prefix of the file name is the Request/Response pair name
   - The suffix of the file name should be either -req.txt or -rsp.txt
   - Do not use spaces in the name
   - **Example: order1-req.txt, order2-req.txt, order2-req.xml, order2-rsp.xml**

   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).

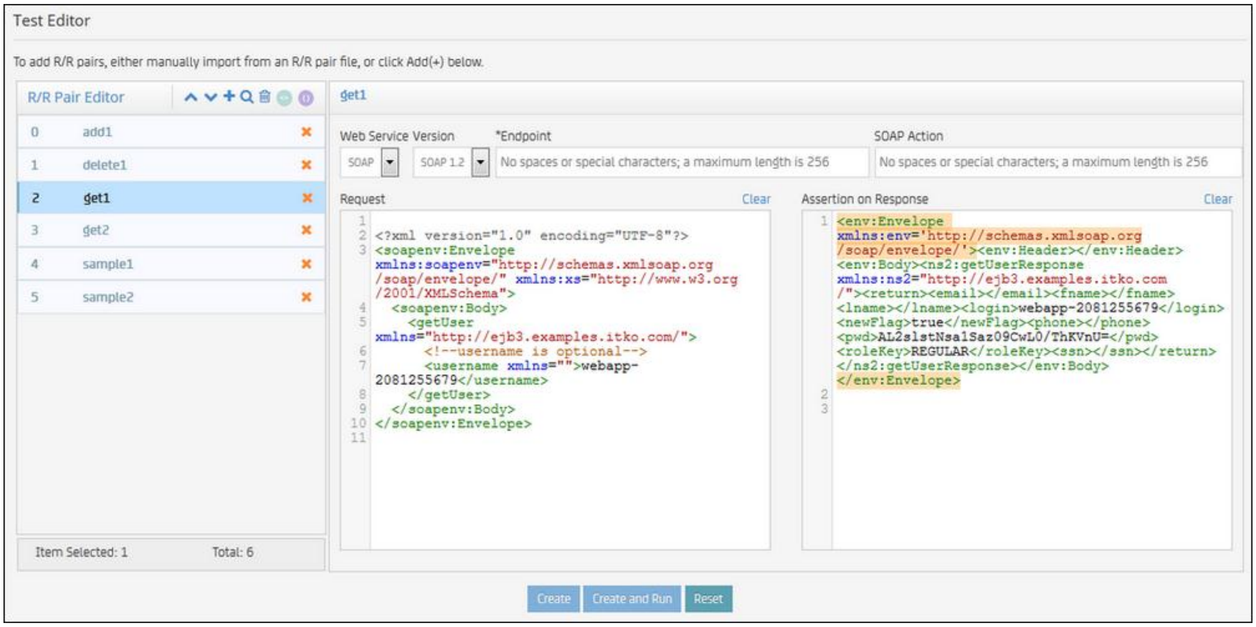
   - For SOAP, enter a Version (SOAP 1.2 or SOAP 1.1), an Endpoint (the specific endpoint for this request) and a SOAP Action (optional for SOAP 1.2).

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.



Test Editor

To add R/R pairs, either manually import from an R/R pair file, or click Add(+) below.

R/R Pair Editor

0	add1	✖
1	delete1	✖
2	get1	✖
3	get2	✖
4	sample1	✖
5	sample2	✖

Item Selected: 1 Total: 6

get1

Web Service Version: SOAP, SOAP 1.2

*Endpoint: No spaces or special characters; a maximum length is 256

SOAP Action: No spaces or special characters; a maximum length is 256

Request

```

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org
/soap/envelope/" xmlns:xs="http://www.w3.org
/2001/XMLSchema">
  <soapenv:Body>
    <getUser
xmlns="http://ejb3.examples.itko.com/">
      <!--username is optional-->
      <username xmlns="">webapp-
2081255679</username>
    </getUser>
  </soapenv:Body>
</soapenv:Envelope>

```

Assertion on Response

```
<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
/"><return><email></email><fname></fname>
<lname></lname><login>webapp-2081255679</login>
<newFlag>true</newFlag><phone></phone>
<pwd>AL2slstNsa1Saz09CwL0/ThKVnU=</pwd>
<roleKey>REGULAR</roleKey><ssn></ssn></return>
</ns2:getUserResponse></env:Body>
</env:Envelope>

```

To change the name of a test, double-click the name, enter a new name, and click Enter.

To create an API test, click the Create button. 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 the Create and Run button. 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.

Chapter 4: Monitor Tests

The Monitor Test window lets you see API tests, test cases, and test suites that have been run on CA Application Test.

This section contains the following topics:

Filter Tests and Suites (see page 26)
Monitor Tests and Suites (see page 28)

Filter Tests and Suites

To filter the test cases and test suites that are returned from the database, click Search 🔍.

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.

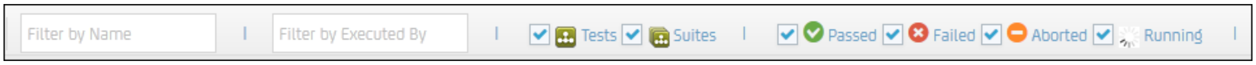
Executed By

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

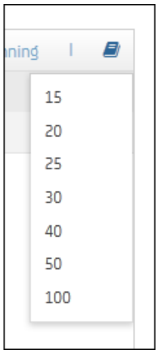
To show only the results from a specific test case or suite, enter a name or partial name of the test or suite.

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.

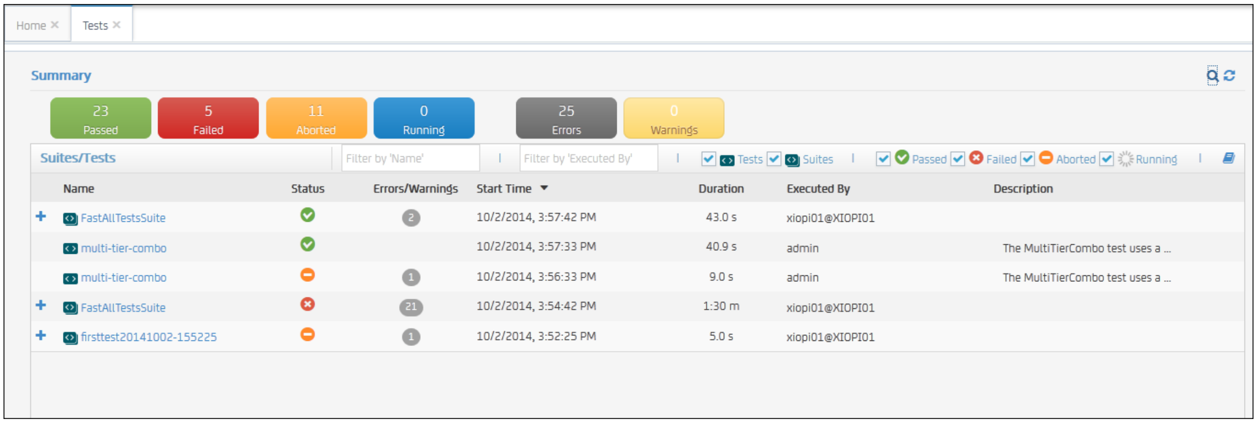
To select how many rows to display, click Page Size.


nning
15
20
25
30
40
50
100

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.



	Name	Status	Errors/Warnings	Start Time ▼	Duration	Executed By	Description
+	FastAllTestsSuite	✓	2	10/2/2014, 3:57:42 PM	43.0 s	xiopi01@XIOPI01	
	multi-tier-combo	✓		10/2/2014, 3:57:33 PM	40.9 s	admin	The MultiTierCombo test uses a ...
	multi-tier-combo	⊖	1	10/2/2014, 3:56:33 PM	9.0 s	admin	The MultiTierCombo test uses a ...
+	FastAllTestsSuite	✗	21	10/2/2014, 3:54:42 PM	1:30 m	xiopi01@XIOPI01	
+	firsttest20141002-155225	⊖	1	10/2/2014, 3:52:25 PM	5.0 s	xiopi01@XIOPI01	

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 Solutions 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

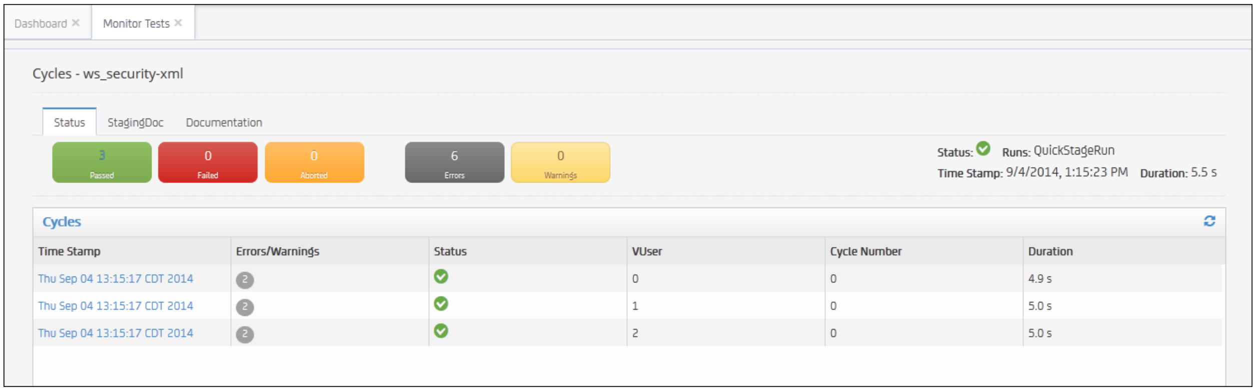
Displays the description of the test case or test suite.

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.



Cycles - ws_security-xml

3	0	0	6	0
Passed	Failed	Aborted	Errors	Warnings

Status: ✓ Runs: QuickStageRun
Time Stamp: 9/4/2014, 1:15:23 PM Duration: 5.5 s

Cycles

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.

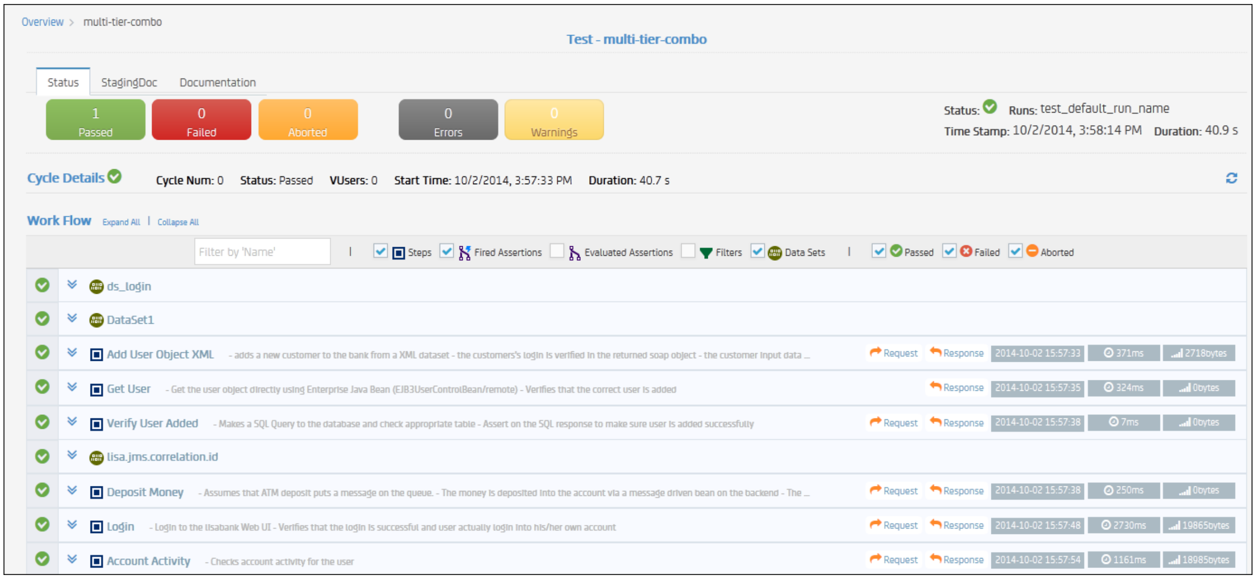
Duration

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



Overview > multi-tier-combo

Test - multi-tier-combo

Status | StagingDoc | Documentation

1 Passed | 0 Failed | 0 Aborted | 0 Errors | 0 Warnings

Status: ✓ **Runs:** test_default_run_name

Time Stamp: 10/2/2014, 3:58:14 PM **Duration:** 40.9 s

Cycle Details ✓ **Cycle Num:** 0 **Status:** Passed **VUsers:** 0 **Start Time:** 10/2/2014, 3:57:33 PM **Duration:** 40.7 s

Work Flow Expand All | Collapse All

Filter by 'Name' | ☑ Steps ☑ Fired Assertions □ Evaluated Assertions □ Filters ☑ Data Sets | ☑ Passed ☑ Failed ☑ Aborted

✓	ds_login					
✓	DataSet1					
✓	Add User Object XML - adds a new customer to the bank from a XML dataset - the customers's login is verified in the returned soap object - the customer input data ...	Request	Response	2014-10-02 15:57:33	371ms	2718bytes
✓	Get User - Get the user object directly using Enterprise Java Bean (EJB3UserControlBean/remote) - Verifies that the correct user is added		Response	2014-10-02 15:57:35	324ms	0bytes
✓	Verify User Added - Makes a SQL Query to the database and check appropriate table - Assert on the SQL response to make sure user is added successfully	Request	Response	2014-10-02 15:57:38	7ms	0bytes
✓	lisa.jms.correlation.id					
✓	Deposit Money - Assumes that ATM deposit puts a message on the queue. - The money is deposited into the account via a message driven bean on the backend - The ...	Request	Response	2014-10-02 15:57:38	250ms	0bytes
✓	Login - Login to the lisabank Web UI - Verifies that the login is successful and user actually login into his/her own account	Request	Response	2014-10-02 15:57:48	2730ms	19865bytes
✓	Account Activity - Checks account activity for the user	Request	Response	2014-10-02 15:57:54	1161ms	18985bytes

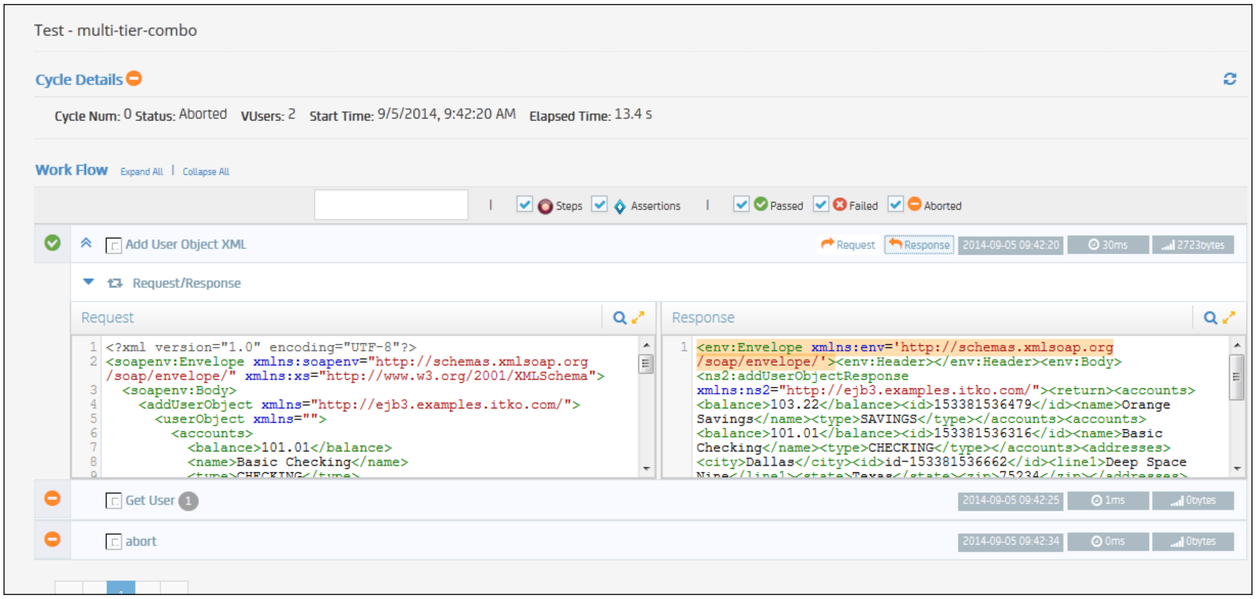
Each test step in the test case displays. All data sets, filters (global and nonglobal), and assertions (global and nonglobal) that are associated with each step also display.

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.



Test - multi-tier-combo

Cycle Details

Cycle Num: 0 Status: Aborted VUsers: 2 Start Time: 9/5/2014, 9:42:20 AM Elapsed Time: 13.4 s

Work Flow Expand All | Collapse All

☑ Steps ☑ Assertions | ☑ Passed ☑ Failed ☑ Aborted

Add User Object XML — Request | Response | 2014-09-05 09:42:20 | 30ms | 2723bytes

Request/Response

Request

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org
/soap/envelope/" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <soapenv:Body>
    <addUserObject xmlns="http://ejb3.examples.itko.com/">
      <userObject xmlns="">
        <accounts>
          <balance>101.01</balance>
          <name>Basic Checking</name>
          <type>CHECKING</type>
```

Response

```
<env:Envelope xmlns:env='http://schemas.xmlsoap.org
/soap/envelope/'><env:Header></env:Header><env:Body>
<ns2:addUserObjectResponse
xmlns:ns2="http://ejb3.examples.itko.com/"><return><accounts>
<balance>103.22</balance><id>153381536479</id><name>Orange
Savings</name><type>SAVINGS</type></accounts><accounts>
<balance>101.01</balance><id>153381536316</id><name>Basic
Checking</name><type>CHECKING</type></accounts><addresses>
<city>Dallas</city><id>id-153381536662</id><line1>Deep Space
Nine</line1><state>Texas</state><zip>75234</zip></addresses>
```

Get User 1 — 2014-09-05 09:42:25 | 1ms | 0bytes

abort — 2014-09-05 09:42:34 | 0ms | 0bytes

To search the text of a request or a response, click Search 🔍. Enter a query string and click Enter.

Note: If you do not see events or properties for a step that had events or properties, inspect the staging document for that test to be sure that it is set to record events, requests, responses, screenshots, and properties that are set or referenced. You can use the **Run1User1CycleShowAll** staging document to ensure all is captured.

To expand the view of a request or a response, click Expand ⤢.

To view the staging document that a test uses, click the Staging Doc tab.

To view documentation that is associated with a test, click the Documentation tab.

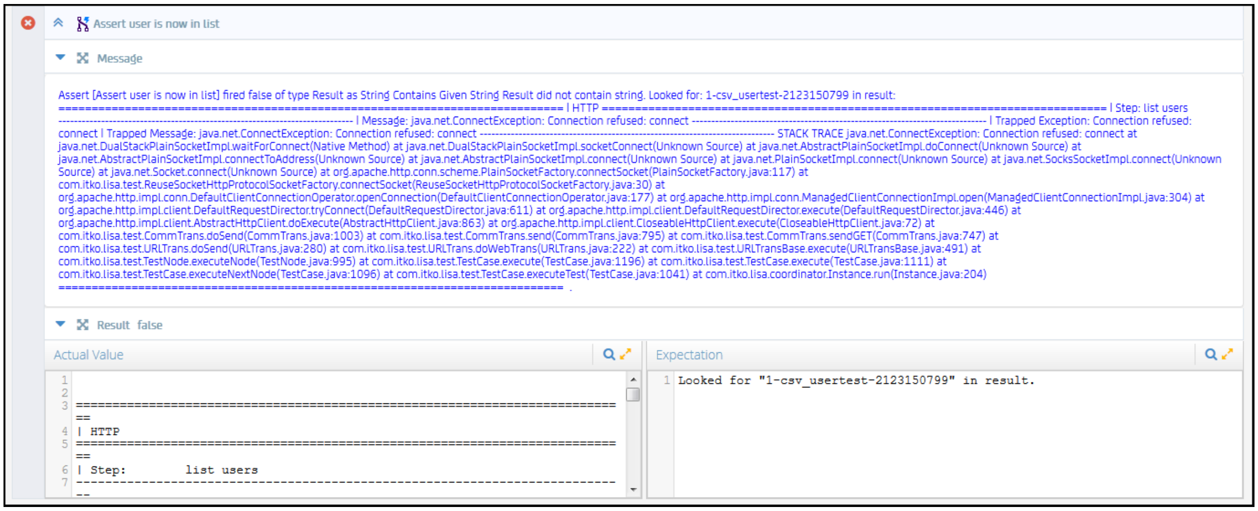
Display Details of Assertions, Filters, Data Sets, and Companions

To display the details of the results of assertions, filters, data sets, and companions, click the arrow ︾ to the left of the element name.

Details of Assertions

You can select to see details for fired assertions or evaluated assertions by selecting the appropriate check box on the filter bar.

Details for fired assertions display the message that the assertion produced and the result of the assertion.



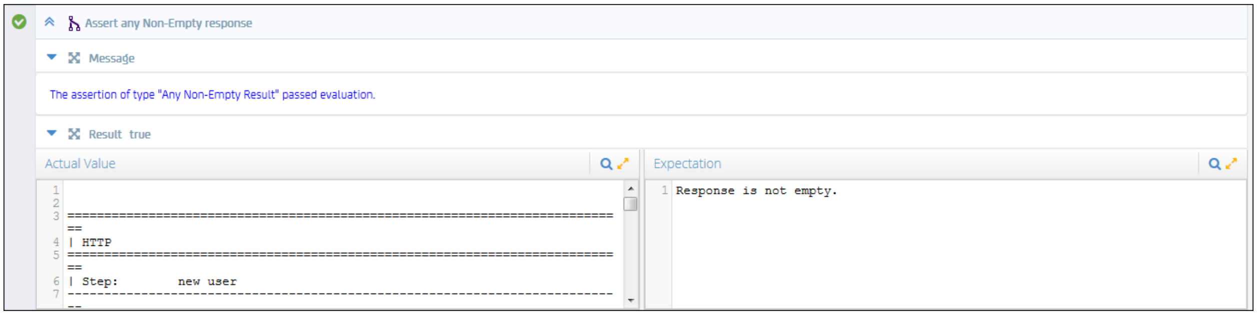
```
Assert [Assert user is now in list] fired false of type Result as String Contains Given String Result did not contain string. Looked for: 1-csv_usertest-2123150799 in result:
============================================================================== | HTTP ============================================================================== | Step: list users
------------------------------------------------------------------------------ | Message: java.net.ConnectException: Connection refused: connect ------------------------------------------------------------------------------ | Trapped Exception: Connection refused:
connect | Trapped Message: java.net.ConnectException: Connection refused: connect ------------------------------------------------------------------------------ STACK TRACE java.net.ConnectException: Connection refused: connect at
java.net.DualStackPlainSocketImpl.waitForConnect(Native Method) at java.net.DualStackPlainSocketImpl.socketConnect(Unknown Source) at java.net.AbstractPlainSocketImpl.doConnect(Unknown Source) at
java.net.AbstractPlainSocketImpl.connectToAddress(Unknown Source) at java.net.AbstractPlainSocketImpl.connect(Unknown Source) at java.net.PlainSocketImpl.connect(Unknown Source) at java.net.SocksSocketImpl.connect(Unknown
Source) at java.net.Socket.connect(Unknown Source) at org.apache.http.conn.scheme.PlainSocketFactory.connectSocket(PlainSocketFactory.java:117) at
com.itko.lisa.test.ReuseSocketHttpProtocolSocketFactory.connectSocket(ReuseSocketHttpProtocolSocketFactory.java:30) at
org.apache.http.impl.conn.DefaultClientConnectionOperator.openConnection(DefaultClientConnectionOperator.java:177) at org.apache.http.impl.conn.ManagedClientConnectionImpl.open(ManagedClientConnectionImpl.java:304) at
org.apache.http.impl.client.DefaultRequestDirector.tryConnect(DefaultRequestDirector.java:611) at org.apache.http.impl.client.DefaultRequestDirector.execute(DefaultRequestDirector.java:446) at
org.apache.http.impl.client.AbstractHttpClient.doExecute(AbstractHttpClient.java:863) at org.apache.http.impl.client.CloseableHttpClient.execute(CloseableHttpClient.java:72) at
com.itko.lisa.test.CommTrans.doSend(CommTrans.java:1003) at com.itko.lisa.test.CommTrans.send(CommTrans.java:795) at com.itko.lisa.test.CommTrans.sendGET(CommTrans.java:747) at
com.itko.lisa.test.URLTrans.doSend(URLTrans.java:280) at com.itko.lisa.test.URLTrans.doWebTrans(URLTrans.java:222) at com.itko.lisa.test.URLTransBase.execute(URLTransBase.java:491) at
com.itko.lisa.test.TestNode.executeNode(TestNode.java:995) at com.itko.lisa.test.TestCase.execute(TestCase.java:1196) at com.itko.lisa.test.TestCase.execute(TestCase.java:1111) at
com.itko.lisa.test.TestCase.executeNextNode(TestCase.java:1096) at com.itko.lisa.test.TestCase.executeTest(TestCase.java:1041) at com.itko.lisa.coordinator.Instance.run(Instance.java:204)
============================================================================== .
```

```


==============================================================================
==
| HTTP
==============================================================================
==
| Step:        list users
------------------------------------------------------------------------------
--
```

```
Looked for "1-csv_usertest-2123150799" in result.
```

Details for evaluated assertions display the message that the assertion produced and the result of the assertion.



The assertion of type "Any Non-Empty Result" passed evaluation.

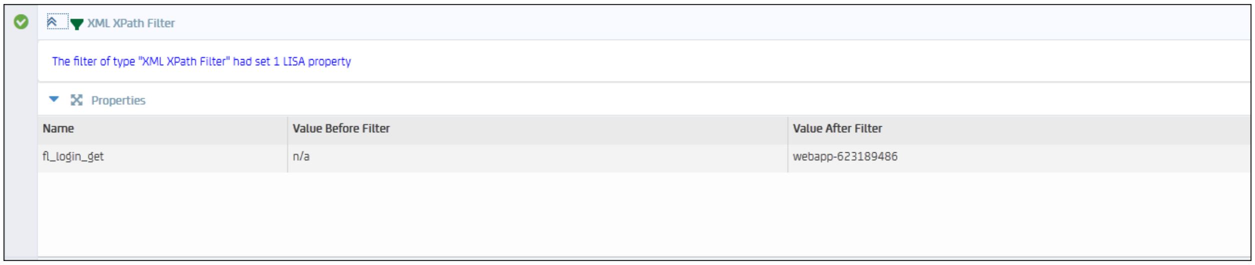
```


==============================================================================
==
| HTTP
==============================================================================
==
| Step:        new user
------------------------------------------------------------------------------
--
```

```
Response is not empty.
```

Details of Filters

Details for filters display the properties that the filter set.



XML XPath Filter

The filter of type "XML XPath Filter" had set 1 LISA property

Properties

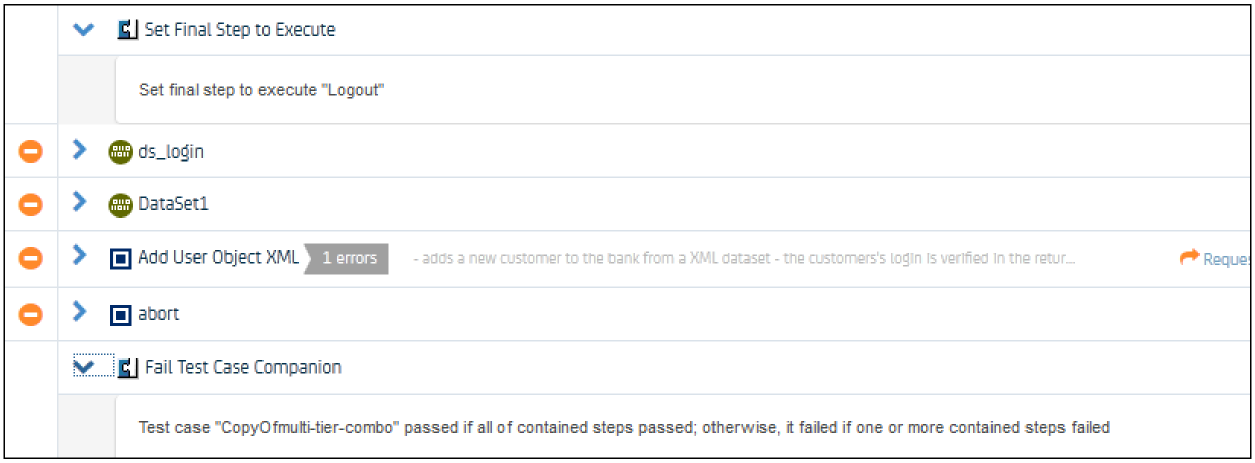
Name	Value Before Filter	Value After Filter
fl_login_get	n/a	webapp-623189486

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.



Set Final Step to Execute

Set final step to execute "Logout"

ds_login

DataSet1

Add User Object XML 1 errors - adds a new customer to the bank from a XML dataset - the customers's login is verified in the retur... Reque

abort

Fail Test Case Companion

Test case "CopyOfmulti-tier-combo" passed if all of contained steps passed; otherwise, it failed if one or more contained steps failed

Chapter 5: Manage Test Artifacts

To manage test artifacts, use the Manage option on the left nav bar or select Manage Test Artifacts from the Quick Links portlet on the portal.

To manage tests that were created with the DevTest Portal, select Manage API Test.

To manage test cases that were created through DevTest Workstation, select Manage Tests.

To manage test suites that were created through DevTest Workstation, select Manage Test Suites.

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.

This section contains the following topics:

Manage API Tests (see page 37)
Manage Tests (see page 39)
Manage Test Suites (see page 41)

Chapter 6: Manage API Tests

The Manage API Tests option lets you edit tests that were created in the DevTest Portal.

Follow these steps:

1. Select a test from the list of tests, and click it.

   The Test Editor opens.

2. Use the Test Editor as described in Edit an API Test (see page 24).

Chapter 7: Manage Tests

To run a test, right-click the test name and select Run.

To delete a test, right-click the test name and select Delete.

To download the Mar (see page 256) file with the test, right-click the Test name and select Download Mar.

To view information and documentation about the test, click the test name.

Chapter 8: Manage Test Suites

To run a test suite, right-click the suite name and select Run.

To delete a test suite, right-click the suite name and select Delete.

To download the Mar (see page 256) file for the test suite, right-click the suite name and select Download Mar.

To view suite information and documentation about the test suite, click the suite name.

Chapter 9: Using the Workstation and Console with CA Application Test

This section contains the following topics:

Workstation and Console Overview (see page 43)
Building Test Cases (see page 73)
Building Documents (see page 215)
Working with Model Archives (MARs) (see page 255)
Running Test Cases and Suites (see page 269)
Cloud DevTest Labs (see page 333)
Continuous Validation Service (CVS) (see page 353)
Reports (see page 369)
Recorders and Test Generators (see page 409)
Mobile Testing (see page 419)
Advanced Features (see page 433)

Workstation and Console Overview

This section contains the following topics:

DevTest Workstation (see page 44)
DevTest Console (see page 70)

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.

Open DevTest Workstation

When you open DevTest Workstation, you are prompted to specify a registry (see page 11).

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

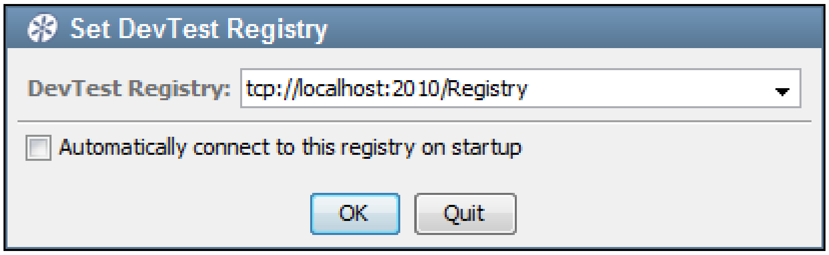
Use a registry that is running on a remote computer if your computer has an installation of DevTest Workstation.

For more information about how to specify the registry with SSL enabled, see Use SSL to Secure Communication Between Components in *Administering.*

To open DevTest Workstation:

1. Do one:
   - Open a command prompt, go to the **LISA_HOME\bin** directory, and run the DevTest Workstation executable.
   - If you have a DevTest application icon DevTest Workstation on your desktop, double-click the application icon.
   - Click Start Menu, All Programs, DevTest, DevTest Workstation.

   The Set DevTest Registry dialog appears.



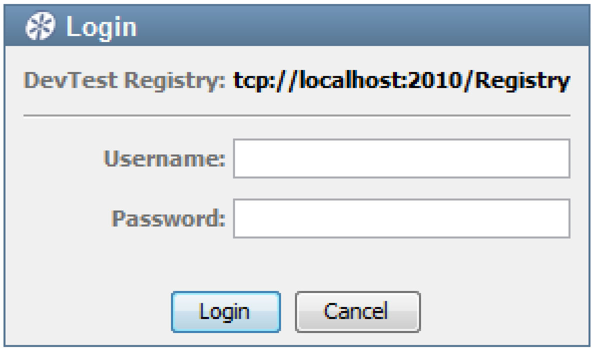
Set DevTest Registry

DevTest Registry: tcp://localhost:2010/Registry

□ Automatically connect to this registry on startup

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 appears.



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 59) 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 200) in *Using CA Application Test.*

Create New VS Model

For detailed information, see Work with Virtual Service Models in *Using CA Service Virtualization.* This feature requires an extra license.

Create New VS Image

For detailed information, see Create a Service Image in *Using CA Service Virtualization.* This feature requires an extra license.

Create New Staging Document

For detailed information, see Building Staging Documents (see page 215) in *Using CA Application Test.*

Create New Suite

For detailed information, see Building Test Suites (see page 243) in *Using CA Application Test.*

Create New Test Audit

For detailed information, see Building Audit Documents (see page 237) in *Using CA Application Test.*

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 (see page 257)

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 (see page 237)

DefaultAudit

main_all_should_fail

ws_security-xml

Configs (see page 100)

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 (see page 215)

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 (see page 209)

ws-sec-sub

This one-step test case is marked as a subprocess and can be called from any Execute Subprocess step.

Test Suites (see page 243)

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 (see page 73)

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.

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/webservic
e
```

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.

Lisa_config_info

This test case fetches diagnostic information about the computer running DevTest. The results can help Support solve configuration issues.

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

Tests the ability of VSEasy to create a JMS service from request/response pairs.

service-validation

This test is a simple example of service validation. The test calls one web service and one EJB service and inspects the underlying database with SQL to validate that the services function appropriately.

web-application

This test is a simple web test that was generated using the web recorder. The test contains some basic assertions such as "assert nonempty response," which is automatically generated. The test also contains some "assert title" assertions that were created by parsing the HTML responses for the <title> tag. These assertions help to ensure that the recorded page is the same when we play back the test.

webservices

This test case adds, gets, and deletes a user from the EJB3 web services. The test uses a unique code generator to create a number that has a prefix that is a value {{user}} from the config file. The password is hard-coded in the config file.

ws_attachments

This test case tests our ability to send and receive inline base64 encoded data blobs and XOP/MTOM attachments. Filters and assertions on steps verify that the requests and responses look correct.

ws_security

The ws_security test case shows how to use signed and encrypted SOAP messages. The first two steps succeed and the last two steps fail. The calls are the same, but the web service does not accept messages that are unencrypted or unsigned.

ws_security-xml

This test model shows how to use signed and encrypted SOAP messages. The first two steps should succeed and the last two steps should fail. (The calls are the same but the web service does not accept messages that are unencrypted or signed.)

This test plan requires that your Java environment supports unlimited strength encryption. If either of the first two steps fail, a likely suspect is that your JCE jars must be updated to enable unlimited strength encryption. The JCE jars can be downloaded from www.oracle.com. Search on the keywords "JCE unlimited strength" and pick the JCE library matching your Java version, such as JCE 7 for Java 7. After you install the JCE jars, your DevTest services must be restarted.

In the audit document, two occurrences of Step Error are expected. We audit for two Step Errors because we expect both of the last two steps here to raise Step Errors. Thus, the audit document can also audit how many occurrences of an event are received. If we add a third Step Error entry to the audit document, the auditor will fail this test because only two Step Error events are raised.

Tests that Fail

webservices-xml-fail

This test case adds, gets, and deletes a user from the EJB3 web services. The test uses a unique code generator to create a number that has a prefix that is a value {{user}} from the config file. The password is hard-coded in the config file.

VServices

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:

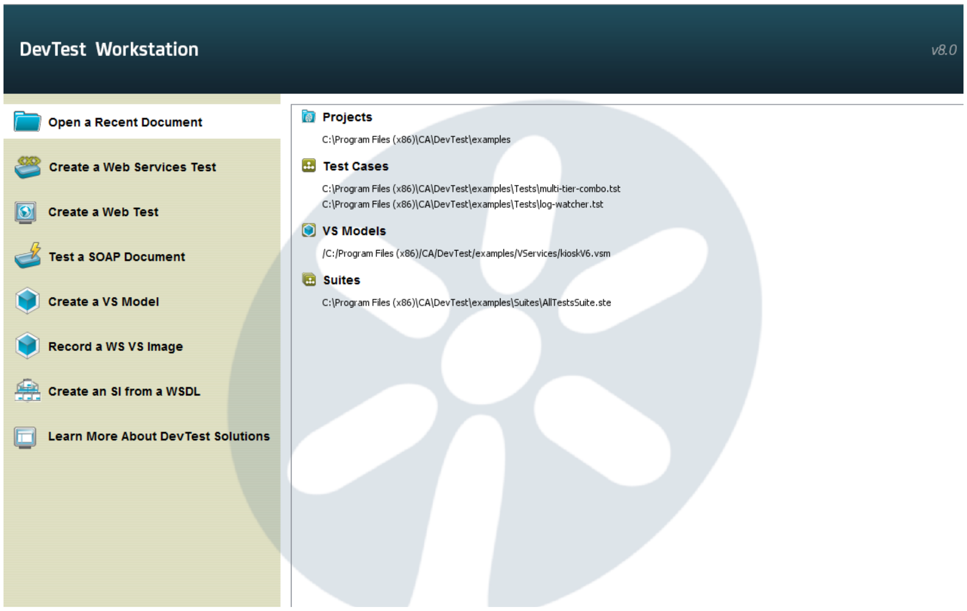
1. Navigate to the **LISA_HOME\bin** directory.
2. Run the DevTest Workstation executable and specify the project directory as an argument. The project directory can be an absolute path or a relative path. The following example uses an absolute path:

   ```
   LISAWorkstation "C:\Program Files\CA\Lisa\Projects\Project1"
   ```

Quick Start Window

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.



Projects
C:\Program Files (x86)\CA\DevTest\examples

Test Cases
C:\Program Files (x86)\CA\DevTest\examples\Tests\multi-tier-combo.tst
C:\Program Files (x86)\CA\DevTest\examples\Tests\log-watcher.tst

VS Models
/C:/Program Files (x86)/CA/DevTest/examples/VServices/kioskV6.vsm

Suites
C:\Program Files (x86)\CA\DevTest\examples\Suites\AllTestsSuite.ste

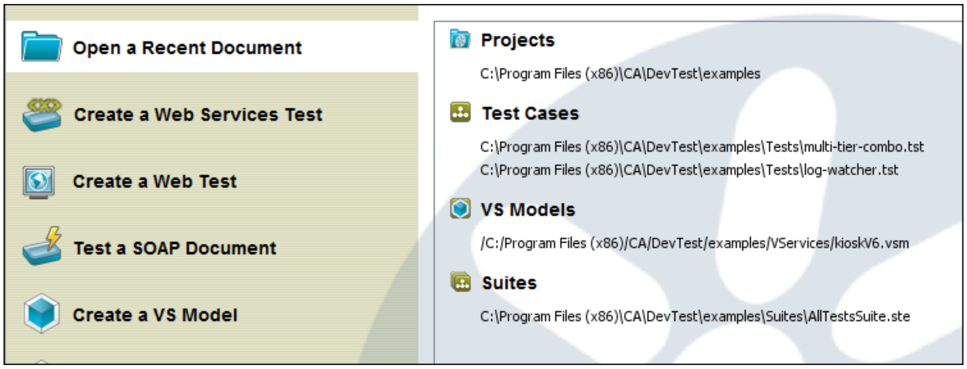
The Quick Start window has some of the most useful options in DevTest Workstation. When you click an option, its parameters are listed on the right side of the window.

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).



Projects

C:\Program Files (x86)\CA\DevTest\examples

Test Cases

C:\Program Files (x86)\CA\DevTest\examples\Tests\multi-tier-combo.tst
C:\Program Files (x86)\CA\DevTest\examples\Tests\log-watcher.tst

VS Models

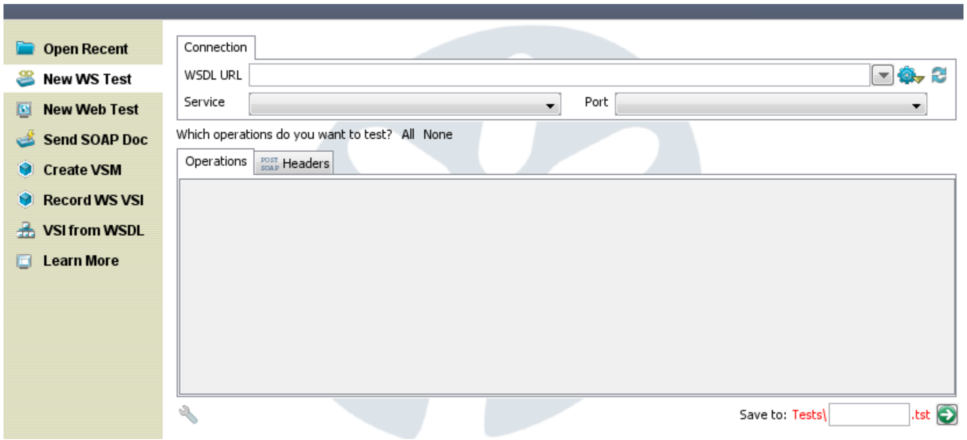
/C:/Program Files (x86)/CA/DevTest/examples/VServices/kioskV6.vsm

Suites

C:\Program Files (x86)\CA\DevTest\examples\Suites\AllTestsSuite.ste

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.



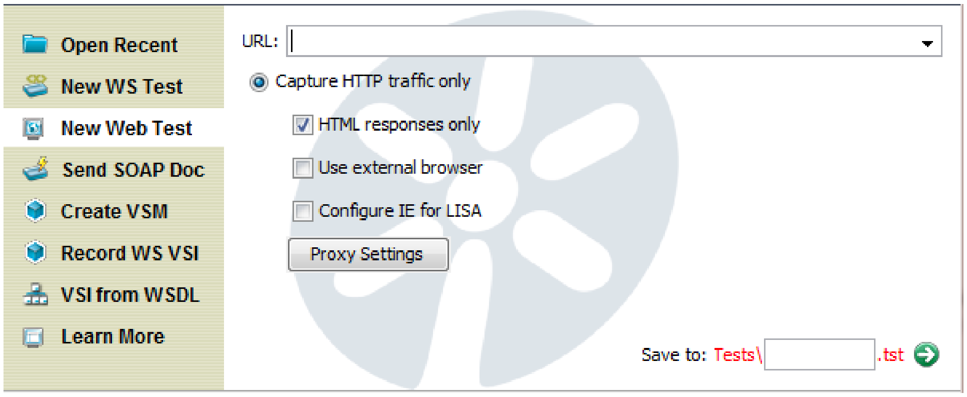
Follow these steps:

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 409) in *Using CA Application Test.*

New Web Test

The New Web Test option in the Quick Start window lets you create a web test.



URL:

● Capture HTTP traffic only

☑ HTML responses only

□ Use external browser

□ Configure IE for LISA

Save to: Tests\ .tst

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

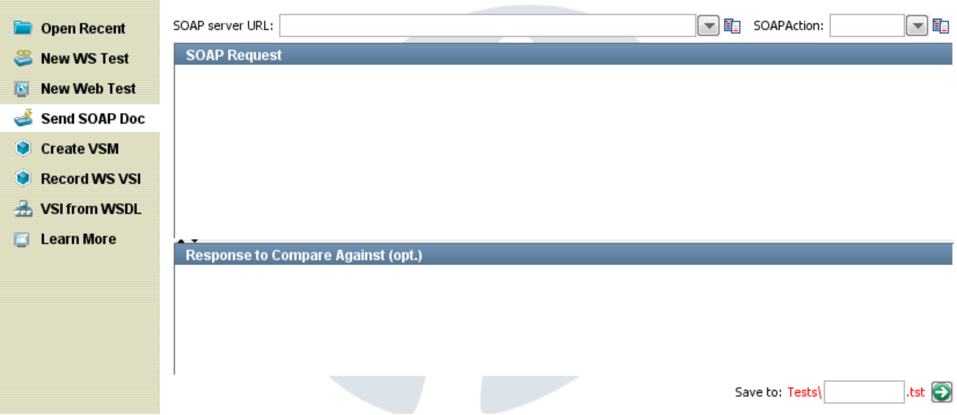
   Configures Internet Explorer for DevTest.

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 411) in *Using CA Application Test.*

Send SOAP Doc

The Send SOAP Doc option in the Quick Start window lets you create a SOAP Request test case.

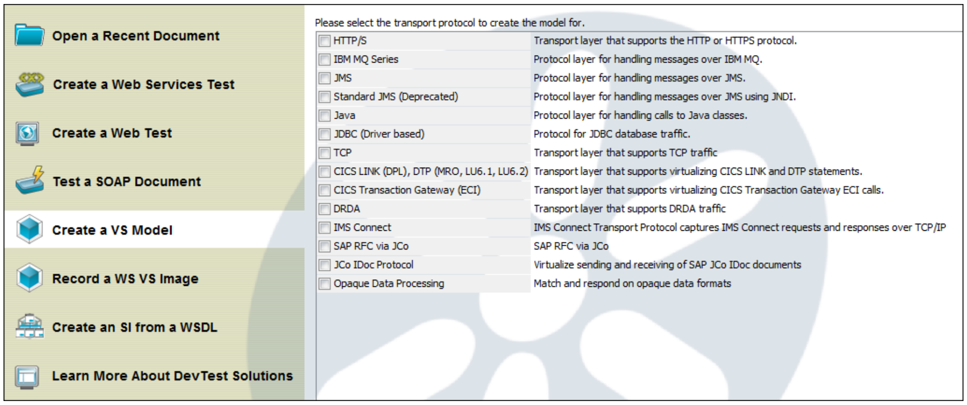


Follow these steps:

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.



Please select the transport protocol to create the model for.

□ HTTP/S	Transport layer that supports the HTTP or HTTPS protocol.
□ IBM MQ Series	Protocol layer for handling messages over IBM MQ.
□ JMS	Protocol layer for handling messages over JMS.
□ Standard JMS (Deprecated)	Protocol layer for handling messages over JMS using JNDI.
□ Java	Protocol layer for handling calls to Java classes.
□ JDBC (Driver based)	Protocol for JDBC database traffic.
□ TCP	Transport layer that supports TCP traffic
□ CICS LINK (DPL), DTP (MRO, LU6.1, LU6.2)	Transport layer that supports virtualizing CICS LINK and DTP statements.
□ CICS Transaction Gateway (ECI)	Transport layer that supports virtualizing CICS Transaction Gateway ECI calls.
□ DRDA	Transport layer that supports DRDA traffic
□ IMS Connect	IMS Connect Transport Protocol captures IMS Connect requests and responses over TCP/IP
□ SAP RFC via JCo	SAP RFC via JCo
□ JCo IDoc Protocol	Virtualize sending and receiving of SAP JCo IDoc documents
□ Opaque Data Processing	Match and respond on opaque data formats

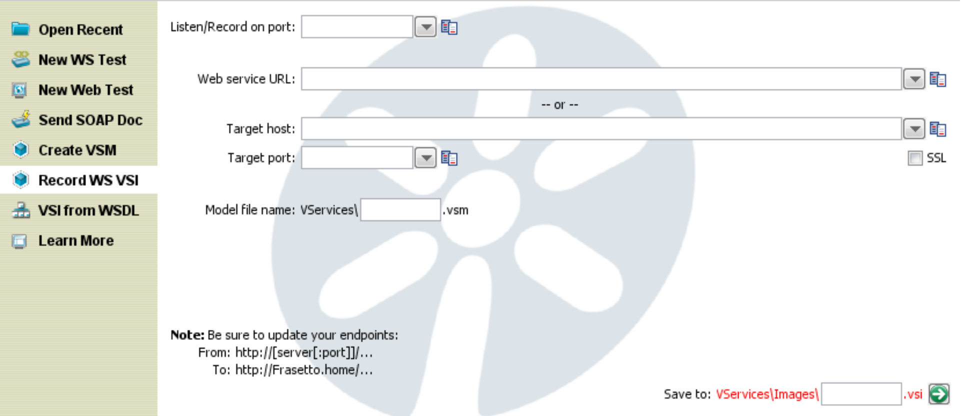
Follow these steps:

1. Select the transport protocol from the list of available protocols. Your selection here will determine the next sequence of windows.
2. Enter the name of the service image.
3. Enter the name of the VSM and select the path where you would like to save it in the project folder. When you select the path, it is seen in the Path field.
4. To create the test case, click the green arrow ➔.

For more detailed information about the windows and options for VSM and VSI creation, see Create a Service Image.

Record 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.



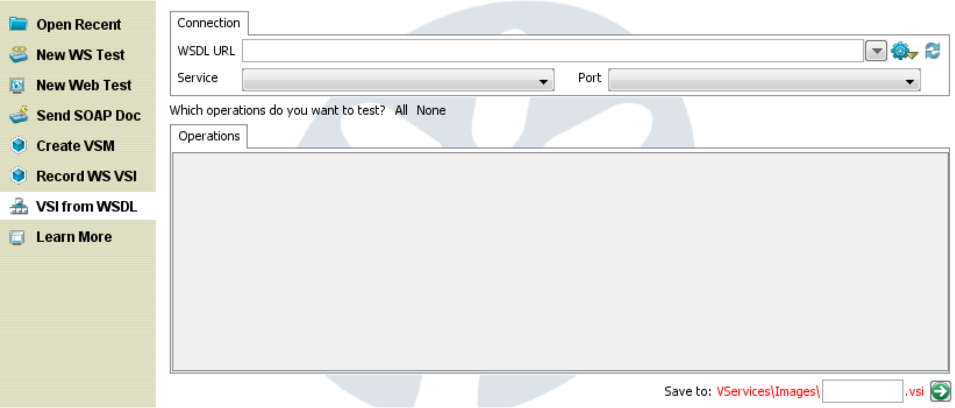
Note: Be sure to update your endpoints:
From: http://[server[:port]]/...
To: http://Frasetto.home/...

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.



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 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
```

DevTest Console

The web-based DevTest Console provides access to the following consoles and dashboards:

- Reporting Dashboard
- Continuous Validation Service
- Server Console
- VSEasy

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 369) in *Using CA Application Test.*

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 353) in *Using CA Application Test.*

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.

VSEasy

VSEasy lets you create HTTP service images and service images from VRS files. You can use VSEasy to create stateless virtual services for HTTP with SOAP, JSON, XML, or HTML payloads.

For more information, see Cloud DevTest Labs (see page 333), Access Control (ACL), VSE Dashboard, and Create and Deploy a Virtual Service with VSEasy in *Using CA Application Test, Administering, and Using CA Service Virtualization*.

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, the Server Console, and VSEasy. 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 11) 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
```

Chapter 10: 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.

This section contains the following topics:

Anatomy of a Test Case (see page 73)
Properties (see page 86)
Configurations (see page 100)
Assets (see page 106)
Filters (see page 114)
Assertions (see page 129)
Data Sets (see page 145)
Companions (see page 155)
Complex Object Editor (COE) (see page 158)
Building Test Steps (see page 188)
Creating Test Cases (see page 199)
Building Subprocesses (see page 209)

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.

The following topics are included.

Test Case Quick Start (see page 75)
Multi-tier-combo Test Case (see page 79)
Elements of a Test Case (see page 81)
Elements of a Test Step (see page 83)

Test Case Quick Start

To start working with test cases:

1. Start the registry.

   See DevTest Registry (see page 11) in *Using CA Application Test.*

2. Create or open a project in DevTest Workstation.

   See Project Panel (see page 46) in *Using CA Application Test.*

3. Create a test case in the project.

   See Create a Test Case (see page 200) in *Using CA Application Test.*

   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 79) in *Using CA Application Test.*

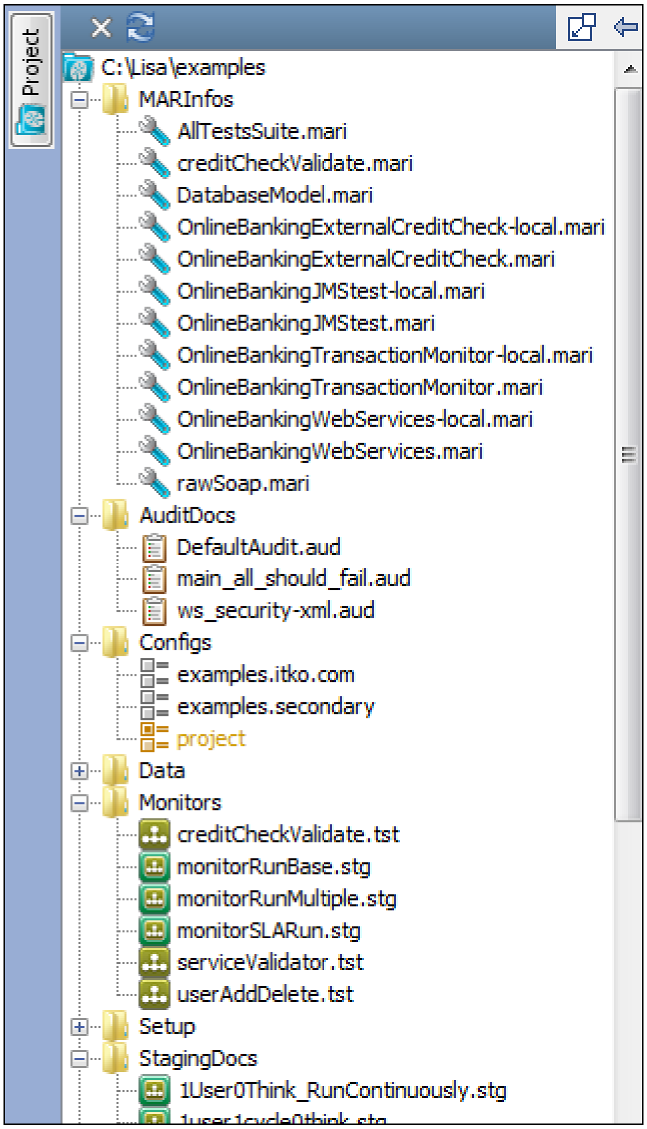
DevTest Workstation is divided into the following main areas (from left to right):

- 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 Project button on the left.

When DevTest Workstation first opens, the last project you had open opens by default.

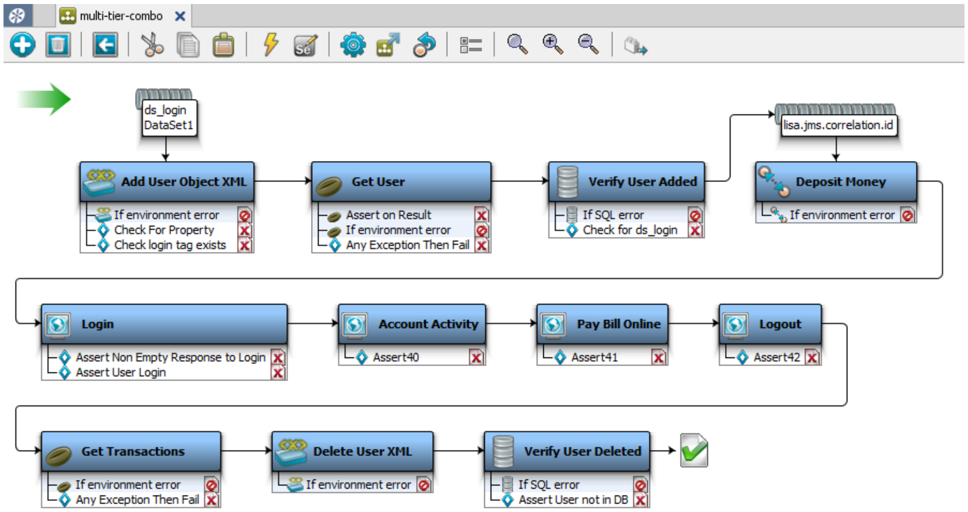


For more information, see Project Panel (see page 46) in *Using CA Application Test.*

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.



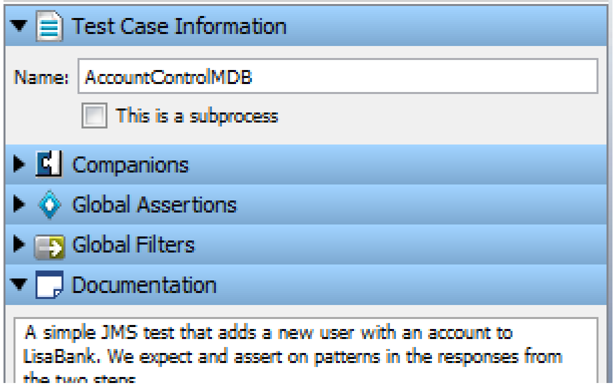
The green arrow marks the start of a test case.

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.



Test Case Information

Name: AccountControlMDB

☐ This is a subprocess

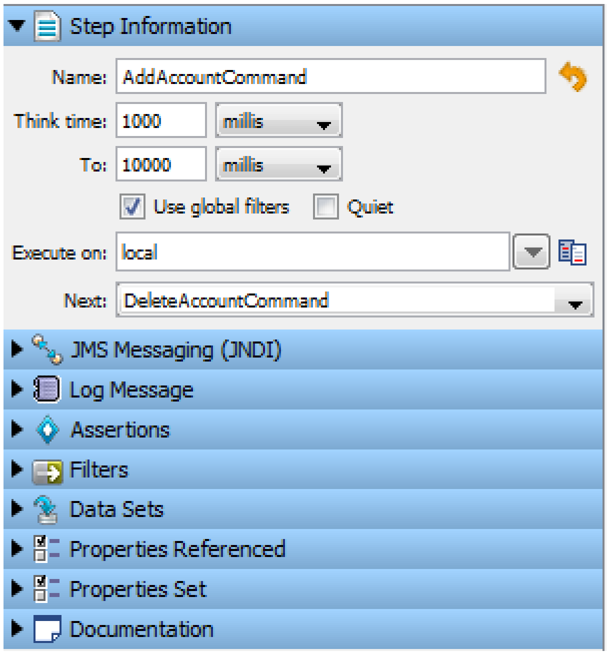
Companions

Global Assertions

Global Filters

Documentation

A simple JMS test that adds a new user with an account to LisaBank. We expect and assert on patterns in the responses from the two steps



Name: AddAccountCommand

Think time: 1000 millis

To: 10000 millis

☑ Use global filters ☐ Quiet

Execute on: local

Next: DeleteAccountCommand

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 50) 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 270) window, the test creates a single user from the first row of the spreadsheet and finishes.

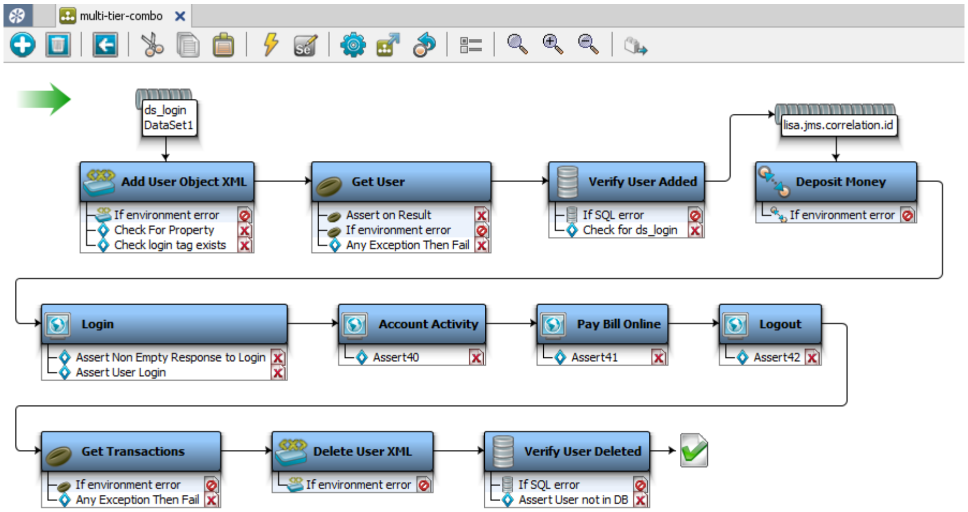
If you stage the test (see page 289) with the example **1User0Think_RunContinuously** staging document, the test restarts until it reaches the end of the data set. This method is the preferred way to iterate repeatedly over a large data set. You can introduce a loop in the test case, but that is not 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."

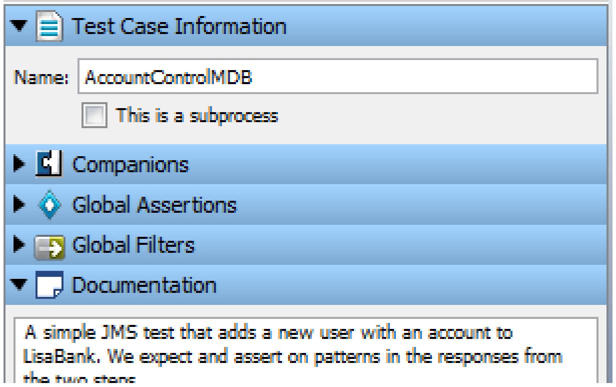
Notice the **example** project folders being opened in the Project panel and a set of test case elements in the Element panel.

Here you can see in the model editor section the test case information.


multi-tier-combo
ds_login
DataSet1
Add User Object XML
If environment error
Check For Property
Check login tag exists
Get User
Assert on Result
If environment error
Any Exception Then Fail
Verify User Added
If SQL error
Check for ds_login
lisa.jms.correlation.id
Deposit Money
If environment error
Login
Assert Non Empty Response to Login
Assert User Login
Account Activity
Assert40
Pay Bill Online
Assert41
Logout
Assert42
Get Transactions
If environment error
Any Exception Then Fail
Delete User XML
If environment error
Verify User Deleted
If SQL error
Assert User not in DB

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.



Name: AccountControlMDB

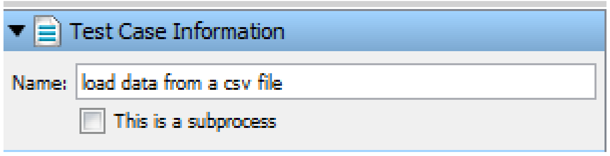
□ This is a subprocess

A simple JMS test that adds a new user with an account to LisaBank. We expect and assert on patterns in the responses from the two steps

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.



Name: load data from a csv file

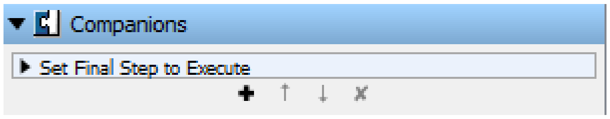
□ This is a subprocess

For more information about subprocesses, see Building Subprocesses (see page 209) in *Using CA Application Test.*

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.

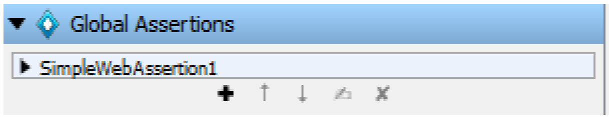


For more information, see Companions (see page 155) in *Using CA Application Test.*

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.

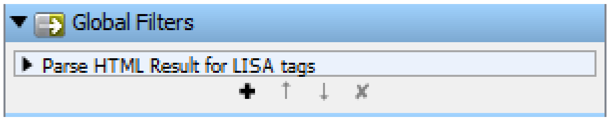


For more information, see Assertions (see page 129) in *Using CA Application Test.*

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.

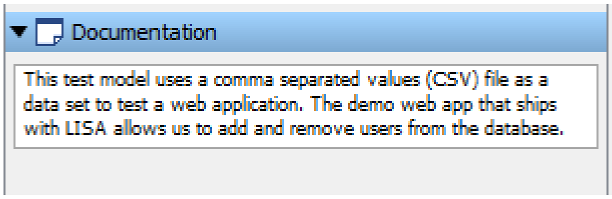
To open the filter editor, double-click the filter in the Global Filter list.



For more information, see Filters (see page 114) in *Using CA Application Test.*

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.

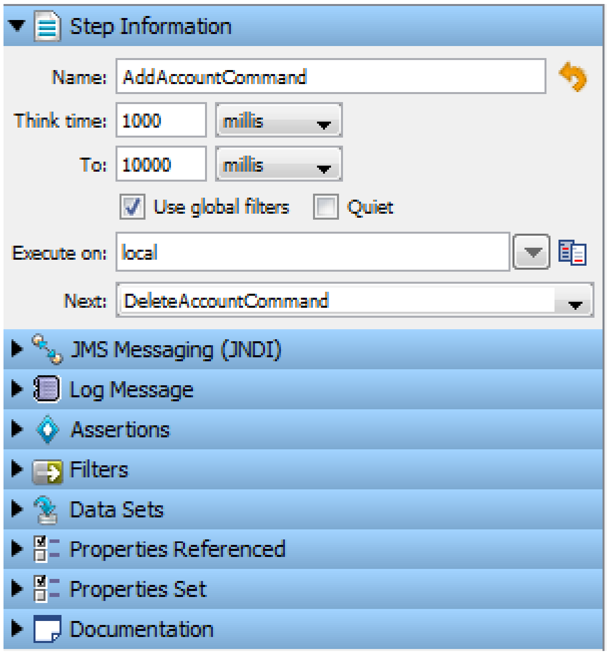


This test model uses a comma separated values (CSV) file as a data set to test a web application. The demo web app that ships with LISA allows us to add and remove users from the database.

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 188) in *Using CA Application Test.*



Step Information

Name: AddAccountCommand

Think time: 1000 millis

To: 10000 millis

☑ Use global filters ☐ Quiet

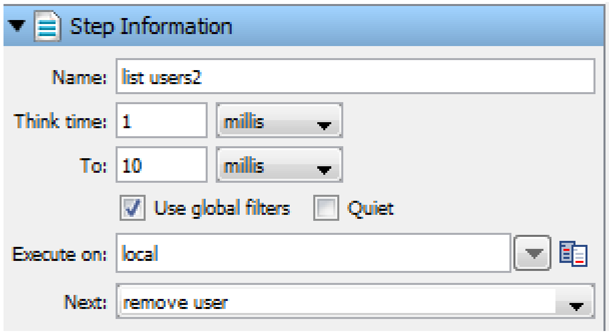
Execute on: local

Next: DeleteAccountCommand

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.



Name: list users2

Think time: 1 millis

To: 10 millis

☑ Use global filters □ Quiet

Execute on: local

Next: remove user

For more information, see Building Test Steps (see page 188) in *Using CA Application Test.*

Log Message

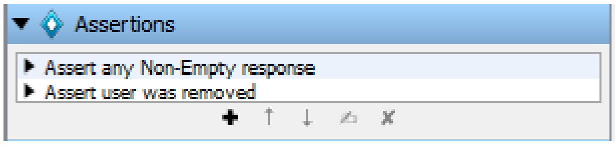
A log message is a text field in which you can enter a message for a step. This message is seen upon execution of the test step or case.

For more information, see Test Step Logger in *Administering.*

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.

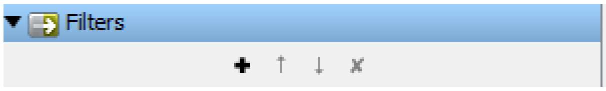


- Assert any Non-Empty response
- Assert user was removed

For more information, see Assertions (see page 129) in *Using CA Application Test.*

Filters

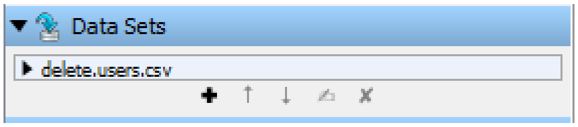
A filter is an element that runs before and after a test step. Filters give you the opportunity to change the data in the result, or store values in properties.



For more information, see Filters (see page 114) in *Using CA Application Test.*

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.



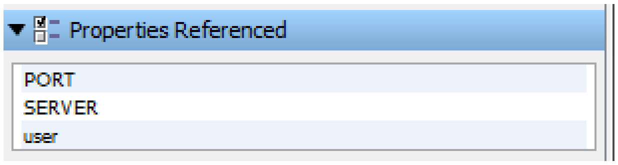
delete.users.csv

For more information, see Data Sets (see page 145) in *Using CA Application Test.*

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.

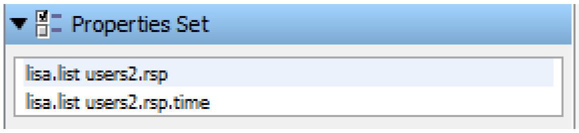


PORT
SERVER
user

For more information, see Properties (see page 86) in *Using CA Application Test.*

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.

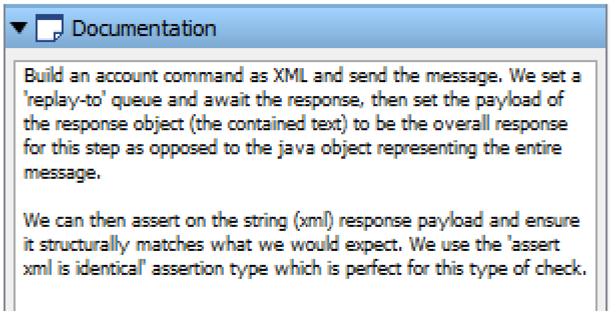


lisa.list users2.rsp
lisa.list users2.rsp.time

For more information, see Properties (see page 86) in *Using CA Application Test.*

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.



Build an account command as XML and send the message. We set a 'replay-to' queue and await the response, then set the payload of the response object (the contained text) to be the overall response for this step as opposed to the java object representing the entire message.

We can then assert on the string (xml) response payload and ensure it structurally matches what we would expect. We use the 'assert xml is identical' assertion type which is perfect for this type of check.

Properties

Test properties are name–value pairs, also known as key–value pairs.

The key to data independence, reusability, and portability in test cases is the ability to replace specific data values abstracted from them with variables. These variables are referred to as *properties*. Some properties are predefined and guide how the application operates. You create other properties while you are building your tests.

A sound understanding of properties is important to the creation of test cases. In the context of a test case, any time there is something that can change, it is appropriate to use a property. This scenario includes values in test steps and values in configurations, for example.

Properties can be defined in several ways. After they are defined, they are available to any subsequent steps, assertions, and filters in the test case (they are global to the test case). Properties can, with few exceptions, be overridden in a test case.

Whenever a property value is set, a Property set event is recorded. The event contains the property name and value.

Property values are not limited to string values. A property can hold strings, numbers, XML fragments, serialized Java objects, or the complete response from a test step. Many properties that are created during a test run that are available to the subsequent test steps. For example, the **lisa.stepname.rsp** property contains the response for the stepname step.

The following topics are included.

Specify a Property (see page 87)
Property Expressions (see page 88)
String Patterns (see page 89)
Property Sources (see page 96)
Common Properties and Environment Variables (see page 97)
Property Files (see page 99)
Use the Properties Pane (see page 99)

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 433) or www.beanshell.org.

To use a property expression, use one of the following formats:

{{=expression}}

> BeanShell is used to evaluate the expression and replace {{expression}} with the result of the evaluation. For example, {{=Math.random()}} evaluates the static Java method and replaces the {{}} construct with the random number that was returned.

{{key=expression}}

> Using {{rand=Math.random()}} sets a property **rand** equal to the random number that was returned, and replaces the {{}} construct with the random number.

You can reference properties by name only in a property expression (that is, without the braces) because they are already defined as properties. If the property is not found, the property expression is returned inside braces to indicate that there is a problem in the expression.

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:

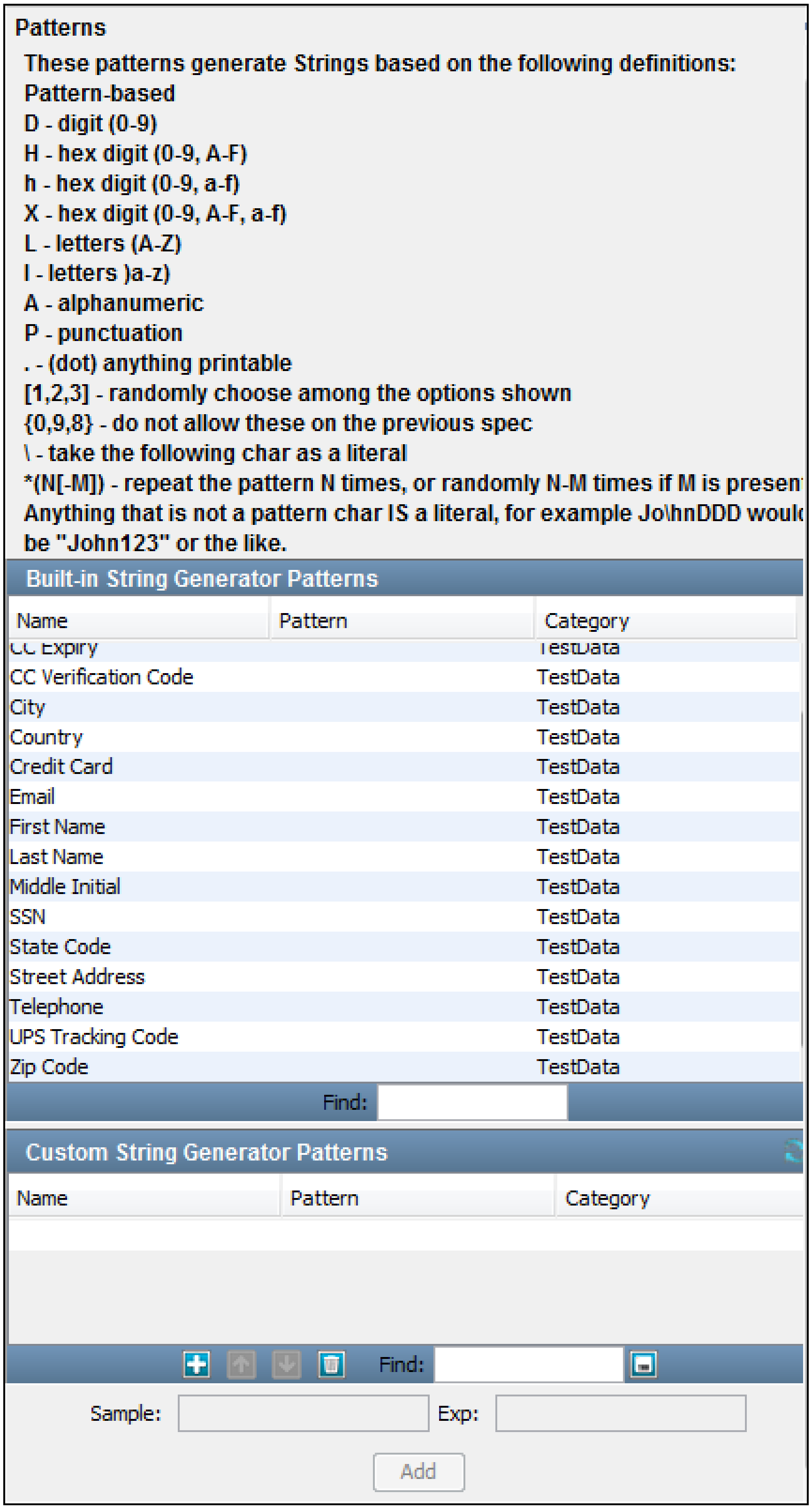
```
{{=[:First Name:]}} {{=[:Middle Initial:]}} {{=[:Last Name:]}}
{{=[:Street Address:]}}
{{=[:City:]}}, {{=[:State Code:]}}
{{=[:ZIP Code:]}} {{=[:Country:]}}
SSN: {{=[:SSN:]}}
Card: {{=[:Credit Card:]}} Expires {{=[:CC Expiry:]}}
Phone: {{=[:Telephone:]}}
Email: {{=[:Email:]}}
```

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)
l - letters)a-z)
A - alphanumeric
P - punctuation
. - (dot) anything printable
[1,2,3] - randomly choose among the options shown
{0,9,8} - do not allow these on the previous spec
\ - take the following char as a literal
*(N[-M]) - repeat the pattern N times, or randomly N-M times if M is presen
Anything that is not a pattern char IS a literal, for example Jo\hnDDD woul
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

Find:

Sample: Exp:

Add

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

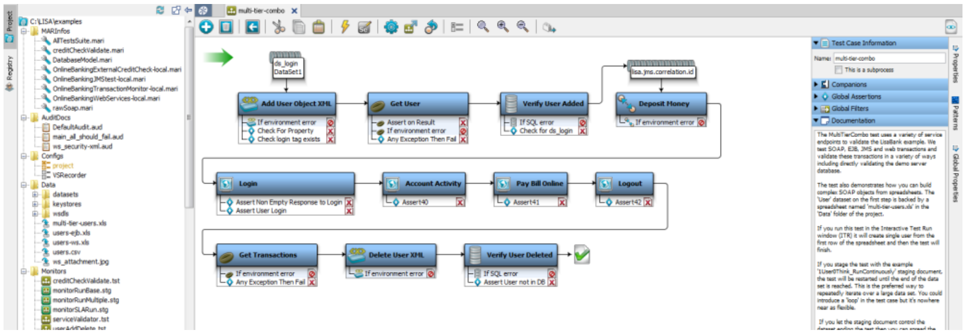
When you add your own data, the only thing to be careful about is to assign the ID correctly. Start with 1 and increase with no gaps until you get to your number of rows.

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):



Name: multi-tier-combo

□ This is a subprocess

The MultiTierCombo test uses a variety of service endpoints to validate the LisaBank example. We test SOAP, EJB, JMS and web transactions and validate these transactions in a variety of ways including directly validating the demo server database.

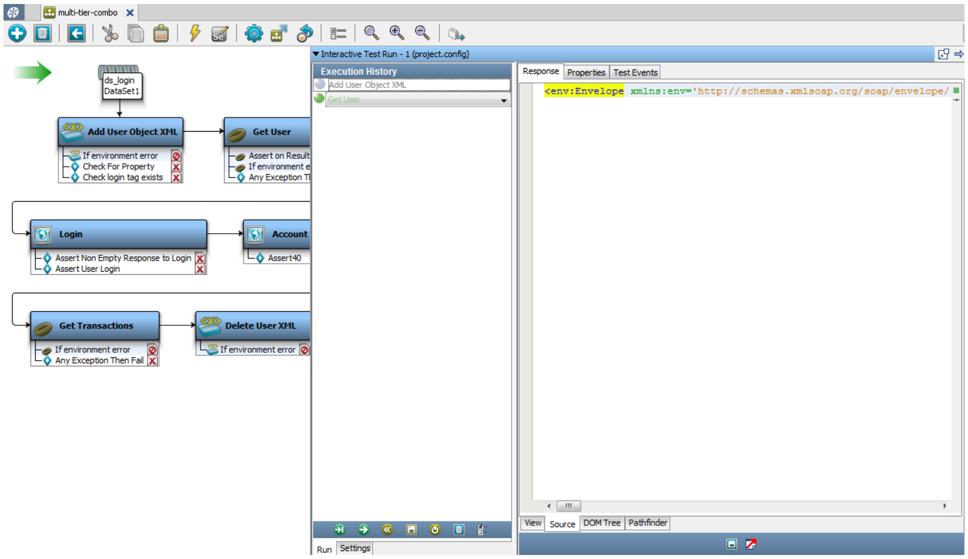
The test also demonstrates how you can build complex SOAP objects from spreadsheets. The 'User' dataset on the first step is backed by a spreadsheet named 'multi-tier-users.xls' in the 'Data' folder of the project.

If you run this test in the Interactive Test Run window (ITR) it will create single user from the first row of the spreadsheet and then the test will finish.

If you stage the test with the example '1User0Think_RunContinuously' staging document, the test will be restarted until the end of the data set is reached. This is the preferred way to repeatedly iterate over a large data set. You could introduce a 'loop' in the test case but it's nowhere near as flexible.

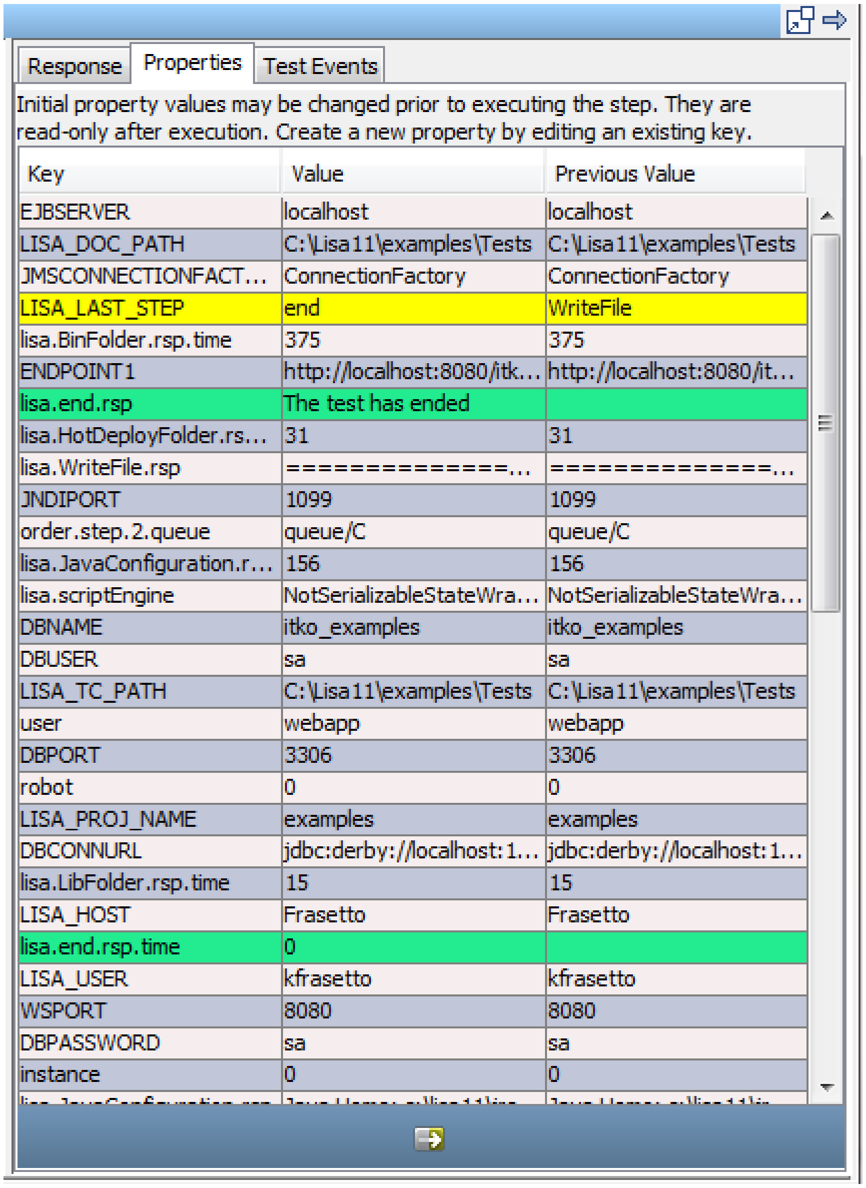
If you let the staging document control the

This graphic shows our example in the ITR utility.



```
<env:Envelope xmlns:env='http://schemas.xmlsoap.org/soap/envelope/
```

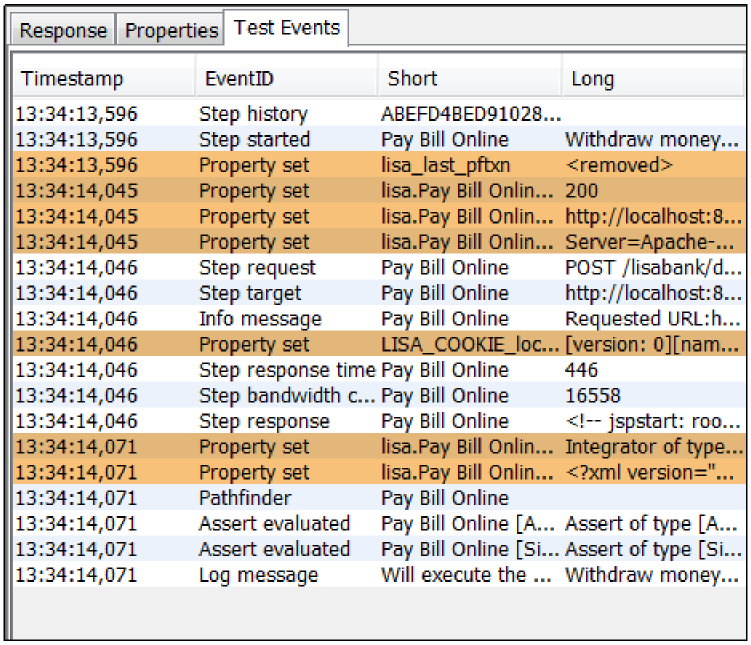
These graphics show the Properties tab in the ITR. The tab that is shown is for the step Get User.



Initial property values may be changed prior to executing the step. They are read-only after execution. Create a new property by editing an existing key.

Key	Value	Previous Value
EJBSERVER	localhost	localhost
LISA_DOC_PATH	C:\Lisa11\examples\Tests	C:\Lisa11\examples\Tests
JMSCONNECTIONFACT...	ConnectionFactory	ConnectionFactory
LISA_LAST_STEP	end	WriteFile
lisa.BinFolder.rsp.time	375	375
ENDPOINT1	http://localhost:8080/itk...	http://localhost:8080/it...
lisa.end.rsp	The test has ended	
lisa.HotDeployFolder.rs...	31	31
lisa.WriteFile.rsp	==============...	==============...
JNDIPORT	1099	1099
order.step.2.queue	queue/C	queue/C
lisa.JavaConfiguration.r...	156	156
lisa.scriptEngine	NotSerializableStateWra...	NotSerializableStateWra...
DBNAME	itko_examples	itko_examples
DBUSER	sa	sa
LISA_TC_PATH	C:\Lisa11\examples\Tests	C:\Lisa11\examples\Tests
user	webapp	webapp
DBPORT	3306	3306
robot	0	0
LISA_PROJ_NAME	examples	examples
DBCONNURL	jdbc:derby://localhost:1...	jdbc:derby://localhost:1...
lisa.LibFolder.rsp.time	15	15
LISA_HOST	Frasetto	Frasetto
lisa.end.rsp.time	0	
LISA_USER	kfrasetto	kfrasetto
WSPORT	8080	8080
DBPASSWORD	sa	sa
instance	0	0
lisa.JavaConfiguration.rsp	Java Home: c:\lisa11\jre	Java Home: c:\lisa11\jr

This graphic shows the Test Events tabs in the ITR.



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...

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

Detailed information about properties is available in these appendixes:

- Appendix A (see page 445) - DevTest Property File (lisa.properties)
- Appendix B (see page 489) - Custom Property Files (local.properties, site.properties)

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.

Properties are added to your configuration automatically while you develop your test.

For example, when you enter the WSDL name in a test, the defined server name and port are replaced with properties such as WSSERVER and WSPORT. The values of these properties are automatically added to your default project configuration. Now you can change the location of the web service merely by editing the configuration, rather than looking for hard-coded values in several test steps.

As another example, when you work with Enterprise JavaBeans (EJBs) or Java objects, you can switch hot deploy directories, or add extra JAR files to your class path, to use different versions of your Java code. Two standard properties exist for these locations: HOT_DEPLOY and MORE_JARS. You can set these properties in your configuration.

For information about other properties, see Properties (see page 86).

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 155).

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 105) in *Using CA Application Test.*

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.

The following topics are included.

Project Configuration (see page 102)
Add a Configuration (see page 103)
Mark a Configuration as Active (see page 103)
Edit a Configuration (see page 104)
Apply a Configuration While Running a Test Case (see page 105)

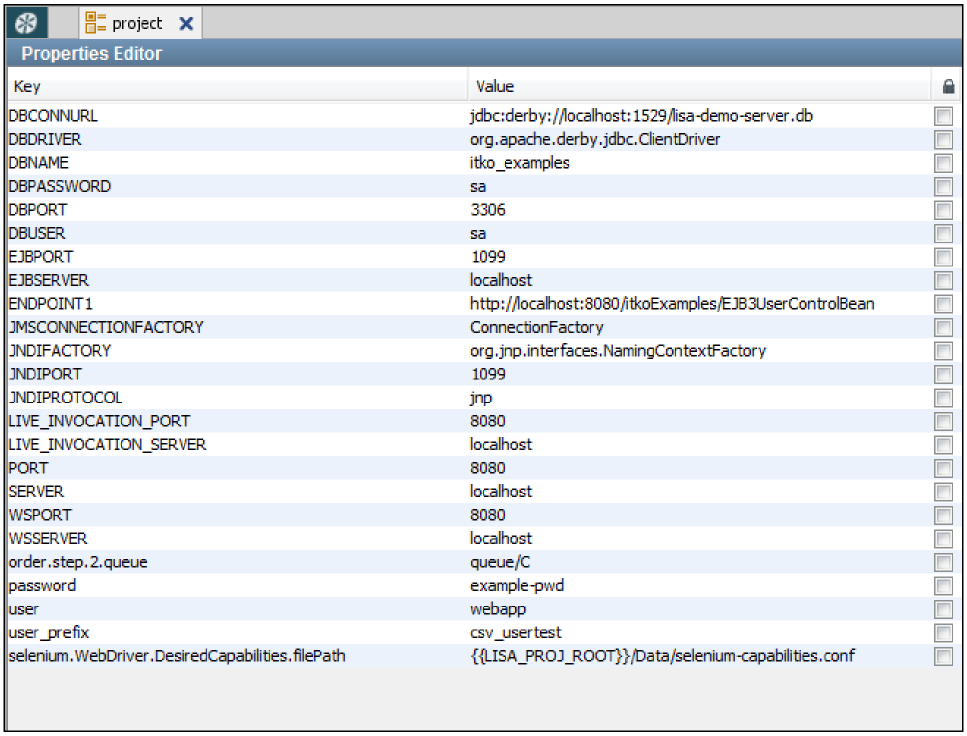
Project Configuration

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.



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	□
EJBSERVER	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	□
WSSERVER	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	□

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 103) 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.

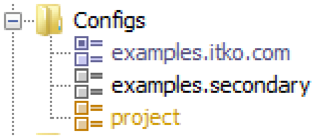
Follow these steps:

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.



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

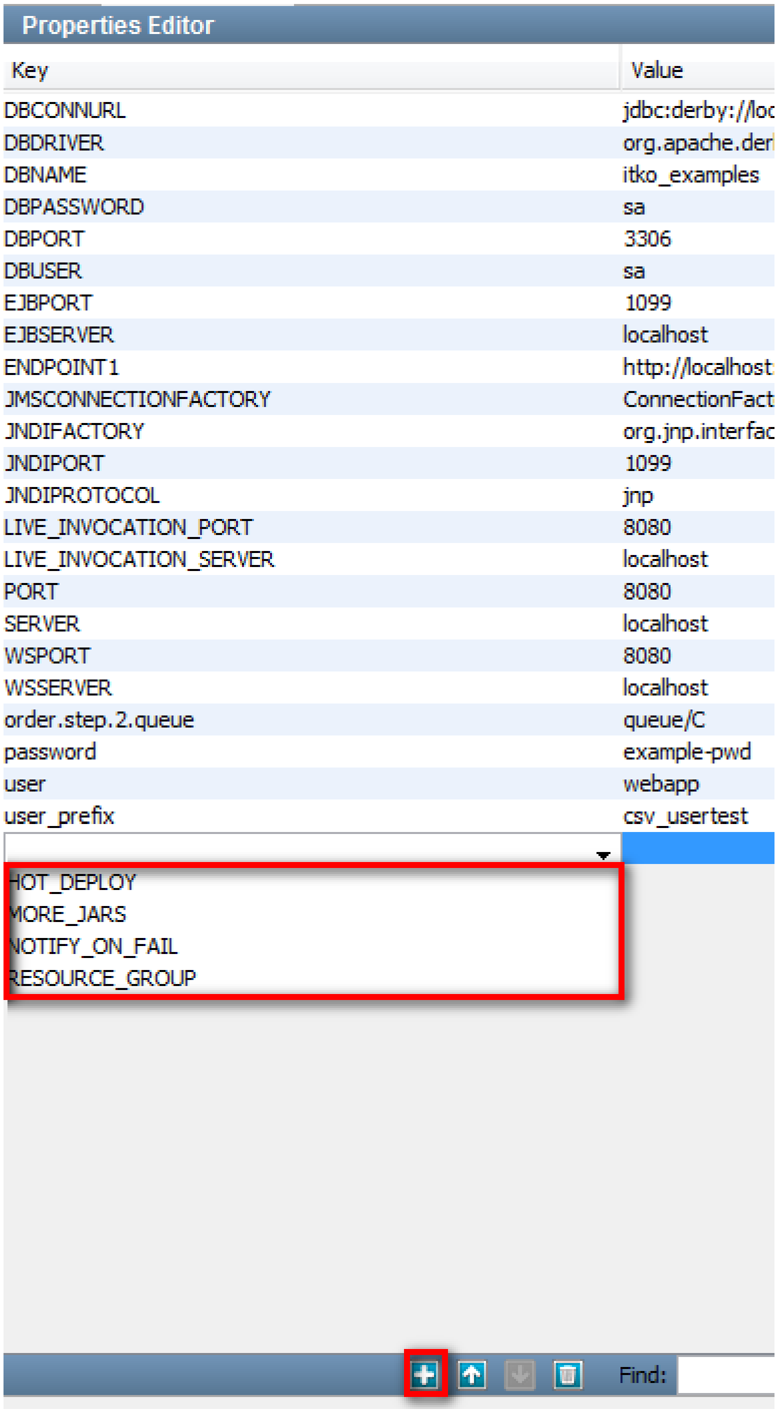
   Add more JAR files to your class path.

   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.



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

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 289) in *Using CA Application Test.*

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 100). 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.

Asset Browser

When you open a configuration, the asset browser is located to the right of the properties editor.

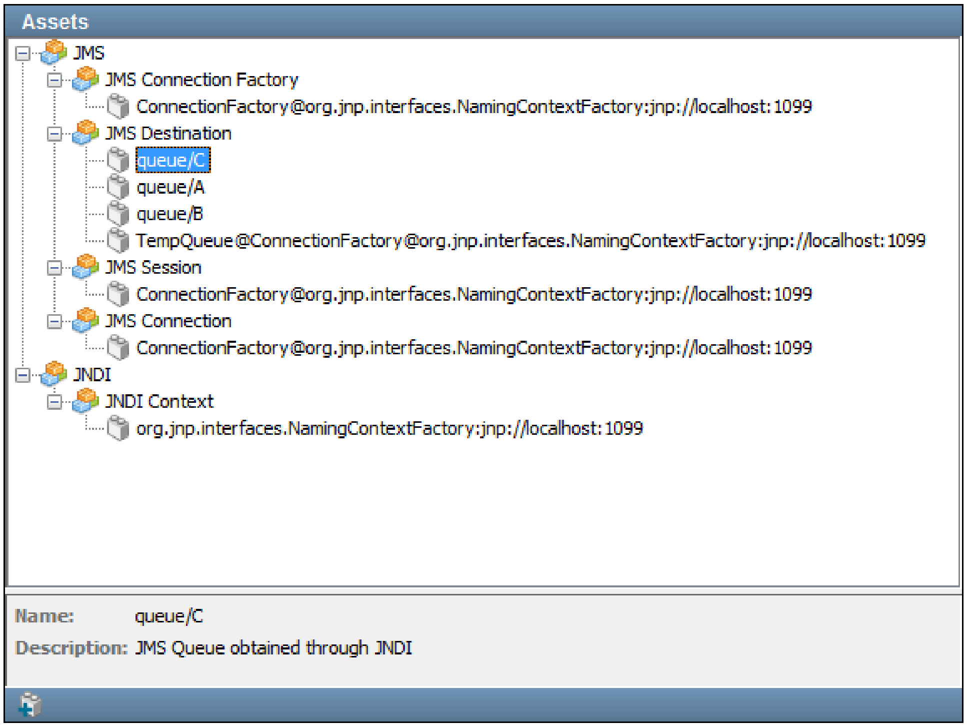
Note: If the asset browser is empty, then no assets have been created.

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.



- JMS
  - JMS Connection Factory
    - ConnectionFactory@org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099
  - JMS Destination
    - queue/C
    - queue/A
    - queue/B
    - TempQueue@ConnectionFactory@org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099
  - JMS Session
    - ConnectionFactory@org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099
  - JMS Connection
    - ConnectionFactory@org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099
- JNDI
  - JNDI Context
    - org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099

Name: queue/C

Description: JMS Queue obtained through JNDI

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.

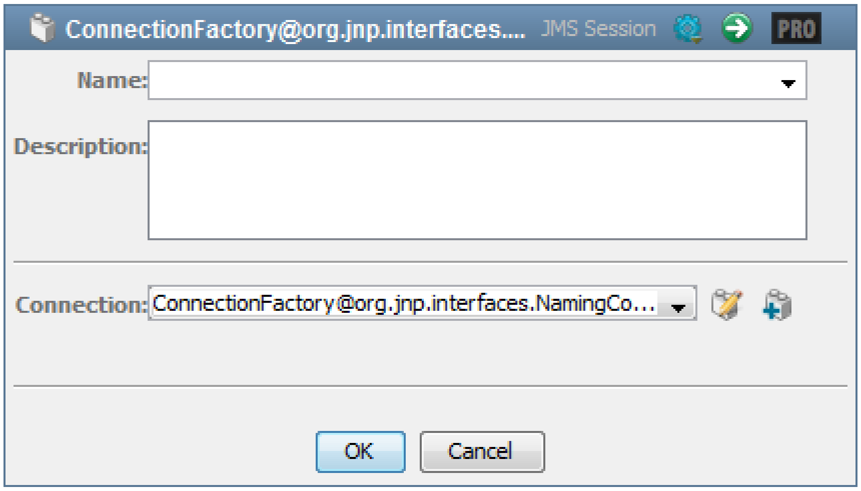
Asset Editor

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.



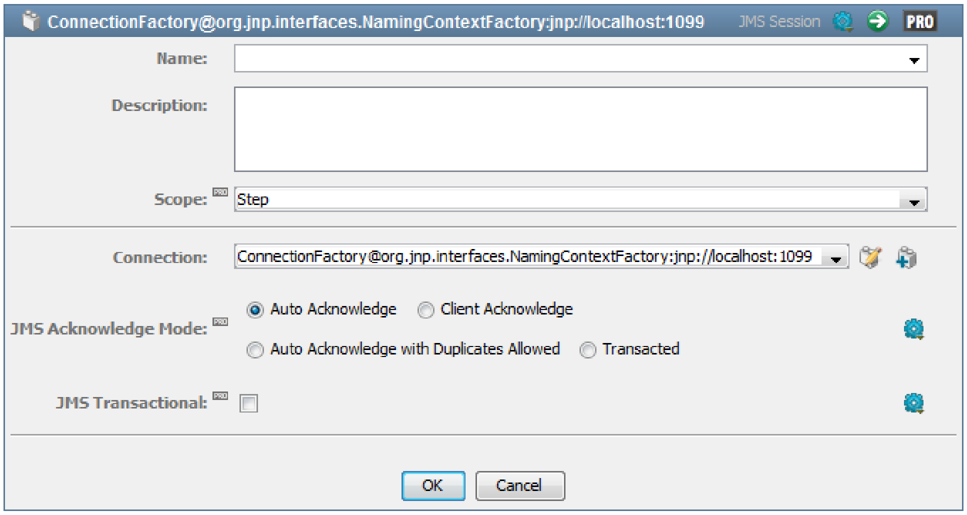
ConnectionFactory@org.jnp.interfaces.... JMS Session

Name:

Description:

Connection: ConnectionFactory@org.jnp.interfaces.NamingCo...

The following graphic shows the advanced mode. You can display the advanced mode by clicking PRO in the upper right corner.



ConnectionFactory@org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099 JMS Session

Name:

Description:

Scope: Step

Connection: ConnectionFactory@org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099

JMS Acknowledge Mode: ● Auto Acknowledge ○ Client Acknowledge ○ Auto Acknowledge with Duplicates Allowed ○ Transacted

JMS Transactional: □

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 113) the asset.

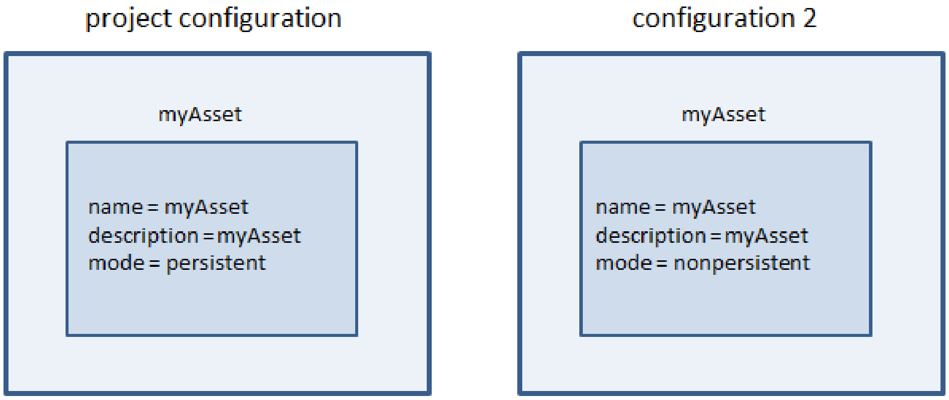
Asset Inheritance

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 107) 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 107) or the asset editor (see page 108).

Follow these steps:

1. Complete one of the following actions:
   - In the asset browser, click the Add button in the lower left corner and select the asset type.
   - In the asset editor, click the Add New Asset icon 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 topic 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 108) 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.

The following topics are included.

Add a Filter (see page 114)
Drag and Drop a Filter (see page 128)

Add a Filter

The ways to add a filter are:

Add a Filter Manually (see page 115)
Add a Filter from an HTTP Response (see page 117)
Add a Filter from a JDBC Result Set (see page 120)
Add a Filter from a Returned Java Object (see page 125)

Add a Filter Manually

To add a filter manually, select the filter type from a list and enter the parameters for the filter.

You can add two types of filters manually: global filters and step filters.

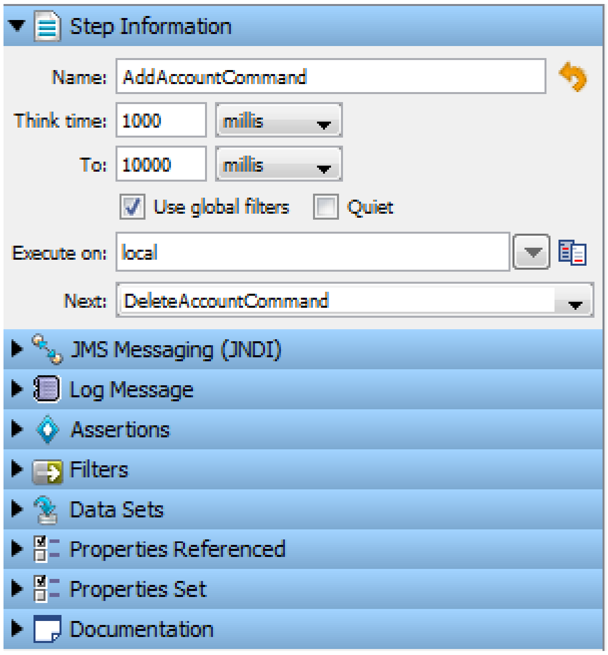
Global filters apply to and automatically run for every step in the test case, unless a step is instructed otherwise.

A step filter applies only to one step and executes for that step only.

To add a global filter manually:

1. To open the Test Case Elements panel, open a test case and click anywhere in the editor.
2. On the Global Filters element, click Add + to add a global filter.
3. You are prompted to select a filter type of Integration Support for CAI or Integration Support for webMethods Integration Server.

   For more information about adding each of these types of filters, see Integration Support for CAI or Integration Support for webMethods Integration Server in *Using CA Application Test.*
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.



Step Information

Name: AddAccountCommand

Think time: 1000 millis

To: 10000 millis

☑ Use global filters ☐ Quiet

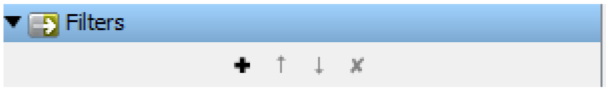
Execute on: local

Next: DeleteAccountCommand

To add a step filter manually:

Follow these steps:

1. Select the step for which you want to apply the filter and in the right panel, click the Filter element.

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 in *Reference*.

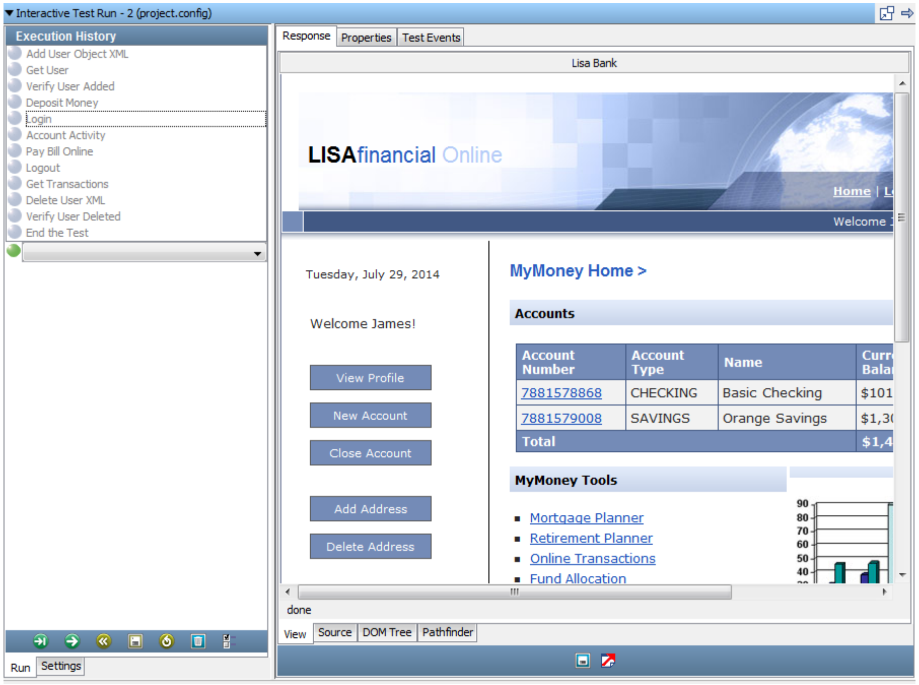
Add a Filter from an HTTP Response

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:

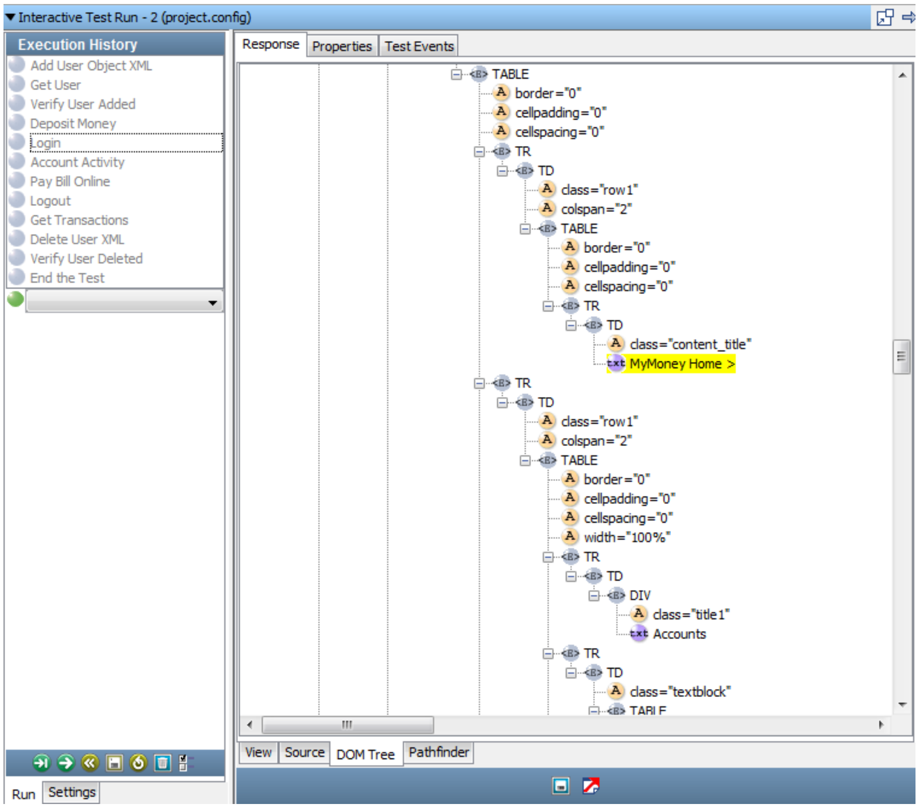
1. Run the multi-tier-combo test case in the ITR, and then select the Login test step.



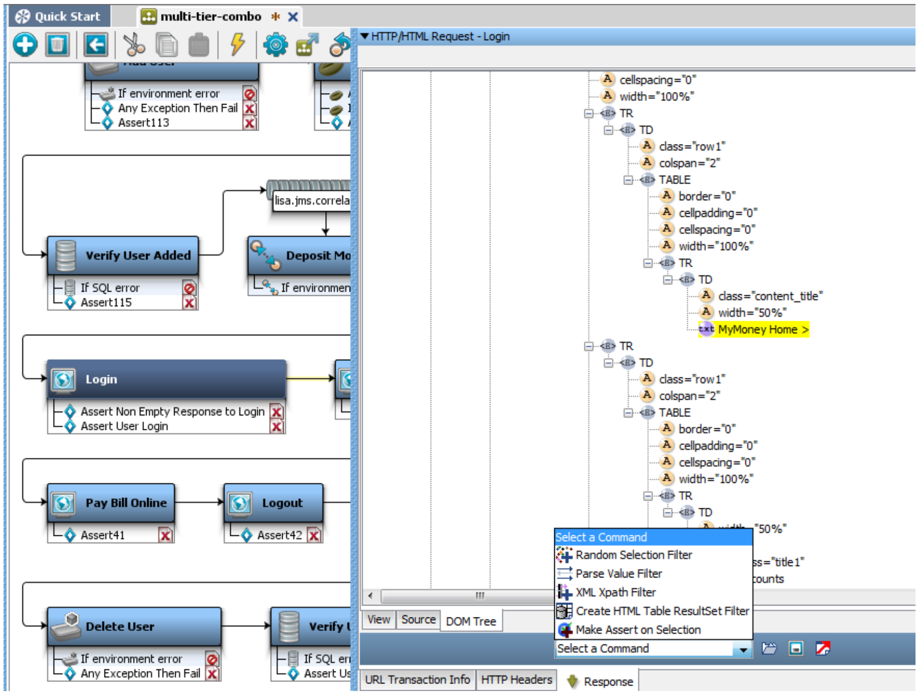
Account Number	Account Type	Name	Curr Bala
7881578868	CHECKING	Basic Checking	$101
7881579008	SAVINGS	Orange Savings	$1,3
Total			$1,4

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.



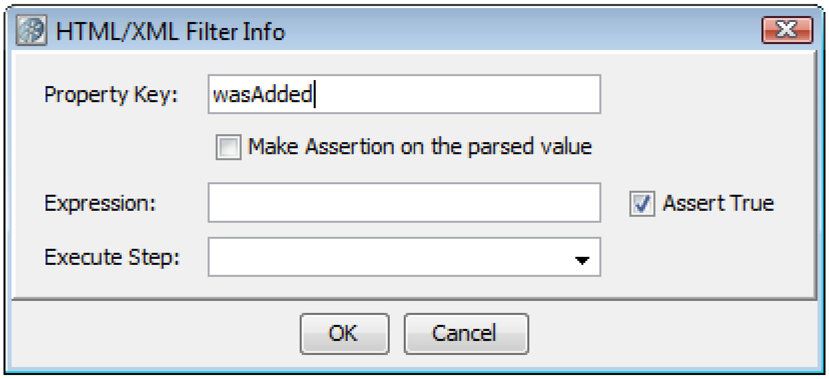
4. To apply an inline filter, double-click the login step in the model editor to open the HTTP/HTML Step Editor.



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

Property Key: wasAdded

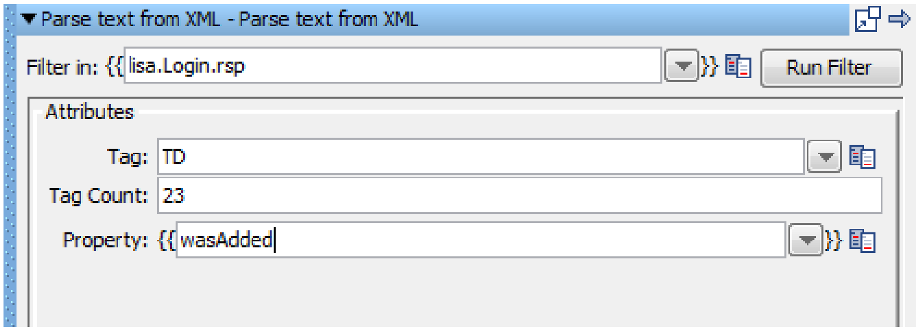
☐ Make Assertion on the parsed value

Expression:

☑ Assert True

Execute Step:

8. Click OK.



Parse text from XML - Parse text from XML

Filter in: {{lisa.Login.rsp}}

Attributes

Tag: TD

Tag Count: 23

Property: {{wasAdded}}

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.

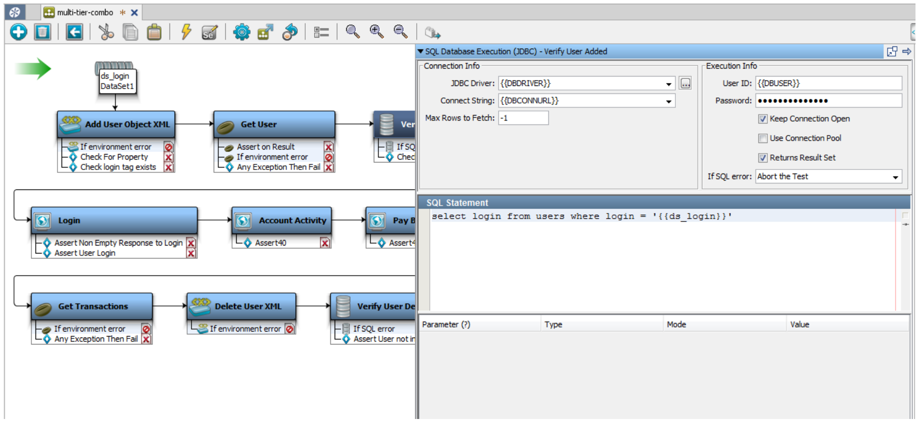
Add a Filter from a JDBC Result Set

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.



SQL Database Execution (JDBC) - Verify User Added

Connection Info

JDBC Driver: {{DBDRIVER}}

Connect String: {{DBCONNURL}}

Max Rows to Fetch: -1

Execution Info

User ID: {{DBUSER}}

Password: ••••••••••••••

☑ Keep Connection Open

☐ Use Connection Pool

☑ Returns Result Set

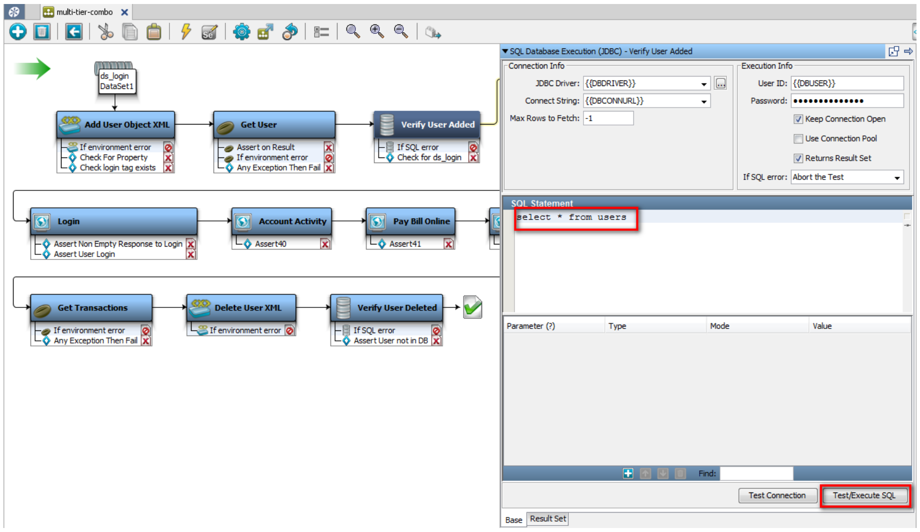
If SQL error: Abort the Test

SQL Statement

```
select login from users where login = '{{ds_login}}'
```

Parameter (?)	Type	Mode	Value

2. To get values in the result set, edit the SQL statement to read **select * from users** and click Test/Execute SQL.



SQL Database Execution (JDBC) - Verify User Added

Connection Info

JDBC Driver: {{DBDRIVER}}

Connect String: {{DBCONNURL}}

Max Rows to Fetch: -1

Execution Info

User ID: {{DBUSER}}

Password: ••••••••••••••

☑ Keep Connection Open

☐ Use Connection Pool

☑ Returns Result Set

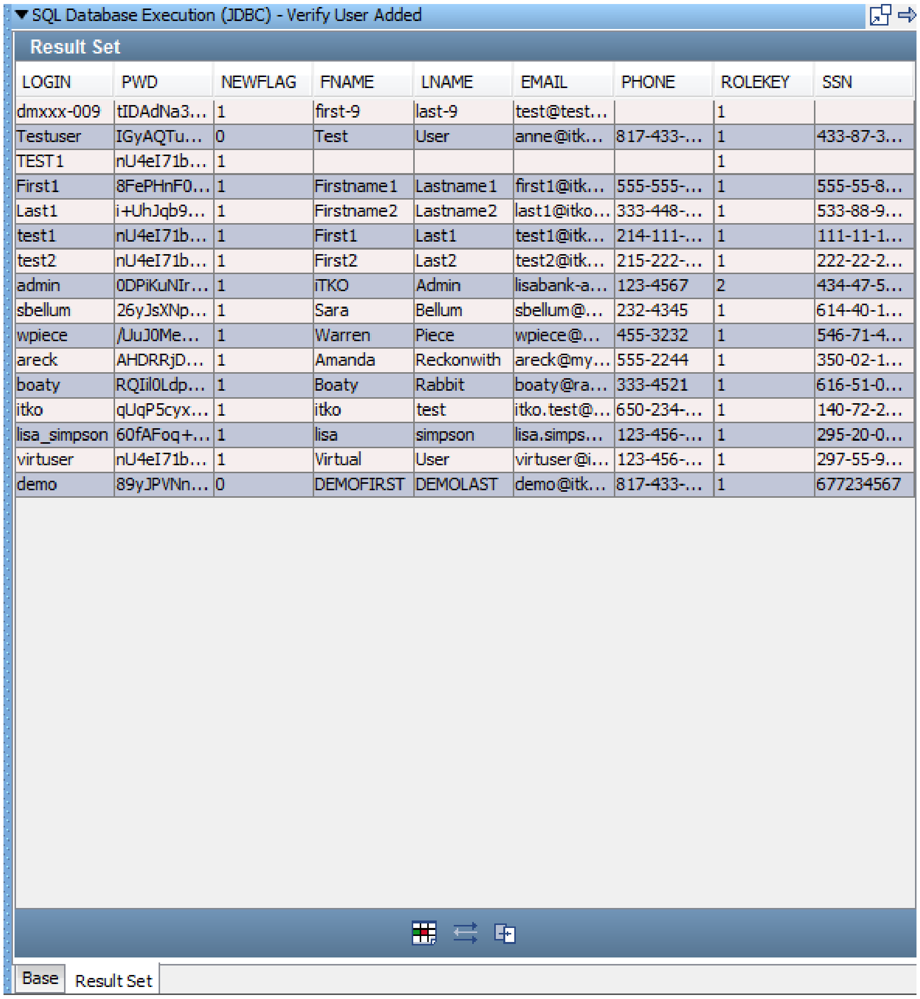
If SQL error: Abort the Test

SQL Statement

```
select * from users
```

Parameter (?)	Type	Mode	Value

3. To get values in the result set, click the Result Set tab and click Test/Execute SQL.



SQL Database Execution (JDBC) - Verify User Added

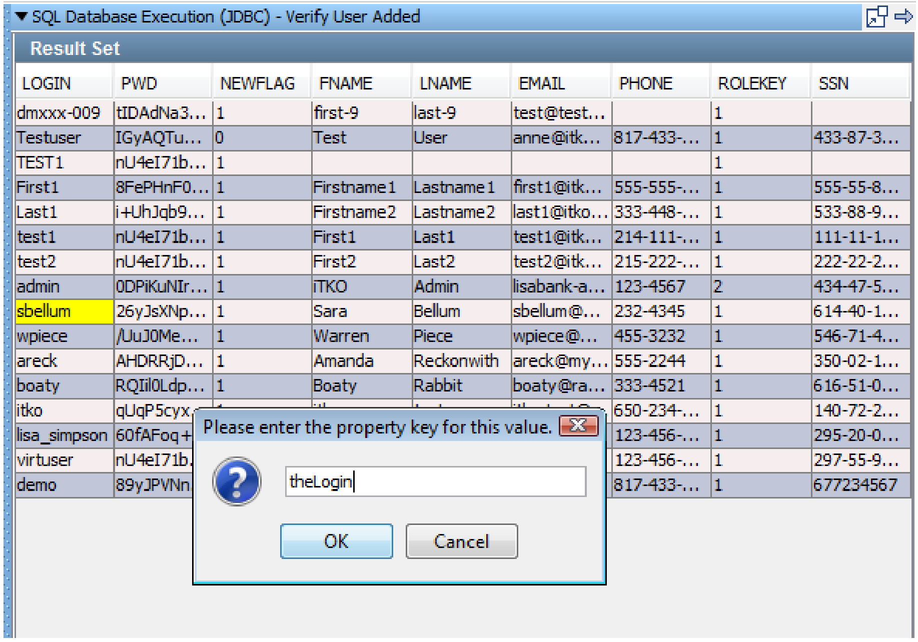
Result Set

LOGIN	PWD	NEWFLAG	FNAME	LNAME	EMAIL	PHONE	ROLEKEY	SSN
dmxxx-009	tIDAdNa3…	1	first-9	last-9	test@test…		1	
Testuser	IGyAQTu…	0	Test	User	anne@itk…	817-433-…	1	433-87-3…
TEST1	nU4eI71b…	1					1	
First1	8FePHnF0…	1	Firstname1	Lastname1	first1@itk…	555-555-…	1	555-55-8…
Last1	i+UhJqb9…	1	Firstname2	Lastname2	last1@itko…	333-448-…	1	533-88-9…
test1	nU4eI71b…	1	First1	Last1	test1@itk…	214-111-…	1	111-11-1…
test2	nU4eI71b…	1	First2	Last2	test2@itk…	215-222-…	1	222-22-2…
admin	0DPiKuNIr…	1	iTKO	Admin	lisabank-a…	123-4567	2	434-47-5…
sbellum	26yJsXNp…	1	Sara	Bellum	sbellum@…	232-4345	1	614-40-1…
wpiece	/UuJ0Me…	1	Warren	Piece	wpiece@…	455-3232	1	546-71-4…
areck	AHDRRjD…	1	Amanda	Reckonwith	areck@my…	555-2244	1	350-02-1…
boaty	RQIil0Ldp…	1	Boaty	Rabbit	boaty@ra…	333-4521	1	616-51-0…
itko	qUqP5cyx…	1	itko	test	itko.test@…	650-234-…	1	140-72-2…
lisa_simpson	60fAFoq+…	1	lisa	simpson	lisa.simps…	123-456-…	1	295-20-0…
virtuser	nU4eI71b…	1	Virtual	User	virtuser@i…	123-456-…	1	297-55-9…
demo	89yJPVNn…	0	DEMOFIRST	DEMOLAST	demo@itk…	817-433-…	1	677234567

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	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					650-234-…	1	140-72-2…
lisa_simpson	60fAFoq+					123-456-…	1	295-20-0…
virtuser	nU4eI71b.					123-456-…	1	297-55-9…
demo	89yJPVNn					817-433-…	1	677234567

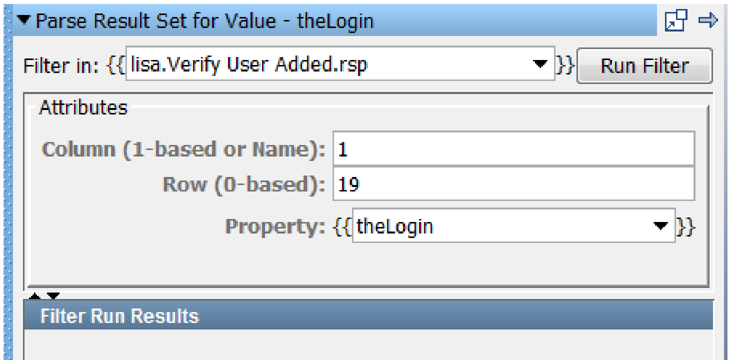
Please enter the property key for this value.

theLogin

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}}

Attributes

Column (1-based or Name): 1

Row (0-based): 19

Property: {{theLogin}}

Filter Run Results

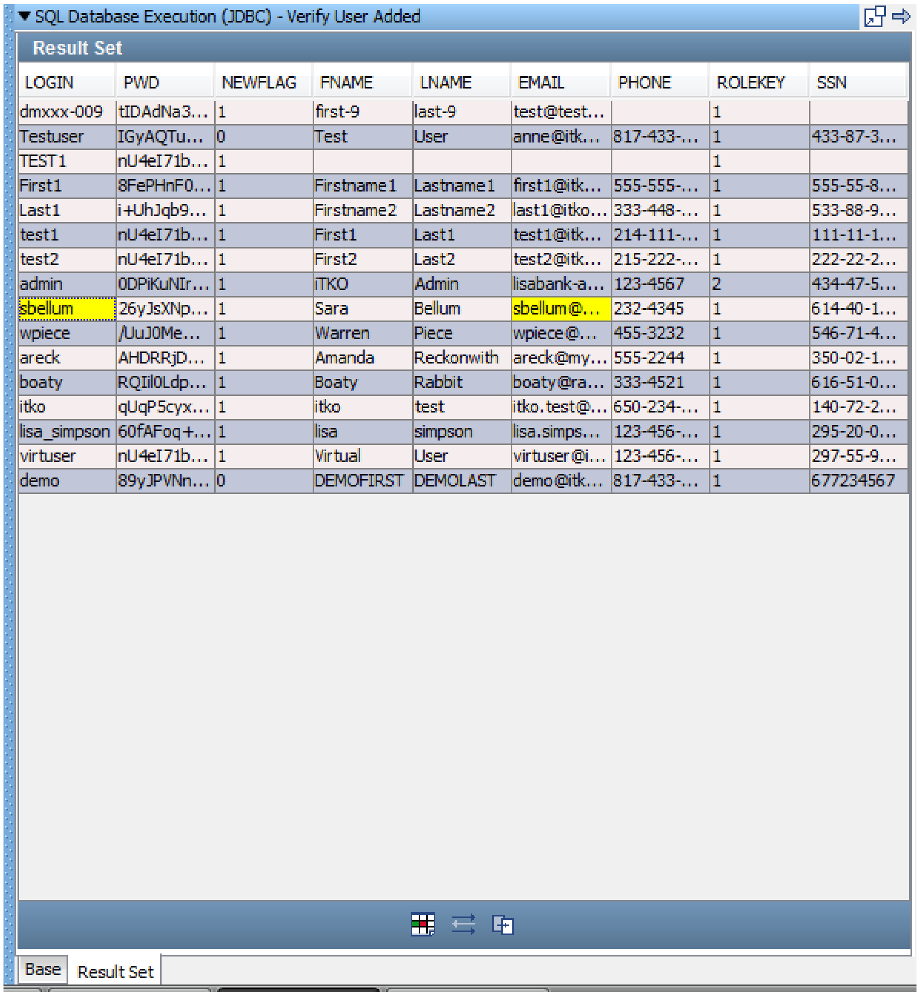
In the example, the value in the cell in the first column, and eighth row, **sbellum**, is stored in the property theLogin.

Applying a Second Filter

A second filter can be applied here. You can look for a value in one column of the result set, and then capture a value from another column in the same row.

Follow these steps:

1. From the result set, select the two values in two different columns from the same row, using the Ctrl key.



SQL Database Execution (JDBC) - Verify User Added

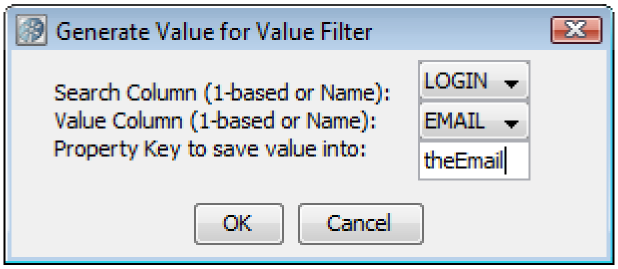
Result Set

LOGIN	PWD	NEWFLAG	FNAME	LNAME	EMAIL	PHONE	ROLEKEY	SSN
dmxxx-009	tIDAdNa3…	1	first-9	last-9	test@test…		1	
Testuser	IGyAQTu…	0	Test	User	anne@itk…	817-433-…	1	433-87-3…
TEST1	nU4eI71b…	1					1	
First1	8FePHnF0…	1	Firstname1	Lastname1	first1@itk…	555-555-…	1	555-55-8…
Last1	i+UhJqb9…	1	Firstname2	Lastname2	last1@itko…	333-448-…	1	533-88-9…
test1	nU4eI71b…	1	First1	Last1	test1@itk…	214-111-…	1	111-11-1…
test2	nU4eI71b…	1	First2	Last2	test2@itk…	215-222-…	1	222-22-2…
admin	0DPiKuNIr…	1	iTKO	Admin	lisabank-a…	123-4567	2	434-47-5…
sbellum	26yJsXNp…	1	Sara	Bellum	sbellum@…	232-4345	1	614-40-1…
wpiece	/UuJ0Me…	1	Warren	Piece	wpiece@…	455-3232	1	546-71-4…
areck	AHDRRjD…	1	Amanda	Reckonwith	areck@my…	555-2244	1	350-02-1…
boaty	RQIil0Ldp…	1	Boaty	Rabbit	boaty@ra…	333-4521	1	616-51-0…
itko	qUqP5cyx…	1	itko	test	itko.test@…	650-234-…	1	140-72-2…
lisa_simpson	60fAFoq+…	1	lisa	simpson	lisa.simps…	123-456-…	1	295-20-0…
virtuser	nU4eI71b…	1	Virtual	User	virtuser@i…	123-456-…	1	297-55-9…
demo	89yJPVNn…	0	DEMOFIRST	DEMOLAST	demo@itk…	817-433-…	1	677234567

2. Select Filter for a value and then get another column value filter using the ▦ icon. To create this filter, select two cells in the same row. One is the search column and the other is the column whose value you want to extract.

3. In the dialog that opens, select or reassign the columns for the search and the value.

4. Enter the property key *theEmail*.



Generate Value for Value Filter

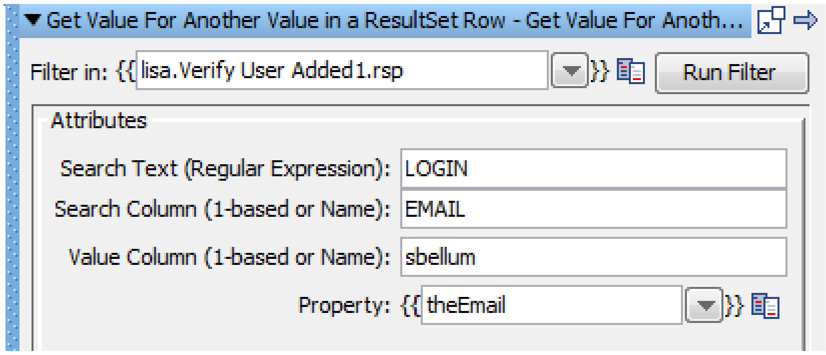
Search Column (1-based or Name): LOGIN

Value Column (1-based or Name): EMAIL

Property Key to save value into: theEmail

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 - Get Value For Anoth...
Run Filter
Attributes

Filter in: {{lisa.Verify User Added1.rsp}}

Search Text (Regular Expression): LOGIN

Search Column (1-based or Name): EMAIL

Value Column (1-based or Name): sbellum

Property: {{theEmail}}

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.

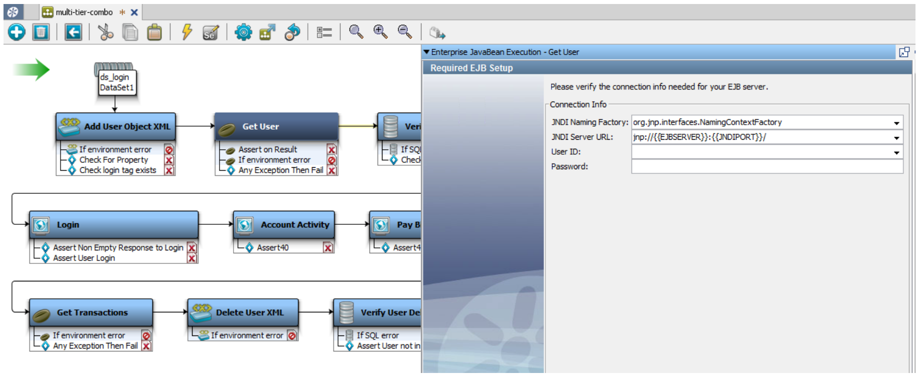
Add a Filter from a Returned Java Object

When the test step result is a Java object, you can use the Inline Filter panel in the Complex Object Editor (see page 158) 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.



Please verify the connection info needed for your EJB server.

Connection Info

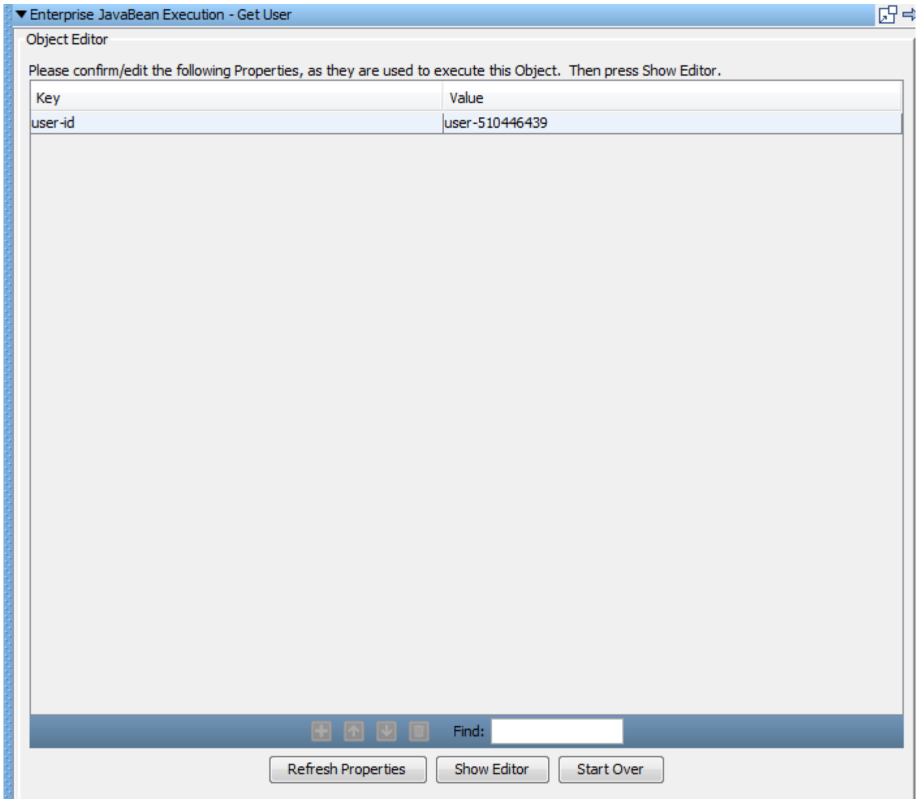
JNDI Naming Factory: org.jnp.interfaces.NamingContextFactory

JNDI Server URL: jnp://{{EJBSERVER}}:{{JNDIPORT}}/

User ID:

Password:

2. Click Next. On the next window, click Finish.



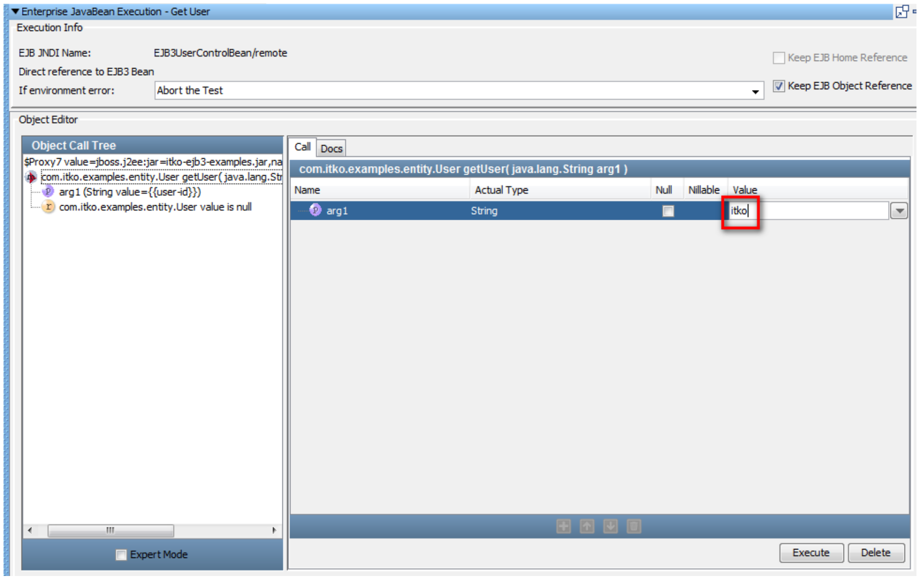
Enterprise JavaBean Execution - Get User

Object Editor

Please confirm/edit the following Properties, as they are used to execute this Object. Then press Show Editor.

Key	Value
user-id	user-510446439

3. To open the Object Call Tree, click Show Editor.



Enterprise JavaBean Execution - Get User

Execution Info

EJB JNDI Name: EJB3UserControlBean/remote

Direct reference to EJB3 Bean

If environment error: Abort the Test

□ Keep EJB Home Reference

☑ Keep EJB Object Reference

Object Editor

Object Call Tree

$Proxy7 value=jboss.j2ee:jar=itko-ejb3-examples.jar,na
com.itko.examples.entity.User getUser(java.lang.Str
arg1 (String value={{user-id}})
com.itko.examples.entity.User value is null

com.itko.examples.entity.User getUser(java.lang.String arg1)

Name	Actual Type	Null	Nillable	Value
arg1	String	□		itko

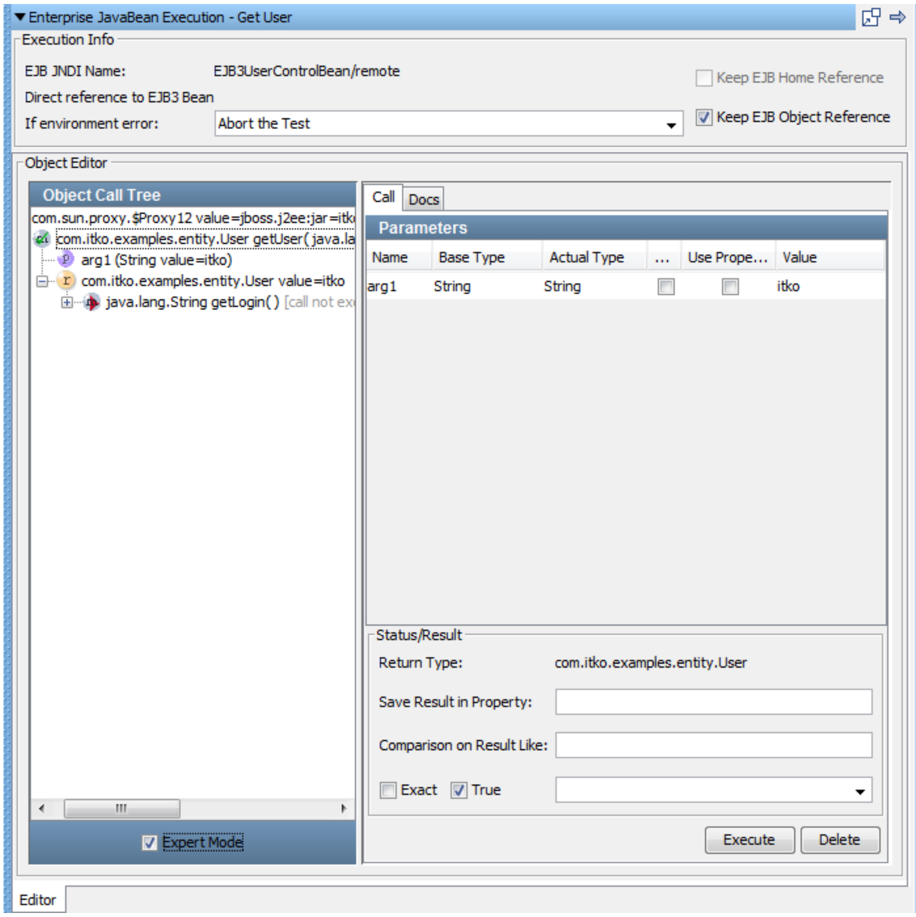
4. Enter an input parameter "itko" in the value field.

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.



Enterprise JavaBean Execution - Get User

Execution Info

EJB JNDI Name: EJB3UserControlBean/remote

Direct reference to EJB3 Bean

If environment error: Abort the Test

□ Keep EJB Home Reference

☑ Keep EJB Object Reference

Object Editor

Object Call Tree

com.sun.proxy.$Proxy12 value=jboss.j2ee:jar=itk

com.itko.examples.entity.User getUser(java.la

arg1 (String value=itko)

com.itko.examples.entity.User value=itko

java.lang.String getLogin( ) [call not ex

☑ Expert Mode

Parameters

Name	Base Type	Actual Type	...	Use Prope...	Value
arg1	String	String	□	□	itko

Status/Result

Return Type: com.itko.examples.entity.User

Save Result in Property:

Comparison on Result Like:

□ Exact ☑ True

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 158) in *Using CA Application Test.*

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.

This section contains the following topics:

Add an Assertion (see page 130)
Assertions Toolbar (see page 143)
Reorder an Assertion (see page 144)
Rename an Assertion (see page 144)
Drag and Drop an Assertion (see page 144)
Configure the Next Step of an Assertion (see page 145)

Add an Assertion

You can add assertions to the test case in the following ways:

Add an Assertion Manually (see page 131)
Add an Assertion from an HTTP Response (see page 134)
Add an Assertion from a JDBC Result Set (see page 138)
Add an Assertion for Returned Java Object (see page 140)
Add an Assertion to a Mobile Test Step (see page 142)

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.

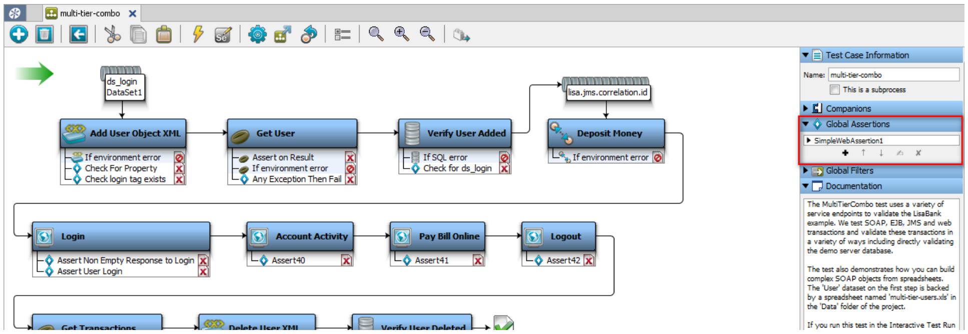
To add a global assertion:

1. Open a test case and in the right panel click the Global Assertions element.

You can apply the following types of global assertions:

- **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.



Name: multi-tier-combo

□ This is a subprocess

The MultiTierCombo test uses a variety of service endpoints to validate the LisaBank example. We test SOAP, EJB, JMS and web transactions and validate these transactions in a variety of ways including directly validating the demo server database.

The test also demonstrates how you can build complex SOAP objects from spreadsheets. The 'User' dataset on the first step is backed by a spreadsheet named 'multi-tier-users.xls' in the 'Data' folder of the project.

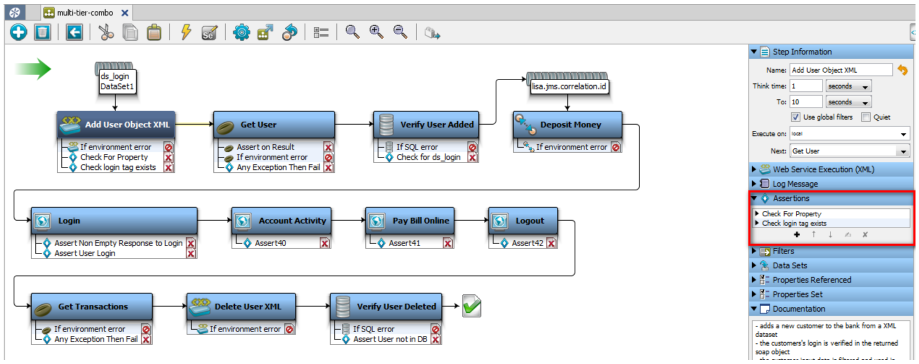
If you run this test in the Interactive Test Run

To add a step assertion:

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.

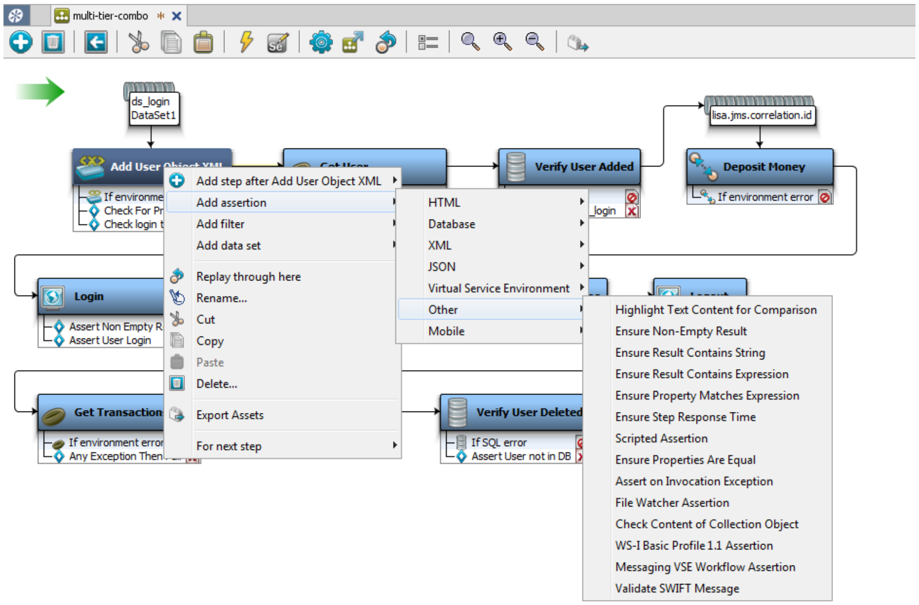


Name: Add User Object XML
Think time: 1 seconds
To: 10 seconds
☑ Use global filters □ Quiet
Execute on: local
Next: Get User

- adds a new customer to the bank from a XML dataset
- the customers's login is verified in the returned soap object

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.



To select and edit an assertion:

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.

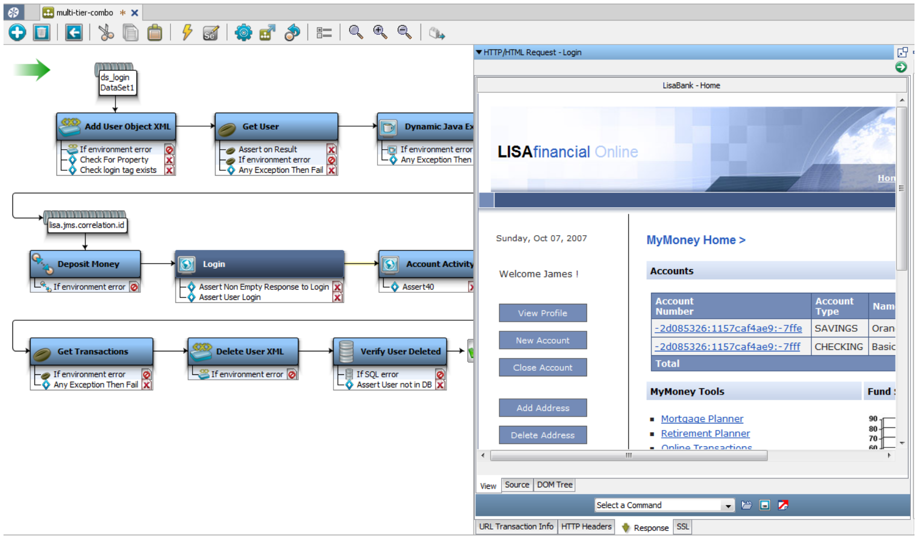
Add an Assertion from an HTTP Response

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.

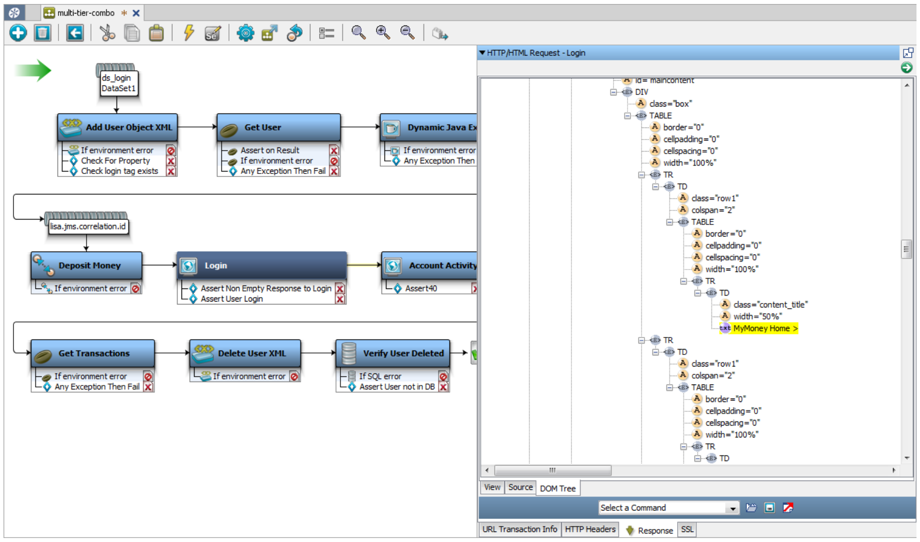
Follow these steps:

1. Run the multi-tier-combo test case in the ITR.
2. Double-click the Login step in the model editor.

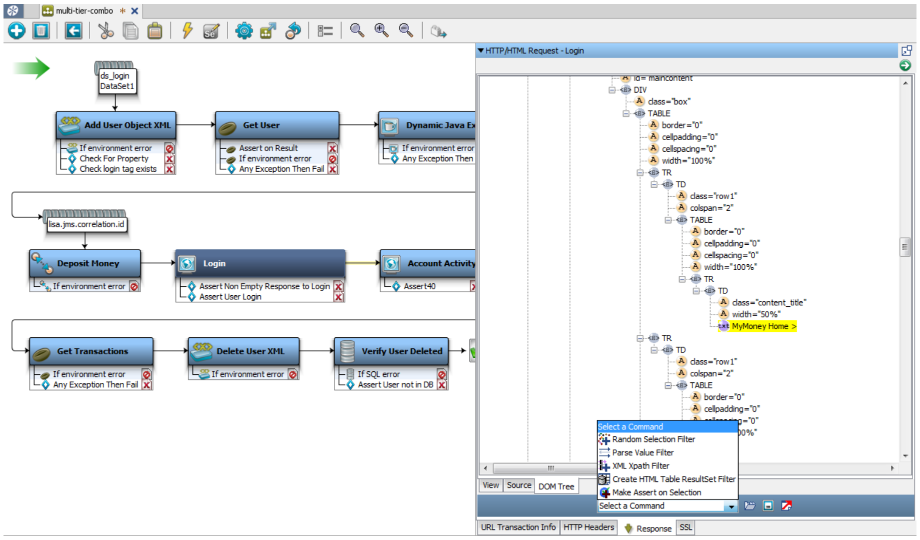


Account Number	Account Type	Nam
-2d085326:1157caf4ae9:-7ffe	SAVINGS	Oran
-2d085326:1157caf4ae9:-7fff	CHECKING	Basic
Total		

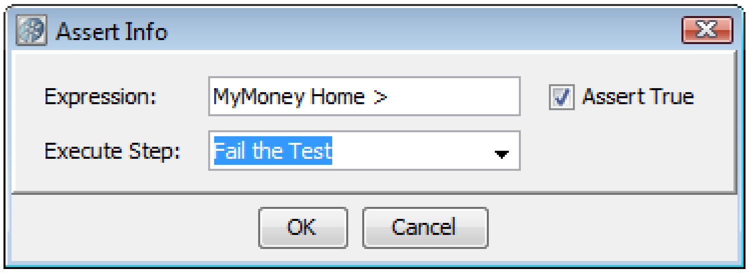
3. Select the text "MyMoney Home" in the View tab.
4. To view and verify that this text is selected in the tree, click the DOM Tree tab.



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.



Assert Info

Expression: MyMoney Home >

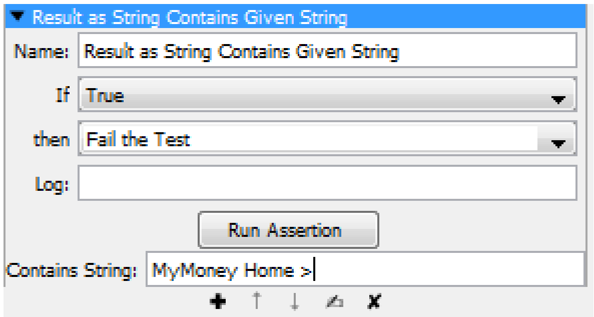
☑ Assert True

Execute Step: Fail the Test

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.



Result as String Contains Given String

Name: Result as String Contains Given String

If: True

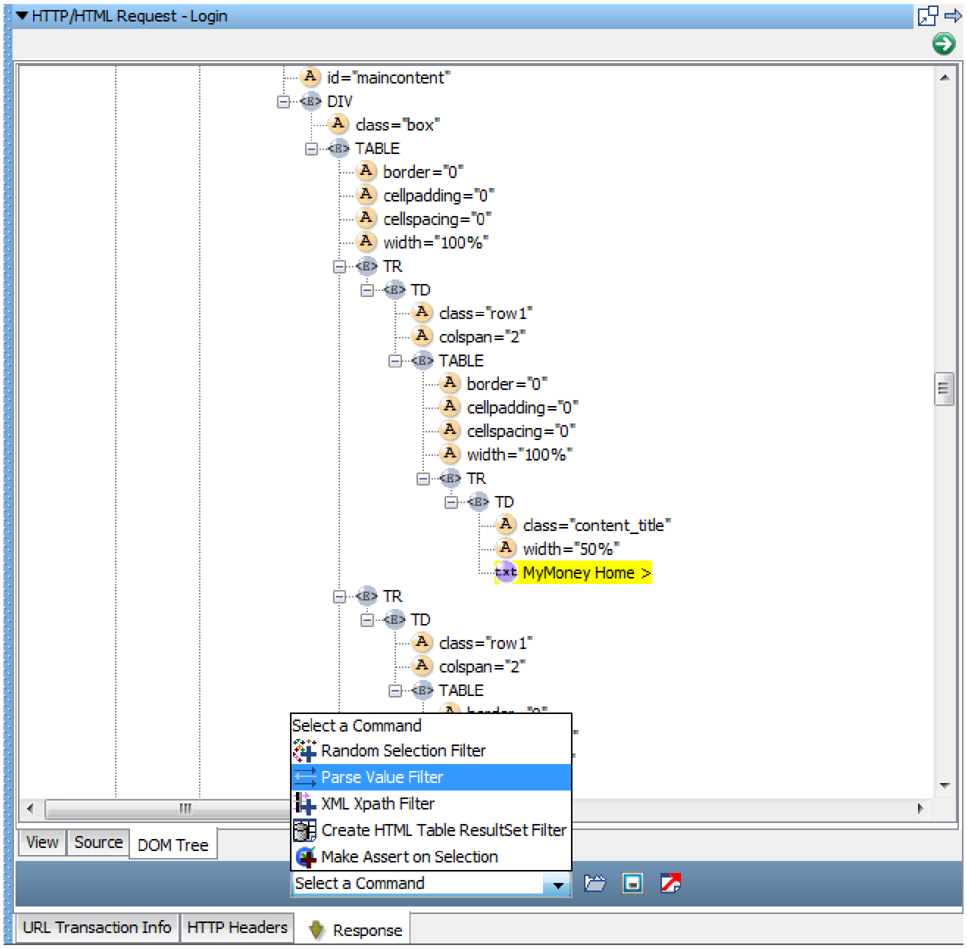
then: Fail the Test

Log:

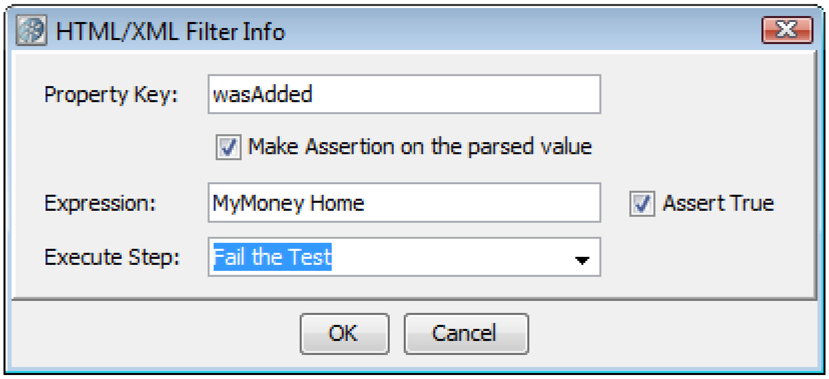
Contains String: MyMoney Home >

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.



HTML/XML Filter Info

Property Key: wasAdded

☑ Make Assertion on the parsed value

Expression: MyMoney Home ☑ Assert True

Execute Step: Fail the Test

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.

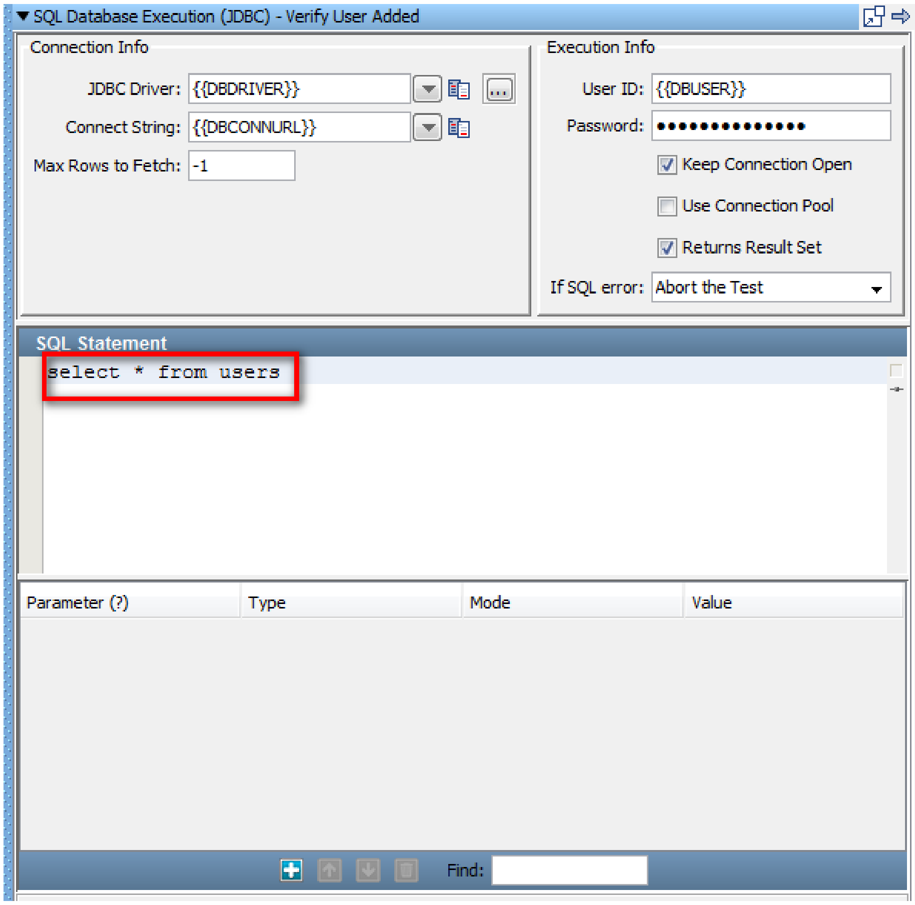
Add an Assertion from a JDBC Result Set

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**.



SQL Database Execution (JDBC) - Verify User Added

Connection Info

JDBC Driver: {{DBDRIVER}}

Connect String: {{DBCONNURL}}

Max Rows to Fetch: -1

Execution Info

User ID: {{DBUSER}}

Password: ••••••••••••••

☑ Keep Connection Open

□ Use Connection Pool

☑ Returns Result Set

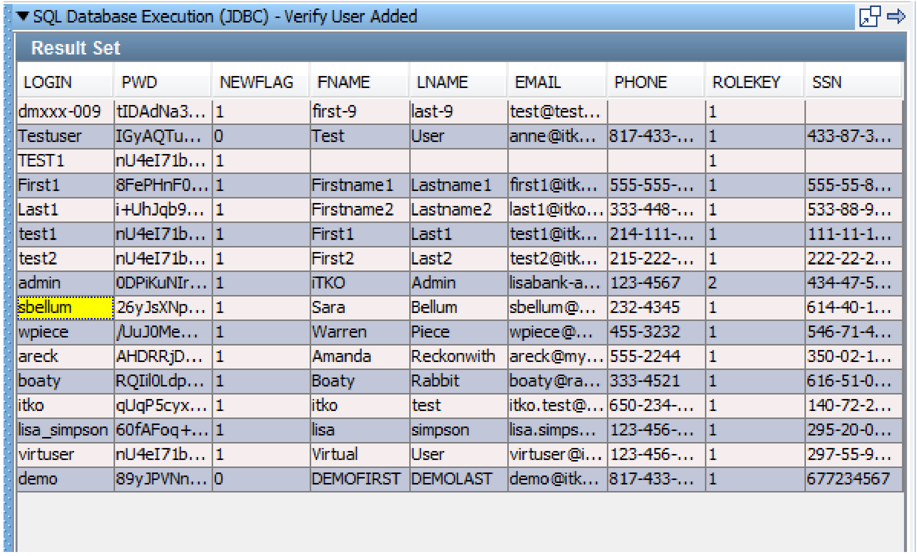
If SQL error: Abort the Test

SQL Statement

```
select * from users
```

Parameter (?)	Type	Mode	Value

2. To run the query, click the Test/Execute SQL button.
3. Select the Result Set tab and click the cell in the result set that represents the information that you want to test for (for example, **sbellum**).



SQL Database Execution (JDBC) - Verify User Added

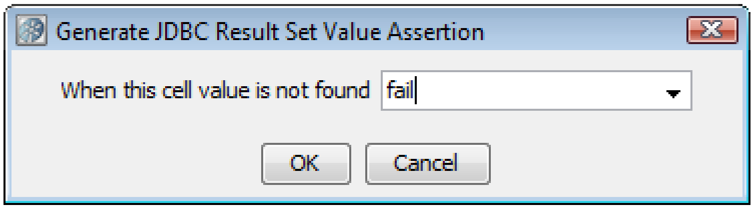
Result Set

LOGIN	PWD	NEWFLAG	FNAME	LNAME	EMAIL	PHONE	ROLEKEY	SSN
dmxxx-009	tIDAdNa3...	1	first-9	last-9	test@test...		1	
Testuser	IGyAQTu...	0	Test	User	anne@itk...	817-433-...	1	433-87-3...
TEST1	nU4eI71b...	1					1	
First1	8FePHnF0...	1	Firstname1	Lastname1	first1@itk...	555-555-...	1	555-55-8...
Last1	i+UhJqb9...	1	Firstname2	Lastname2	last1@itko...	333-448-...	1	533-88-9...
test1	nU4eI71b...	1	First1	Last1	test1@itk...	214-111-...	1	111-11-1...
test2	nU4eI71b...	1	First2	Last2	test2@itk...	215-222-...	1	222-22-2...
admin	0DPiKuNIr...	1	iTKO	Admin	lisabank-a...	123-4567	2	434-47-5...
sbellum	26yJsXNp...	1	Sara	Bellum	sbellum@...	232-4345	1	614-40-1...
wpiece	/UuJ0Me...	1	Warren	Piece	wpiece@...	455-3232	1	546-71-4...
areck	AHDRRjD...	1	Amanda	Reckonwith	areck@my...	555-2244	1	350-02-1...
boaty	RQIil0Ldp...	1	Boaty	Rabbit	boaty@ra...	333-4521	1	616-51-0...
itko	qUqP5cyx...	1	itko	test	itko.test@...	650-234-...	1	140-72-2...
lisa_simpson	60fAFoq+...	1	lisa	simpson	lisa.simps...	123-456-...	1	295-20-0...
virtuser	nU4eI71b...	1	Virtual	User	virtuser@i...	123-456-...	1	297-55-9...
demo	89yJPVNn...	0	DEMOFIRST	DEMOLAST	demo@itk...	817-433-...	1	677234567

4. Click Generate Assertions for the Value of a Cell in the toolbar below the Result set window.

   We want to test that **sbellum** appears in a cell in the LOGIN column.

5. In the Generate JDBC Result Set Value Assertion dialog, enter the test step (fail) to redirect if the value is not found:

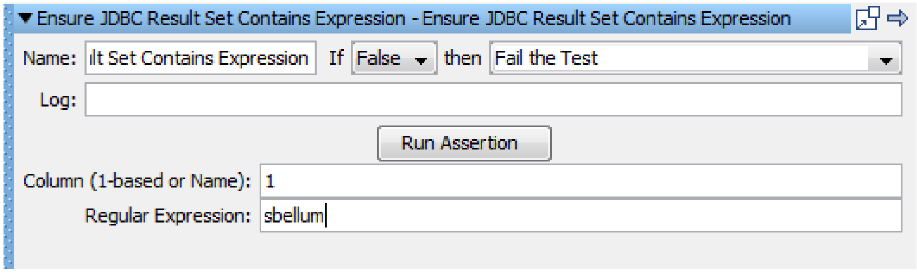


Generate JDBC Result Set Value Assertion

When this cell value is not found: fail

DevTest creates an assertion with the name Ensure JDBC Result Set Contains Expression in the Verify User Added step.

Figure 1: Assertion: Adding an assertion from a JDBC Result Set showing assertion added



Ensure JDBC Result Set Contains Expression - Ensure JDBC Result Set Contains Expression

Name: ilt Set Contains Expression If: False then: Fail the Test

Log:

Column (1-based or Name): 1

Regular Expression: sbellum

Note: The same assertion capabilities are available when a JDBC result set is displayed in the step editor.

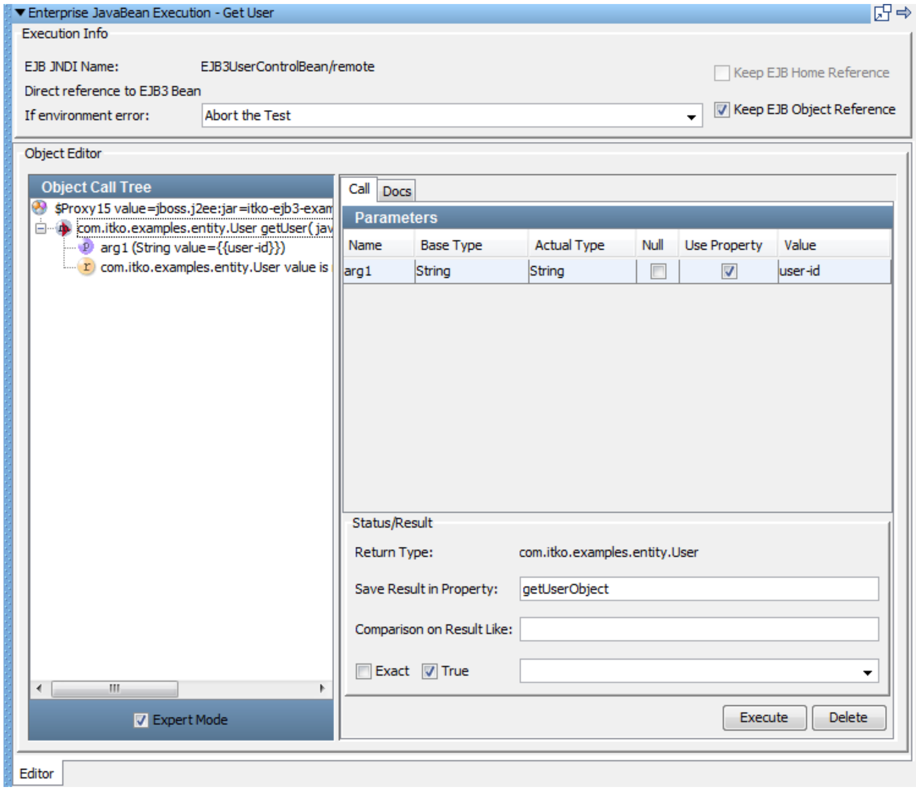
Add an Assertion for Returned Java Object

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.



Enterprise JavaBean Execution - Get User

Execution Info

EJB JNDI Name: EJB3UserControlBean/remote

Direct reference to EJB3 Bean

If environment error: Abort the Test

□ Keep EJB Home Reference

☑ Keep EJB Object Reference

Object Editor

Object Call Tree

$Proxy15 value=jboss.j2ee:jar=itko-ejb3-exam

com.itko.examples.entity.User getUser(jav

arg1 (String value={{user-id}})

com.itko.examples.entity.User value is

☑ Expert Mode

Parameters

Name	Base Type	Actual Type	Null	Use Property	Value
arg1	String	String	□	☑	user-id

Status/Result

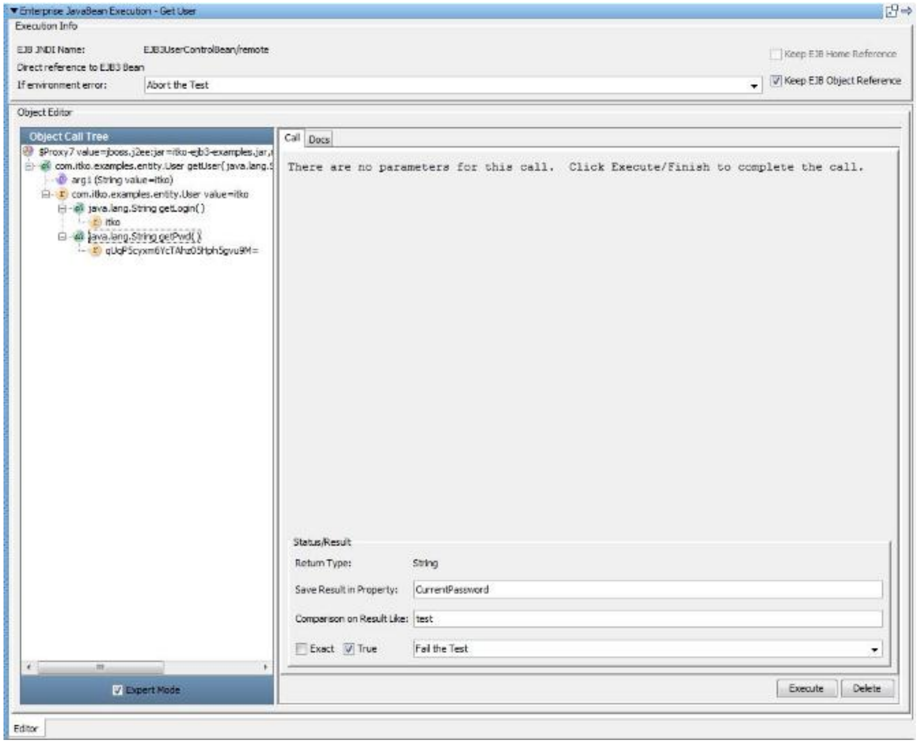
Return Type: com.itko.examples.entity.User

Save Result in Property: getUserObject

Comparison on Result Like:

□ Exact ☑ True

2. To open the Status/Result pane where you can add the assertion, select the Expert Mode check box in the left pane.



EJB JNDI Name: EJB3UserControlBean/remote

Direct reference to EJB3 Bean

If environment error: Abort the Test

□ Keep EJB Home Reference

☑ Keep EJB Object Reference

```
$Proxy7 value=jboss.j2ee:jar=itko-ejb3-examples.jar,
com.itko.examples.entity.User getUser( java.lang.S
    arg1 (String value=itko)
    com.itko.examples.entity.User value=itko
        java.lang.String getLogin( )
            itko
        java.lang.String getPwd( )
            qUqP5cyxm6YcTAhz05Hph5gvu9M=
```

```
There are no parameters for this call.  Click Execute/Finish to complete the call.
```

Return Type: String

Save Result in Property: CurrentPassword

Comparison on Result Like: test

□ Exact ☑ True Fail the Test

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 158) in *Using CA Application Test.*

Add an Assertion to a Mobile Test Step

You can apply assertions to:

- A mobile test step.
- Each action that is contained in a 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.

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.

More Information:

Add an Assertion (see page 130)

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.

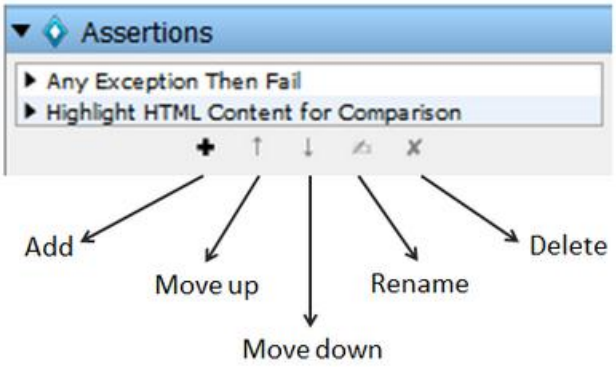
   The Assertion editor opens.
4. Complete the fields for the selected assertion.

   The editor is unique for each type of assertion. For more information about a specific type of assertion, see the following topics:

   - Ensure Same Mobile Screen
   - Ensure Screen Element Matches Expression
   - Ensure Screen Element is Removed
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

Follow 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.

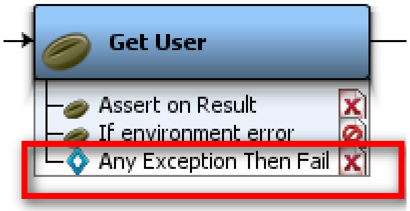
Drag and Drop an Assertion

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.



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 526) 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.

The following topics are included.

Global and Local Data Sets (see page 147)
Random Data Sets (see page 150)
Example Scenarios (see page 151)
Add a Data Set (see page 152)
Rename a Data Set (see page 154)
Move a Data Set (see page 154)
Data Set Next Step Selection (see page 154)
Data Sets and Properties (see page 155)

Global and Local Data Sets

Data sets can be global or local.

Global Data Sets

By default, a data set is global.

The coordinator server is responsible for providing data to all the test steps.

Here, all the model instances share a single instance of the data set.

The global data set is shared and applied to all instances of the model, even if they are run in different simulators.

Local Data Sets

You can make the data set local by selecting the Local check box while building the data set.

Each instance gets (essentially) its own copy of the data set.

A local data set provides one copy of the data set to each instance being run.

Example

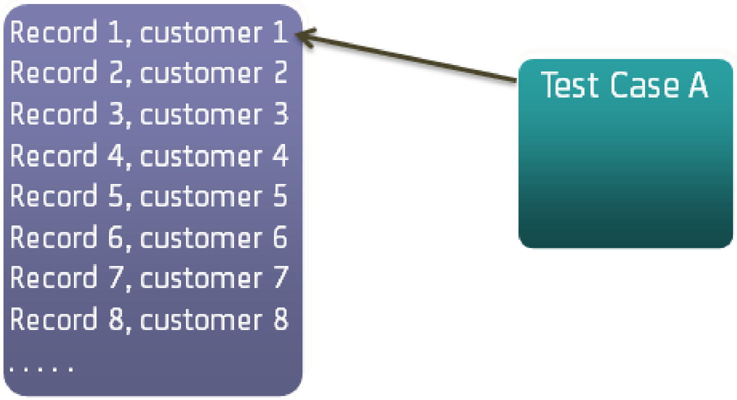
This example has the following components:

- Three concurrent virtual users
- A local data set with 100 rows of data
- A test case that loops over 100 rows of data and stops

Each virtual user sees all the 100 rows of data in the data set.

For a local data set

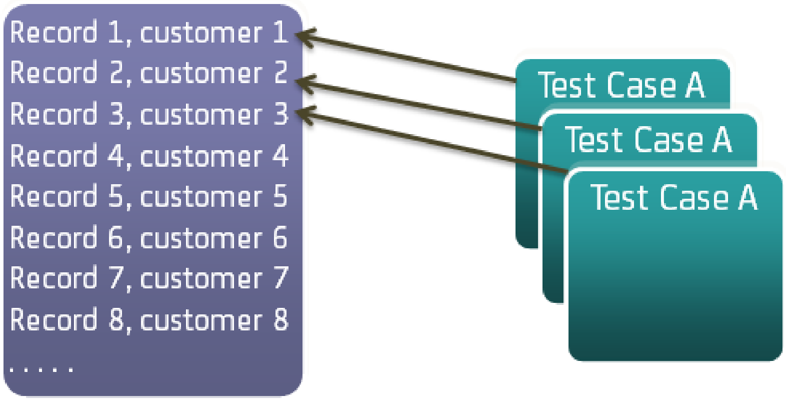
A single run (one vuser): Test Case A, gets the first row of data: Record 1, customer 1



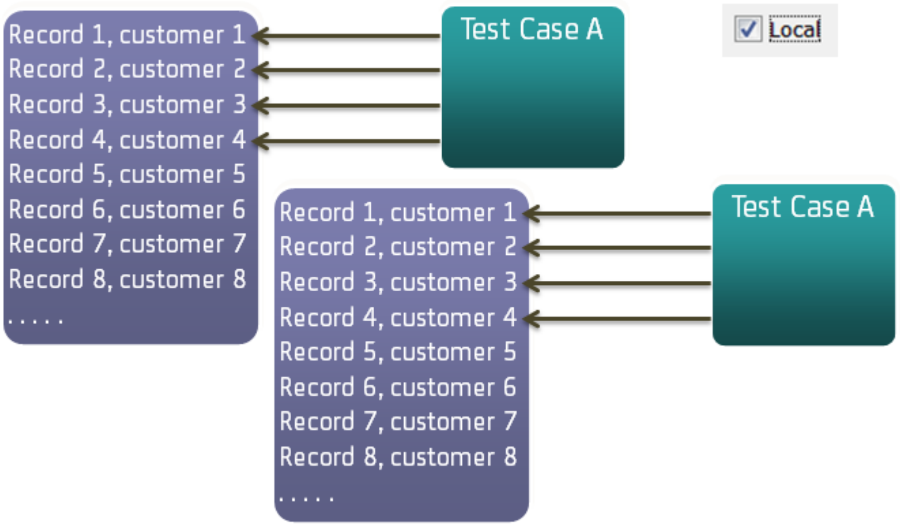
A data set that is shared across virtual users is global.

For a global data set

When Test Case A is staged with three virtual users or it is staged to run continuously, each test case receives the next row of data. DevTest shares a data set across multiple runs, using the instructions that are specified in the staging document.



Each virtual user (instance) gets its own copy of the data set that is local.



When local is marked and Test Case A has a loop, the test reads every row of data in the data set. Each run gets its own copy of the data set.

Test Case Looping

Often the data in a data set drives the number of times a test case is run.

Looping can be implemented several ways:

- A test is set to finish when all the data in the data set is exhausted.
- A test is set to reuse the data set when the data is exhausted.
- A test step can call itself, or you can configure a series of steps that loops until the data in the data set is exhausted.

Use a numeric counter data set to cause a specific step to run a fixed number of times. You can set the frequency at the step level or the test case level.

For example, consider a test where we want to test the login functionality of our application using 100 user ID/password pairs. To achieve this test, set a single step to call itself until the data set (with 100 rows of data) is exhausted. When the data set is exhausted, the test can redirect to the next natural step in the case, or the End step. Alternatively, you can use a counter data set with the user ID/password data set.

If you configure a global data set in the first step of a test case to end the test when the data set is exhausted, all test instances end for a staged run.

This overrides other staging parameters such as steady state time. Local data sets do not end the staged run in this fashion nor will data sets on steps other than the first test.

Random Data Sets

A random data set is a special type of data set; you can think of it as a wrapper around another data set.

You can select a data set to be randomized for specific steps, and also select the maximum number of records for randomization.

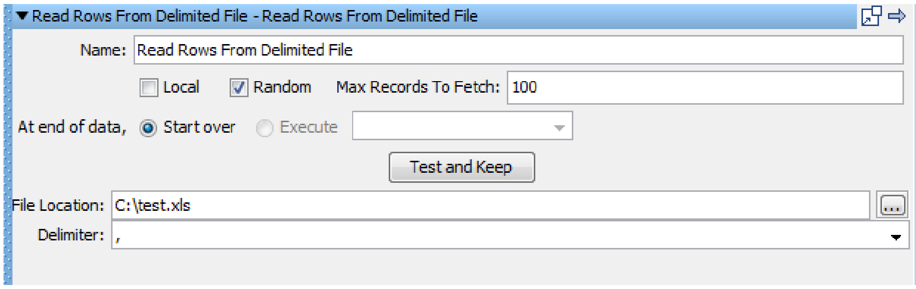
When a random wraps a data set, *n* copies of the data set are added to the random list, where *n* = Max Number of rows. So when *n*=10, DevTest makes 10 copies of the rows from the data set.

When a step references the random data set, a random row is selected from the data set.

If you have a random data set with the name **RAND1** reads one column with the name **Fruit**, and the Max Records to Fetch is set to 10, the random number **RAND1** is generated to pick a row. **RAND1** is in the range 0 through *n*-1. Fruit is the value of the "Fruit" column that is found in that row.

To make a data set random:

1. Select the data set to make random. This example uses the Read Rows from Delimited File data set.



Read Rows From Delimited File - Read Rows From Delimited File

Name: Read Rows From Delimited File

□ Local ☑ Random Max Records To Fetch: 100

At end of data, ● Start over ○ Execute

File Location: C:\test.xls

Delimiter: ,

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.

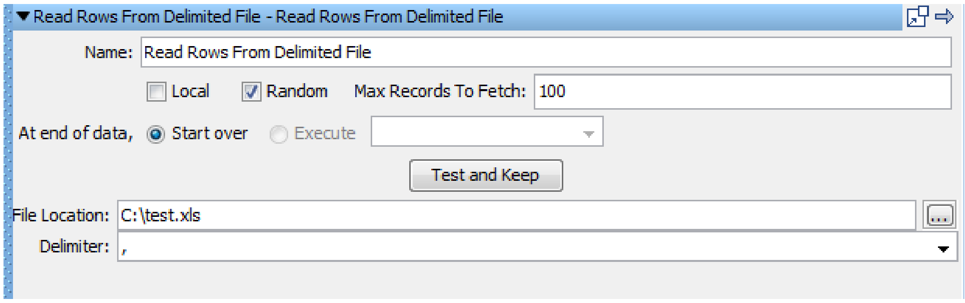
Add a Data Set

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 - Read Rows From Delimited File

Name: Read Rows From Delimited File

□ Local ☑ Random Max Records To Fetch: 100

At end of data, ● Start over ○ Execute

File Location: C:\test.xls

Delimiter: ,

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:

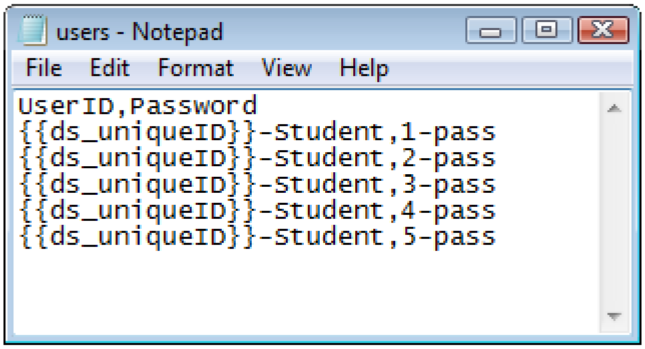
File Location

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

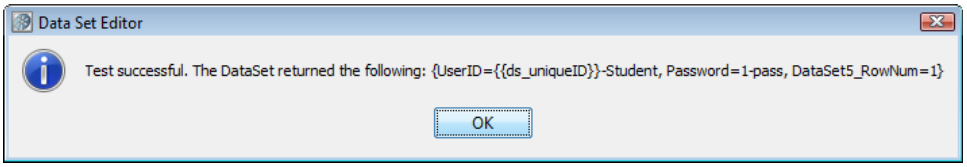
Enter the delimiter being used. Any value can be a delimiter. Some common delimiters are provided in the drop-down list.

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.



```
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
```

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.



Test successful. The DataSet returned the following: {UserID={{ds_uniqueID}}-Student, Password=1-pass, DataSet5_RowNum=1}

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.

The following topics are included.

Add a Companion (see page 156)
Companion Toolbar (see page 156)
DevTest Hooks (see page 157)

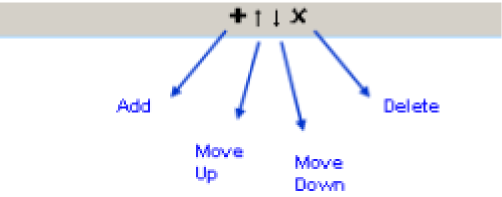
Add a Companion

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 in *Reference.*

Companion Toolbar

At the bottom of every element, there is a toolbar that has icons to add, delete, and reorder.



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)).

This section starts by describing the user interface. Then it looks at several usage scenarios that become progressively more complex.

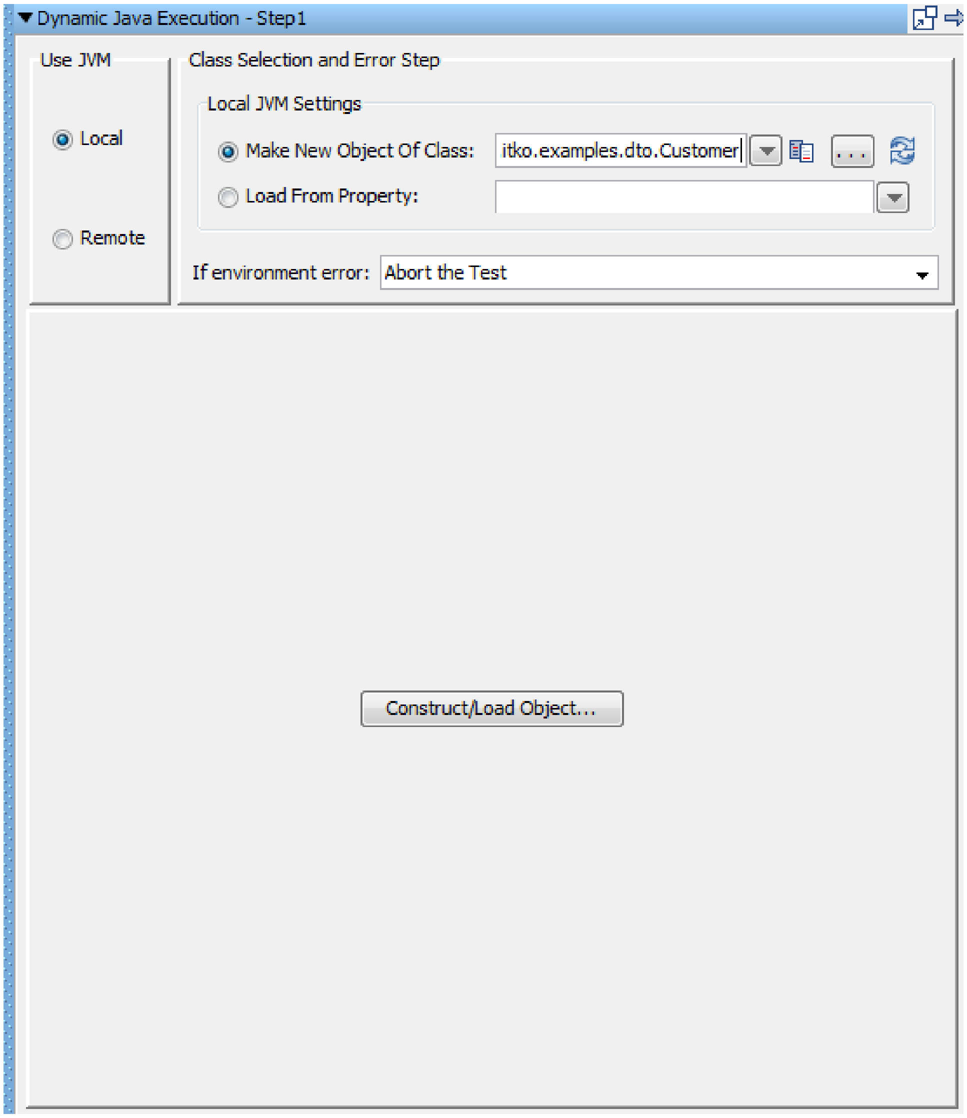
The following topics are included.

Invoke the COE (see page 159)
Object Call Tree Panel (see page 161)
Data Sheet and Call Sheet Panels (see page 163)
Object Interaction Panels (see page 165)
Using Data Sets in the COE (see page 172)
Usage Scenarios for Simple Objects (see page 174)
Usage Scenarios for Complex Objects (see page 179)

Invoke the COE

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:


Dynamic Java Execution - Step1
Use JVM
Class Selection and Error Step
Local JVM Settings
Construct/Load Object...

● Local
○ Remote

● Make New Object Of Class: itko.examples.dto.Customer
○ Load From Property:

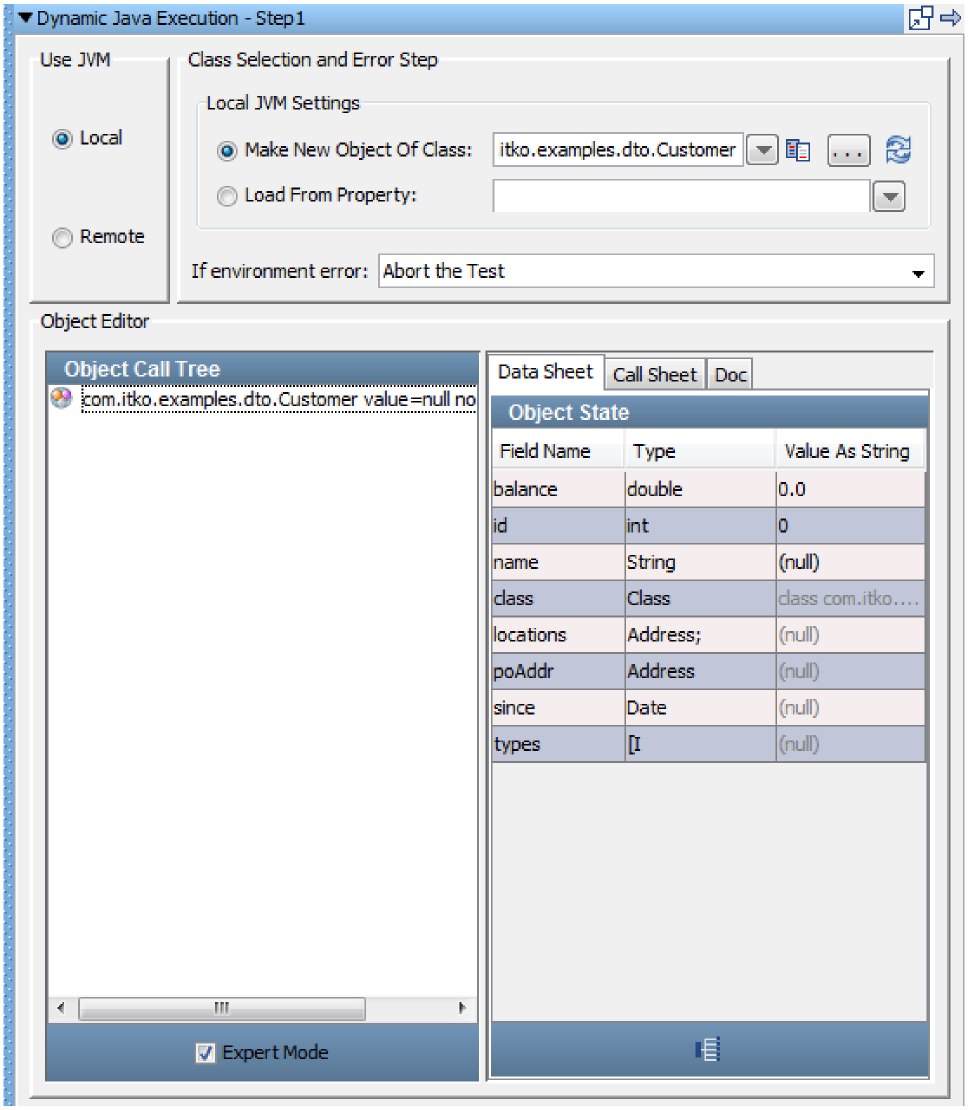
If environment error: Abort the Test

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.

Complex Object Editor

After the object is loaded, the Complex Object Editor is invoked.



Dynamic Java Execution - Step1

Use JVM

● Local

○ Remote

Class Selection and Error Step

Local JVM Settings

● Make New Object Of Class: itko.examples.dto.Customer

○ Load From Property:

If environment error: Abort the Test

Object Editor

Object Call Tree

com.itko.examples.dto.Customer value=null no

Data Sheet | Call Sheet | Doc

Object State

Field Name	Type	Value As String
balance	double	0.0
id	int	0
name	String	(null)
class	Class	class com.itko....
locations	Address;	(null)
poAddr	Address	(null)
since	Date	(null)
types	[I	(null)

☑ Expert Mode

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 161) is described in detail in the next topic.

The right panel is the Object State, with a set of dynamic tabs (Data Sheet Panel, Call Sheet Panel, Doc Panel (see page 163)) that show you available options.

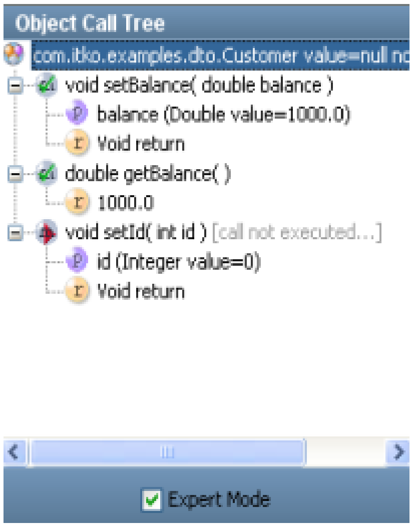
Note: If the response is XML and it is larger than 5 MB, it is displayed in plain text with no DOM view. This 5-MB default limit can be adjusted with the **gui.viewxml.maxResponseSize** property.

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.

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

```
com.itko.examples.dto.Customer value=null nc
⊟ void setBalance( double balance )
     balance (Double value=1000.0)
     Void return
⊟ double getBalance( )
     1000.0
⊟ void setId( int id ) [call not executed...]
     id (Integer value=0)
     Void return
```

☑ Expert Mode

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 toString method of object.
✓	The constructor that was called. This is shown if multiple constructors exist.
✗	A method call that has not been executed.
✓	A method call that has been executed.
p	The input parameters (type and current value) for the enclosing method.
r	The return value (current value if call has been executed) for the enclosing method.

Clicking an item in the Object Call Tree displays the appropriate set of tabs in the Data Sheet and Call Sheet in the right panel.

Right-clicking an item in the Object Call Tree displays a menu.

You can execute all calls or you can mark all calls as unexecuted in the Object Call Tree.

Data Sheet and Call Sheet Panels

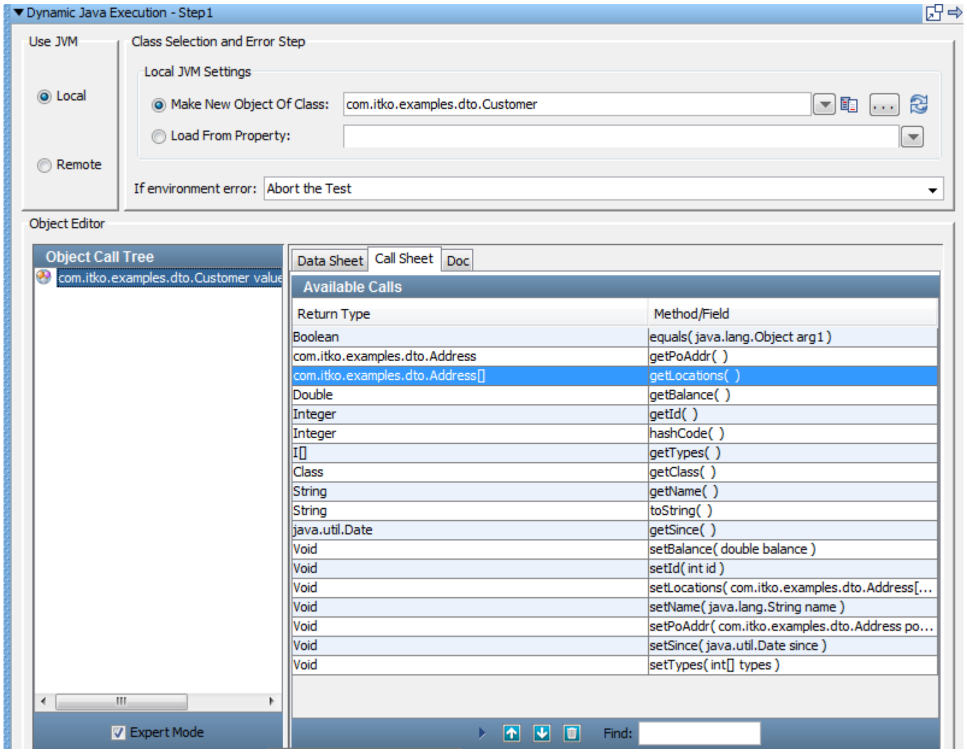
Data Sheet Panel

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.



● Local
○ Remote

● Make New Object Of Class: com.itko.examples.dto.Customer
○ Load From Property:

If environment error: Abort the Test

com.itko.examples.dto.Customer value

Return Type	Method/Field
Boolean	equals(java.lang.Object arg1)
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.Address[...
Void	setName(java.lang.String name)
Void	setPoAddr(com.itko.examples.dto.Address po...
Void	setSince(java.util.Date since)
Void	setTypes(int[] types)

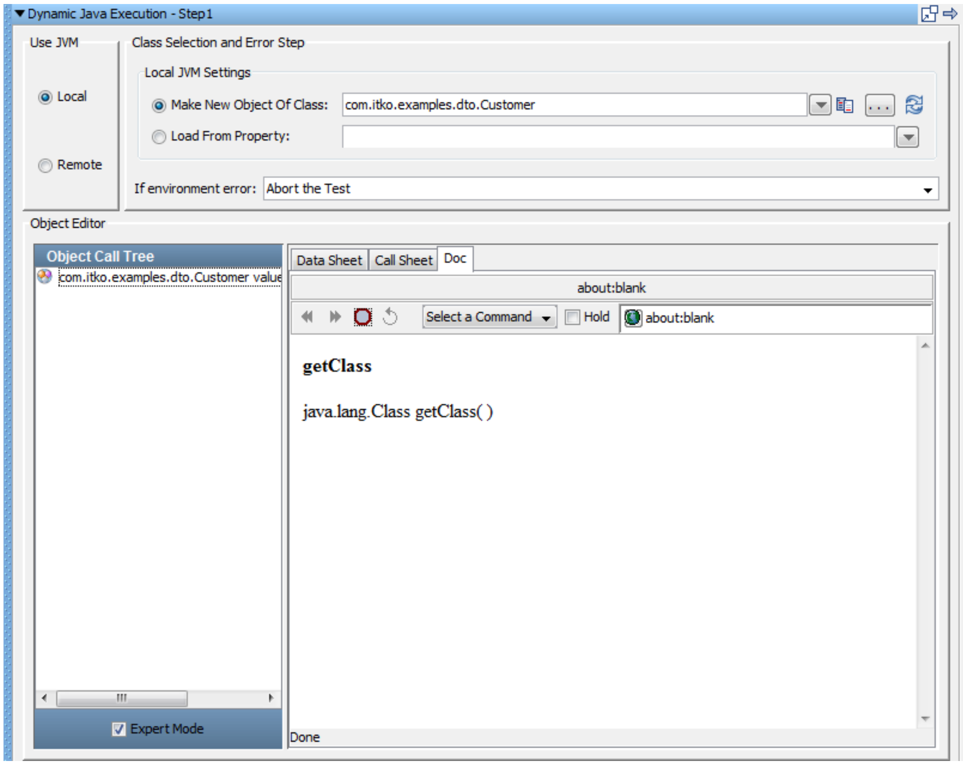
☑ Expert Mode

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.



Use JVM: ● Local ○ Remote

● Make New Object Of Class: com.itko.examples.dto.Customer

○ Load From Property:

If environment error: Abort the Test

com.itko.examples.dto.Customer value

☑ Expert Mode

about:blank

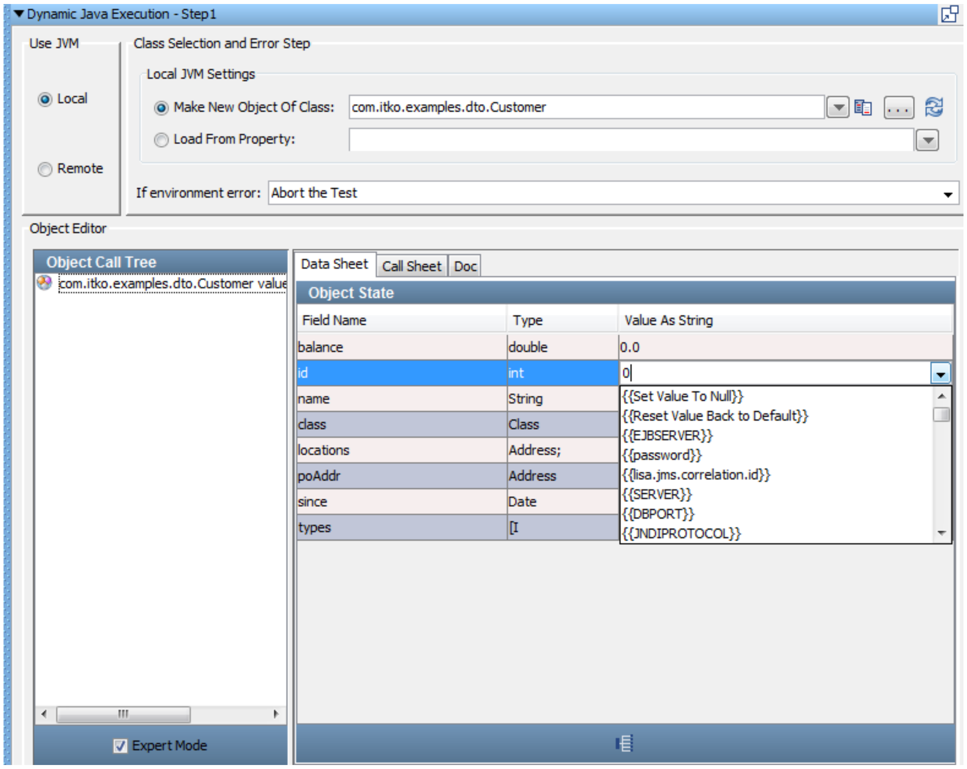
getClass

java.lang.Class getClass( )

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.



Dynamic Java Execution - Step1

Use JVM

● Local

○ Remote

Class Selection and Error Step

Local JVM Settings

● Make New Object Of Class: com.itko.examples.dto.Customer

○ Load From Property:

If environment error: Abort the Test

Object Editor

Object Call Tree

com.itko.examples.dto.Customer value

Object State

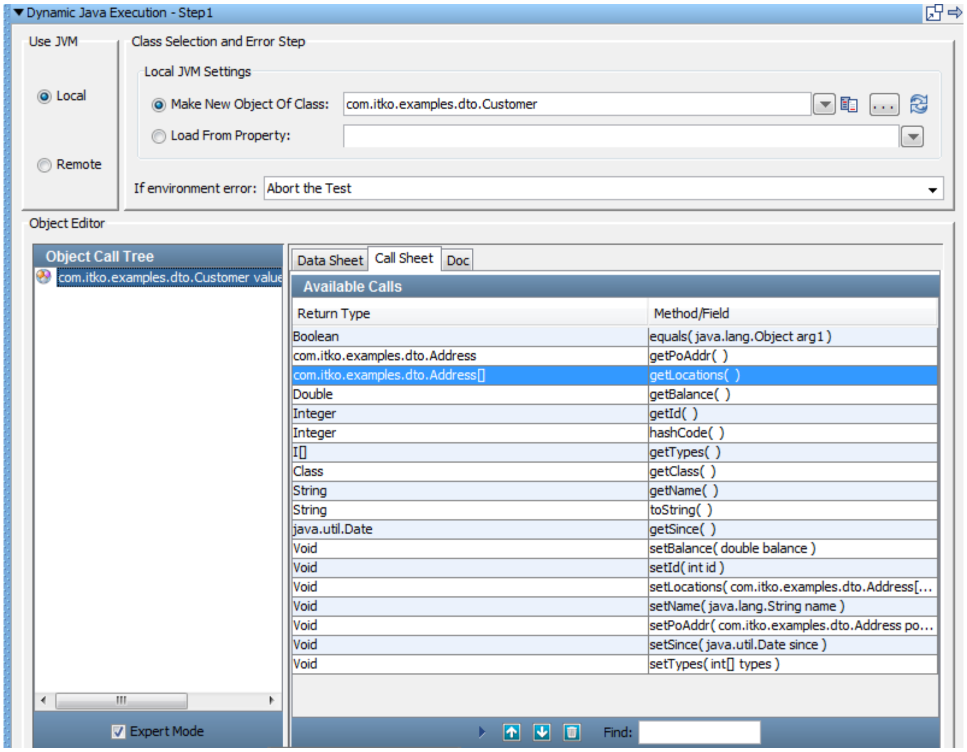
Field Name	Type	Value As String
balance	double	0.0
id	int	0
name	String	
class	Class	
locations	Address;	
poAddr	Address	
since	Date	
types	[I	

{{Set Value To Null}}
{{Reset Value Back to Default}}
{{EJBSERVER}}
{{password}}
{{lisa.jms.correlation.id}}
{{SERVER}}
{{DBPORT}}
{{JNDIPROTOCOL}}

☑ Expert Mode

Object Panels

The Data Sheet, Call Sheet, and Doc tabs are available when an object is selected in the Object Call Tree.



Dynamic Java Execution - Step1

Use JVM

● Local

○ Remote

Class Selection and Error Step

Local JVM Settings

● Make New Object Of Class: com.itko.examples.dto.Customer

○ Load From Property:

If environment error: Abort the Test

Object Editor

Object Call Tree
com.itko.examples.dto.Customer value
Data Sheet
Call Sheet
Doc

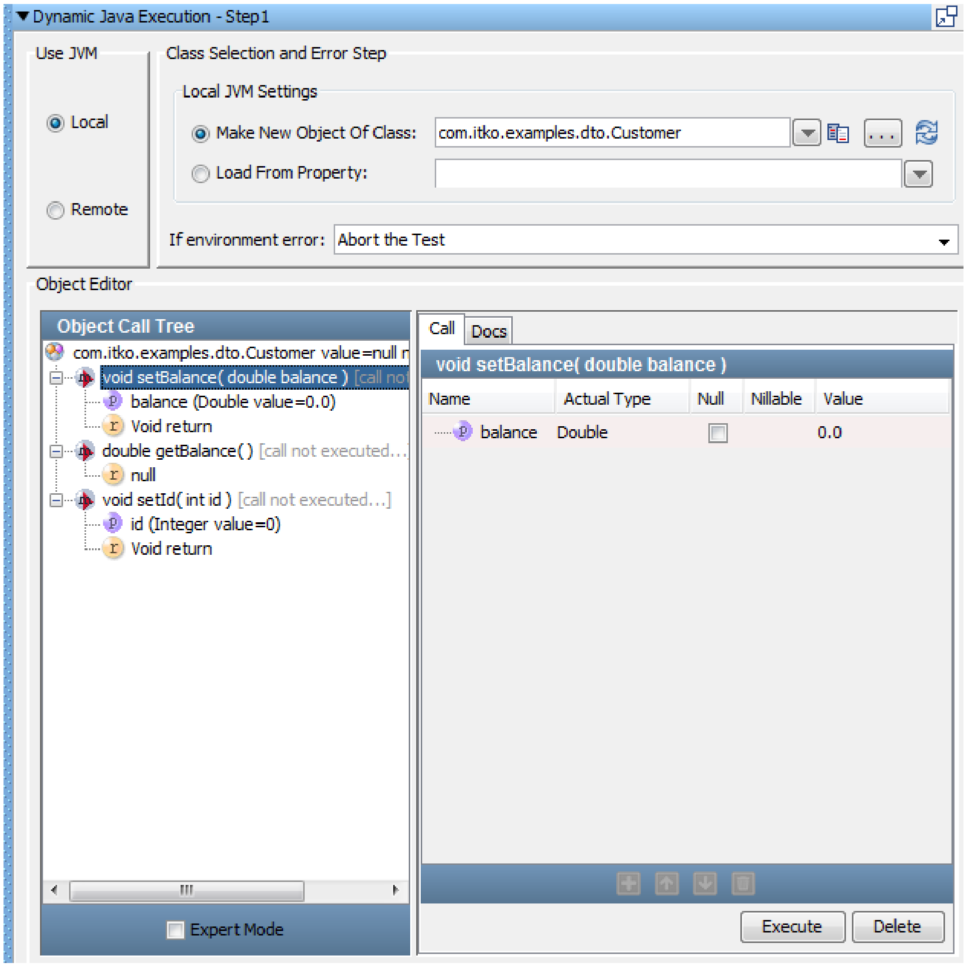
Available Calls

Return Type	Method/Field
Boolean	equals(java.lang.Object arg1)
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.Address[...
Void	setName(java.lang.String name)
Void	setPoAddr(com.itko.examples.dto.Address po...
Void	setSince(java.util.Date since)
Void	setTypes(int[] types)

☑ Expert Mode

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.



Dynamic Java Execution - Step1

Use JVM

● Local

○ Remote

Class Selection and Error Step

Local JVM Settings

● Make New Object Of Class: com.itko.examples.dto.Customer

○ Load From Property:

If environment error: Abort the Test

Object Editor

Object Call Tree

```
com.itko.examples.dto.Customer value=null r
  void setBalance( double balance ) [call not
    balance (Double value=0.0)
    Void return
  double getBalance( ) [call not executed...]
    null
  void setId( int id ) [call not executed...]
    id (Integer value=0)
    Void return
```

□ Expert Mode

Call | Docs

void setBalance(double balance)

Name	Actual Type	Null	Nillable	Value
balance	Double	□		0.0

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

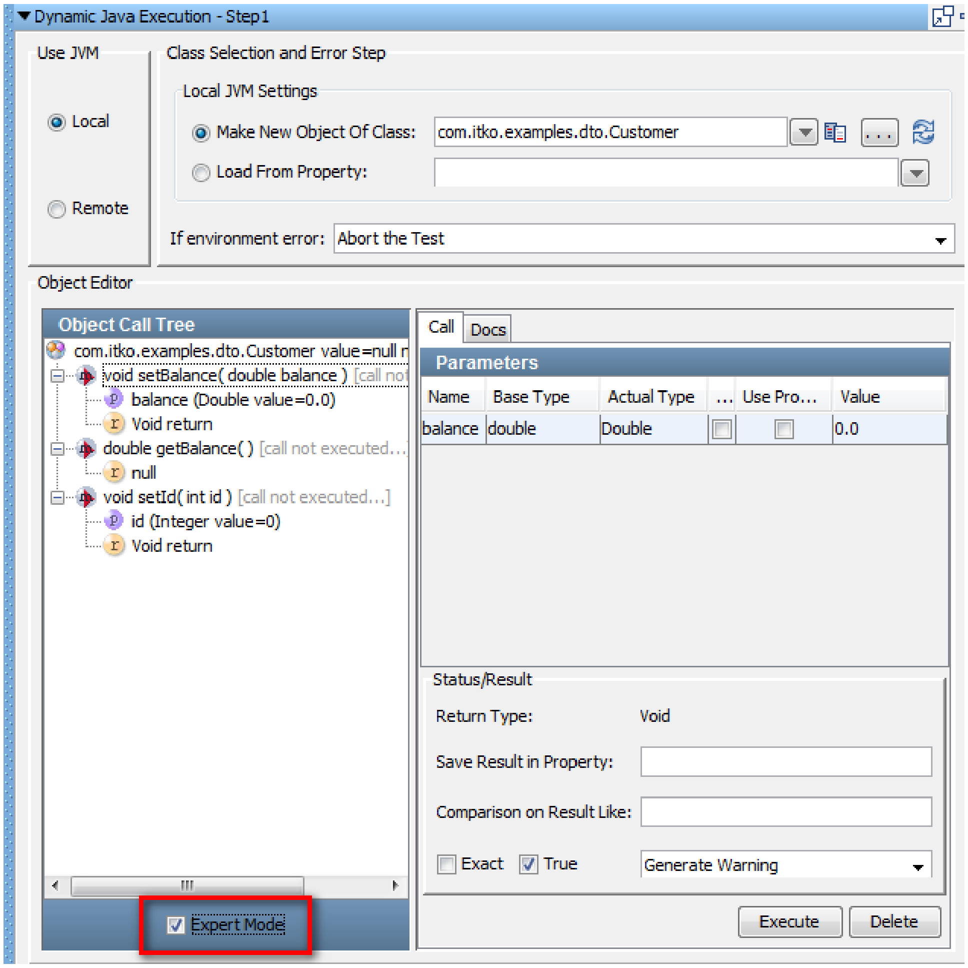
Two editing modes are available: simple and expert.

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.



▼ Dynamic Java Execution - Step1

Use JVM: ● Local ○ Remote

Class Selection and Error Step

Local JVM Settings

● Make New Object Of Class: com.itko.examples.dto.Customer

○ Load From Property:

If environment error: Abort the Test

Object Editor

Object Call Tree

com.itko.examples.dto.Customer value=null r
- void setBalance(double balance) [call not
  - balance (Double value=0.0)
  - Void return
- double getBalance( ) [call not executed...
  - null
- void setId(int id) [call not executed...]
  - id (Integer value=0)
  - Void return

☑ Expert Mode

Parameters

Name	Base Type	Actual Type	...	Use Pro...	Value
balance	double	Double	□	□	0.0

Status/Result

Return Type: Void

Save Result in Property:

Comparison on Result Like:

□ Exact ☑ True Generate Warning

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.

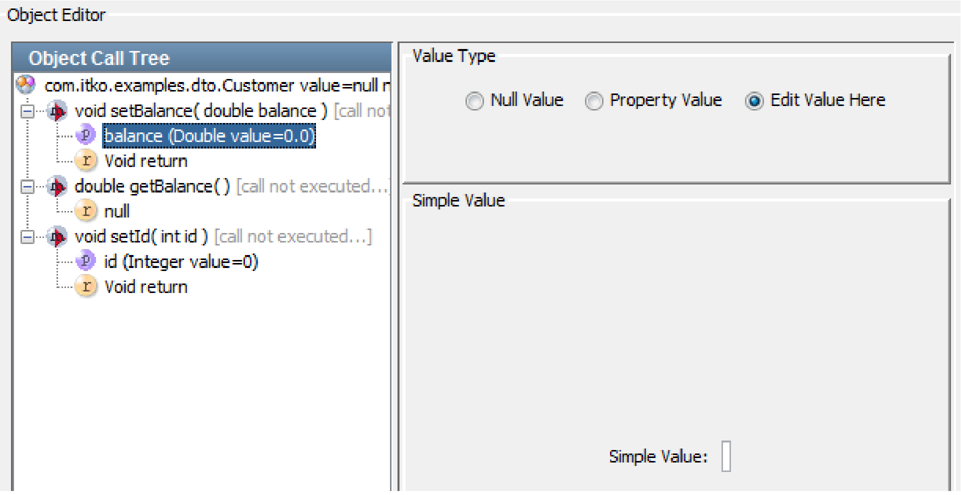
Note: The in-line filters and assertions that are applied are not seen in the filters or assertions list.

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.



Object Editor

Object Call Tree

```
com.itko.examples.dto.Customer value=null r
  void setBalance( double balance ) [call not
    balance (Double value=0.0)
    Void return
  double getBalance( ) [call not executed...
    null
  void setId( int id ) [call not executed...]
    id (Integer value=0)
    Void return
```

Value Type

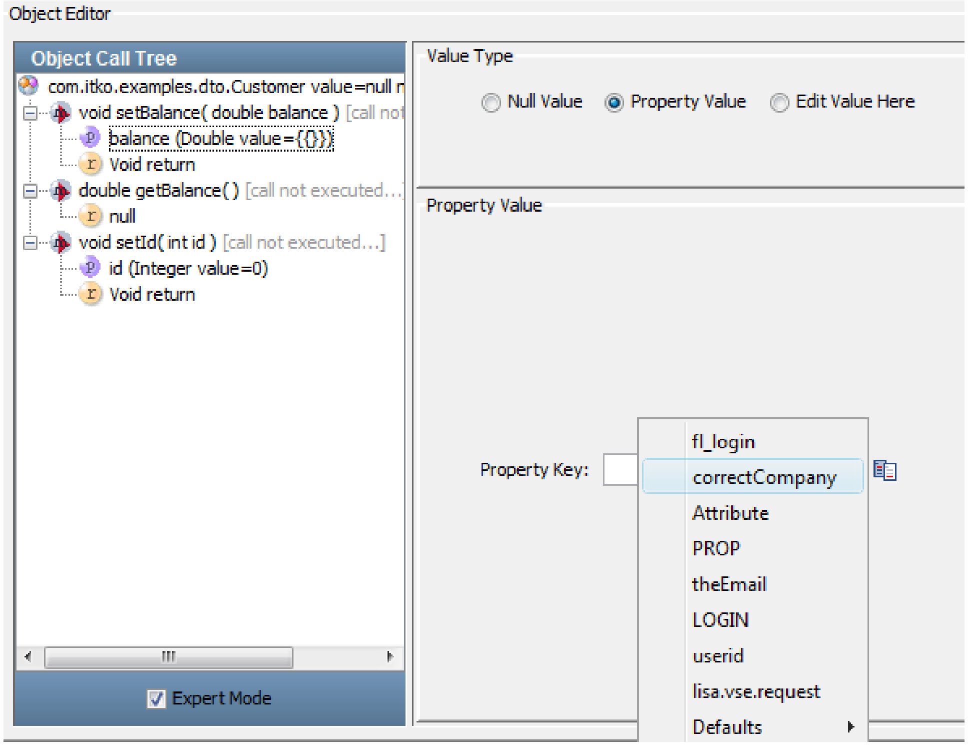
○ Null Value ○ Property Value ● Edit Value Here

Simple Value

Simple Value:

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.

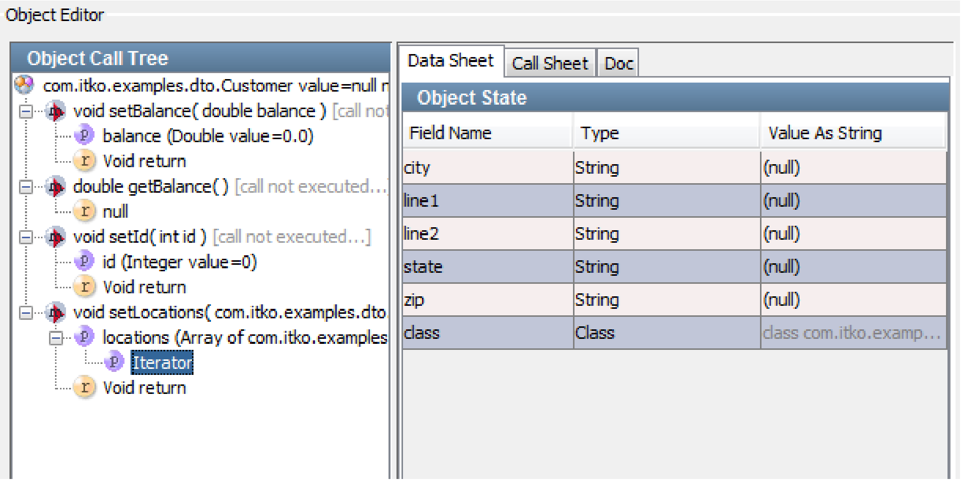


☑ Expert Mode

Value Type: ○ Null Value ● Property Value ○ Edit Value Here

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.

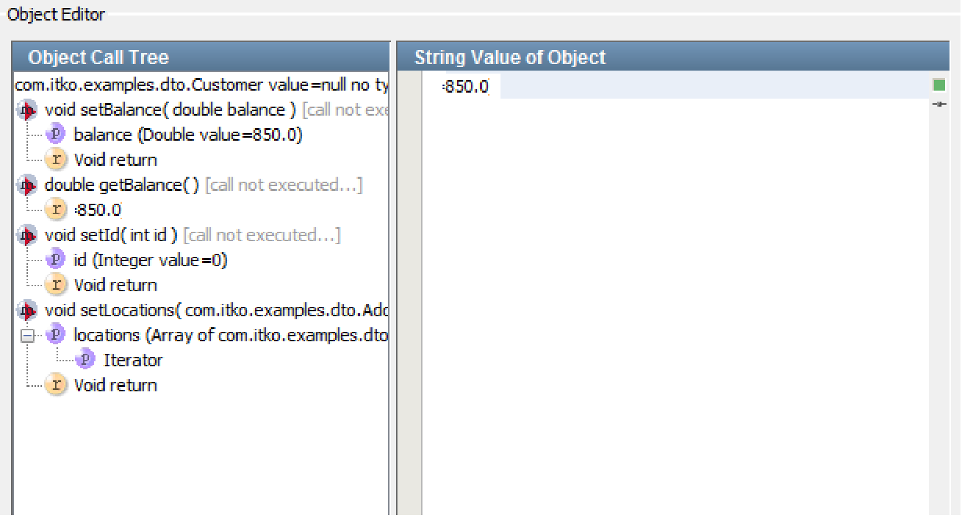

Object Editor
Object Call Tree
com.itko.examples.dto.Customer value=null r
void setBalance(double balance) [call not
balance (Double value=0.0)
Void return
double getBalance( ) [call not executed...
null
void setId(int id) [call not executed...]
id (Integer value=0)
Void return
void setLocations(com.itko.examples.dto
locations (Array of com.itko.examples
Iterator
Void return
Data Sheet
Call Sheet
Doc
Object State

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.examp...

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.



Object Editor

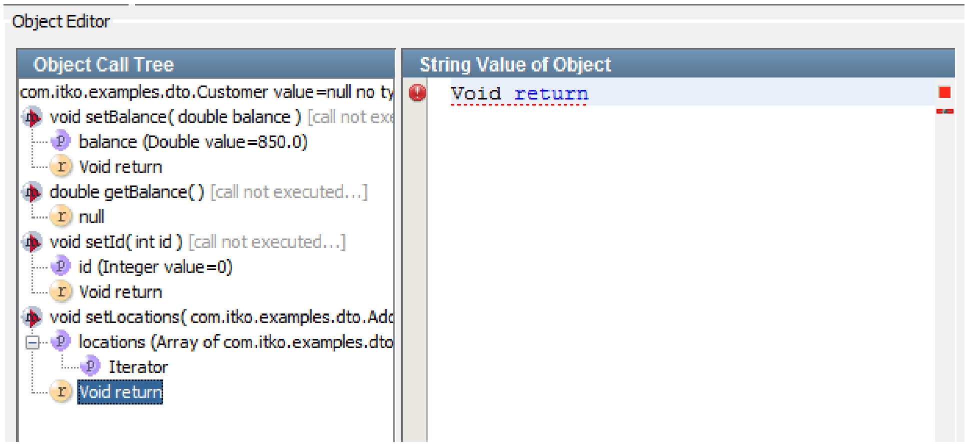
Object Call Tree

com.itko.examples.dto.Customer value=null no ty
void setBalance(double balance) [call not exe
balance (Double value=850.0)
Void return
double getBalance( ) [call not executed...]
:850.0
void setId(int id) [call not executed...]
id (Integer value=0)
Void return
void setLocations(com.itko.examples.dto.Add
locations (Array of com.itko.examples.dto
Iterator
Void return

String Value of Object

:850.0

For input parameters that are objects, the following panel is displayed.



Object Editor

Object Call Tree

com.itko.examples.dto.Customer value=null no ty
void setBalance(double balance) [call not exe
balance (Double value=850.0)
Void return
double getBalance( ) [call not executed...]
null
void setId(int id) [call not executed...]
id (Integer value=0)
Void return
void setLocations(com.itko.examples.dto.Add
locations (Array of com.itko.examples.dto
Iterator
Void return

String Value of Object

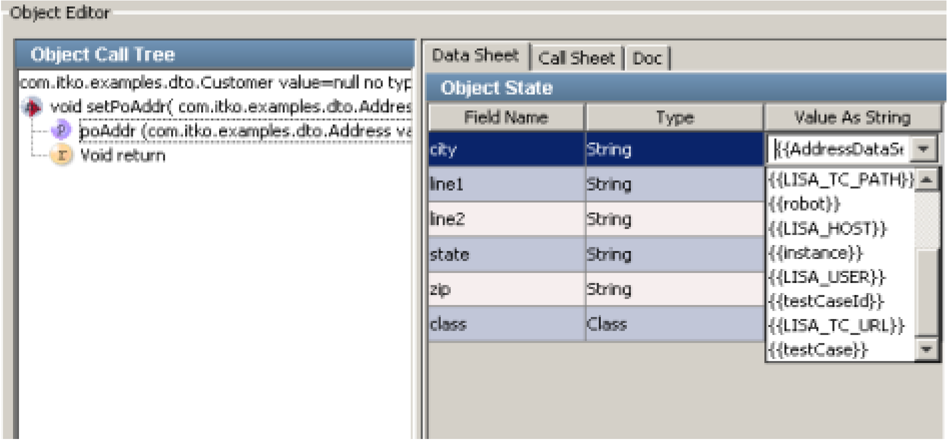
```
Void return
```

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.

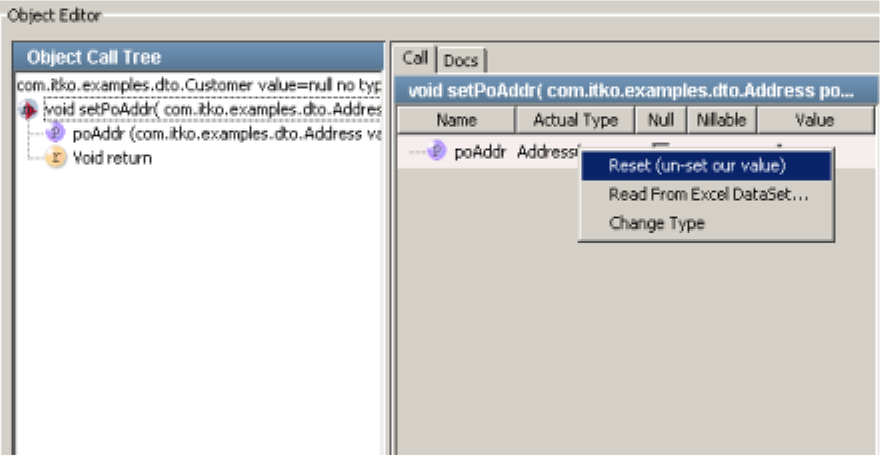
If the Excel data set exists, the property that contains the data set can be entered as a value for the DTO object.



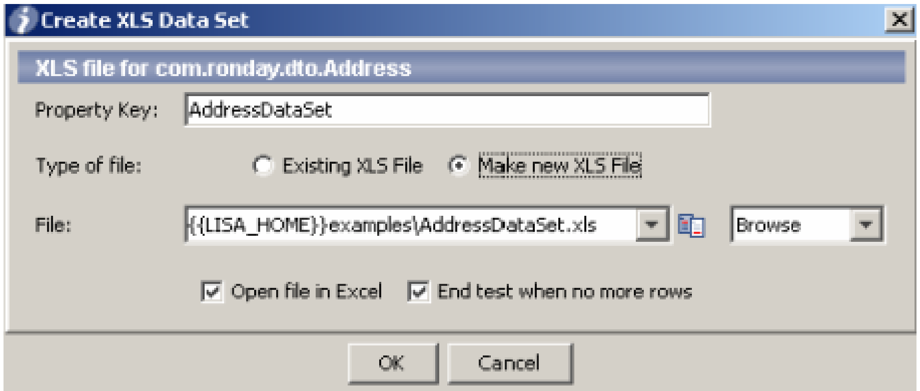
Field Name	Type	Value As String
city	String	{{AddressDataS
line1	String	
line2	String	
state	String	
zip	String	
class	Class	

To initiate the creation of a new DTO data set from the object editor:

1. When a DTO object appears in the call list, right-click the Name or Actual Type to open a menu.


Object Editor
Object Call Tree
com.itko.examples.dto.Customer value=null no typ
void setPoAddr(com.itko.examples.dto.Addres
poAddr (com.itko.examples.dto.Address va
Void return
Call
Docs
void setPoAddr(com.itko.examples.dto.Address po...
Name
Actual Type
Null
Nillable
Value
poAddr
Address
Reset (un-set our value)
Read From Excel DataSet...
Change Type

2. Select the data set Read From Excel Data set to display the following window.


Create XLS Data Set

XLS file for com.ronday.dto.Address

Property Key: AddressDataSet

Type of file: ○ Existing XLS File ● Make new XLS File

File: {{LISA_HOME}}examples\AddressDataSet.xls

☑ Open file in Excel ☑ End test when no more rows

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

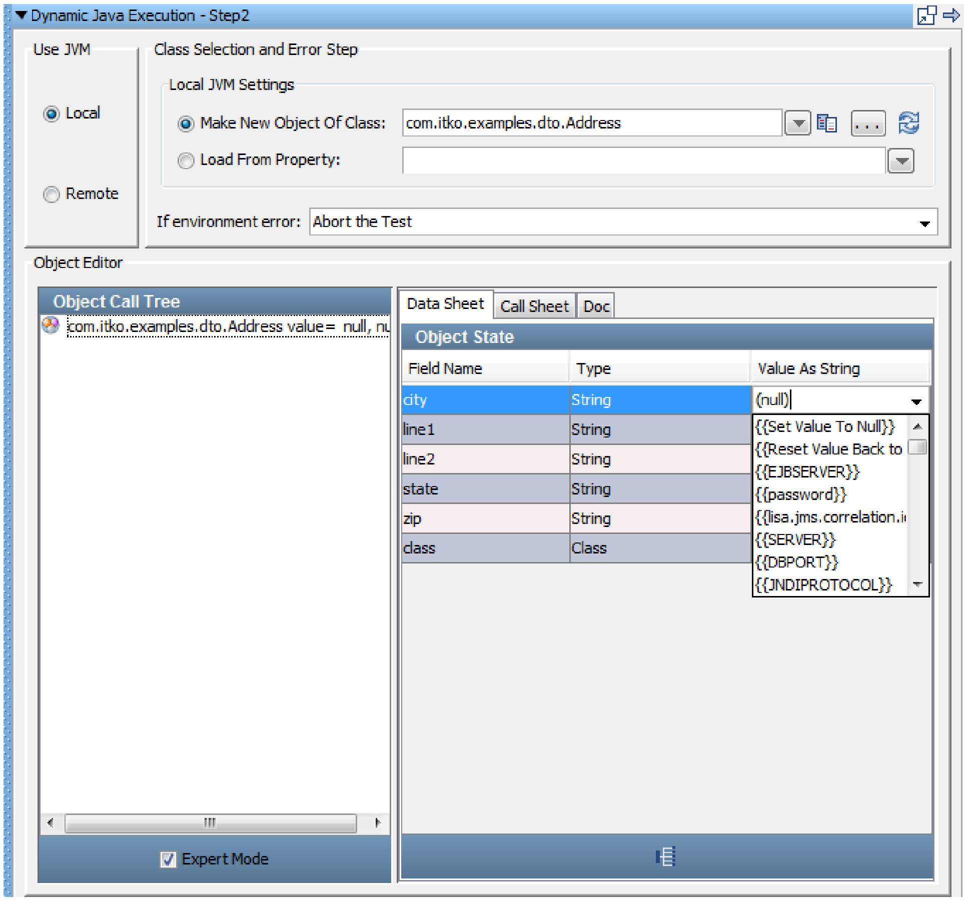
Ends the test after all rows are read.

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.



● Local

○ Remote

● Make New Object Of Class: com.itko.examples.dto.Address

○ Load From Property:

If environment error: Abort the Test

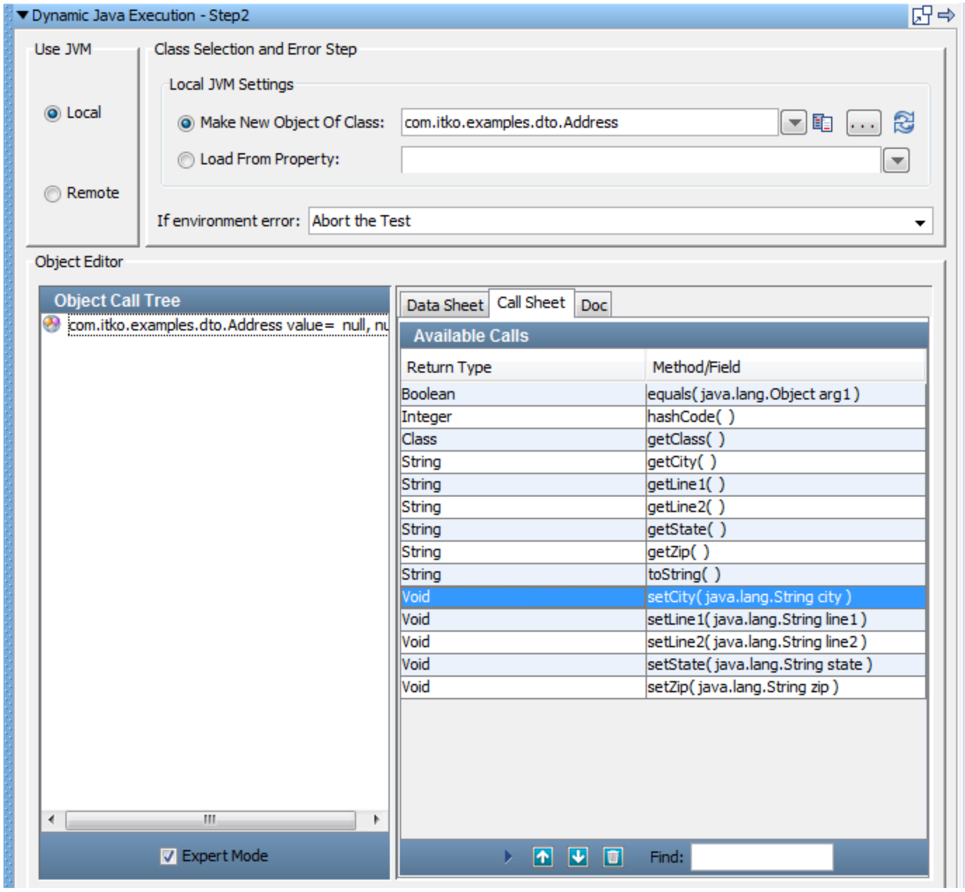
com.itko.examples.dto.Address value= null, nu

Field Name	Type	Value As String
city	String	(null)
line1	String	
line2	String	
state	String	
zip	String	
class	Class	

☑ Expert Mode

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:



Use JVM

● Local

○ Remote

● Make New Object Of Class: com.itko.examples.dto.Address

○ Load From Property:

If environment error: Abort the Test

com.itko.examples.dto.Address value= null, nu

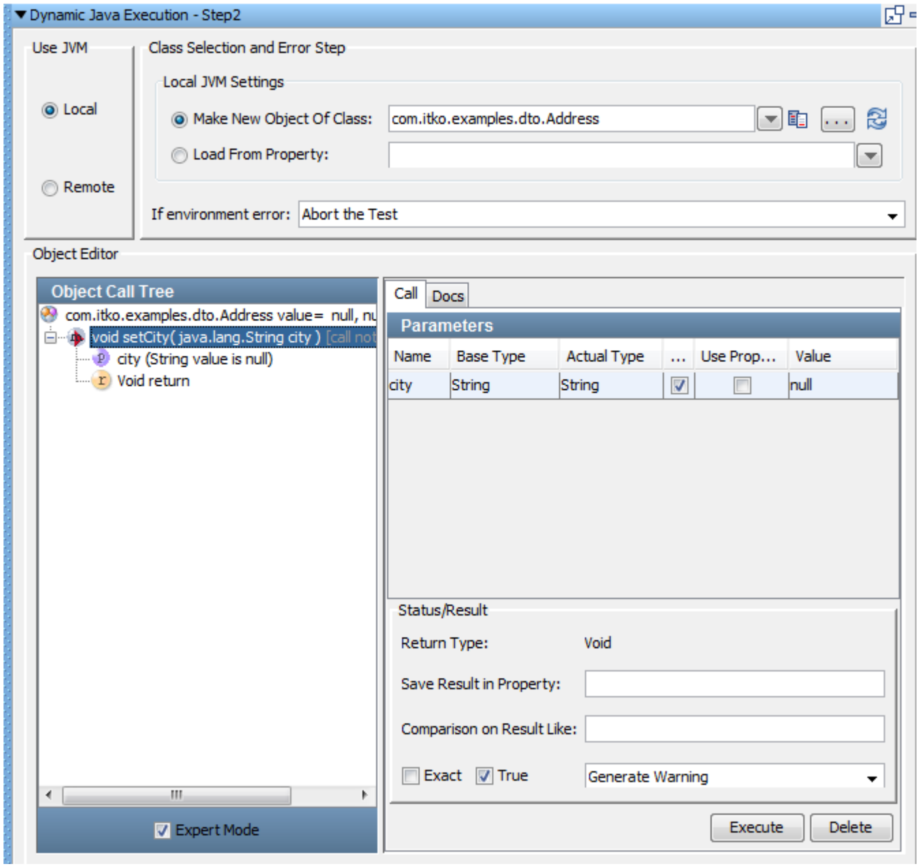
Return Type	Method/Field
Boolean	equals(java.lang.Object arg1)
Integer	hashCode( )
Class	getClass( )
String	getCity( )
String	getLine1( )
String	getLine2( )
String	getState( )
String	getZip( )
String	toString( )
Void	setCity(java.lang.String city)
Void	setLine1(java.lang.String line1)
Void	setLine2(java.lang.String line2)
Void	setState(java.lang.String state)
Void	setZip(java.lang.String zip)

☑ Expert Mode

Find:

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.



Dynamic Java Execution - Step2

Use JVM

● Local

○ Remote

Class Selection and Error Step

Local JVM Settings

● Make New Object Of Class: com.itko.examples.dto.Address

○ Load From Property:

If environment error: Abort the Test

Object Editor

Object Call Tree

com.itko.examples.dto.Address value= null, nu

void setCity(java.lang.String city) [call not

city (String value is null)

Void return

☑ Expert Mode

Call | Docs

Parameters

Name	Base Type	Actual Type	...	Use Prop...	Value
city	String	String	☑	☐	null

Status/Result

Return Type: Void

Save Result in Property:

Comparison on Result Like:

☐ Exact ☑ True Generate Warning

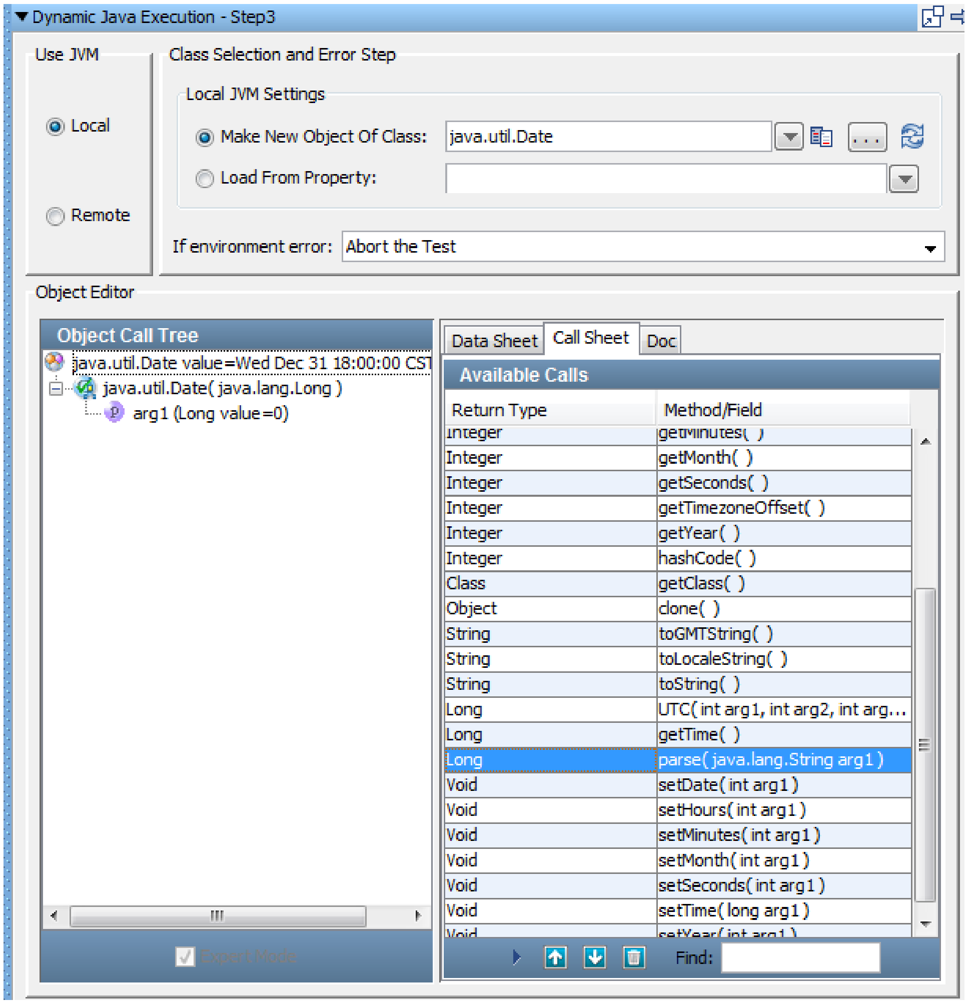
2. Enter the parameter value as a fixed value or a property. Use the DevTest property syntax, propname.

3. Click the Execute button 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.



Dynamic Java Execution - Step3

Use JVM

● Local

○ Remote

Class Selection and Error Step

Local JVM Settings

● Make New Object Of Class: java.util.Date

○ Load From Property:

If environment error: Abort the Test

Object Editor

Object Call Tree

- java.util.Date value=Wed Dec 31 18:00:00 CST
  - java.util.Date(java.lang.Long)
    - arg1 (Long value=0)

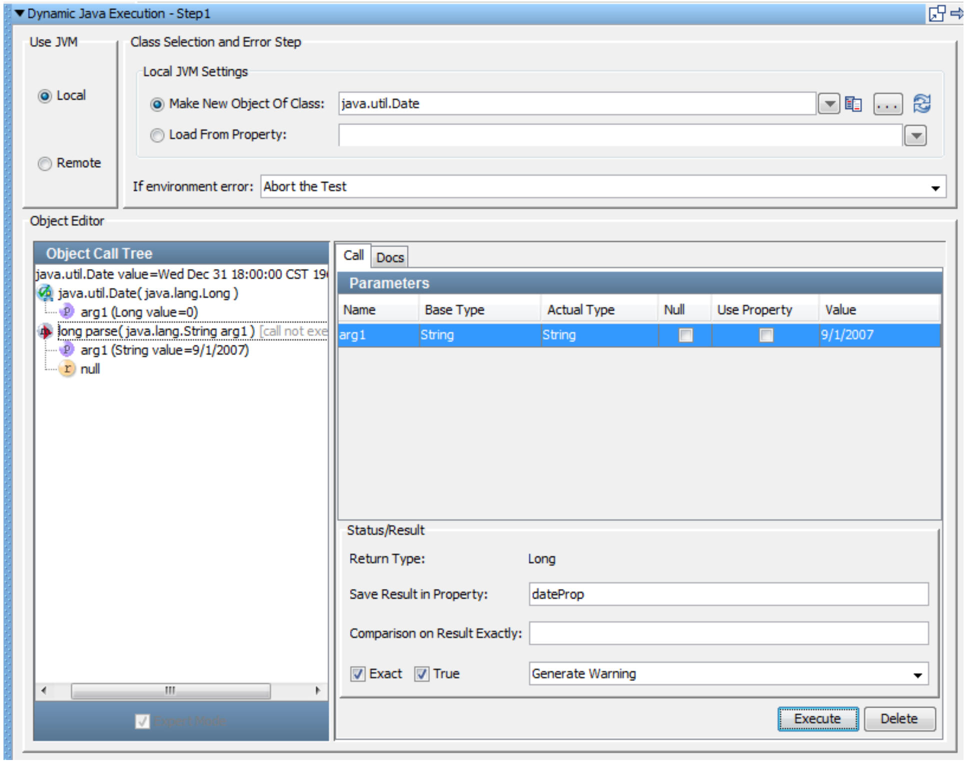
☑ Expert Mode

Available Calls

Return Type	Method/Field
Integer	getMinutes( )
Integer	getMonth( )
Integer	getSeconds( )
Integer	getTimezoneOffset( )
Integer	getYear( )
Integer	hashCode( )
Class	getClass( )
Object	clone( )
String	toGMTString( )
String	toLocaleString( )
String	toString( )
Long	UTC(int arg1, int arg2, int arg...
Long	getTime( )
Long	parse(java.lang.String arg1)
Void	setDate(int arg1)
Void	setHours(int arg1)
Void	setMinutes(int arg1)
Void	setMonth(int arg1)
Void	setSeconds(int arg1)
Void	setTime(long arg1)
Void	setYear(int arg1)

Find:

In this case, Expert Mode must be used, because the Date type is not a DTO. In fact the COE forces Expert mode.



Dynamic Java Execution - Step1

Use JVM

● Local

○ Remote

Class Selection and Error Step

Local JVM Settings

● Make New Object Of Class: java.util.Date

○ Load From Property:

If environment error: Abort the Test

Object Editor

Object Call Tree

```
java.util.Date value=Wed Dec 31 18:00:00 CST 19
  java.util.Date( java.lang.Long )
    arg1 (Long value=0)
  long parse( java.lang.String arg1 ) [call not exe
    arg1 (String value=9/1/2007)
    null
```

☑ Expert Mode

Parameters

Name	Base Type	Actual Type	Null	Use Property	Value
arg1	String	String	□	□	9/1/2007

Status/Result

Return Type: Long

Save Result in Property: dateProp

Comparison on Result Exactly:

☑ Exact ☑ True Generate Warning

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.

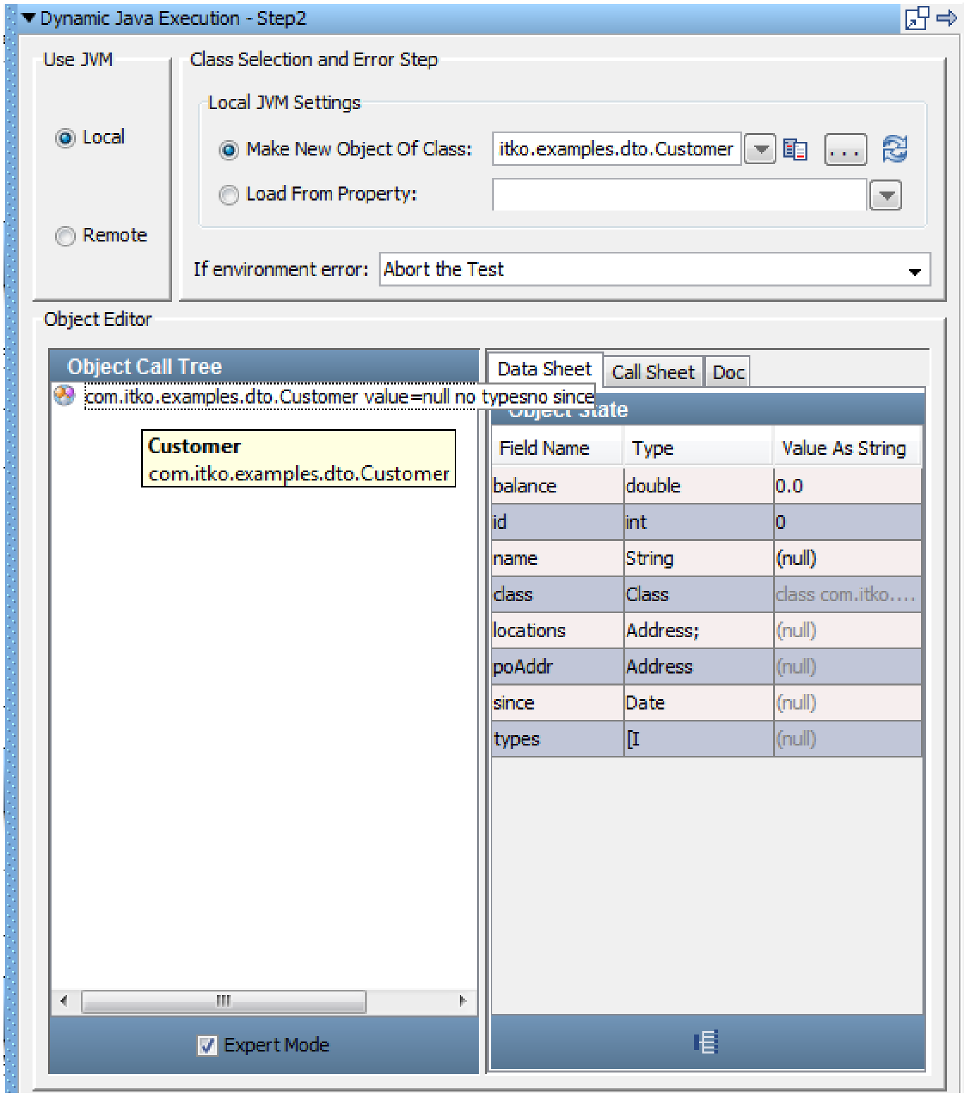
Because we are in Expert Mode, we can add inline filters and assertions.

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.



Dynamic Java Execution - Step2

Use JVM

● Local

○ Remote

Class Selection and Error Step

Local JVM Settings

● Make New Object Of Class: itko.examples.dto.Customer

○ Load From Property:

If environment error: Abort the Test

Object Editor

Object Call Tree

com.itko.examples.dto.Customer value=null no typesno since

Customer
com.itko.examples.dto.Customer

Object State

Field Name	Type	Value As String
balance	double	0.0
id	int	0
name	String	(null)
class	Class	class com.itko....
locations	Address;	(null)
poAddr	Address	(null)
since	Date	(null)
types	[I	(null)

☑ Expert Mode

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

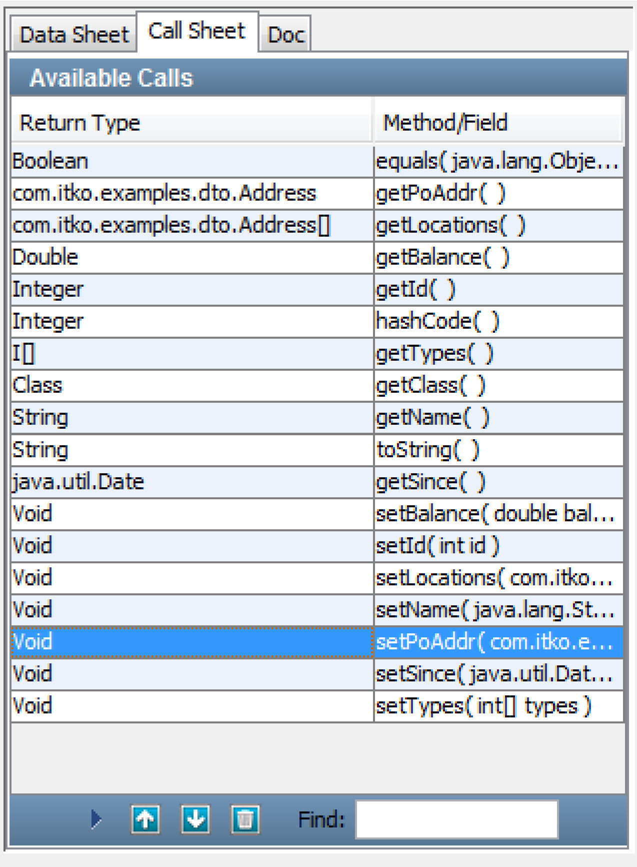
since

A Java Date object

types

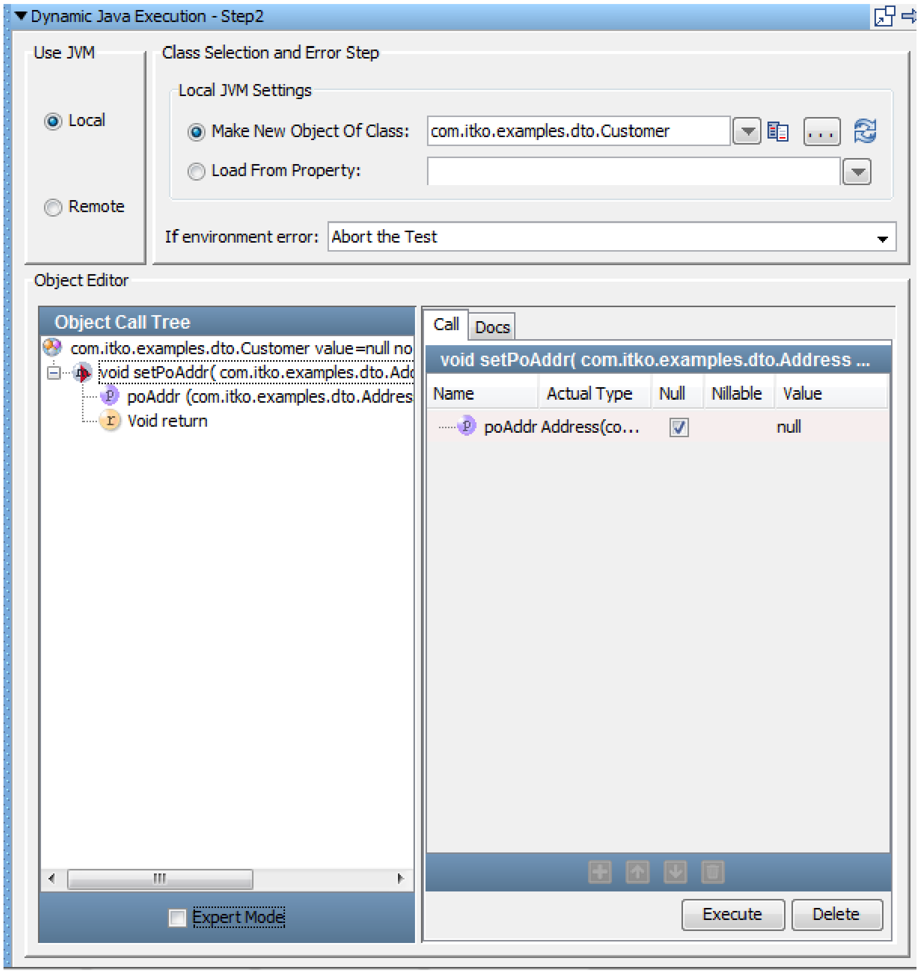
An array of integers

1. Starting with the poAddr object, identify the setPoAddr method in the Call Sheet.



Return Type	Method/Field
Boolean	equals(java.lang.Obje…
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 bal…
Void	setId(int id)
Void	setLocations(com.itko…
Void	setName(java.lang.St…
Void	setPoAddr(com.itko.e…
Void	setSince(java.util.Dat…
Void	setTypes(int[] types)

2. Select the setPoAddr method and double-click or click ▶ Add Method to run this method.



Dynamic Java Execution - Step2

Use JVM

● Local

○ Remote

Class Selection and Error Step

Local JVM Settings

● Make New Object Of Class: com.itko.examples.dto.Customer

○ Load From Property:

If environment error: Abort the Test

Object Editor

Object Call Tree

com.itko.examples.dto.Customer value=null no
void setPoAddr(com.itko.examples.dto.Add
poAddr (com.itko.examples.dto.Addres
Void return

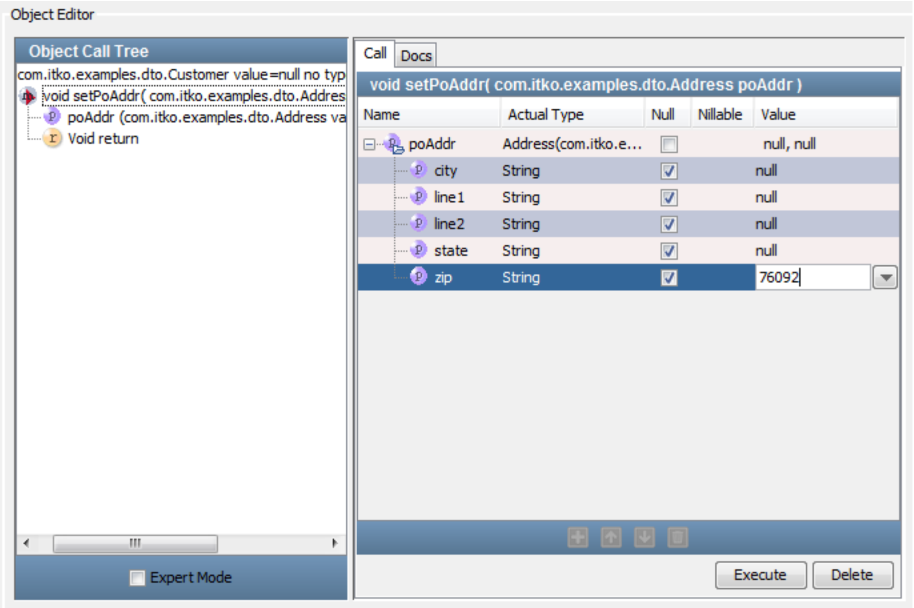
void setPoAddr(com.itko.examples.dto.Address ...

Name	Actual Type	Null	Nillable	Value
poAddr Address(co...		☑		null

□ Expert Mode

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.



Object Editor

Object Call Tree

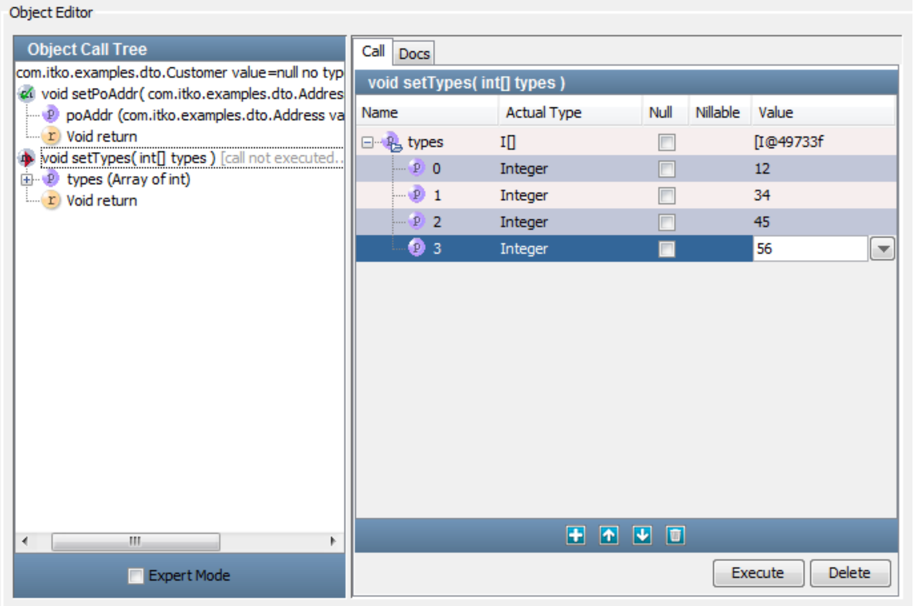
```
com.itko.examples.dto.Customer value=null no typ
void setPoAddr( com.itko.examples.dto.Addres
    poAddr (com.itko.examples.dto.Address va
    Void return
```

void setPoAddr(com.itko.examples.dto.Address poAddr)

Name	Actual Type	Null	Nillable	Value
poAddr	Address(com.itko.e...	☐		null, null
city	String	☑		null
line1	String	☑		null
line2	String	☑		null
state	String	☑		null
zip	String	☑		76092

5. Click the Execute button to invoke the setPoAddr method.

6. Select the setTypes method on the Call Sheet and click ▶ Invoke Method.



Object Editor

Object Call Tree

```
com.itko.examples.dto.Customer value=null no typ
void setPoAddr( com.itko.examples.dto.Addres
    poAddr (com.itko.examples.dto.Address va
    Void return
void setTypes( int[] types ) [call not executed...
    types (Array of int)
    Void return
```

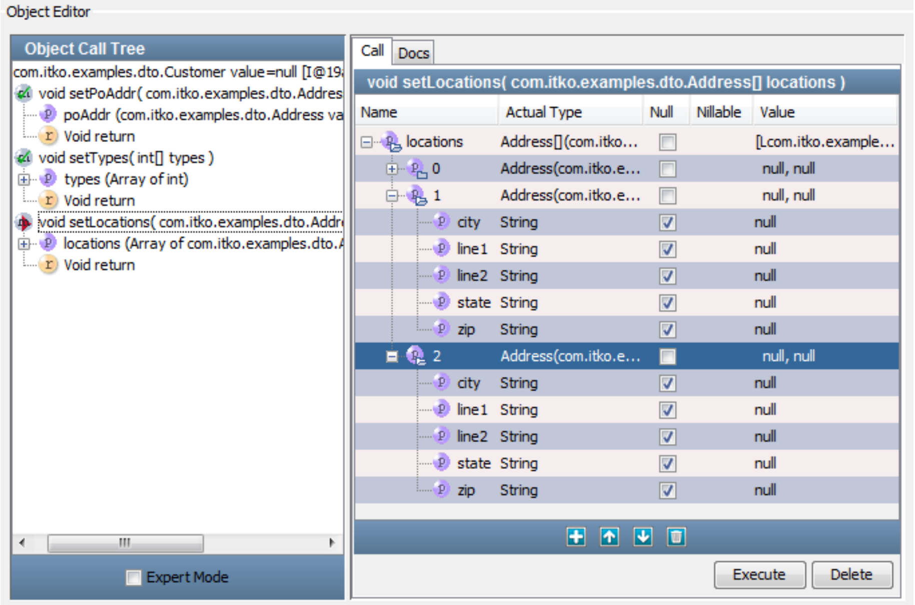
void setTypes(int[] types)

Name	Actual Type	Null	Nillable	Value
types	I[]	☐		[I@49733f
0	Integer	☐		12
1	Integer	☐		34
2	Integer	☐		45
3	Integer	☐		56

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 the Execute button to invoke the setTypes method.

9. Select the setLocations method on the Call Sheet and click ▶ Invoke Method.



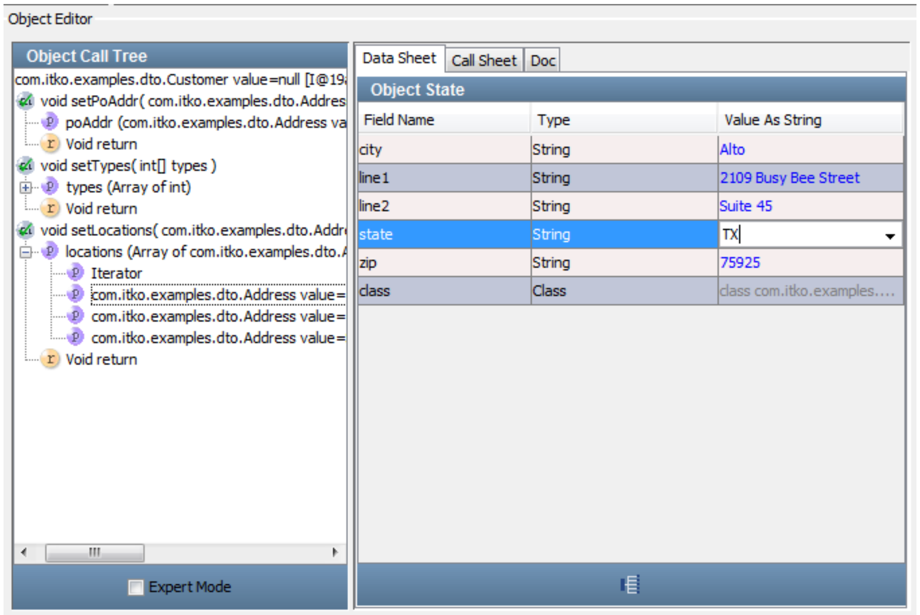
Object Editor
Object Call Tree
com.itko.examples.dto.Customer value=null [I@19
void setPoAddr(com.itko.examples.dto.Addres
poAddr (com.itko.examples.dto.Address va
Void return
void setTypes(int[] types)
types (Array of int)
Void return
void setLocations(com.itko.examples.dto.Addr
locations (Array of com.itko.examples.dto.A
Void return
Expert Mode
Call
Docs
Execute
Delete

void setLocations(com.itko.examples.dto.Address[] locations)

Name	Actual Type	Null	Nillable	Value
locations	Address[](com.itko...	□		[Lcom.itko.example...
0	Address(com.itko.e...	□		null, null
1	Address(com.itko.e...	□		null, null
city	String	☑		null
line1	String	☑		null
line2	String	☑		null
state	String	☑		null
zip	String	☑		null
2	Address(com.itko.e...	□		null, null
city	String	☑		null
line1	String	☑		null
line2	String	☑		null
state	String	☑		null
zip	String	☑		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.



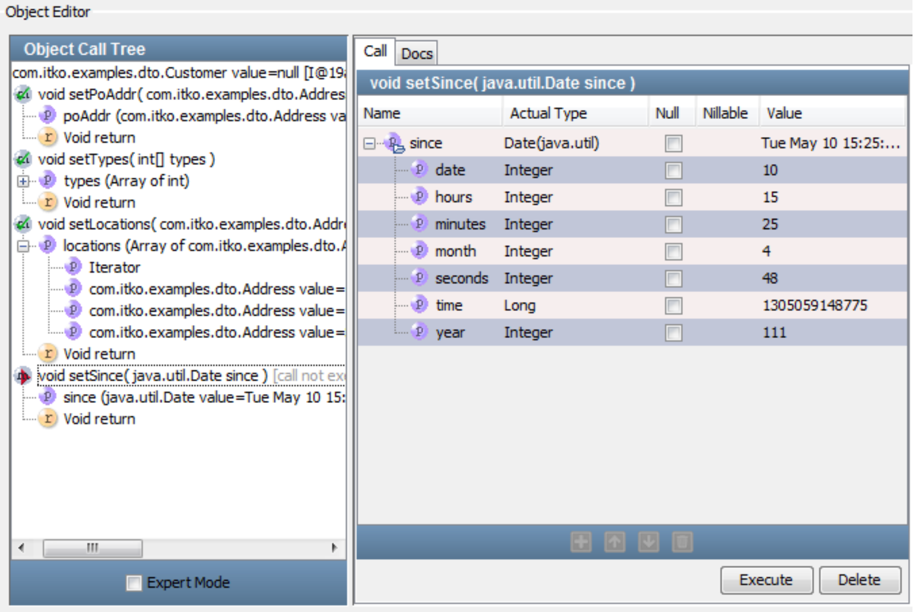
Object Editor
Object Call Tree
com.itko.examples.dto.Customer value=null [I@19
void setPoAddr(com.itko.examples.dto.Addres
poAddr (com.itko.examples.dto.Address va
Void return
void setTypes(int[] types)
types (Array of int)
Void return
void setLocations(com.itko.examples.dto.Addr
locations (Array of com.itko.examples.dto.A
Iterator
com.itko.examples.dto.Address value=
com.itko.examples.dto.Address value=
com.itko.examples.dto.Address value=
Void return
Expert Mode
Data Sheet
Call Sheet
Doc

Object State

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.



Object Editor

Object Call Tree

com.itko.examples.dto.Customer value=null [I@19
- void setPoAddr(com.itko.examples.dto.Addres
  - poAddr (com.itko.examples.dto.Address va
  - Void return
- void setTypes(int[] types)
  - types (Array of int)
  - Void return
- void setLocations(com.itko.examples.dto.Addr
  - locations (Array of com.itko.examples.dto.A
    - Iterator
    - com.itko.examples.dto.Address value=
    - com.itko.examples.dto.Address value=
    - com.itko.examples.dto.Address value=
  - Void return
- void setSince(java.util.Date since) [call not ex
  - since (java.util.Date value=Tue May 10 15:
  - Void return

□ Expert Mode

void setSince(java.util.Date since)

Name	Actual Type	Null	Nillable	Value
since	Date(java.util)	□		Tue May 10 15:25:...
date	Integer	□		10
hours	Integer	□		15
minutes	Integer	□		25
month	Integer	□		4
seconds	Integer	□		48
time	Long	□		1305059148775
year	Integer	□		111

The input parameters for the Data object are displayed and can be given values.

13. Click the Execute button.

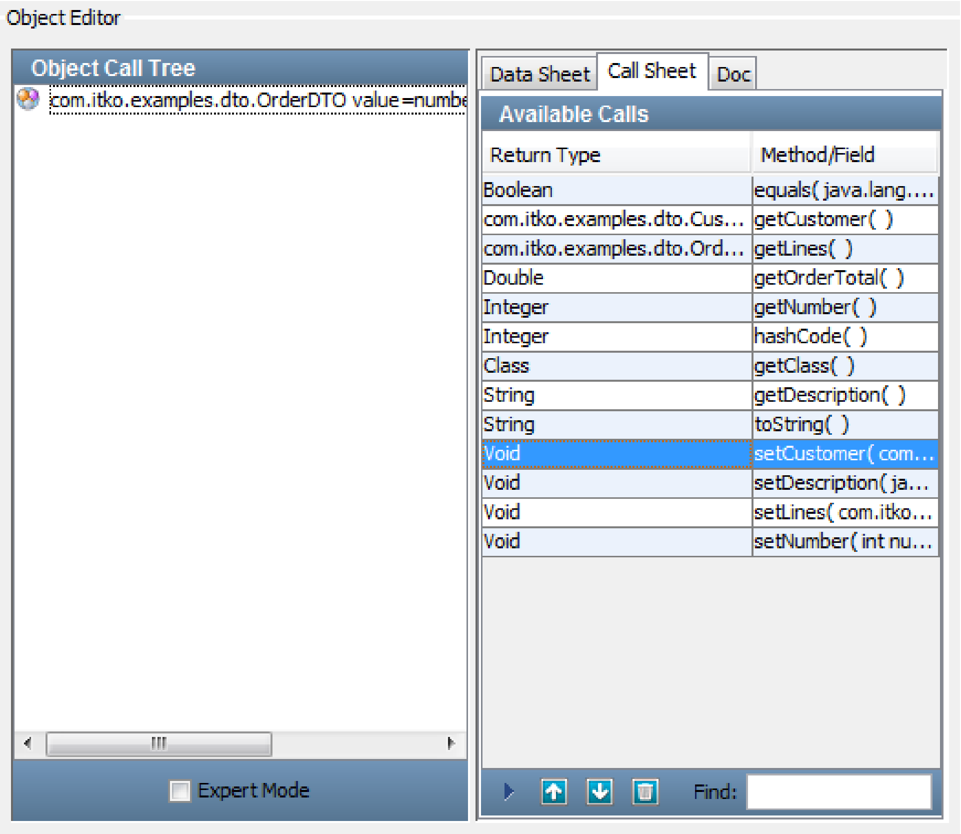
    The Customer object is now fully specified and can be used in your test case.

Complex DTO Object Scenario 2

The last scenario shows an example that builds on the last three scenarios.

This DTO, com.itko.examples.dto.OrderDTO, has a Customer object as one of its properties. This scenario shows how easy it is to build a Customer object in simple mode without calling any setter methods.

The OrderDTO object was loaded in COE using a Dynamic Java Execution step.



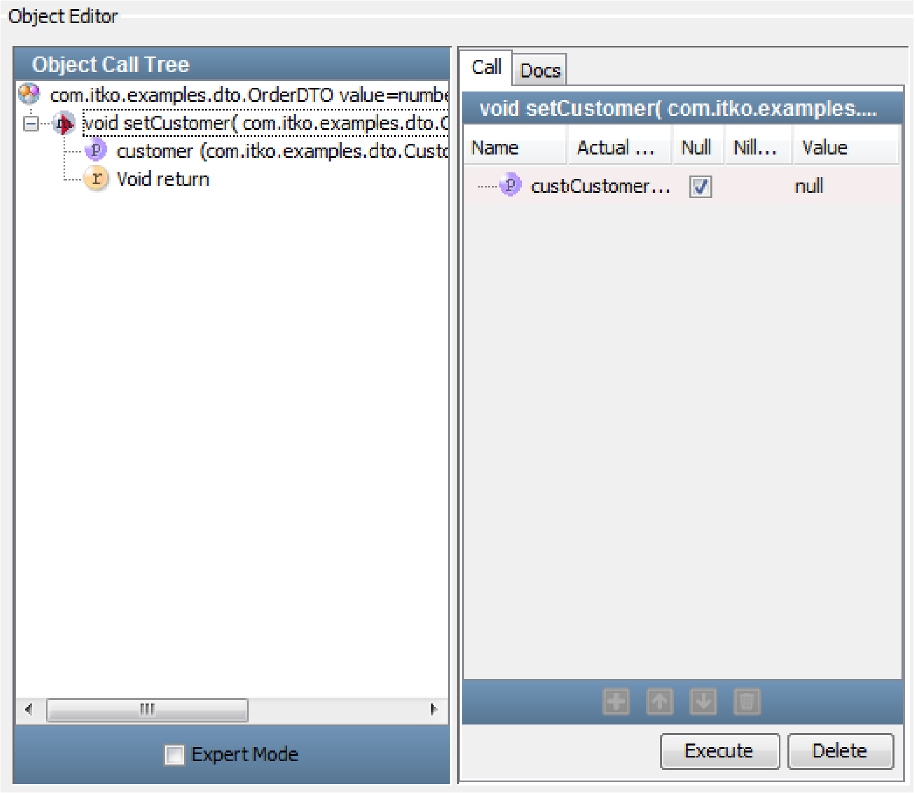
com.itko.examples.dto.OrderDTO value=numbe

Return Type	Method/Field
Boolean	equals(java.lang.…
com.itko.examples.dto.Cus…	getCustomer( )
com.itko.examples.dto.Ord…	getLines( )
Double	getOrderTotal( )
Integer	getNumber( )
Integer	hashCode( )
Class	getClass( )
String	getDescription( )
String	toString( )
Void	setCustomer(com…
Void	setDescription(ja…
Void	setLines(com.itko…
Void	setNumber(int nu…

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.


Object Editor

Object Call Tree

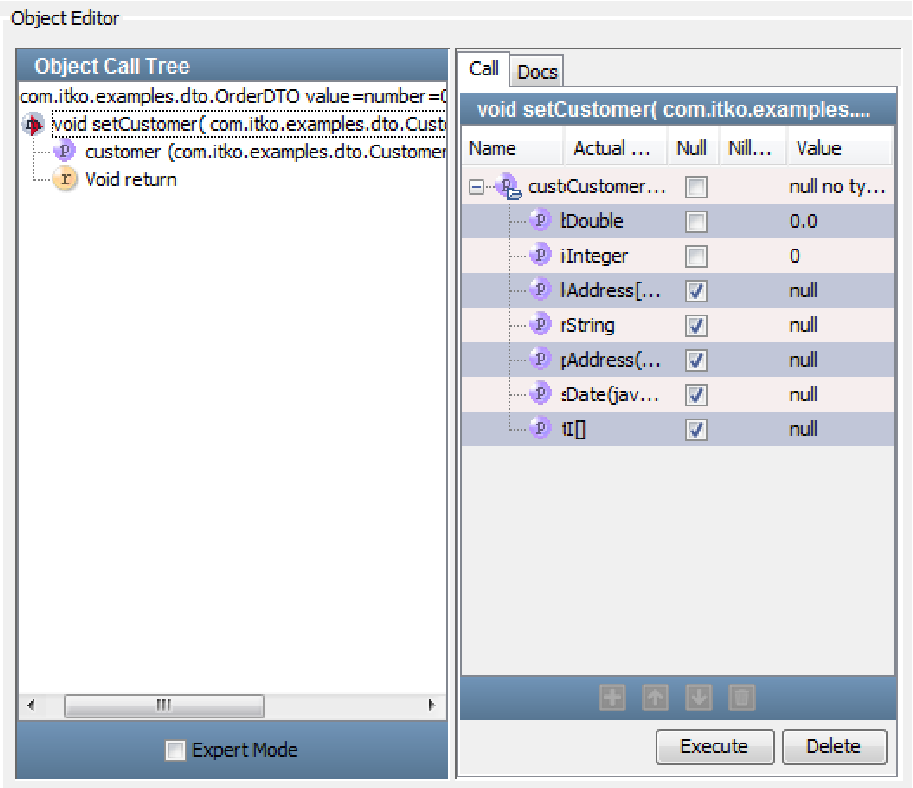
com.itko.examples.dto.OrderDTO value=numbe
void setCustomer(com.itko.examples.dto.C
customer (com.itko.examples.dto.Custo
Void return

□ Expert Mode

Name	Actual ...	Null	Nill...	Value
custCustomer...		☑		null

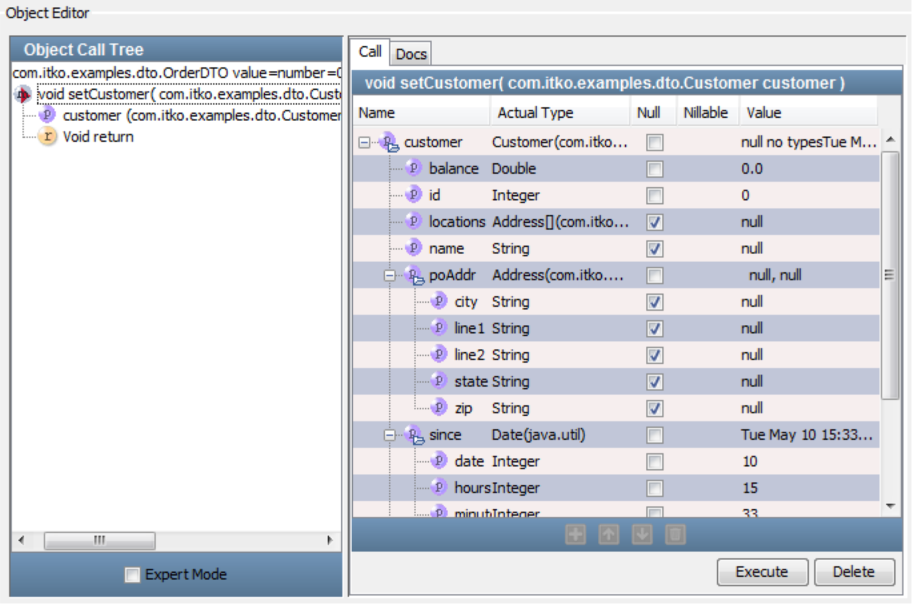
2. Clear the check box in the Null column.

   The Customer row expands to expose its properties.


Object Editor
Object Call Tree
com.itko.examples.dto.OrderDTO value=number=(
void setCustomer(com.itko.examples.dto.Cust
customer (com.itko.examples.dto.Customer
Void return
Expert Mode
Call
Docs
void setCustomer(com.itko.examples....
Execute
Delete

Name	Actual ...	Null	Nil...	Value
cust	Customer...	□		null no ty...
t	Double	□		0.0
i	Integer	□		0
l	Address[...	☑		null
r	String	☑		null
;	Address(...	☑		null
s	Date(jav...	☑		null
t	I[]	☑		null

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.



com.itko.examples.dto.OrderDTO value=number=(
void setCustomer(com.itko.examples.dto.Cust
customer (com.itko.examples.dto.Customer
Void return

void setCustomer(com.itko.examples.dto.Customer customer)

Name	Actual Type	Null	Nillable	Value
customer	Customer(com.itko...	□		null no typesTue M...
balance	Double	□		0.0
id	Integer	□		0
locations	Address[](com.itko...	☑		null
name	String	☑		null
poAddr	Address(com.itko....	□		null, null
city	String	☑		null
line1	String	☑		null
line2	String	☑		null
state	String	☑		null
zip	String	☑		null
since	Date(java.util)	□		Tue May 10 15:33...
date	Integer	□		10
hours	Integer	□		15
minut	Integer	□		33

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.

This section contains the following topics:

Add a Test Step (see page 189)
Configure Test Steps (see page 194)
Add Filters - Assertions - Data Sets to a Step (see page 196)
Configure Next Step (see page 196)
Replay to a Step (see page 197)
Set a Starter Step (see page 197)
Generate Warnings and Errors (see page 198)

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 190)

Add a Test Step - Example

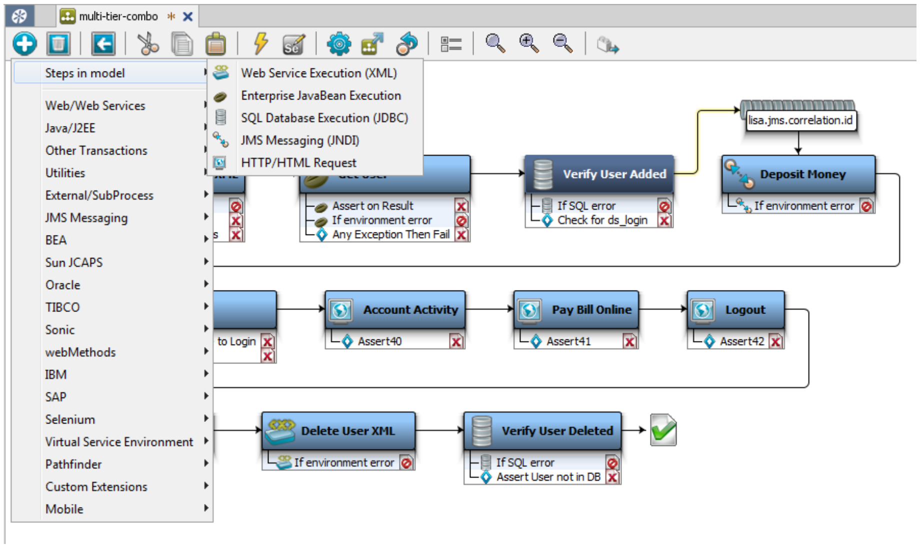
This example adds a step to the multi-tier-combo test case in the examples directory. A new Dynamic Java Execution step will be added to the multi-tier-combo test case, after the Get User step.

Follow these steps:

1. Click the Add Step button.

   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).

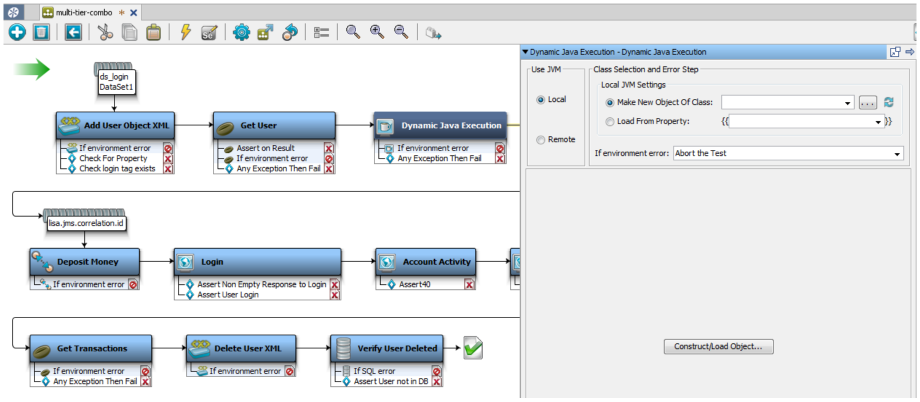
   The sub category opens.

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.



Dynamic Java Execution - Dynamic Java Execution

Use JVM

● Local

○ Remote

Class Selection and Error Step

Local JVM Settings

● Make New Object Of Class:

○ Load From Property: {{ }}

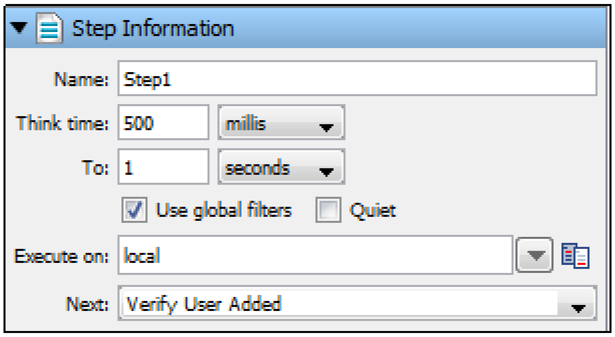
If environment error: Abort the Test

Add Step Information

Follow these steps:

1. To add the basic step information, open and expand the Step Information tab in the Elements tree in the right panel.

   The Step Information editor opens.

   Step Information

   Name: Step1

   Think time: 500 millis

   To: 1 seconds

   ☑ Use global filters ☐ Quiet

   Execute on: local

   Next: Verify User Added

2. Enter the following parameters:

   Name

   The name of the step. You can rename the step in this text box.

   Think Time

   The amount of time the test case waits before executing this step. Think time lets DevTest simulate the amount of time it takes a user to decide what to do before taking action. To specify how the time is calculated:

   a. Select the time unit by clicking the appropriate drop-down (millis, seconds, minutes)

   b. Enter a starting value in the Think Time field

   c. In the To field, enter an end value and a time indicator.

   DevTest picks a random think time in the specified range. For example, to simulate a user think time for a random interval between 500 milliseconds and 1 second, enter "500 millis" and "1 seconds."

   Use Global Filters

   To instruct the step to use global filters, select this box. For more information about filters, see Add a Filter (see page 114) in *Using CA Application Test.*

   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

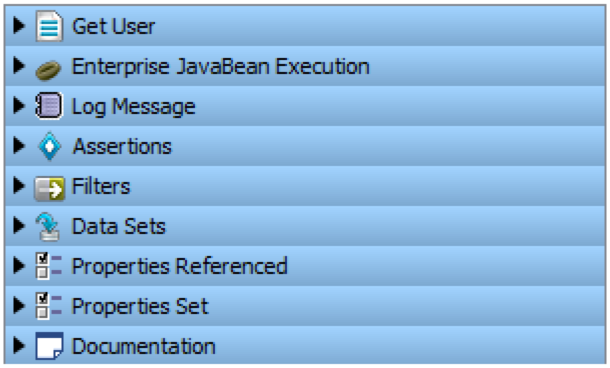
Follow these steps:

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

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 189).

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

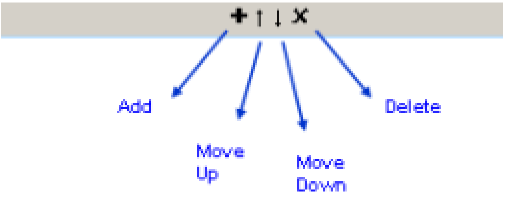
A read-only list of properties the step sets (assigns a value).

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.

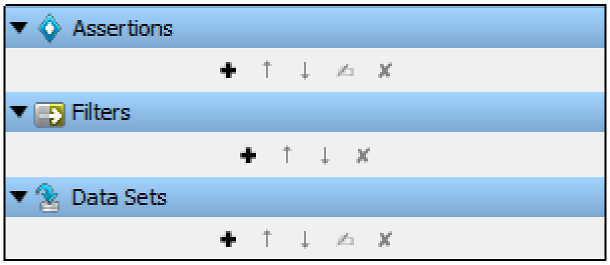


You can add/delete an element to a step by clicking the icons at the bottom of the individual elements.

- For detailed information about adding filters, see Add a Filter (see page 114).
- For detailed information about adding assertions, see Add an Assertion (see page 130).

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.



To add a filter, assertion, or data set, click Add ✚ on the toolbar.

For more information, see:

- Filters (see page 114)
- Assertions (see page 129)
- Data Sets (see page 145)

Configure Next Step

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:

Click the step and right-click to select Replay to here.

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:

Click the step and right-click to select Set as starter.

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.

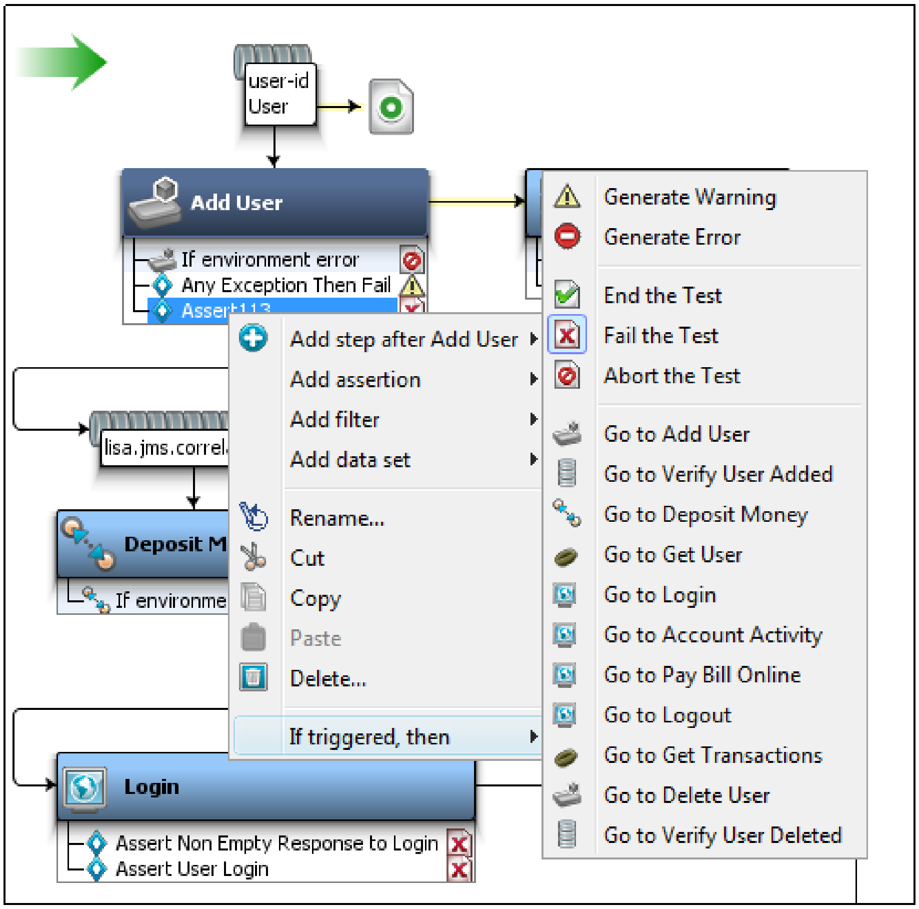
- Generate Warning
- Generate Error

For an example, we have selected the assertion in the Get User step.

Follow these steps:

1. To open a menu, select a step and right-click on the assertion.
2. Select Generate Warning.

   The test case will go to this next step (Generate Warning), only when the assertion is triggered.



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 46). In this project, you can create single or multiple test cases.

This section contains the following topics:

Create a Test Case (see page 200)
Open a Test Case (see page 200)
Save a Test Case (see page 201)
Test Cases in Model Editor (see page 202)
Add Test Steps (see page 203)
Configure the Next Step (see page 203)
Branching and Looping in a Test Case (see page 204)
Import Test Cases (see page 205)
Response (.rsp) Documents (see page 206)
Accessing Files in Another Project (see page 207)
Test Case Toolbar (see page 207)

Create a Test Case

You can create a test case by opening a project or creating a project (see page 46).

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 202) 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 206).

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:

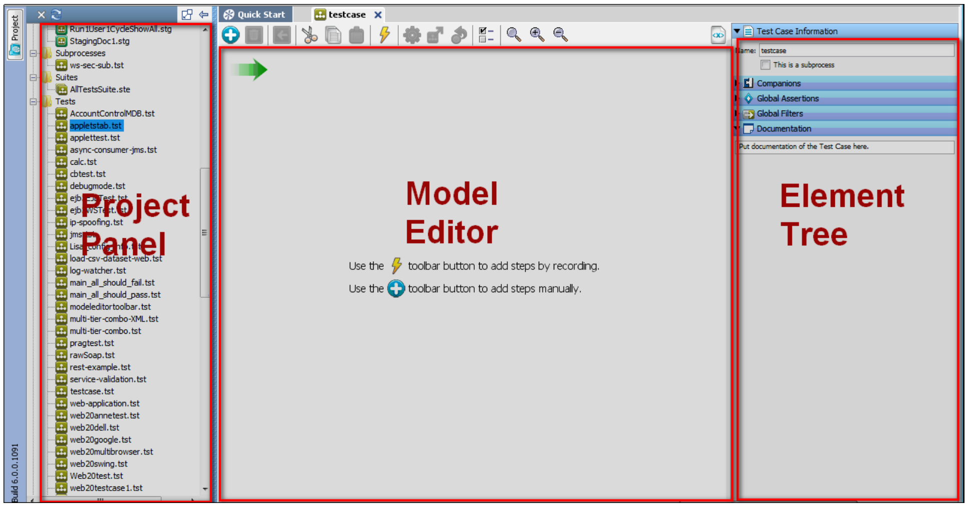
Do one:

- Click Save 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.



Use the toolbar button to add steps by recording.

Use the toolbar button to add steps manually.

Put documentation of the Test Case here.

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 130).

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 145).

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 114).

For more information about using the HTTP response to create assertions, see Assertions (see page 129).

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

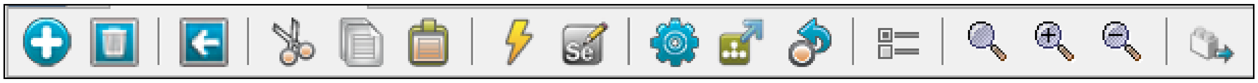
When you work in Project2, you can access a file in Project1 by specifying the following type of path:

```
{{LISA_PROJ_ROOT}}/../Project1/Data/AccountNumbers.xls
```

Be sure to provide a path that is valid across all your environments.

Test Case Toolbar

This toolbar opens after you open a test case in the model editor. All the tasks here are specific to a test case.



Icon	Description
⊕	Create a new step.
🗑	Delete a test step.
←	Set the currently selected step as the starting step in the workflow.
✂	Cut the selected text.
📄	Copy the selected text.
📋	Paste the selected text.

⚡	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
Se	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
⚙	Start a new Interactive Test Run.
↗	Stage a quick test.
↺	Replay a test to a specific point.
☑	Show model properties.
🔍	Reset zoom scale to 1:1.
⊕	Zoom in.
⊖	Zoom out.
<x>	View XML source.

Building Subprocesses

A subprocess is a test case that is called from another test case instead of run as a stand-alone test case.

Subprocesses can be used as modules in other test cases, which increases their ability to be reused. You can build a library of subprocesses that can be shared across many test cases.

In a programming language, a subprocess would be referred to as a function or a subroutine.

A test case must be self-contained. The value for all the properties that are used in the test case must come from in the test case. A subprocess expects the test step that runs it to provide some property values (input properties). When the subprocess completes, it makes property values available to the calling step (return properties).

Subprocesses can be nested. A subprocess can call another subprocess.

Note: If a subprocess that calls another subprocess is marked as Quiet, all called subprocesses will be Quiet.

You create the steps in the subprocess in the same way you do for a regular test case, with the following differences:

- Mark the test case as a subprocess in the Test Case Information tab of the test case (as explained in Create a Subprocess Test Case (see page 210) in *Using CA Application Test*).
- Do not add data sets as you would in a test case. Instead, the data set must be part of the calling case, and the current values are passed to the subprocess when it is invoked. The exception is when a data set is a part of the subprocess logic itself. In that scenario, do not use a global data set.
- Do not use a configuration file or a companion in the subprocess to initialize any parameters that you expect to be passed from the calling step. For testing purposes, we add these values elsewhere. When the subprocess is called, the calling step passes these values.

Note: The think time parameter in the parent test case (the test case that calls the subprocess) is propagated to the subprocess. To make your subprocess think times run independently of the calling process, set a testExec property with the name **lisa.subprocess.setThinkScaleFromParent** to "false" so that you can decide for each subprocess. For a global override, set **lisa.subprocess.setThinkScaleFromParent=false** in local.properties.

You can build subprocesses from scratch, or you can convert an existing test case into a subprocess.

The Execute Subprocess test step simplifes calling a subprocess test case.

This section contains the following topics:

Create a Subprocess Test Case (see page 210)
Convert an Existing Test Case into a Subprocess (see page 212)
Subprocess Example (see page 213)

Create a Subprocess Test Case

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, use the Delete button ✘.

   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, use the Add button ✚ at the bottom of this panel.

To remove properties that you do not want to be made available to the calling step, use the Delete ✘ button.

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

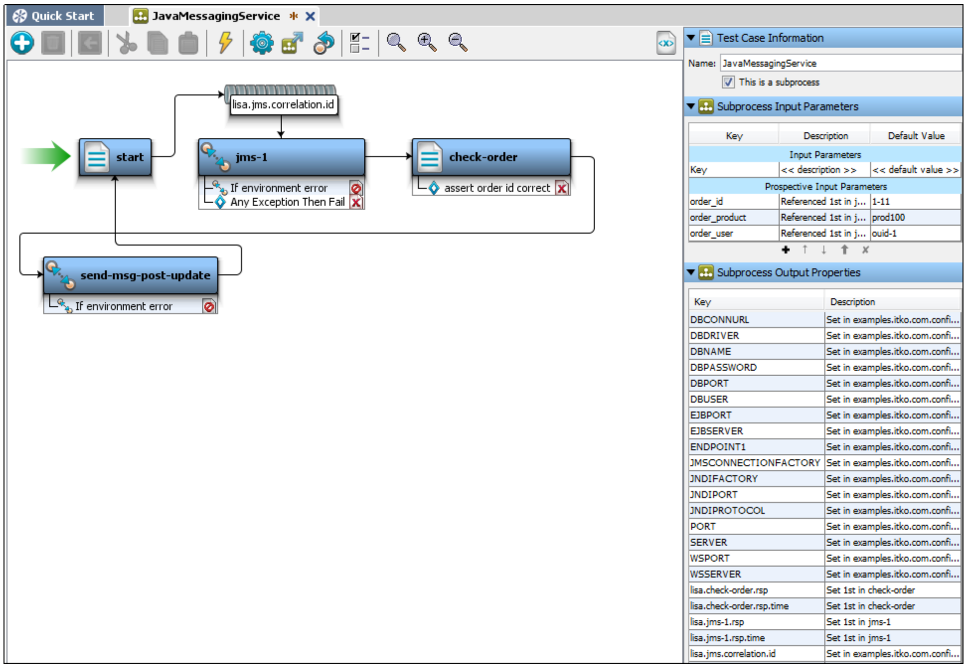
Follow these steps:

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.



Test Case Information

Name: JavaMessagingService

☑ This is a subprocess

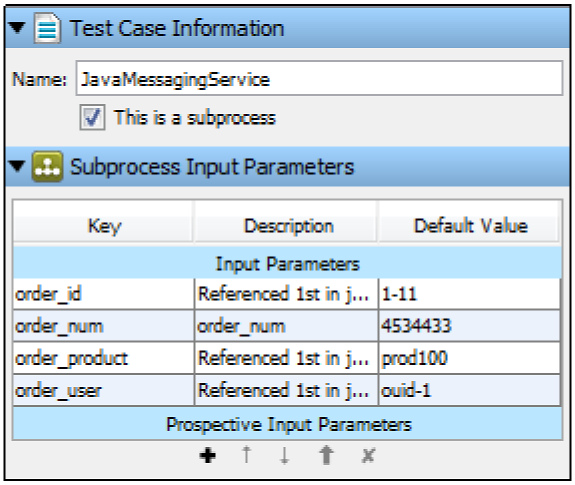
Subprocess Input Parameters

Key	Description	Default Value
Input Parameters		
Key	<< description >>	<< default value >>
Prospective Input Parameters		
order_id	Referenced 1st in j...	1-11
order_product	Referenced 1st in j...	prod100
order_user	Referenced 1st in j...	ouid-1

Subprocess Output Properties

Key	Description
DBCONNURL	Set in examples.itko.com.confi...
DBDRIVER	Set in examples.itko.com.confi...
DBNAME	Set in examples.itko.com.confi...
DBPASSWORD	Set in examples.itko.com.confi...
DBPORT	Set in examples.itko.com.confi...
DBUSER	Set in examples.itko.com.confi...
EJBPORT	Set in examples.itko.com.confi...
EJBSERVER	Set in examples.itko.com.confi...
ENDPOINT1	Set in examples.itko.com.confi...
JMSCONNECTIONFACTORY	Set in examples.itko.com.confi...
JNDIFACTORY	Set in examples.itko.com.confi...
JNDIPORT	Set in examples.itko.com.confi...
JNDIPROTOCOL	Set in examples.itko.com.confi...
PORT	Set in examples.itko.com.confi...
SERVER	Set in examples.itko.com.confi...
WSPORT	Set in examples.itko.com.confi...
WSSERVER	Set in examples.itko.com.confi...
lisa.check-order.rsp	Set 1st in check-order
lisa.check-order.rsp.time	Set 1st in check-order
lisa.jms-1.rsp	Set 1st in jms-1
lisa.jms-1.rsp.time	Set 1st in jms-1
lisa.jms.correlation.id	Set in examples.itko.com.confi...

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

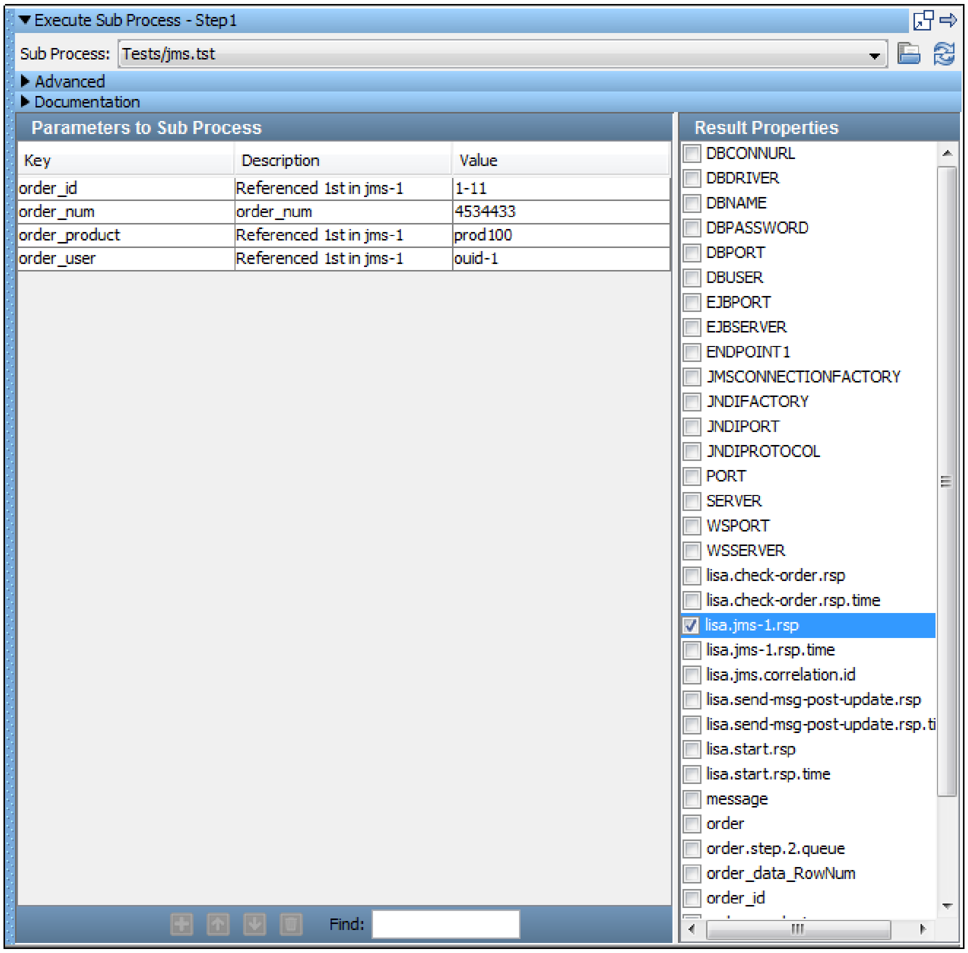
Name: JavaMessagingService

☑ 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...	ouid-1
Prospective Input Parameters		

The following graphic shows a test case with one test step: Execute Subprocess.



Sub Process: Tests/jms.tst

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
- □ EJBSERVER
- □ ENDPOINT1
- □ JMSCONNECTIONFACTORY
- □ JNDIFACTORY
- □ JNDIPORT
- □ JNDIPROTOCOL
- □ PORT
- □ SERVER
- □ WSPORT
- □ 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.ti
- □ lisa.start.rsp
- □ lisa.start.rsp.time
- □ message
- □ order
- □ order.step.2.queue
- □ order_data_RowNum
- □ order_id

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.

Chapter 11: 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.

This section contains the following topics:

Building Staging Documents (see page 215)
Building Audit Documents (see page 237)
Understanding Events (see page 239)
Generating Metrics (see page 243)
Building Test Suites (see page 243)

Building Staging Documents

A staging document contains the information about how to run a test case.

The section contains the following topics:

- Create a Staging Document (see page 216)
- Staging Document Editor (see page 217)
- Staging Document Examples (see page 236)

Create a Staging Document

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 218), 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 227), specify the type of report that you want to create at run time.
6. In the Metrics tab (see page 229), specify the metrics that you want to record at run time.
7. In the Documentation tab (see page 231), enter descriptive information about the staging document.
8. (Optional) In the IP Spoofing tab (see page 232), enable, and configure IP spoofing.
9. (Optional) In the Source View tab (see page 235), 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.

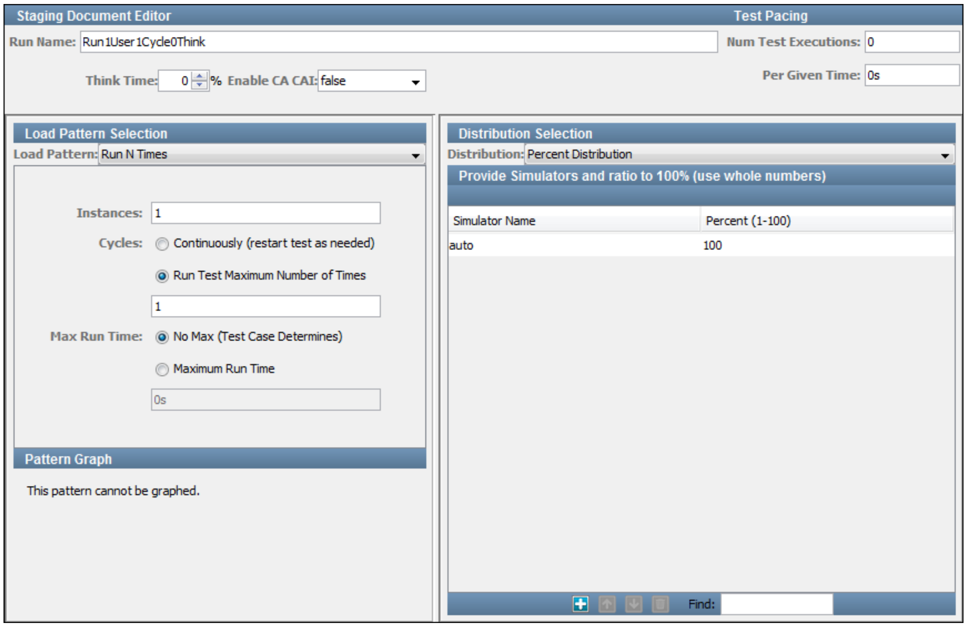
The Staging Document Editor contains the following tabs:

- Base (see page 218): To specify the basic parameters.
- Reports (see page 227): To select and add reports.
- Metrics (see page 229): To select metrics and specify your sampling intervals.
- Documentation (see page 231): To enter descriptive information about the staging document.
- IP Spoofing (see page 232): 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 235): 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



Staging Document Editor

Run Name: Run1User1Cycle0Think

Think Time: 0 % Enable CA CAI: false

Test Pacing

Num Test Executions: 0

Per Given Time: 0s

Load Pattern Selection

Load Pattern: Run N Times

Instances: 1

Cycles: ○ Continuously (restart test as needed)

● Run Test Maximum Number of Times

1

Max Run Time: ● No Max (Test Case Determines)

○ Maximum Run Time

0s

Pattern Graph

This pattern cannot be graphed.

Distribution Selection

Distribution: Percent Distribution

Provide Simulators and ratio to 100% (use whole numbers)

Simulator Name	Percent (1-100)
auto	100

Find:

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 303) 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 221).

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 225).

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

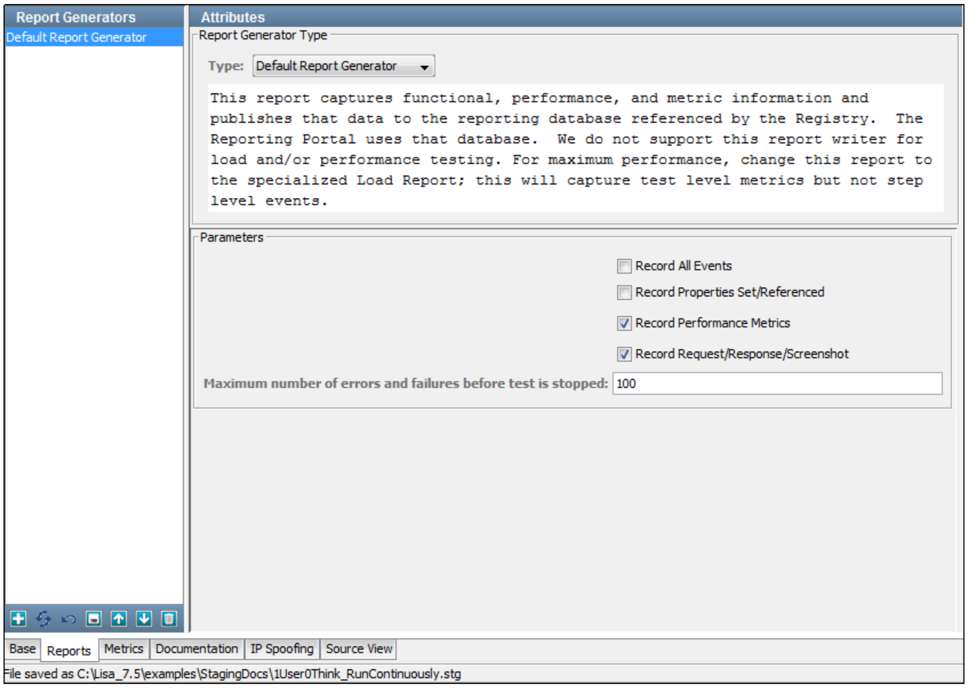
The Round Robin Distribution requests that DevTest control the allocation of virtual users (instances), based on a simple round-robin distribution pattern.

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.



Type: Default Report Generator

```
This report captures functional, performance, and metric information and
publishes that data to the reporting database referenced by the Registry.  The
Reporting Portal uses that database.  We do not support this report writer for
load and/or performance testing. For maximum performance, change this report to
the specialized Load Report; this will capture test level metrics but not step
level events.
```

☐ Record All Events

☐ Record Properties Set/Referenced

☑ Record Performance Metrics

☑ Record Request/Response/Screenshot

Maximum number of errors and failures before test is stopped: 100

File saved as C:\Lisa_7.5\examples\StagingDocs\1User0Think_RunContinuously.stg

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 369).

The metrics to capture for the reports are discussed more fully in Generating Metrics (see page 243).

Reports can be selected in the following modules and can be seen in the Report Viewer:

- Stage a Quick Test (see page 281)
- Building Staging Documents (see page 215)
- Run a Test Suite (see page 297)

After they are requested, they can be viewed and managed later.

When you select a report from the drop-down list, a summary of the report appears in the area below.

According to the selected report, the parameters change. Set the parameters as and where necessary.

The Reports tab is divided into the following areas:

Right panel

The right panel is where you select the report to be added.

Attributes

The Attributes area contains the Report Generator Type and the required parameters for the report.

- Report Generator Type**:** A pull-down menu lists the available report types.
- Parameters**:** The parameters that are required to set the selected report generator.

  The Maximum number of errors before test is stopped field limits the number of errors that the test can have before LISA aborts it. A zero entry means to allow all errors.

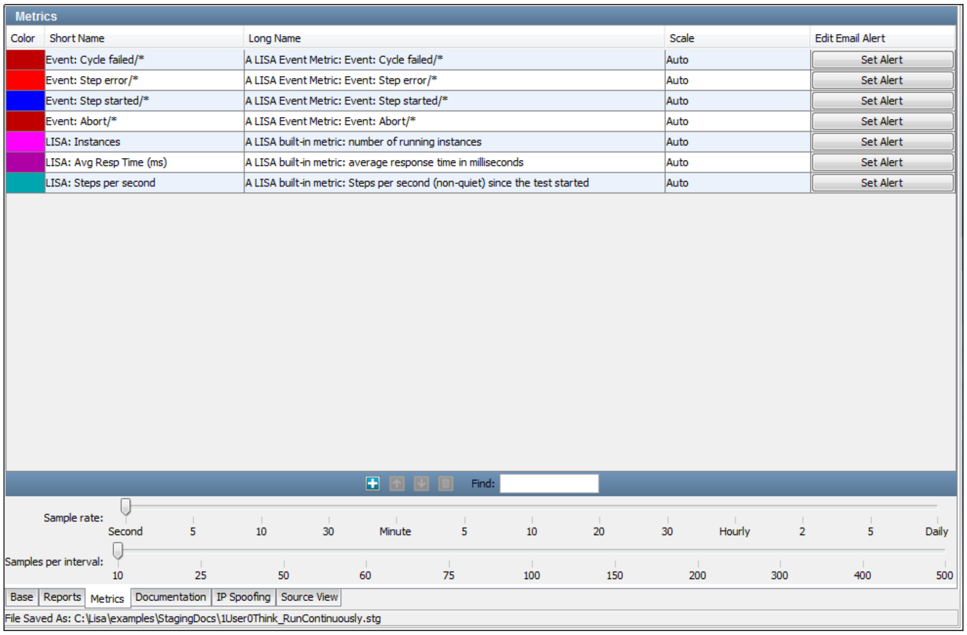
Left panel

The left panel shows the list of added reports. You can add, save, move, and delete reports by using the toolbar at the bottom of the panel.

Staging Document Editor - Metrics Tab

The Metrics tab of the Staging Document Editor lets you select the test metrics to record.

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
🟥	Event: Cycle failed/*	A LISA Event Metric: Event: Cycle failed/*	Auto	Set Alert
🟥	Event: Step error/*	A LISA Event Metric: Event: Step error/*	Auto	Set Alert
🟦	Event: Step started/*	A LISA Event Metric: Event: Step started/*	Auto	Set Alert
🟥	Event: Abort/*	A LISA Event Metric: Event: Abort/*	Auto	Set Alert
🟪	LISA: Instances	A LISA built-in metric: number of running instances	Auto	Set Alert
🟪	LISA: Avg Resp Time (ms)	A LISA built-in metric: average response time in milliseconds	Auto	Set Alert
🟦	LISA: Steps per second	A LISA built-in metric: Steps per second (non-quiet) since the test started	Auto	Set Alert

File Saved As: C:\Lisa\examples\StagingDocs\1User0Think_RunContinuously.stg

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.

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 243).

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.

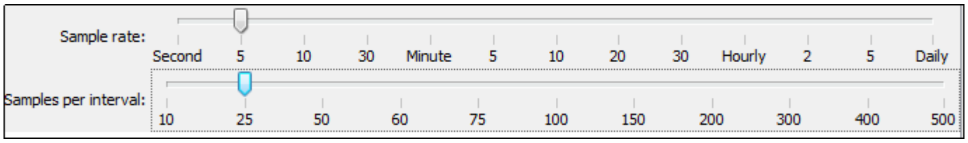
Sampling Parameters

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).


Sample rate: 5
Samples per interval: 25

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 227)).

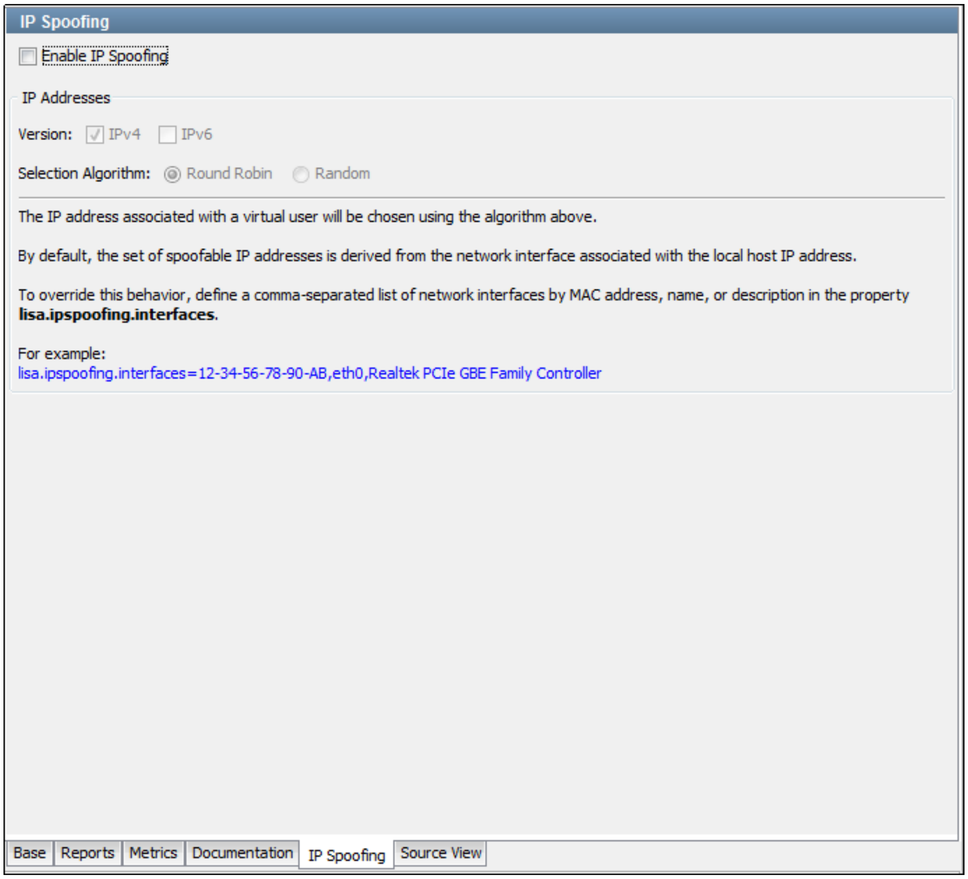
Staging Document Editor - Documentation Tab

The Documentation tab of the Staging Document Editor lets you enter descriptive information about the staging document.

Staging Document Editor - IP Spoofing Tab

The IP Spoofing tab of the Staging Document Editor lets multiple IP addresses on a network interface to be used when making network requests.

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.



□ Enable IP Spoofing

IP Addresses

Version: ☑ IPv4 □ IPv6

Selection Algorithm: ● Round Robin ○ Random

The IP address associated with a virtual user will be chosen using the algorithm above.

By default, the set of spoofable IP addresses is derived from the network interface associated with the local host IP address.

To override this behavior, define a comma-separated list of network interfaces by MAC address, name, or description in the property **lisa.ipspoofing.interfaces**.

For example:
lisa.ipspoofing.interfaces=12-34-56-78-90-AB,eth0,Realtek PCIe GBE Family Controller

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.

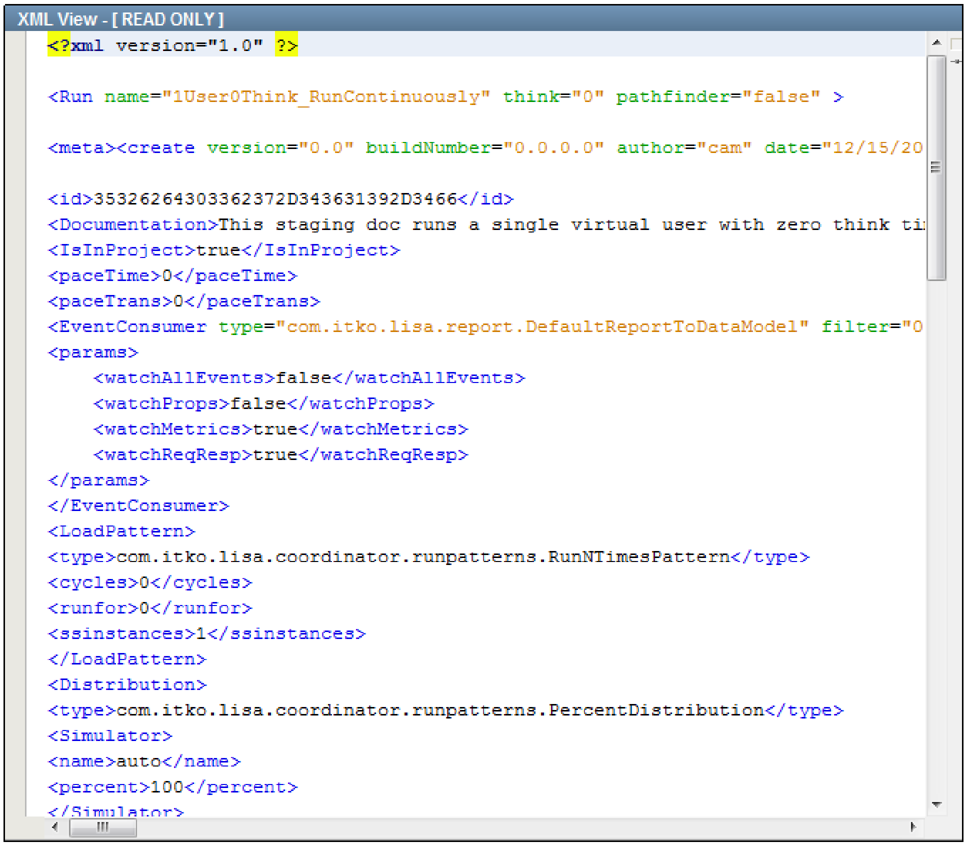
For example, the following command adds 9 more IP addresses between 192.168.0.202 and 192.168.0.210:

```
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.



```
<?xml version="1.0" ?>

<Run name="1User0Think_RunContinuously" think="0" pathfinder="false" >

<meta><create version="0.0" buildNumber="0.0.0.0" author="cam" date="12/15/20

<id>35326264303362372D343631392D3466</id>
<Documentation>This staging doc runs a single virtual user with zero think ti
<IsInProject>true</IsInProject>
<paceTime>0</paceTime>
<paceTrans>0</paceTrans>
<EventConsumer type="com.itko.lisa.report.DefaultReportToDataModel" filter="0
<params>
    <watchAllEvents>false</watchAllEvents>
    <watchProps>false</watchProps>
    <watchMetrics>true</watchMetrics>
    <watchReqResp>true</watchReqResp>
</params>
</EventConsumer>
<LoadPattern>
<type>com.itko.lisa.coordinator.runpatterns.RunNTimesPattern</type>
<cycles>0</cycles>
<runfor>0</runfor>
<ssinstances>1</ssinstances>
</LoadPattern>
<Distribution>
<type>com.itko.lisa.coordinator.runpatterns.PercentDistribution</type>
<Simulator>
<name>auto</name>
<percent>100</percent>
</Simulator>
```

Staging Document Examples

Sample staging documents are provided in the StagingDocs folder of the examples project (see page 50). 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

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

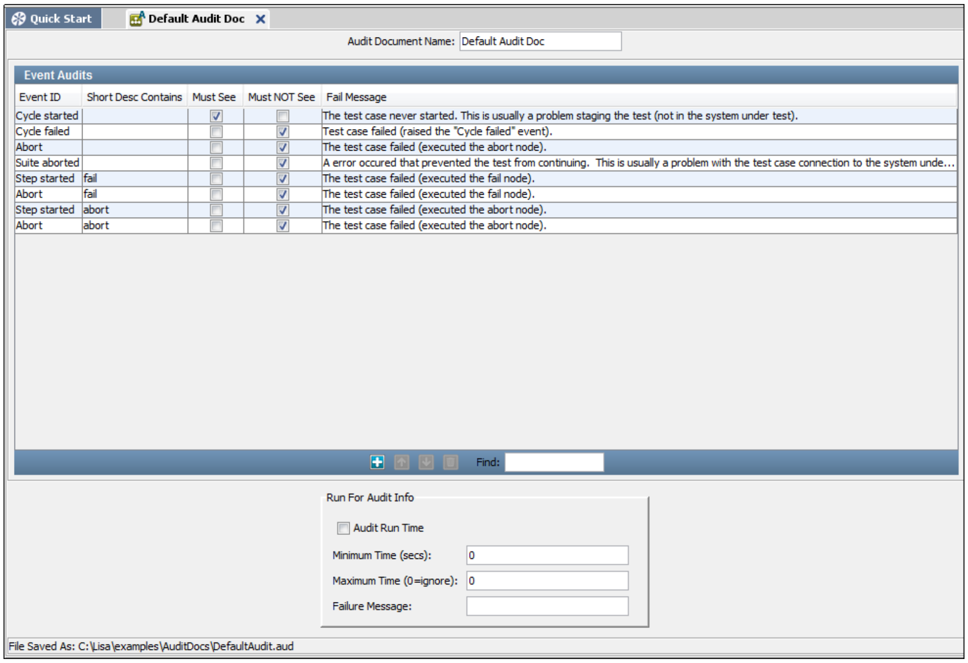
You can specify audit documents in the Base tab (see page 245) of the suite document editor. Be sure to select the Record All Events check box in the Reports tab (see page 249).

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 Name: Default Audit Doc

Event Audits

Event ID	Short Desc Contains	Must See	Must NOT See	Fail Message
Cycle started		☑	☐	The test case never started. This is usually a problem staging the test (not in the system under test).
Cycle failed		☐	☑	Test case failed (raised the "Cycle failed" event).
Abort		☐	☑	The test case failed (executed the abort node).
Suite aborted		☐	☑	A error occured that prevented the test from continuing. This is usually a problem with the test case connection to the system unde...
Step started	fail	☐	☑	The test case failed (executed the fail node).
Abort	fail	☐	☑	The test case failed (executed the fail node).
Step started	abort	☐	☑	The test case failed (executed the abort node).
Abort	abort	☐	☑	The test case failed (executed the abort node).

Run For Audit Info

☐ Audit Run Time

Minimum Time (secs): 0

Maximum Time (0=ignore): 0

Failure Message:

File Saved As: C:\Lisa\examples\AuditDocs\DefaultAudit.aud

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 238) panel.
4. To audit the execution time, configure the parameters in the Run for Audit Info (see page 239) 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 broadcast from DevTest informing any interested parties that an action has occurred.

This section contains the following topics:

- Events Overview (see page 240)
- Adding and Viewing Events (see page 242)

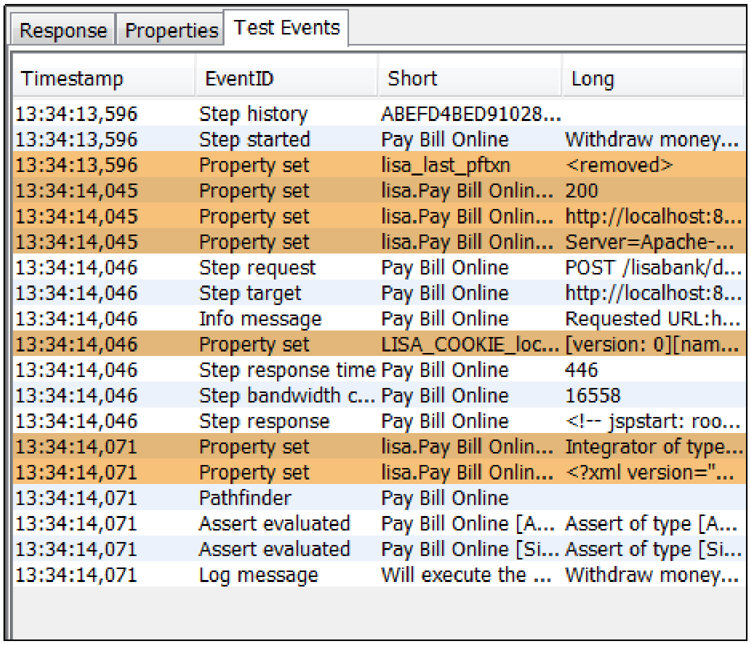
Events Overview

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.

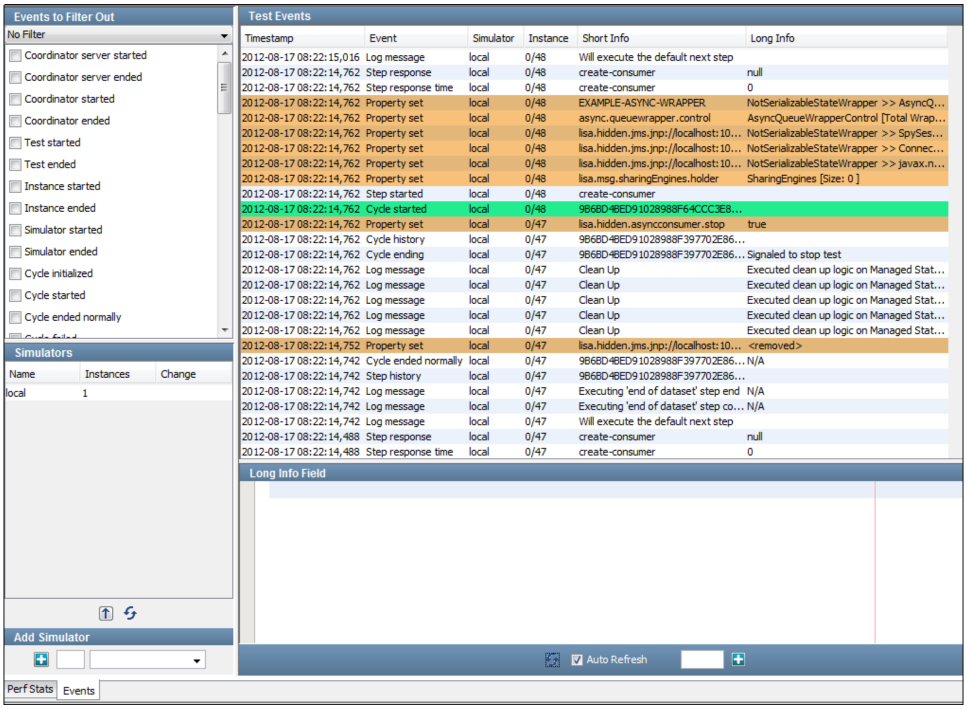
Events are important when you are monitoring tests or analyzing test results.

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.



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...

You can observe and filter events in a Quick Test.



Name	Instances	Change
local	1	

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 >> SpySes…
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	Property set	local	0/48	lisa.msg.sharingEngines.holder	SharingEngines [Size: 0]
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	9B6BD4BED91028988F64CCC3E8…	
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	9B6BD4BED91028988F397702E86…	
2012-08-17 08:22:14,762	Cycle ending	local	0/47	9B6BD4BED91028988F397702E86…	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 08:22:14,752	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	9B6BD4BED91028988F397702E86…	N/A
2012-08-17 08:22:14,742	Step history	local	0/47	9B6BD4BED91028988F397702E86…	
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

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 369). For more information about metrics, see Generating Metrics (see page 243).

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.

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 369). For more information about metrics, see Generating Metrics (see page 243).

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.

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 281): Use to monitor the test.
- Staging Documents (see page 229): Use to include in reports.
- Test Suite Documents (see page 251): Use to include in reports.
- Test Monitors (see page 282): 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.

This section contains the following topics:

Create a Test Suite (see page 244)
Test Suite Editor (see page 244)

Create 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 245), specify basic information about the suite.

   This information includes the suite name and the suite entries.

3. In the Reports tab (see page 249), specify the type of report that you want to create at run time.
4. In the Metrics tab (see page 251), specify the metrics that you want to record at run time.
5. In the Documentation tab (see page 253), enter descriptive information about the suite.
6. Select File, Save from the main menu.

Test Suite Editor

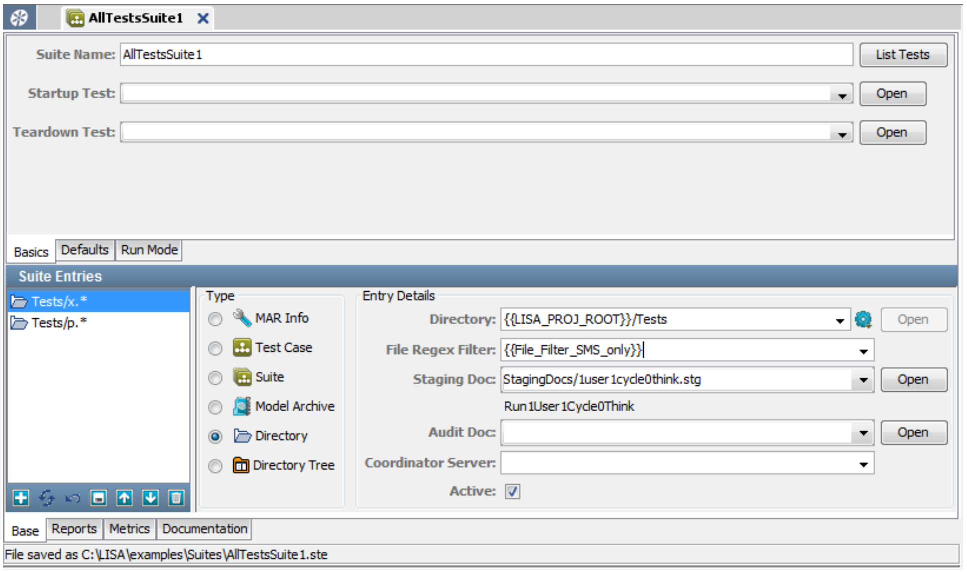
When you create a test suite, and when you open an existing test suite, the Test Suite Editor opens.

For more information about the specific tabs on this page, see the following:

- Test Suite Editor - Base Tab (see page 245)
- Test Suite Editor - Reports Tab (see page 249)
- Test Suite Editor - Metrics Tab (see page 251)
- Test Suite Editor - Documentation Tab (see page 253)

Test Suite Editor - Base Tab

The Base tab of the Test Suite Editor contains basic information about a test suite.



Suite Name: AllTestsSuite1

Startup Test:

Teardown Test:

Suite Entries: Tests/x.*, Tests/p.*

Directory: {{LISA_PROJ_ROOT}}/Tests

File Regex Filter: {{File_Filter_SMS_only}}

Staging Doc: StagingDocs/1user1cycle0think.stg

Run1User1Cycle0Think

Audit Doc:

Coordinator Server:

Active: ☑

File saved as C:\LISA\examples\Suites\AllTestsSuite1.ste

Top Panel

The top panel of the Base tab has basic information about the test suite as a whole.

To configure a suite, you also must enter parameters in all the sub tabs.

Basics Tab

This tab has the following parameters:

Suite Name

The name of the suite.

Startup Test/Teardown Test

You can specify a startup test that runs before the suite has begun, and a teardown test that runs after the suite has completed. The startup test and the teardown test are not included in any test statistics. Events in these tests appear in the reports. If the startup test fails, test suite testing does not continue. You can use a MAR info file as a startup or teardown test. The primary asset of the MAR info file must be a test case.

List Tests

Displays a dialog window with a list of the tests currently included in your suite. Save your suite for this list to be current.

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.

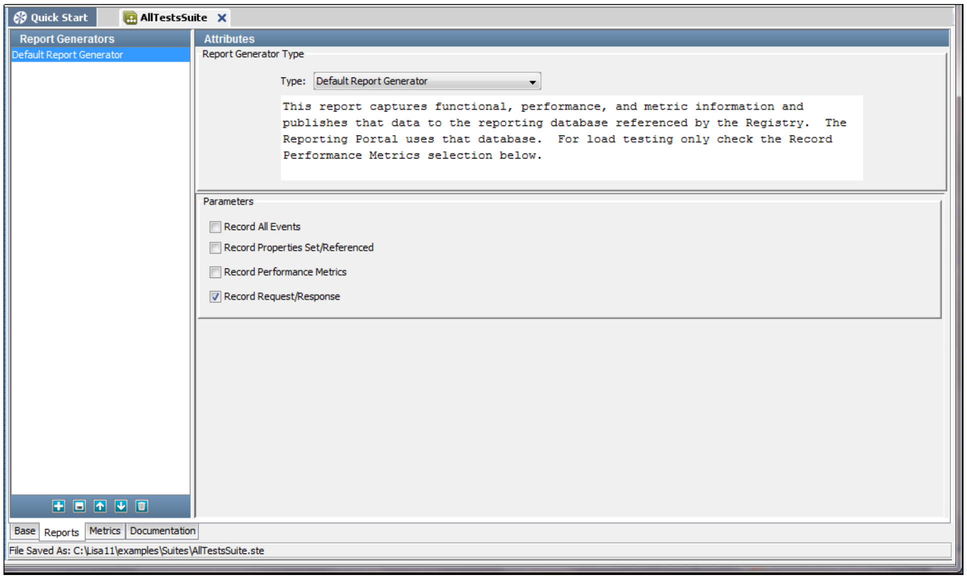
Click Add on the toolbar to add this entry to the list shown in the left panel. You can delete entries by clicking Delete. You can rearrange entries by using the Move Up and Move Down icons.

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 369). The metrics to capture for the reports are discussed more fully in Metrics (see page 243).



Type: Default Report Generator

```
This report captures functional, performance, and metric information and
publishes that data to the reporting database referenced by the Registry.  The
Reporting Portal uses that database.  For load testing only check the Record
Performance Metrics selection below.
```

□ Record All Events

□ Record Properties Set/Referenced

□ Record Performance Metrics

☑ Record Request/Response

File Saved As: C:\Lisa11\examples\Suites\AllTestsSuite.ste

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 370)
- XML Report Generator (see page 371)

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.

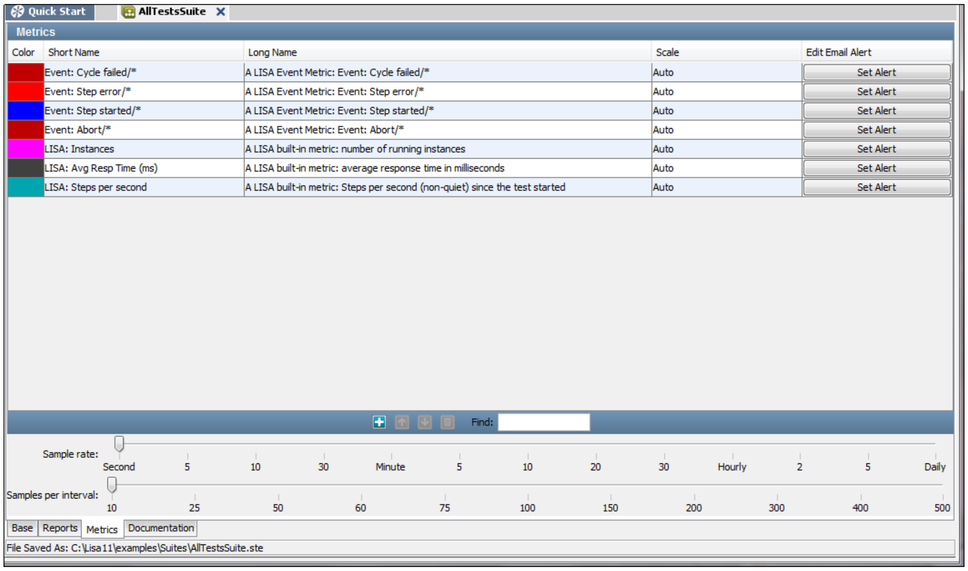
Record Request/Response

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.



Color	Short Name	Long Name	Scale	Edit Email Alert
🟥	Event: Cycle failed/*	A LISA Event Metric: Event: Cycle failed/*	Auto	Set Alert
🟥	Event: Step error/*	A LISA Event Metric: Event: Step error/*	Auto	Set Alert
🟦	Event: Step started/*	A LISA Event Metric: Event: Step started/*	Auto	Set Alert
🟥	Event: Abort/*	A LISA Event Metric: Event: Abort/*	Auto	Set Alert
🟪	LISA: Instances	A LISA built-in metric: number of running instances	Auto	Set Alert
⬛	LISA: Avg Resp Time (ms)	A LISA built-in metric: average response time in milliseconds	Auto	Set Alert
🟦	LISA: Steps per second	A LISA built-in metric: Steps per second (non-quiet) since the test started	Auto	Set Alert

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:\Lisa11\examples\Suites\AllTestsSuite.ste

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).

Adding Metrics

The following metrics categories are available:

- DevTest Whole Test Metrics
- DevTest Test Event Metrics
- JMX Metrics
- SNMP Metrics
- TIBCO Hawk Metrics
- Windows Perfmon Metrics
- UNIX Metrics Via SSH

To add metrics:

1. Click Add ⊞ on the toolbar.

   The Add Metric dialog appears.

2. Select the target metric and click OK.
3. To configure metrics from the other categories, repeat the procedure.

For full descriptions of all the metrics, in all categories, and details on how to configure them for including in your reports, see Generating Metrics (see page 243).

Setting Email Alerts

Follow these steps:

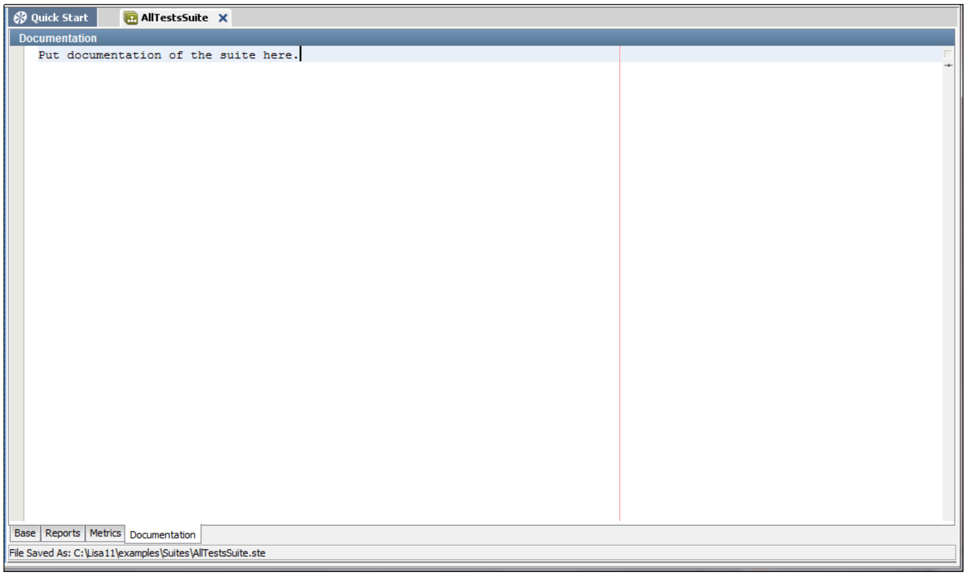
1. Click the Set Alert button corresponding to the metric.

   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.



Put documentation of the suite here.

File Saved As: C:\Lisa11\examples\Suites\AllTestsSuite.ste

Chapter 12: 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.

This section contains the following topics:

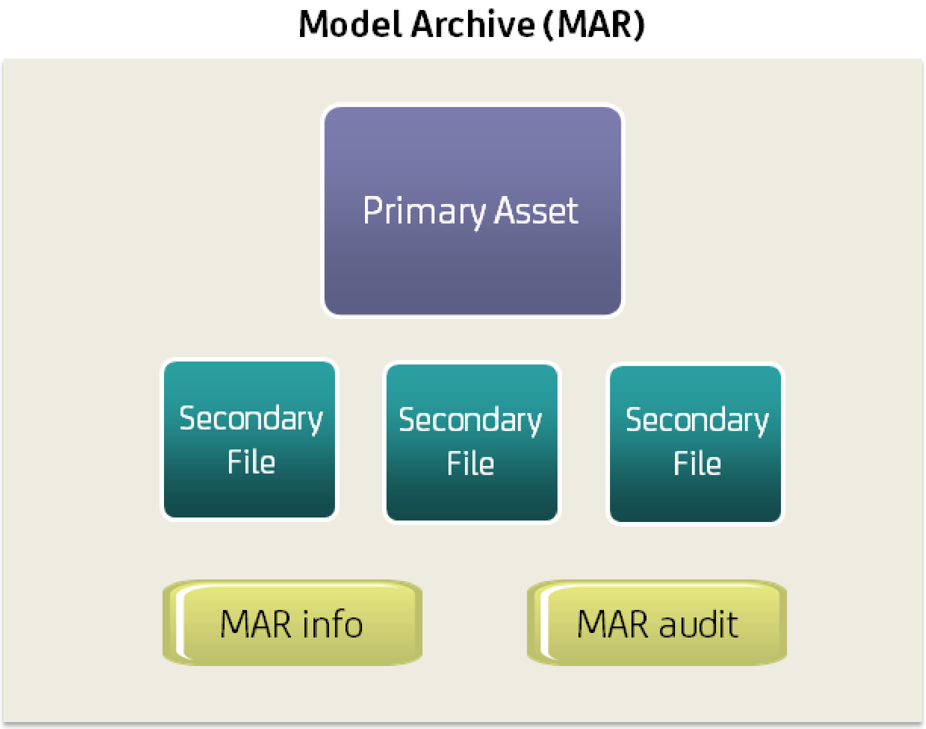
Model Archive (MAR) Overview (see page 256)
Explicit and Implicit MAR Creation (see page 258)
Create MAR Info Files (see page 260)
Create Monitor MAR Info Files (see page 262)
Edit MAR Info Files (see page 265)
Build MARs (see page 266)
Deploy to CVS (see page 266)
Make Mar Utility (see page 267)

Model Archive (MAR) Overview

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



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 257)
- MAR audit file (see page 258)

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 258)
- Implicit (see page 259)

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 310) 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 310):
   - 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 526), 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 353).

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 (see page 262).
4. Specify the time schedule information about the Schedule tab (see page 263).
5. (Test Case) Specify the test case information about the Test Case Info tab (see page 264).
6. (Suite) Specify the suite information about the Suite Info tab (see page 264).
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 fields:

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 256)).

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 262) 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
```

Chapter 13: 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 270) 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 197) in the case.
- In DevTest Workstation, you can run a test case quickly (see page 281) with minimal setup.
- In DevTest Workstation, you can stage a test case (see page 289). 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 310) is a command-line utility that lets you run tests as a batch process.
- LISA Invoke (see page 315) 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 350).
- You can schedule tests to run at regular time intervals by using the Continuous Validation Service (CVS) (see page 353).
- You can run tests as part of an automatic build process by using Ant and JUnit.

This section contains the following topics:

Using the Interactive Test Run (ITR) Utility (see page 270)
Stage a Quick Test (see page 281)
Stage a Test Case (see page 289)
Run a Test Suite (see page 297)
Run a Selenium Integration Test Case (see page 304)
Assumption of Load Testing (see page 308)
Test Runner (see page 310)
LISA Invoke (see page 315)
The REST API (see page 321)
Using the HP ALM - Quality Center Plug-in (see page 322)
HTTP and SSL Debug Viewer (see page 329)

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.

This section contains the following topics:

- Start an ITR Run (see page 270)
- Examine the Results of an ITR Run (see page 274)
- Graphical Text Diff Utility (see page 278)

Start an ITR Run

The Interactive Test Run (ITR) utility is run from an open test case in DevTest Workstation.

Follow these steps:

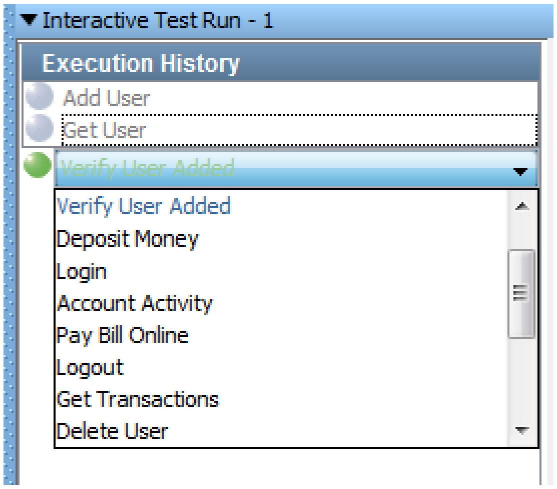
1. Do one:
   - Select Actions, Start Interactive Test Run from the main menu.
   - Click the ITR icon ⚙ on the toolbar. The icon gives you the option of starting a new ITR run or opening a previous ITR run.

   The Interactive Test Run window opens.
2. (Optional) To change any settings, use the Settings tab (see page 273).
3. Use the Run tab (see page 271) to execute all or part of the test case.

ITR Run Tab

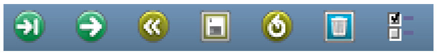
The Run tab on the ITR window shows the steps that have been executed (in gray) and the next step that will be executed (in green).

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 274) 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.

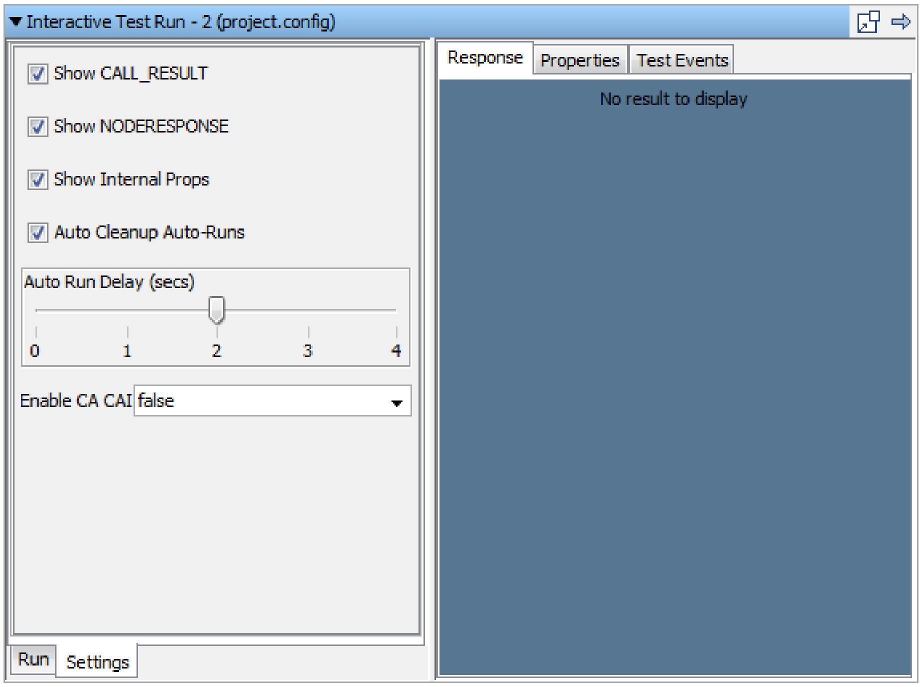
Save the current ITR state. In the dialog, enter the name of the save file. The suffix .itr is appended to the name. Subsequent saves overwrite this file.

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.



Interactive Test Run - 2 (project.config)

☑ Show CALL_RESULT

☑ Show NODERESPONSE

☑ Show Internal Props

☑ Auto Cleanup Auto-Runs

Auto Run Delay (secs): 2

Enable CA CAI: false

No result to display

You can filter out events and properties from the results tabs. You might not want to see some verbose events. Likewise, you do not always want to clutter the property list with DevTest internal properties.

Show CALL_RESULT

Include EVENT_CALL_RESULT events in the Test Events list.

Show NODERESPONSE

Include EVENT_NODERESPONSE events in the Test Events list.

Show Internal Props

Include all internal events in the property lists in the Initial State and Post Exec State tabs.

Auto Cleanup Auto-Runs

Clean up the Auto Runs automatically.

Auto Run Delay (secs)

In the Automatically Execute Test mode, the ITR pauses between each step execution. This setting lets you change the pause interval. The ITR does not honor the think time, so this setting lets you add a constant delay between each step execution.

Enable CA CAI

Controls whether 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.

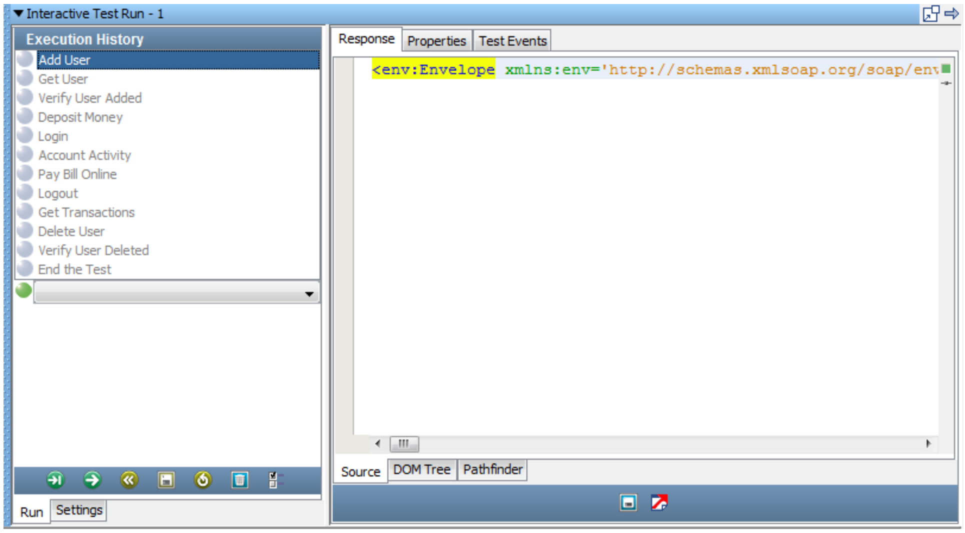
Select the step in the Execution History list. Examine the information in the right panel, which contains the following tabs:

- Response Tab (see page 275)
- Properties Tab (see page 276)
- Test Events Tab (see page 277)

ITR Results Response Tab

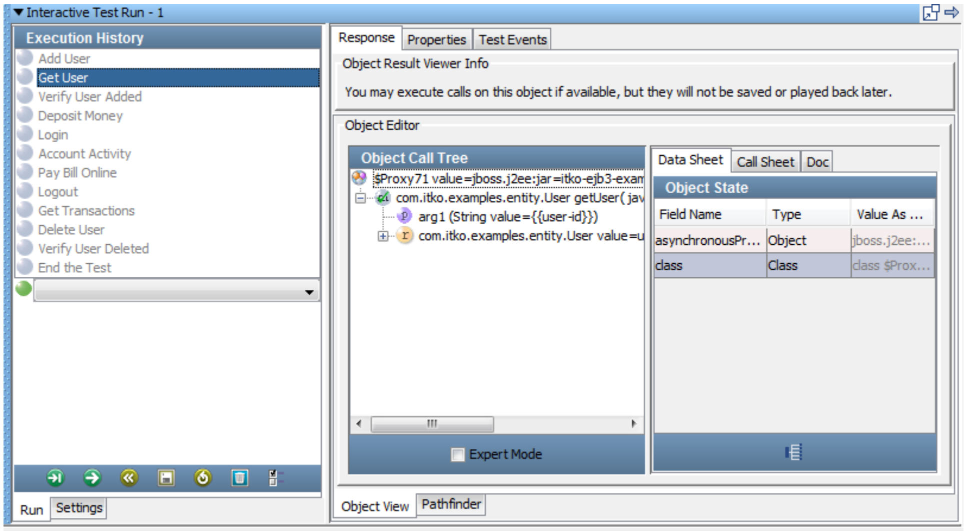
The Response tab displays the step response after executing a step.

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.



```
<env:Envelope xmlns:env='http://schemas.xmlsoap.org/soap/env
```

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 (see page 158).



You may execute calls on this object if available, but they will not be saved or played back later.

```
$Proxy71 value=jboss.j2ee:jar=itko-ejb3-exam
com.itko.examples.entity.User getUser( jav
  arg1 (String value={{user-id}})
  com.itko.examples.entity.User value=u
```

Field Name	Type	Value As ...
asynchronousPr...	Object	jboss.j2ee:...
class	Class	class $Prox...

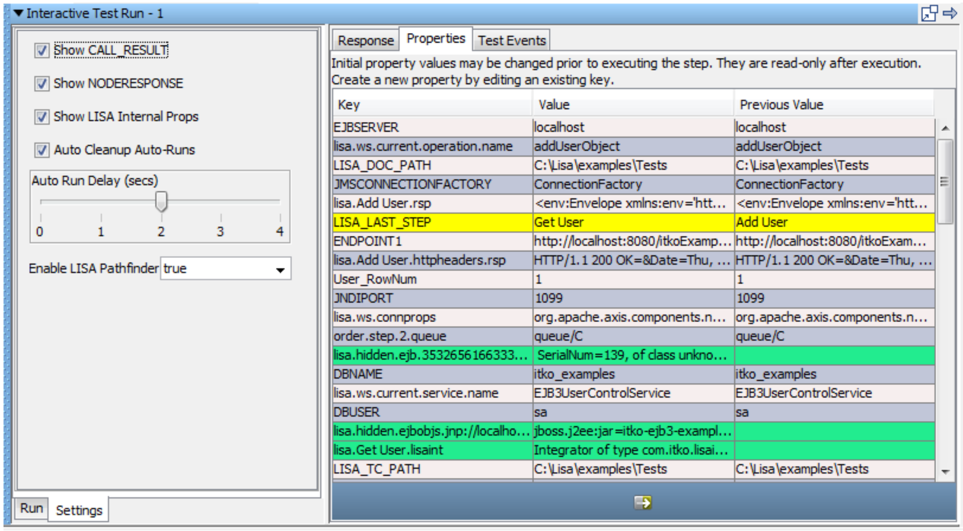
Some editors include two icons that you can use to save the ITR state and start an external browser.

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.

ITR Results 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 LISA Internal Props check box in the Settings tab.



☑ Show CALL_RESULT
☑ Show NODERESPONSE
☑ Show LISA Internal Props
☑ Auto Cleanup Auto-Runs

Auto Run Delay (secs)
0 1 2 3 4

Enable LISA Pathfinder true

Initial property values may be changed prior to executing the step. They are read-only after execution.
Create a new property by editing an existing key.

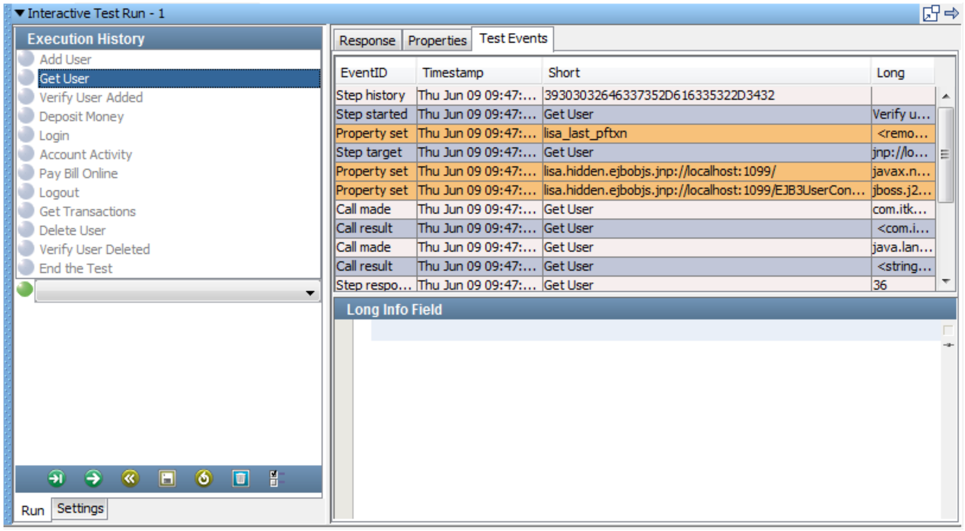
Key	Value	Previous Value
EJBSERVER	localhost	localhost
lisa.ws.current.operation.name	addUserObject	addUserObject
LISA_DOC_PATH	C:\Lisa\examples\Tests	C:\Lisa\examples\Tests
JMSCONNECTIONFACTORY	ConnectionFactory	ConnectionFactory
lisa.Add User.rsp	<env:Envelope xmlns:env='htt…	<env:Envelope xmlns:env='htt…
LISA_LAST_STEP	Get User	Add User
ENDPOINT1	http://localhost:8080/itkoExamp…	http://localhost:8080/itkoExam…
lisa.Add User.httpheaders.rsp	HTTP/1.1 200 OK=&Date=Thu, …	HTTP/1.1 200 OK=&Date=Thu, …
User_RowNum	1	1
JNDIPORT	1099	1099
lisa.ws.connprops	org.apache.axis.components.n…	org.apache.axis.components.n…
order.step.2.queue	queue/C	queue/C
lisa.hidden.ejb.3532656166333…	SerialNum=139, of class unkno…	
DBNAME	itko_examples	itko_examples
lisa.ws.current.service.name	EJB3UserControlService	EJB3UserControlService
DBUSER	sa	sa
lisa.hidden.ejbobjs.jnp://localho…	jboss.j2ee:jar=itko-ejb3-exampl…	
lisa.Get User.lisaint	Integrator of type com.itko.lisai…	
LISA_TC_PATH	C:\Lisa\examples\Tests	C:\Lisa\examples\Tests

Properties that the step set are highlighted in green. Properties that were modified in the step are highlighted in yellow.

ITR Results Test Events Tab

The Test Events tab displays the events that were fired when the step was executed, in the order that the events were fired.

To show or hide the events, select or clear the Show CALL_RESULT and Show NODERESPONSE check boxes in the Settings tab.



EventID	Timestamp	Short	Long
Step history	Thu Jun 09 09:47:…	39303032646337352D616335322D3432	
Step started	Thu Jun 09 09:47:…	Get User	Verify u…
Property set	Thu Jun 09 09:47:…	lisa_last_pftxn	<remo…
Step target	Thu Jun 09 09:47:…	Get User	jnp://lo…
Property set	Thu Jun 09 09:47:…	lisa.hidden.ejbobjs.jnp://localhost:1099/	javax.n…
Property set	Thu Jun 09 09:47:…	lisa.hidden.ejbobjs.jnp://localhost:1099/EJB3UserCon…	jboss.j2…
Call made	Thu Jun 09 09:47:…	Get User	com.itk…
Call result	Thu Jun 09 09:47:…	Get User	<com.i…
Call made	Thu Jun 09 09:47:…	Get User	java.lan…
Call result	Thu Jun 09 09:47:…	Get User	<string…
Step respo…	Thu Jun 09 09:47:…	Get User	36

The Test Events tab contains the following columns:

EventID

The name of the event.

Timestamp

The date and time of the event.

Short

The short description of the event.

Long

The long description of the event. The long description can be truncated in the display. To view the full text, click the cell and its full contents are displayed in the Long Info Field panel.

You can enter the complete or full description of the event in the Long Info Field.

An event is generated on all assertions that are fired or evaluated.

If a warning was generated, the row is yellow.

If an assertion was fired, the row is purple.

If a property was set, the row is orange.

If an error was generated, the test was failed, or the test was aborted, the row is red.

Note: If the step calls a subprocess that contains an assertion, then the Test Events tab includes the appropriate event for the assertion. For example, the Test Events tab can include an Assertion fired event that occurred in the subprocess.

Graphical Text Diff Utility

A graphical XML diff engine and visualizer are available to compare XML files and graphically display their differences.

Start the Graphical Text Diff Utility

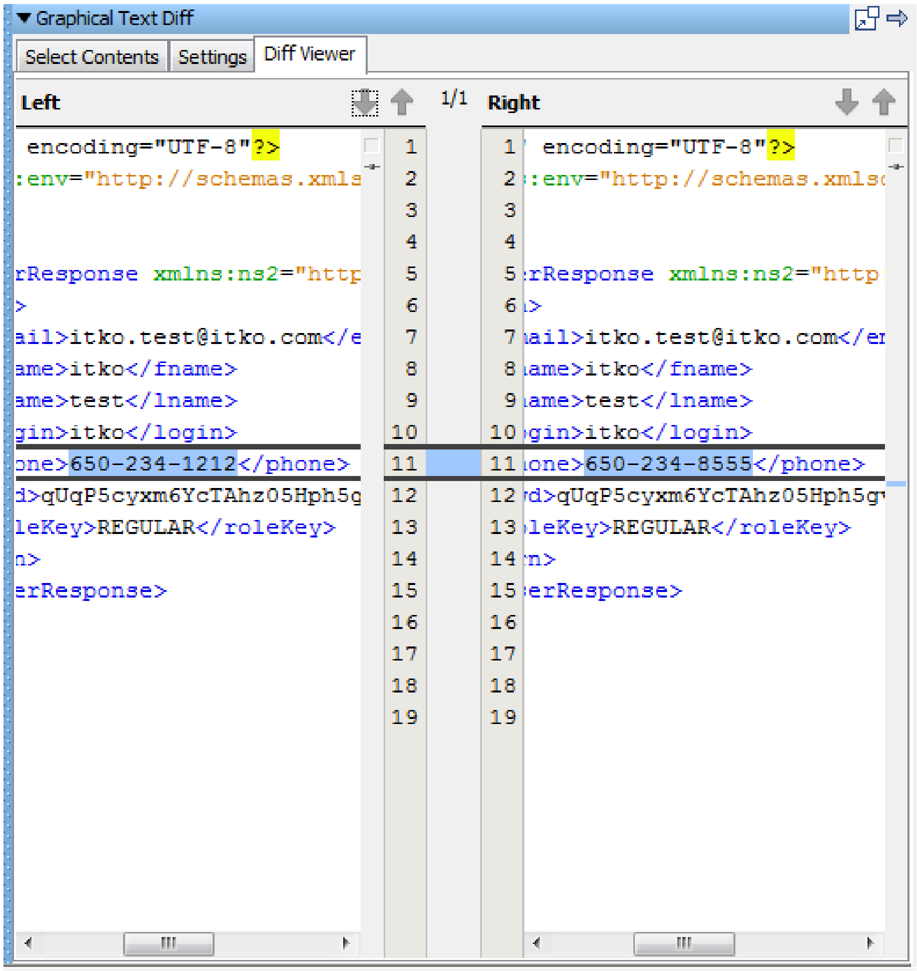
Follow these steps:

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.



```
 encoding="UTF-8"?>
:env="http://schemas.xmls


rResponse xmlns:ns2="http
>
ail>itko.test@itko.com</e
ame>itko</fname>
ame>test</lname>
gin>itko</login>
one>650-234-1212</phone>
d>qUqP5cyxm6YcTAhz05Hph5g
leKey>REGULAR</roleKey>
n>
erResponse>
```

```
 encoding="UTF-8"?>
::env="http://schemas.xmlsc


:rResponse xmlns:ns2="http
ı>
ıail>itko.test@itko.com</er
ıame>itko</fname>
ıame>test</lname>
ıgin>itko</login>
ıone>650-234-8555</phone>
rd>qUqP5cyxm6YcTAhz05Hph5gv
ıleKey>REGULAR</roleKey>
:n>
:erResponse>
```

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 in *Using CA Application Test.*

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 215) as in the case of a proper test case (see page 289). 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 282) window opens.

Test Monitor Window - Quick Test

When you stage a quick test (see page 281), 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 283) Tab displays collected metrics as a function of time. After the test starts, this display cannot be changed.
- The Events Tab (see page 285) displays events as they are recorded during the test run.

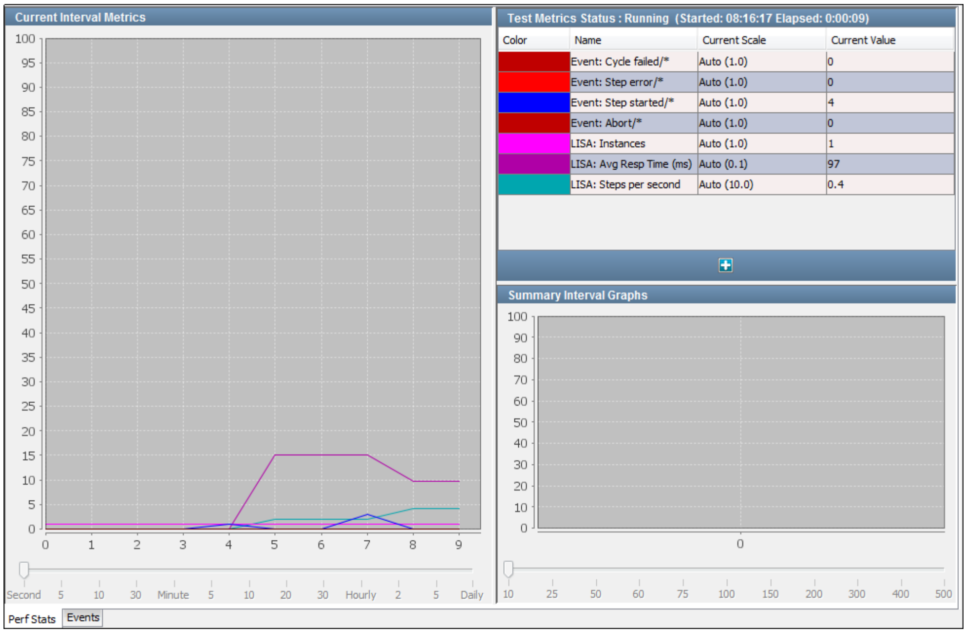
If you run the test case and there are failures in the test, then a third tab that is labeled Failures (see page 287) is displayed.

If you run a baseline test case, then the Consolidated Baseline tab is also displayed. For more information, see *Using CA Continuous Application Insight.*

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.



Test Metrics Status : Running (Started: 08:16:17 Elapsed: 0:00:09)

Color	Name	Current Scale	Current Value
🟥	Event: Cycle failed/*	Auto (1.0)	0
🟥	Event: Step error/*	Auto (1.0)	0
🟦	Event: Step started/*	Auto (1.0)	4
🟥	Event: Abort/*	Auto (1.0)	0
🟪	LISA: Instances	Auto (1.0)	1
🟪	LISA: Avg Resp Time (ms)	Auto (0.1)	97
🟦	LISA: Steps per second	Auto (10.0)	0.4

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 243) in *Using CA Application Test.*

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.

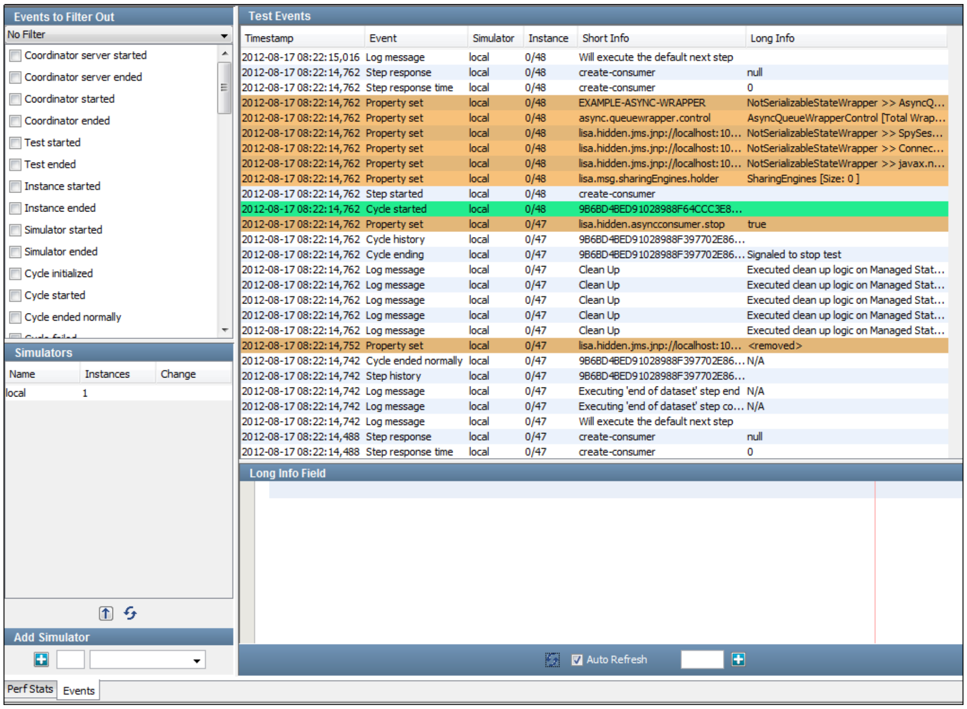
To specify the number of samples per interval, use the slider at the bottom of the panel. You cannot adjust the value after the run has started.

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



No Filter

- □ Coordinator server started
- □ Coordinator server ended
- □ Coordinator started
- □ Coordinator ended
- □ Test started
- □ Test ended
- □ Instance started
- □ Instance ended
- □ Simulator started
- □ Simulator ended
- □ Cycle initialized
- □ Cycle started
- □ Cycle ended normally

Name	Instances	Change
local	1	

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 >> SpySes...
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	Property set	local	0/48	lisa.msg.sharingEngines.holder	SharingEngines [Size: 0]
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	9B6BD4BED91028988F64CCC3E8...	
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	9B6BD4BED91028988F397702E86...	
2012-08-17 08:22:14,762	Cycle ending	local	0/47	9B6BD4BED91028988F397702E86...	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 08:22:14,752	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	9B6BD4BED91028988F397702E86...	N/A
2012-08-17 08:22:14,742	Step history	local	0/47	9B6BD4BED91028988F397702E86...	
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

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

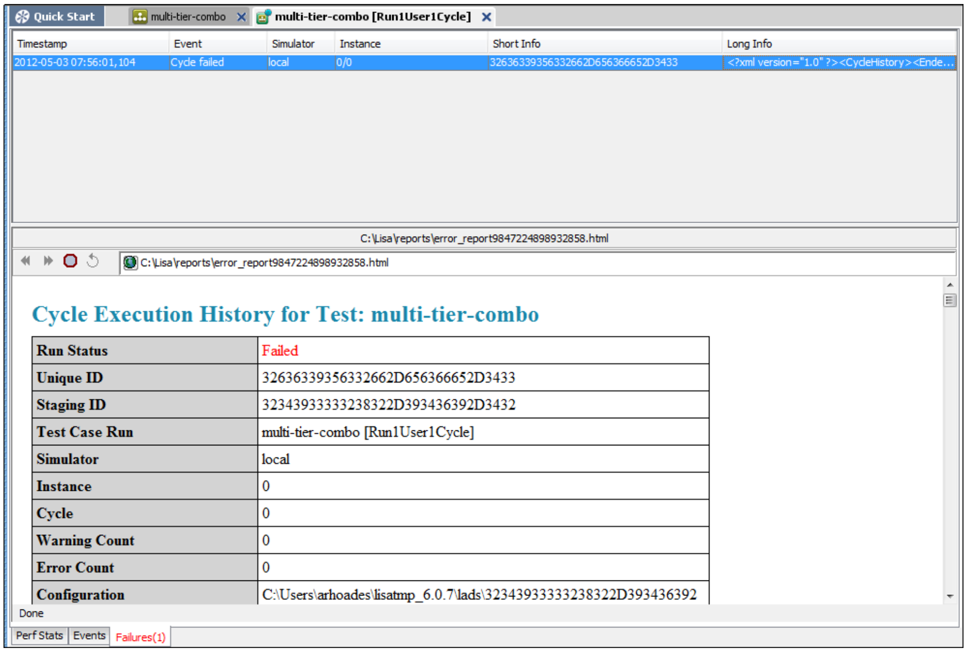
A short piece of information about the event

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.



Timestamp	Event	Simulator	Instance	Short Info	Long Info
2012-05-03 07:56:01,104	Cycle failed	local	0/0	32636339356332662D656366652D3433	<?xml version="1.0" ?><CycleHistory><Ende...

C:\Lisa\reports\error_report9847224898932858.html

C:\Lisa\reports\error_report9847224898932858.html

Cycle Execution History for Test: multi-tier-combo

Run Status	Failed
Unique ID	32636339356332662D656366652D3433
Staging ID	32343933333238322D393436392D3432
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\32343933333238322D393436392

Done

To see a report on any failed test, double-click the Long Info field at the right of the window.

Starting and Stopping Quick Tests

With the Test Monitor window open, you can start and stop a quick test by clicking icons on the main toolbar.

To start a quick test, click Play on the main toolbar.

When the quick test starts, the Play icon changes to a Stop icon.

To stop a quick test, click Stop on the main toolbar.

If you click Stop again, you are asked if you want to kill the test run.

- When you stop a test, you are saying "do not start any new instances." No new instances are started and a running test case does not loop back to the beginning. Tests that are running complete all steps, then go to the end step.
- When you kill a test, you are saying "stop all running instances as soon as possible." No new instances will be started, and a running test case does not go to the next step. No end step is run. Reports show that the test is still running, because the test never goes through an end step.

To replay a quick test, click Replay on the main toolbar.

To close a quick test, click Close on the main toolbar.

Actions Menu

When you run a test and click Play to start the test, the following options are added to the Actions menu.

- Pause Metrics Collection
- Un-pause Metrics Collection
- Save Recent Metrics to XML Document
- Save Recent Metrics to CSV Document
- Save Interval Data to CSV Document
- Save Interval Data to XML Document
- Save Recent Events to CSV Document
- Stage This Test
- Stop Test

- Restart Test (Reloads Documents)

Stage a Test Case

You can run a test case from DevTest Workstation by specifying the configuration, staging document, and coordinator server.

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 100) from the Configuration drop-down list.

   If you leave this field blank, the default configuration is used.
4. Select the staging document (see page 216) from the Staging doc drop-down list.
5. Select the coordinator server (see page 13) or (if available) a cloud-based lab (see page 333) 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.
6. Click Stage.
   - If you selected a coordinator server, the Test Monitor (see page 290) 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 290) 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 289), 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 291) Tab lets you select metrics, and display them as a function of time.
- The Events (see page 293) Tab lets you select and display events as they occur during the test run.

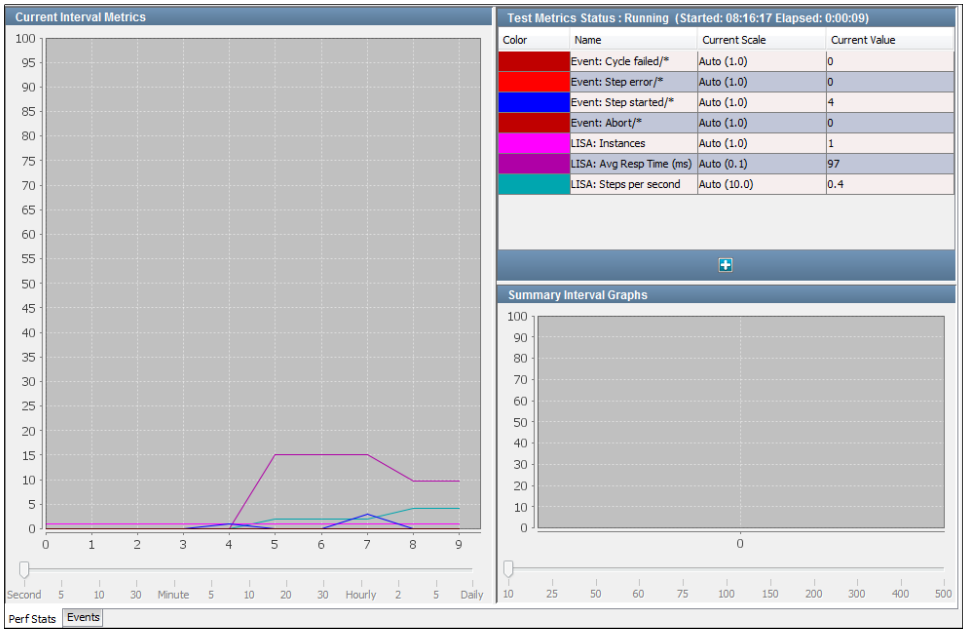
If you run the test case and there are failures in the test, then a third tab that is labeled Failures (see page 295) is also displayed.

If you run a baseline test case, then the Consolidated Baseline tab is also displayed. For more information, see *Using CA Continuous Application Insight.*

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.



Test Metrics Status : Running (Started: 08:16:17 Elapsed: 0:00:09)

Color	Name	Current Scale	Current Value
🟥	Event: Cycle failed/*	Auto (1.0)	0
🟥	Event: Step error/*	Auto (1.0)	0
🟦	Event: Step started/*	Auto (1.0)	4
🟥	Event: Abort/*	Auto (1.0)	0
🟪	LISA: Instances	Auto (1.0)	1
🟪	LISA: Avg Resp Time (ms)	Auto (0.1)	97
🟦	LISA: Steps per second	Auto (10.0)	0.4

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 243) in *Using CA Application Test.*

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.

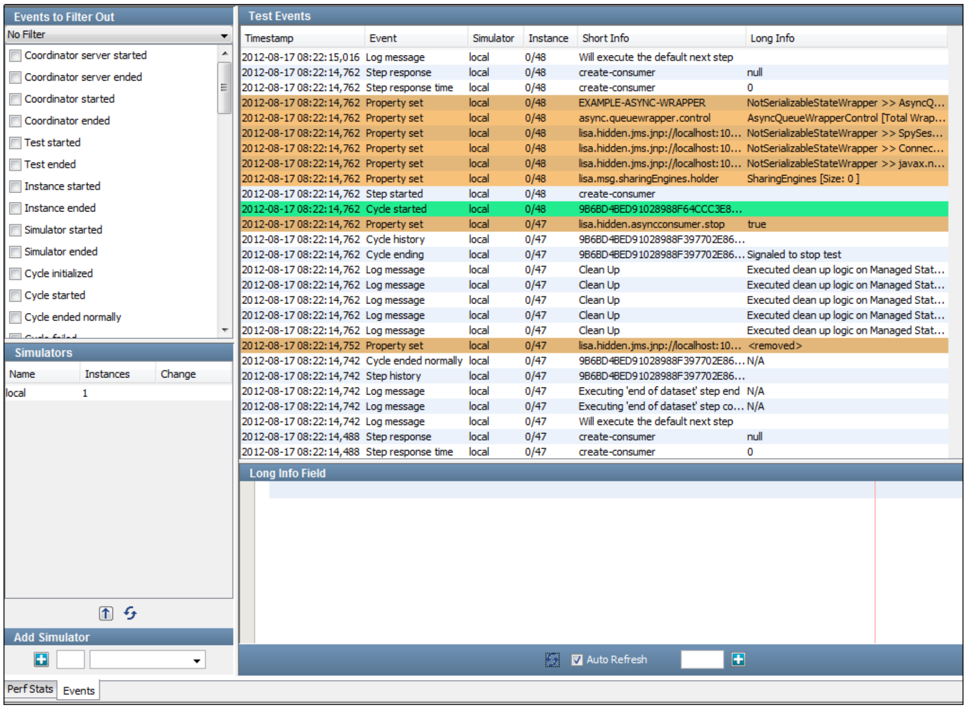
Summary Interval Graphs

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.



Events to Filter Out

No Filter

- □ Coordinator server started
- □ Coordinator server ended
- □ Coordinator started
- □ Coordinator ended
- □ Test started
- □ Test ended
- □ Instance started
- □ Instance ended
- □ Simulator started
- □ Simulator ended
- □ Cycle initialized
- □ Cycle started
- □ Cycle ended normally
- □ Cycle failed

Simulators

Name	Instances	Change
local	1	

Test Events

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 >> SpySes...
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	Property set	local	0/48	lisa.msg.sharingEngines.holder	SharingEngines [Size: 0]
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	9B6BD4BED91028988F64CCC3E8...	
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	9B6BD4BED91028988F397702E86...	
2012-08-17 08:22:14,762	Cycle ending	local	0/47	9B6BD4BED91028988F397702E86...	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 08:22:14,752	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	9B6BD4BED91028988F397702E86...	N/A
2012-08-17 08:22:14,742	Step history	local	0/47	9B6BD4BED91028988F397702E86...	
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

Long Info Field

Add Simulator

☑ Auto Refresh

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

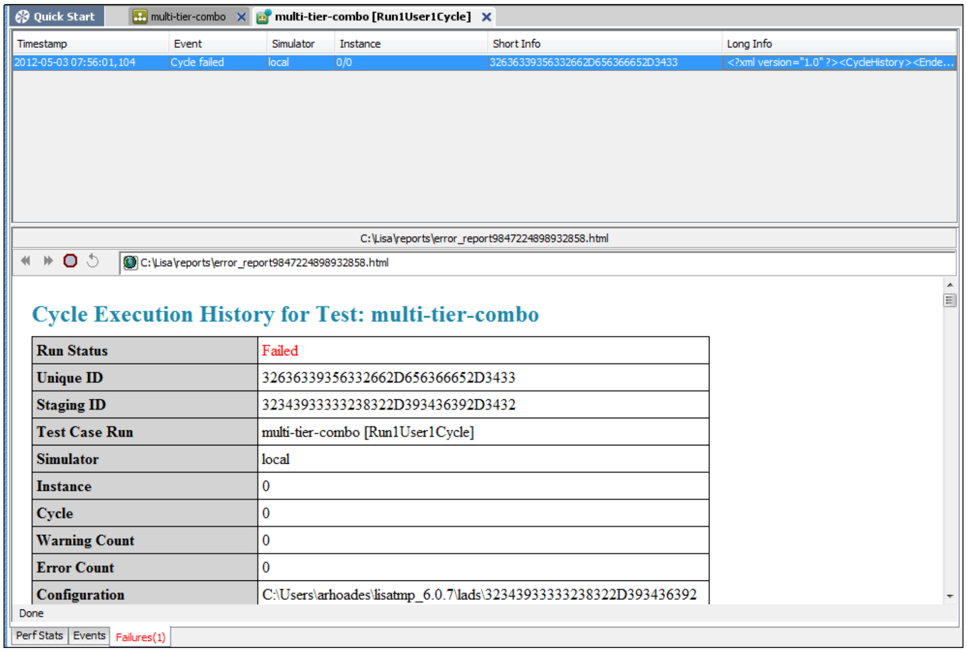
A short piece of information about the event

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, you see a Failures tab at the bottom of the window.



Timestamp	Event	Simulator	Instance	Short Info	Long Info
2012-05-03 07:56:01,104	Cycle failed	local	0/0	32636339356332662D656366652D3433	<?xml version="1.0" ?><CycleHistory><Ende...

C:\Lisa\reports\error_report9847224898932858.html

C:\Lisa\reports\error_report9847224898932858.html

Cycle Execution History for Test: multi-tier-combo

Run Status	Failed
Unique ID	32636339356332662D656366652D3433
Staging ID	32343933333238322D393436392D3432
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\32343933333238322D393436392

To see a report on any failed test, double-click the Long Info field at the right of the window.

Starting and Stopping Test Cases

With the Test Monitor window open, you can start and stop a test case by clicking icons on the main toolbar.

To start a test case, click Play on the main toolbar.

When the test case starts, the Play icon changes to a Stop icon.

To stop a test case, click Stop on the main toolbar.

If you click Stop again, you are asked if you want to kill the test run.

- When you stop a test, you are saying "do not start any new instances." No new instances are started and a running test case does not loop back to the beginning. Tests that are running complete all steps, then go to the end step.
- When you kill a test, you are saying "stop all running instances as soon as possible." No new instances will be started, and a running test case does not go to the next step. No end step is run. Reports show that the test is still running, because the test never goes through an end step.

To replay a test case, click Replay on the main toolbar.

To close a test case, click Close on the main toolbar.

Run a Test Suite

A test suite lets you group related test cases and test suites and run them as a single test. A test suite document specifies the contents of the suite, the reports to generate, and the metrics to collect. These reports and metrics relate to the suite as a whole. Each test in the suite still produces its own reports and metrics. Each test still retains the ability to use its own staging document, configuration, and audit document. Therefore, tests in a suite can be run in a distributed environment.

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.

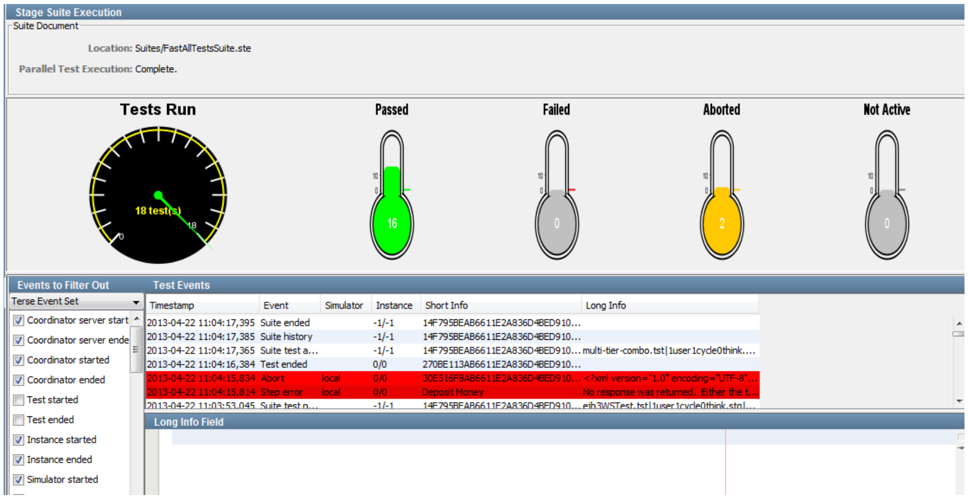
   This action opens a menu where you select whether to run locally or to run with a specified registry.

   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 298) window opens and the tests start.

Stage Suite Execution

When you click Stage during the process of running a test suite (see page 297) in DevTest Workstation, the Stage Suite Execution tab opens and the tests start.



Stage Suite Execution

Suite Document

Location: Suites/FastAllTestsSuite.ste

Parallel Test Execution: Complete.

Terse Event Set

- ☑ Coordinator server start
- ☑ Coordinator server ende
- ☑ Coordinator started
- ☑ Coordinator ended
- ☐ Test started
- ☐ Test ended
- ☑ Instance started
- ☑ Instance ended
- ☑ Simulator started

Timestamp	Event	Simulator	Instance	Short Info	Long Info
2013-04-22 11:04:17,395	Suite ended		-1/-1	14F795BEAB6611E2A836D4BED910...	
2013-04-22 11:04:17,385	Suite history		-1/-1	14F795BEAB6611E2A836D4BED910...	
2013-04-22 11:04:17,365	Suite test a...		-1/-1	14F795BEAB6611E2A836D4BED910...	multi-tier-combo.tst|1user1cycle0think....
2013-04-22 11:04:16,384	Test ended		0/0	270BE113AB6611E2A836D4BED910...	
2013-04-22 11:04:15,834	Abort	local	0/0	30E516F8AB6611E2A836D4BED910...	<?xml version="1.0" encoding="UTF-8"...
2013-04-22 11:04:15,814	Step error	local	0/0	Deposit Money	No response was returned. Either the t...
2013-04-22 11:03:53,045	Suite test n...		-1/-1	14F795BEAB6611E2A836D4BED910...	eih3WSTest.tst|1user1cycle0think.stgl...

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 299): Lists requested events as they occur
- Results Tab (see page 301): 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

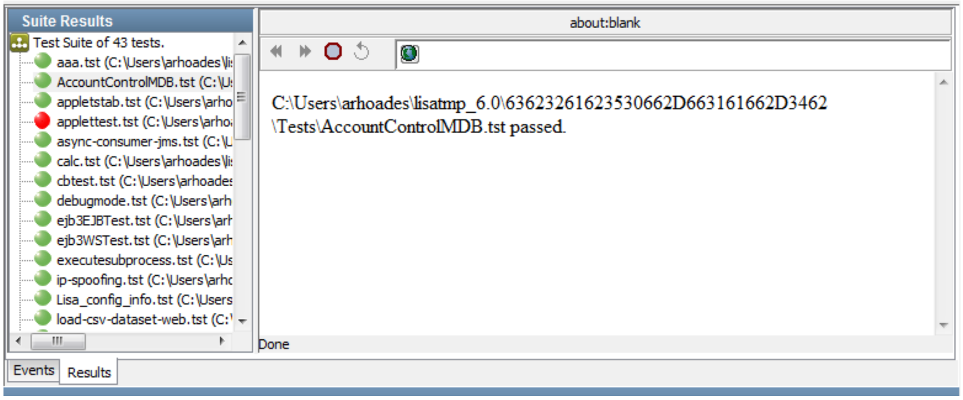
A short piece of information about the event

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.



Test Suite of 43 tests.
aaa.tst (C:\Users\arhoades\li
AccountControlMDB.tst (C:\U
appletstab.tst (C:\Users\arho
appletest.tst (C:\Users\arho
async-consumer-jms.tst (C:\U
calc.tst (C:\Users\arhoades\li
cbtest.tst (C:\Users\arhoades
debugmode.tst (C:\Users\arh
ejb3EJBTest.tst (C:\Users\arh
ejb3WSTest.tst (C:\Users\arh
executesubprocess.tst (C:\Us
ip-spoofing.tst (C:\Users\arhc
Lisa_config_info.tst (C:\Users
load-csv-dataset-web.tst (C:\

C:\Users\arhoades\lisatmp_6.0\63623261623530662D663161662D3462
\Tests\AccountControlMDB.tst passed.

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.

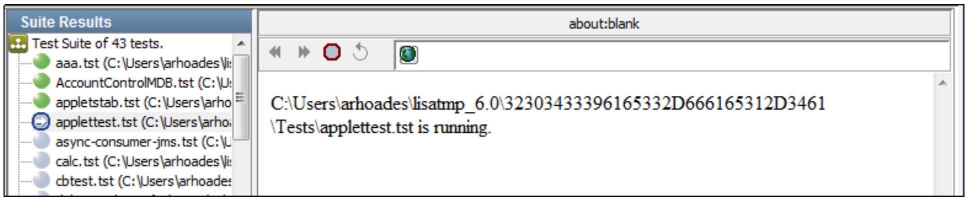
To display the test monitor for tests that are still running, click View Test at the bottom of the panel.

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.

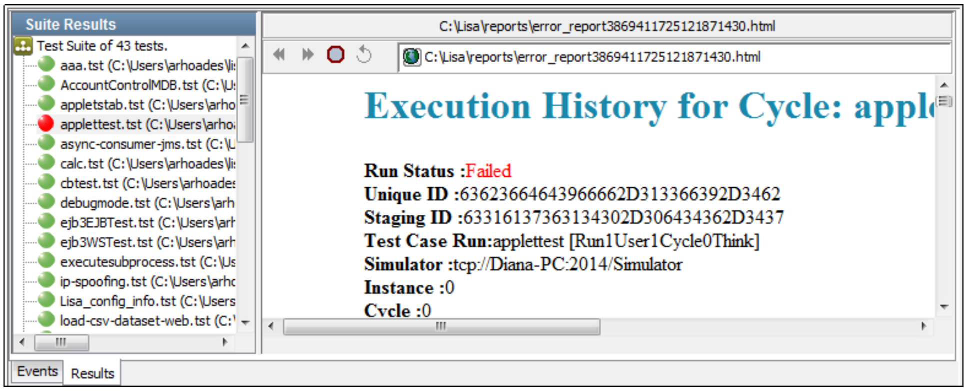
The following test is running:



Test Suite of 43 tests.
aaa.tst (C:\Users\arhoades\li
AccountControlMDB.tst (C:\U
appletstab.tst (C:\Users\arho
appletest.tst (C:\Users\arho
async-consumer-jms.tst (C:\U
calc.tst (C:\Users\arhoades\li
cbtest.tst (C:\Users\arhoades

C:\Users\arhoades\lisatmp_6.0\32303433396165332D666165312D3461
\Tests\applettest.tst is running.

The following test has failed:


Suite Results
Test Suite of 43 tests.
aaa.tst (C:\Users\arhoades\li:
AccountControlMDB.tst (C:\U:
appletstab.tst (C:\Users\arho
applettest.tst (C:\Users\arho:
async-consumer-jms.tst (C:\L
calc.tst (C:\Users\arhoades\li:
cbtest.tst (C:\Users\arhoades
debugmode.tst (C:\Users\arh
ejb3EJBTest.tst (C:\Users\arh
ejb3WSTest.tst (C:\Users\arh
executesubprocess.tst (C:\Us
ip-spoofing.tst (C:\Users\arhc
Lisa_config_info.tst (C:\Users
load-csv-dataset-web.tst (C:\
C:\Lisa\reports\error_report3869411725121871430.html
C:\Lisa\reports\error_report3869411725121871430.html
Events
Results

Execution History for Cycle: appl

Run Status :Failed
Unique ID :6362366464396662D313366392D3462
Staging ID :63316137363134302D306434362D3437
Test Case Run:applettest [Run1User1Cycle0Think]
Simulator :tcp://Diana-PC:2014/Simulator
Instance :0
Cycle :0

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.
🚫	Kill Test	Stops the test immediately after the current step completes.
🔍	View Test	This tool will work only for currently running tests. This starts the test monitor for the selected test.

The second set of icons manages the test suite itself, or when a suite is running:

▶	Run Suite	Starts/restarts the suite test.
🛑	Close Suite	Close the Stage Suite Execution window.
🔍	View Test	Starts the test monitor for the selected test.

For more information about Test Monitor, see Use the Registry Monitor.

Use the Load Test Optimizer

The Load Test Optimizer appears in the Registry Monitor and lets you run a simple load test on the system under test. A load test determines how many users the system under test supports.

In the Load Test Optimizer, the system under test runs continuously. More simulated users continue to access it, until a specific target is reached. This target is typically a predefined average response time.

The Load Test Optimizer helps determine how many users the system under test supports. An optimizer can be set on any DevTest metric, such as average response time. An optimizer can also be set on a metric pulled from an external source, such as SNMP, JMX, or Windows Perfmon. The number of users increases at a preset frequency and informs you when a preset metric threshold is achieved. For example, you can increase the number of test instances by 5 instances every 10 seconds until the average response time hits 2 seconds.

To optimize the test, it must be running.

To configure and start an optimizer:

1. In the Registry Monitor, select a test from the running tests list and click Optimize Test .

   The Optimizer panel opens with the name of the staging document at the top.

2. Configure the following parameters:

   Metric

   The metric to use for the optimization. Select from the pull-down list.

   Simulator

   The simulator that is used to spawn test instances. Select from the pull-down list.

   Threshold Low

   The metric value at which the optimizer reports that the system requires more virtual users. Enter a numeric value.

   Threshold High

   The metric value at which the optimizer reports that the system cannot support any more virtual users. Enter a numeric value.

   Increment (#instances)

   The number of extra virtual users to add to the system at the update frequency. Enter a numeric value.

   Update Frequency (millis)

The number of milliseconds between increments

3. Click Start Optimizer ▶.

   As the optimizer increases the number of virtual users, the number of virtual users display on both the Optimizers section and in the Instances column of the Tests tab in the Registry Monitor.

   As the optimizer executes, you can change the parameters. To update the optimizer with the changed information, click Update ↑.

4. To close the optimizer, click Close ⊗.

Run a Selenium Integration Test Case

Selenium Integration test steps let you import test scripts for web-based user interfaces from Selenium Builder to DevTest Solutions. 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, or Internet Explorer 8.0 or later. 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 additional prerequisite tasks. For general information about running a test case, see Running Test Cases and Suites (see page 269).

Run a Selenium Integration Test on Google Chrome (Local)

This topic 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.

Run a Selenium Integration Test on Microsoft Internet Explorer (Local)

This topic 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 a Remote Browser

This topic describes how to run a Selenium Integration test case on a remote browser. The remote browser can be Mozilla Firefox, Google Chrome, or Internet Explorer 8.0 or later.

Follow these steps:

1. Download the Selenium Server and save it to a local directory.
   a. Go to http://www.seleniumhq.org/download/.
   b. Locate the Selenium Server stand-alone .jar file in the **Selenium Server (formerly the Selenium RC Server)** section and download it.
2. Verify that the driver for the browser that you want to use is available on the remote computer.
3. On the remote computer, run the following command from a command prompt:
   ```
   java -jar selenium-server-standalone-2.xx.0.jar -role hub
   ```
4. On the remote computer, run the following command from a new command prompt:
   ```
   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
   ```
5. On the local computer, add the following properties to the project configuration file you use for running test cases on the remote browser.
   - **Key:** selenium.broswer.type

     Values: IE, Firefox, 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 100).

   Note: You must add these properties to project.config before you add them to other project configuration files.
6. Right-click the selected project configuration file in the Project panel and select Make Active.
7. 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]
```

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 in *Administering.*

This section contains the following topics:

- Run a MAR with Test Runner (see page 311)
- Run a Test Case with Test Runner (see page 311)
- Run a Suite with Test Runner (see page 312)
- Other Test Runner Options (see page 313)
- Multiple Test Runner Instances (see page 314)
- Test Runner Log File (see page 314)

Run a MAR with Test Runner

To run a Model Archive (MAR) (see page 256) 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 313).

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 313).

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 topic 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
```

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 Log File Overview in *Administering.*

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.invok
e.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=61653261643936342D
613636392D3435&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/StagingDoc
s/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=[callback
Key]&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.

The REST API

The REST 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. The API documentation provides details on how to interact with the various RESTful services.

Note: You must use Google Chrome or Mozilla Firefox to access the API documentation.

Invoke the REST API from the Web Interface

DevTest Solutions includes a simple web interface that you can use to explore the REST 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_add
ress>: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.

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*.

This section contains the following topics:

- Set up DevTest Tests in HP ALM - Quality Center (see page 323)
- Run DevTest Tests in HP ALM - Quality Center (see page 325)
- LisaQCRunner Command-Line Interface (see page 327)
- HP ALM - Quality Center Plug-in Troubleshooting (see page 328)

Set up DevTest Tests in HP ALM - Quality Center

The plug-in uses VAPI-XP to integrate with HP ALM - Quality Center.

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, a MAR file, or a MAR info file.

For test cases, you can also attach a staging document and a configuration file. If you do not attach a staging document, a staging document is automatically created with these characteristics:

- One user
- One cycle
- Zero think time

For MAR files and MAR info files, you cannot attach any additional files.

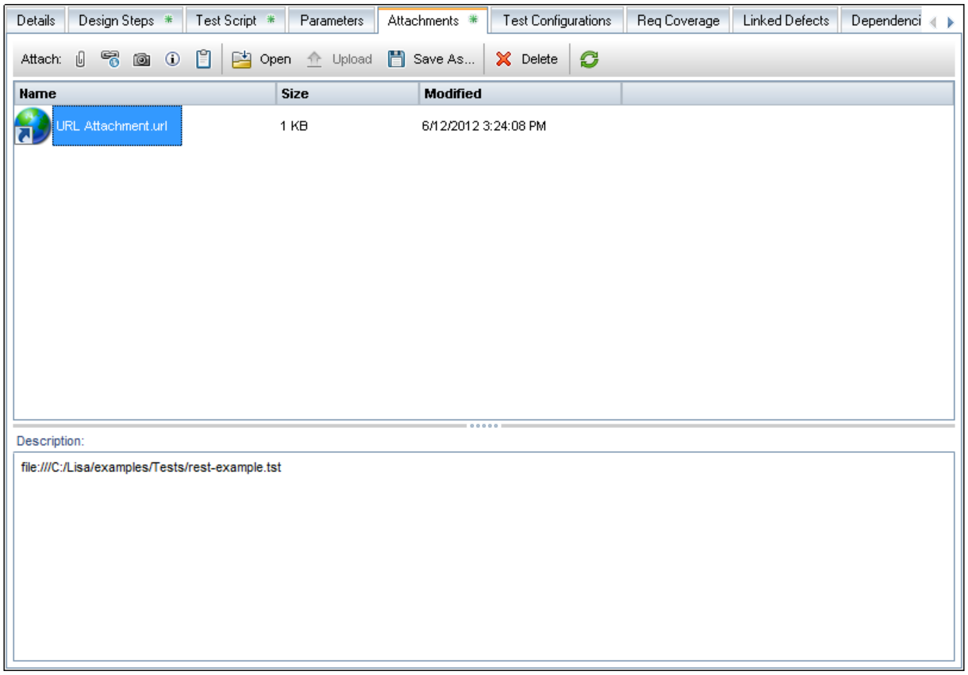
In this procedure, you create one or more URL attachments. Here is an example URL for a test case file:

```
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.



Name	Size	Modified
URL Attachment.url	1 KB	6/12/2012 3:24:08 PM

Description:

file:///C:/Lisa/examples/Tests/rest-example.tst

Follow these steps:

1. Create a VAPI-XP test in Quality Center.
2. Select the Test Script tab and replace the default contents with the contents of the JavaScript or VBScript template included with the plug-in.
3. Select the Attachments tab and add a URL to the test case file, MAR file, or MAR info file.
4. (Optional) Add URLs to the staging document and configuration file. These files are allowed with test cases, but not with MAR files or MAR info files.
5. Save the VAPI-XP test.

Run DevTest Tests in HP ALM - Quality Center

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.

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.

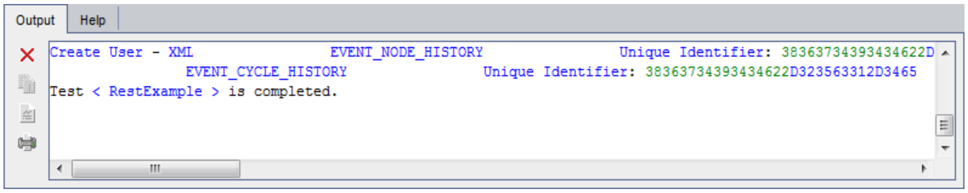
For MAR files and MAR info files:

- 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 module, click the green arrow in the Test Script tab. The output appears in the output window.

The following graphic shows the results from a run of the **rest-example** test case in the Test Plan.

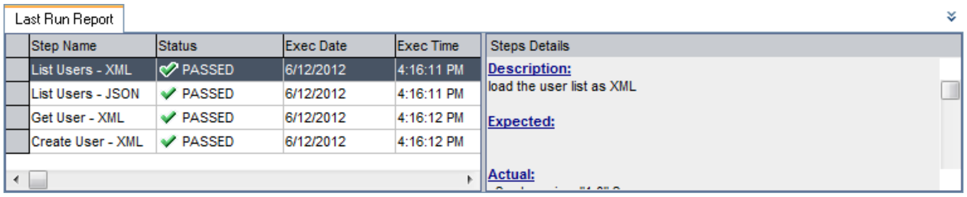


```
Create User - XML                    EVENT_NODE_HISTORY                    Unique Identifier: 38363734393434622D
                   EVENT_CYCLE_HISTORY                    Unique Identifier: 38363734393434622D323563312D3465
Test < RestExample > is completed.
```

To run the test from the Test Lab module, add the test to a test set. From there, you can run the single test or the entire test set.

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.

The following graphic shows the results from a run of the **rest-example** test case in the Test Lab.



Step Name	Status	Exec Date	Exec Time
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

Steps Details

Description:
load the user list as XML

Expected:

Actual:

LisaQCRunner Command-Line Interface

The **LisaQCRunner** 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:

```
LisaQCRunner [-h host] [-P port] [-u user] [-p password] [-D domain] [-l project]
run|debug|reload testname|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.

The project argument is the Quality Center project, not the DevTest project.

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).

The following example shows a run of the **rest-example** test case.

```
LisaQCRunner -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/1user1cycle0think.stg
Config:
Mar:
List Users - XML     EVENT_NODE_HISTORY     Unique Identifier:
63366561643365642D643834302D3461 ...
List Users - JSON     EVENT_NODE_HISTORY     Unique Identifier:
63366561643365642D643834302D3461 ...
Get User - XML     EVENT_NODE_HISTORY     Unique Identifier:
63366561643365642D643834302D3461 ...
Create User - XML     EVENT_NODE_HISTORY     Unique Identifier:
63366561643365642D643834302D3461 ...
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 **LisaQCRunner.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 **LisaQCRunner.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.

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 330) 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.

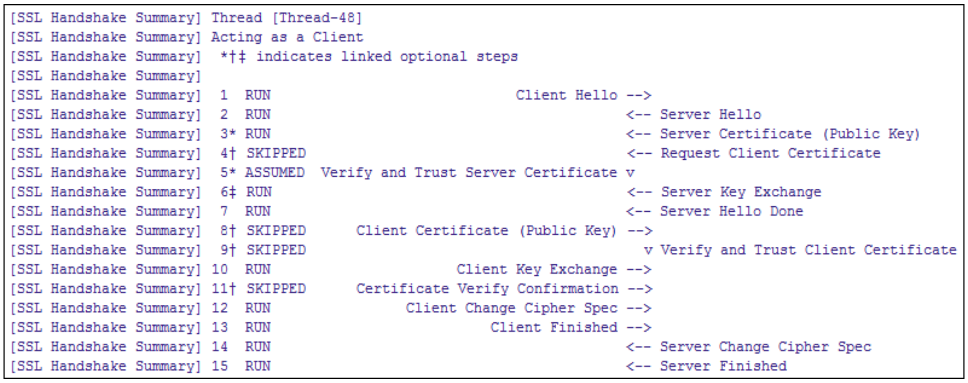
The viewer lets you specify whether to show all the lines, HTTP lines only, or SSL lines only.

SSL Handshake Summary

The SSL output in the HTTP and SSL Debug Viewer (see page 329) 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 topic 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
```

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
```

Chapter 14: Cloud DevTest Labs

You can use a cloud-based infrastructure to provision development and test environments.

This section contains the following topics:

Labs and Lab Members (see page 334)
Virtual Lab Manager (VLM) (see page 335)
DevTest Cloud Manager (see page 336)
Configure DCM Properties (see page 337)
Configure vCloud Director (see page 342)
Dynamic Expansion of Test Labs (see page 343)
List the Available Labs (see page 344)
Start a Lab (see page 346)
Obtain Information About a Lab (see page 349)
Deploy a MAR to a Lab (see page 350)
Stop a Lab (see page 351)

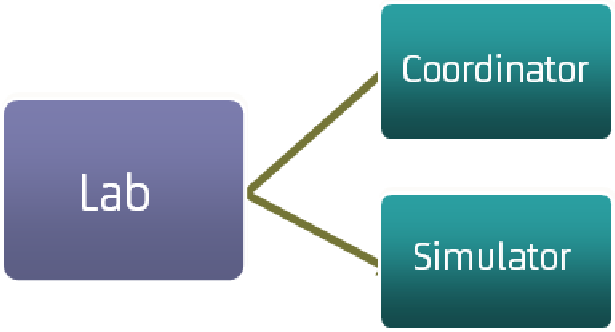
Labs and Lab Members

A *lab* is a logical container for one or more lab members.

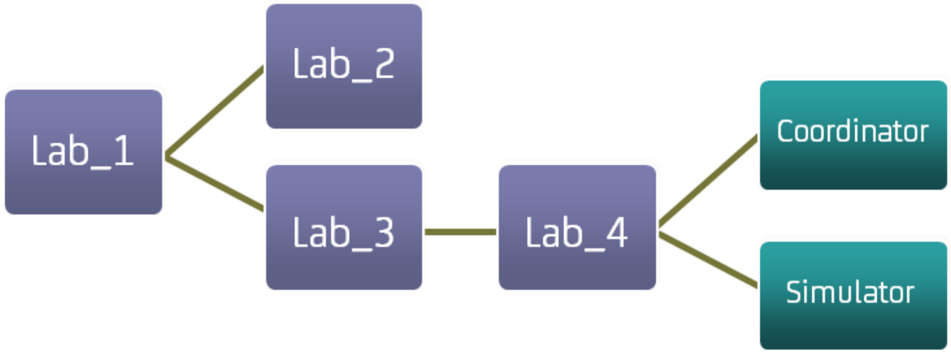
A *lab member* can be a DevTest server or a non-DevTest server.

- 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.



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.



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 (see page 334) 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.

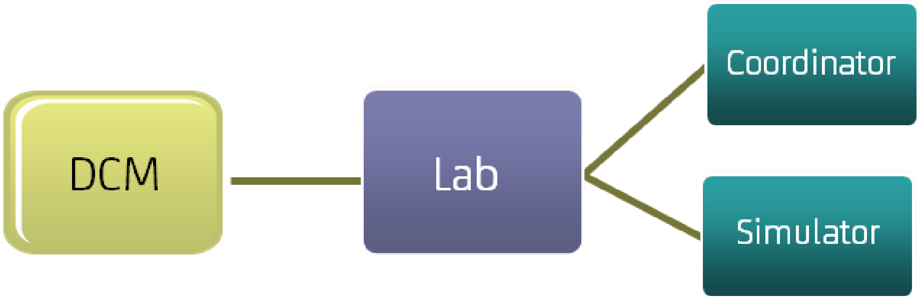
VLM Provider	Prefix
VMware vCloud Director	VCD

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 (see page 334).

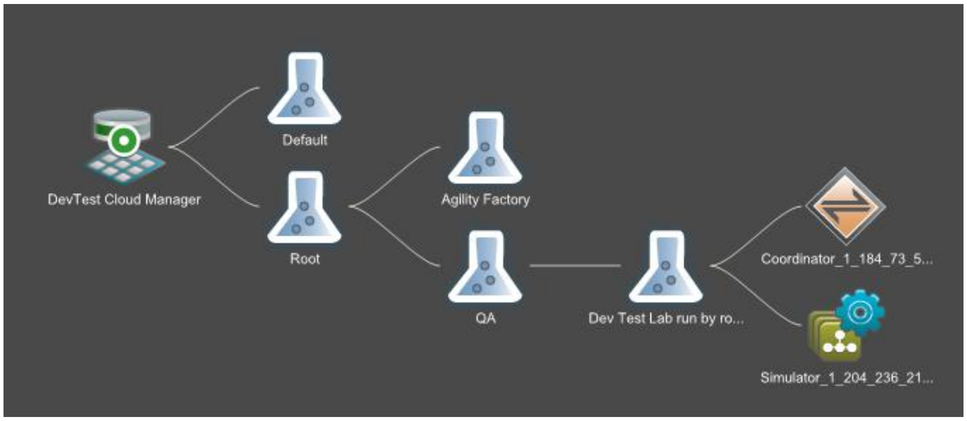


The responsibilities of the DCM include:

- 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 256) 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.


DevTest Cloud Manager
Default
Root
Agility Factory
QA
Dev Test Lab run by ro...
Coordinator_1_184_73_5...
Simulator_1_204_236_21...

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.

- General Properties (see page 338)
- vCloud Director Properties (see page 340)
- DCM Properties Example (see page 341)

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-addre
ss>
```

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.

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 topic 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 **remoteInit** 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:

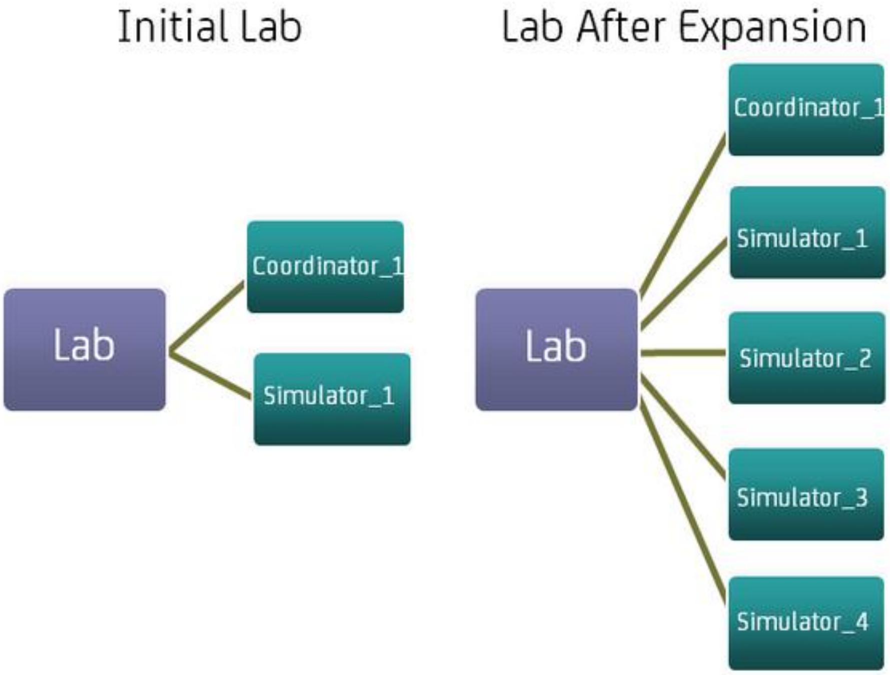
1. Using an application such as VMware Workstation, create a virtual machine image for each server component that you want to include in a lab.
    - To start the server component in **remoteInit** mode, configure each image.
    - The name of a coordinator must include the term **Coordinator**.
    - 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 225).*

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.



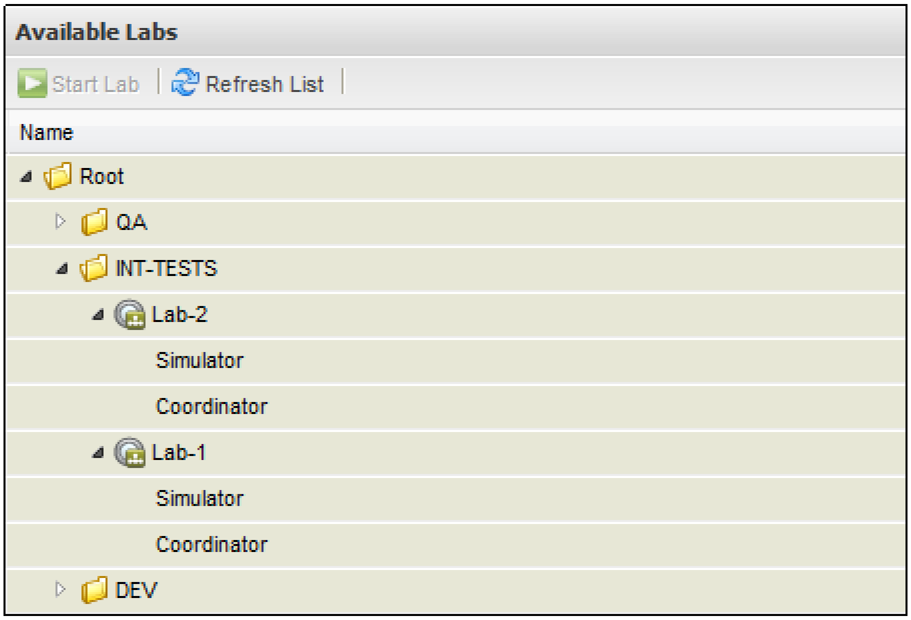
List the Available Labs

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 in *Administering.*

The following graphic shows a list of available labs in the Server Console. The tree structure is expanded to show two child 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 315) 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
```

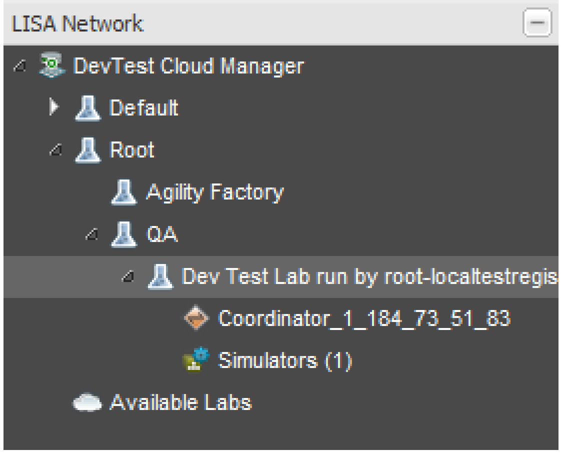
The following response contains a list of three available labs.

```
<?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.

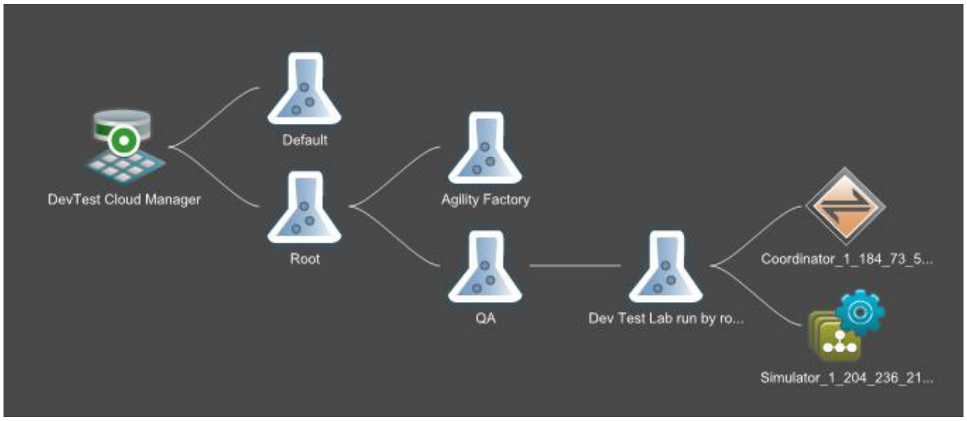
In the Server Console, the Network panel displays the started lab in a tree structure. The following graphic shows the Network panel.



LISA Network

- DevTest Cloud Manager
  - Default
  - Root
    - Agility Factory
    - QA
      - Dev Test Lab run by root-localtestregis
        - Coordinator_1_184_73_51_83
        - Simulators (1)
  - Available Labs

The right panel displays the started lab in the network graph. The following graphic shows the network graph.


DevTest Cloud Manager
Default
Root
Agility Factory
QA
Dev Test Lab run by ro...
Coordinator_1_184_73_5...
Simulator_1_204_236_21...

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 337).*

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 347)
- From the Server Console (see page 347)

- By using LISA Invoke (see page 348)

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 344) 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.

Start a Lab by Using LISA Invoke

You can use LISA Invoke (see page 315) 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 344).

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 315) 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 344).

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 346). 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 315) 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 344).

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>
```

Chapter 15: Continuous Validation Service (CVS)

The 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*.

This section contains the following topics:

Open the CVS Dashboard (see page 354)
CVS Dashboard Overview (see page 355)
Deploy a Monitor to CVS (see page 363)
Run a Monitor Immediately (see page 365)
View Test Details (see page 365)
Email Notification Settings (see page 366)
CVS Manager (see page 367)

Open the CVS Dashboard

You can open the CVS Dashboard from DevTest Workstation or from a web browser.

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 11) 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.

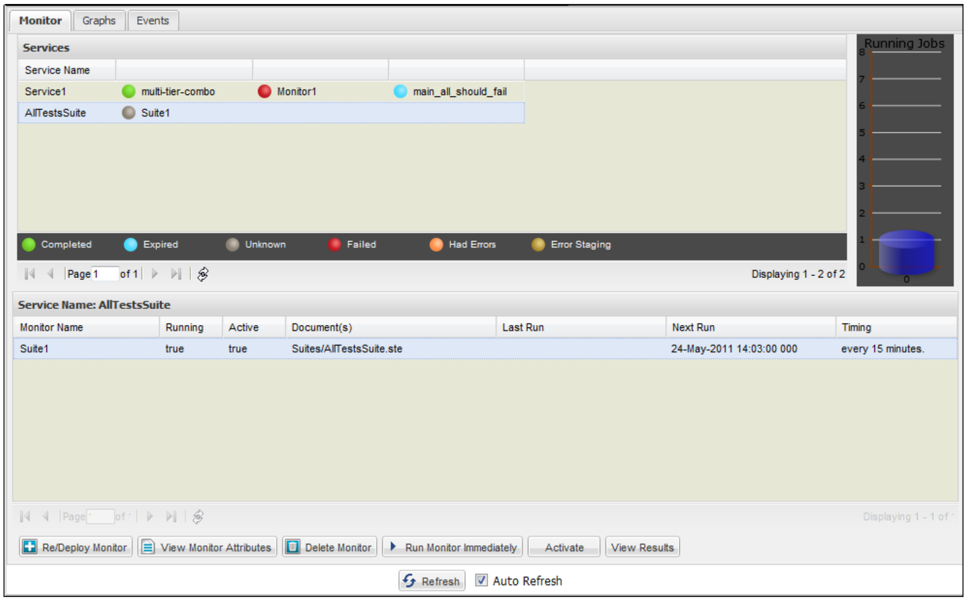
   The DevTest Console appears.
3. Click Continuous Validation Service.

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.

CVS Dashboard Overview

The CVS Dashboard has the following tabs:

- Monitor (see page 356): Displays a list of all the monitors (tests/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.
- Graphs (see page 360): Displays the dashboard status graphically. This tab also shows the percentage of the tests that passed or failed.
- Events (see page 362): Displays the status events that the monitors record.



Services

Service Name			
Service1	● multi-tier-combo	● Monitor1	● main_all_should_fail
AllTestsSuite	● Suite1		

Displaying 1 - 2 of 2

Service Name: AllTestsSuite

Monitor Name	Running	Active	Document(s)	Last Run	Next Run	Timing
Suite1	true	true	Suites/AllTestsSuite.ste		24-May-2011 14:03:00 000	every 15 minutes.

To refresh the list that is displayed on the CVS Dashboard, click Refresh at the bottom of the window.

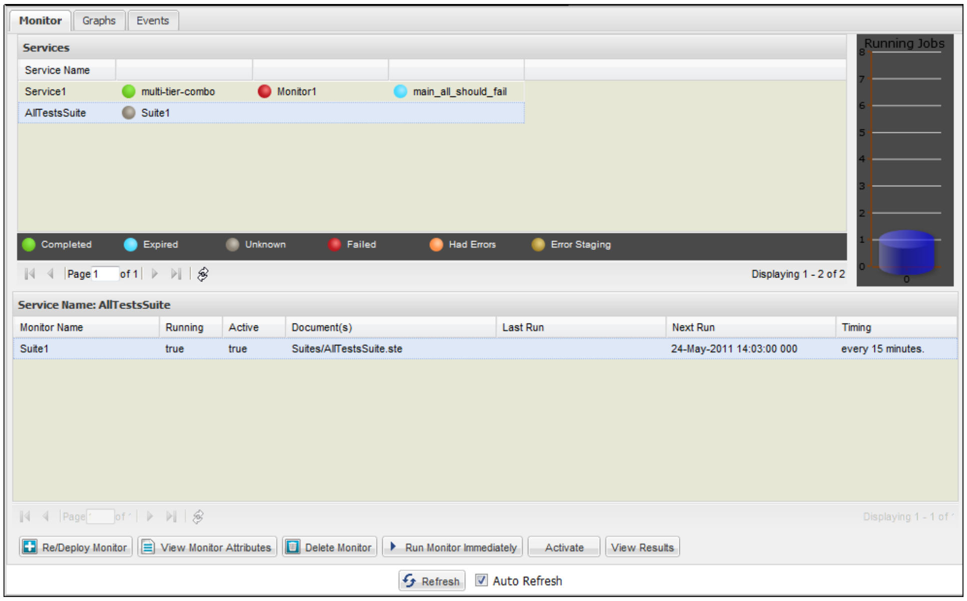
CVS Dashboard - Monitor

The CVS Dashboard opens with the Monitor 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 (in the following example, **Service1**) or add the monitors in different services (in the following example, **Service1** and **AllTestsSuite**). All the monitors added in one service (for example, **Service1**) are shown added after that service name.



Services

Service Name			
Service1	🟢 multi-tier-combo	🔴 Monitor1	🔵 main_all_should_fail
AllTestsSuite	● Suite1		

🟢 Completed 🔵 Expired ● Unknown 🔴 Failed 🟠 Had Errors 🟡 Error Staging

Page 1 of 1 Displaying 1 - 2 of 2

Service Name: AllTestsSuite

Monitor Name	Running	Active	Document(s)	Last Run	Next Run	Timing
Suite1	true	true	Suites/AllTestsSuite.ste		24-May-2011 14:03:00 000	every 15 minutes.

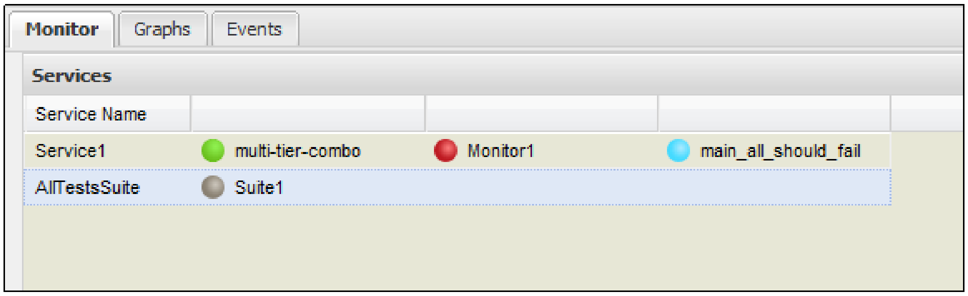
Page of Displaying 1 - 1 of

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.

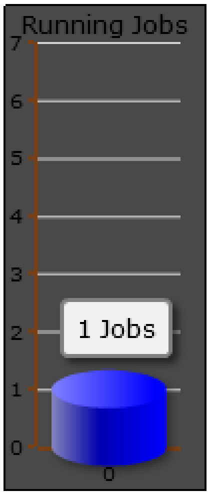
The icons show the Test Run Completion, Failure, Error, Staging Error, or Unknown Error. These icons show the state of the monitor after it has run.



Services

Service Name			
Service1	🟢 multi-tier-combo	🔴 Monitor1	🔵 main_all_should_fail
AllTestsSuite	● Suite1		

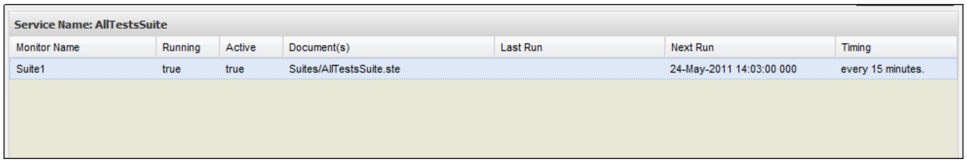
At the right side, the top panel shows a graphical representation of the jobs that are currently running. For example, the following graph shows one job currently running.



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.



Service Name: AllTestsSuite

Monitor Name	Running	Active	Document(s)	Last Run	Next Run	Timing
Suite1	true	true	Suites/AllTestsSuite.ste		24-May-2011 14:03:00 000	every 15 minutes.

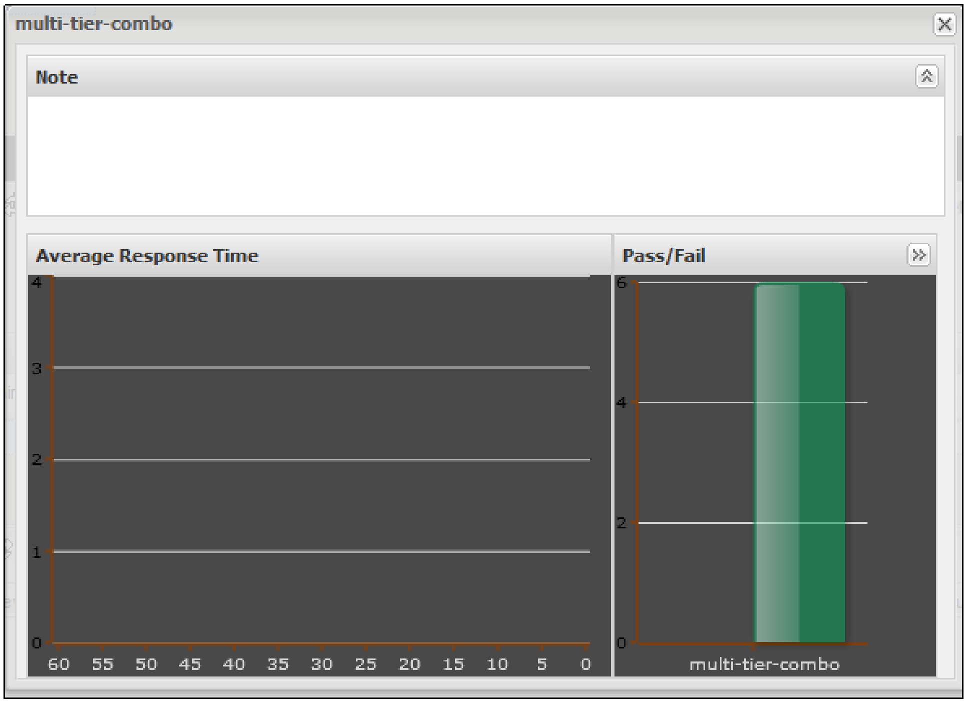
The table shows the following information:

- **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 363).
- **View Monitor Attributes:** View monitor-related information.
- **Delete Monitor:** Delete a monitor from the dashboard.
- **Run Monitor Immediately:** Run the monitor immediately.

- **Deactivate:** 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.

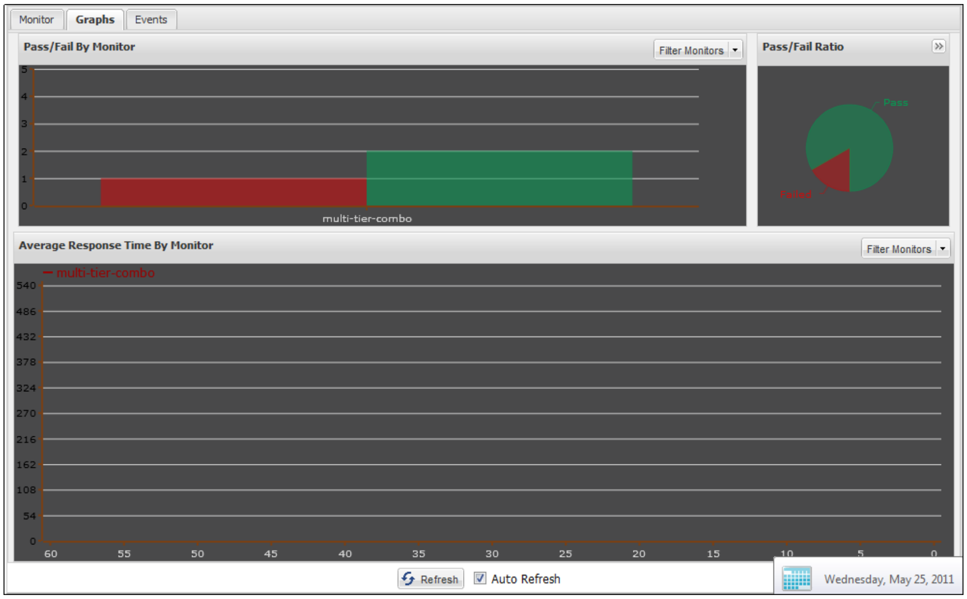
When you are running the CVS Dashboard directly from DevTest Workstation (by selecting to view the dashboard from DevTest), you also see the following buttons:

- 🔄 Refreshes the list on the dashboard
- ☑ Auto refreshes the list on the dashboard after a specific period.

When you run the CVS Dashboard directly from your browser, these buttons do not appear.

CVS Dashboard - Graphs

The Graphs 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 Graphs tab consists of three graphical displays, which show and track the last 60 minutes of activity.

The Filter Monitor lets you select the specific monitors that you want 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.

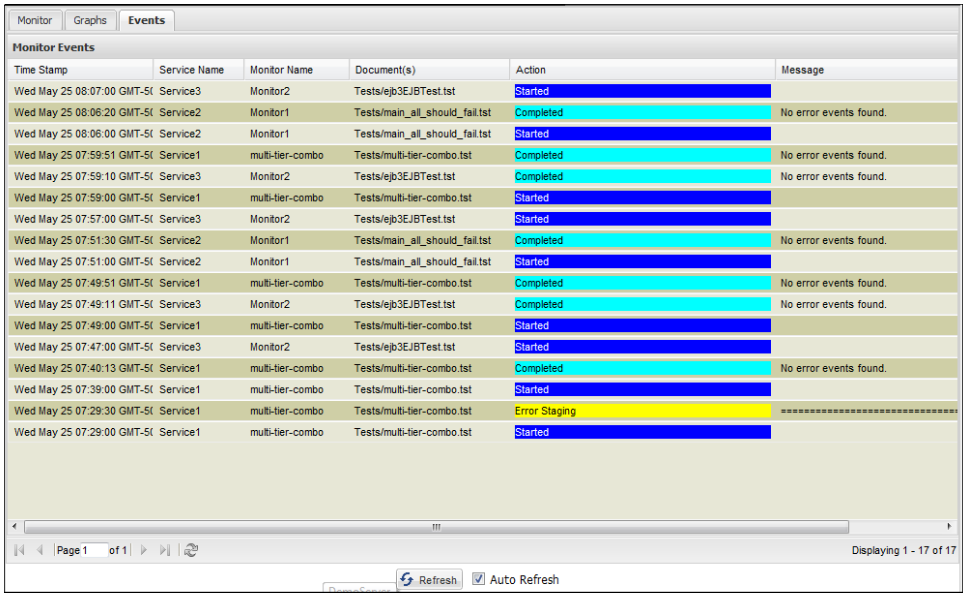
   - Pass/Fail Ratio**:** The Pass/Fail Ratio graph displays the percentage of total number of passing tests (green), and the total number of failing tests (red) as a pie chart.

   - 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.

CVS Dashboard - Events

The Events tab displays various events corresponding to the stages of the monitor run, such as starting of a test run, ending, and failing.



Monitor Events

Time Stamp	Service Name	Monitor Name	Document(s)	Action	Message
Wed May 25 08:07:00 GMT-5(	Service3	Monitor2	Tests/ejb3EJBTest.tst	Started	
Wed May 25 08:06:20 GMT-5(	Service2	Monitor1	Tests/main_all_should_fail.tst	Completed	No error events found.
Wed May 25 08:06:00 GMT-5(	Service2	Monitor1	Tests/main_all_should_fail.tst	Started	
Wed May 25 07:59:51 GMT-5(	Service1	multi-tier-combo	Tests/multi-tier-combo.tst	Completed	No error events found.
Wed May 25 07:59:10 GMT-5(	Service3	Monitor2	Tests/ejb3EJBTest.tst	Completed	No error events found.
Wed May 25 07:59:00 GMT-5(	Service1	multi-tier-combo	Tests/multi-tier-combo.tst	Started	
Wed May 25 07:57:00 GMT-5(	Service3	Monitor2	Tests/ejb3EJBTest.tst	Started	
Wed May 25 07:51:30 GMT-5(	Service2	Monitor1	Tests/main_all_should_fail.tst	Completed	No error events found.
Wed May 25 07:51:00 GMT-5(	Service2	Monitor1	Tests/main_all_should_fail.tst	Started	
Wed May 25 07:49:51 GMT-5(	Service1	multi-tier-combo	Tests/multi-tier-combo.tst	Completed	No error events found.
Wed May 25 07:49:11 GMT-5(	Service3	Monitor2	Tests/ejb3EJBTest.tst	Completed	No error events found.
Wed May 25 07:49:00 GMT-5(	Service1	multi-tier-combo	Tests/multi-tier-combo.tst	Started	
Wed May 25 07:47:00 GMT-5(	Service3	Monitor2	Tests/ejb3EJBTest.tst	Started	
Wed May 25 07:40:13 GMT-5(	Service1	multi-tier-combo	Tests/multi-tier-combo.tst	Completed	No error events found.
Wed May 25 07:39:00 GMT-5(	Service1	multi-tier-combo	Tests/multi-tier-combo.tst	Started	
Wed May 25 07:29:30 GMT-5(	Service1	multi-tier-combo	Tests/multi-tier-combo.tst	Error Staging	================================
Wed May 25 07:29:00 GMT-5(	Service1	multi-tier-combo	Tests/multi-tier-combo.tst	Started	

The following information is displayed:

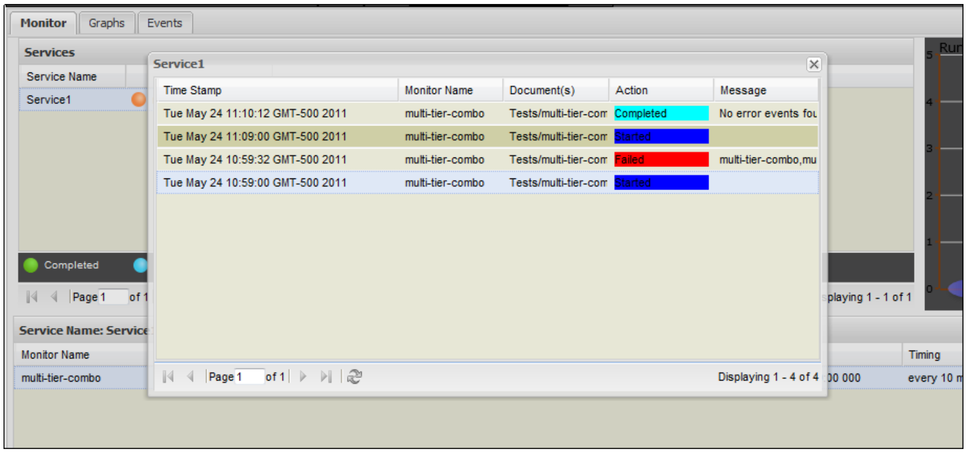
- Time Stamp**:** The time the event occurred.
- Service Name**:** The name of the service in which the monitor resides.
- Monitor Name**:** The name of the monitor in which the event occurred.
- Document(s)**:** A list of the documents that are associated with the monitor for which the event occurred.
- 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.

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 adjust the column sizes so that all columns can display on the window. If you double-click a Message field, you can see the full text of the message.

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.



Service1

Time Stamp	Monitor Name	Document(s)	Action	Message
Tue May 24 11:10:12 GMT-500 2011	multi-tier-combo	Tests/multi-tier-com	Completed	No error events fou
Tue May 24 11:09:00 GMT-500 2011	multi-tier-combo	Tests/multi-tier-com	Started	
Tue May 24 10:59:32 GMT-500 2011	multi-tier-combo	Tests/multi-tier-com	Failed	multi-tier-combo,mu
Tue May 24 10:59:00 GMT-500 2011	multi-tier-combo	Tests/multi-tier-com	Started	

Page 1 of 1 Displaying 1 - 4 of 4

Deploy a Monitor to CVS

The following topics are available.

Deploy a Monitor through DevTest Workstation (see page 363)
Deploy a Monitor through the CVS Dashboard (see page 364)
Deploy a Monitor through the cvsMonitors Directory (see page 364)
Deploy a Monitor through CVS Manager (see page 364)

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 262) in *Using CA Application Test.*

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 266).

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 266).

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 367).

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, select it.
3. Click Run Monitor 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.
- 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.

Email Notification Settings

Every time that you schedule a new test in the CVS utility, 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.

- 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.

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 353). 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
```

Chapter 16: Reports

You determine what data is collected for reports by specifying reporting parameters in one of three areas:

- Quick Tests (see page 281)
- Staging Documents (see page 215)
- Test Suites (see page 243)

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.

This section contains the following topics:

Report Generator Types (see page 369)
Open the Reporting Portal (see page 372)
Reporting Portal Layout (see page 372)
Filtering Reports (see page 375)
Viewing Reports (see page 376)
Exporting Reports (see page 406)
Changing Reporting Databases (see page 407)
Troubleshooting Reporting Performance (see page 408)

Report Generator Types

You can select the following types of report generators in the Reports tab of the Staging Document Editor (see page 217) or the Test Suite Editor (see page 244):

- Default Report Generator (see page 370)
- Load Test Report Generator (see page 371)
- XML Report Generator (see page 371)

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.

Chapter 17: 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 406).

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 11) is running.
2. Enter **http://localhost:1505/** in a web browser.

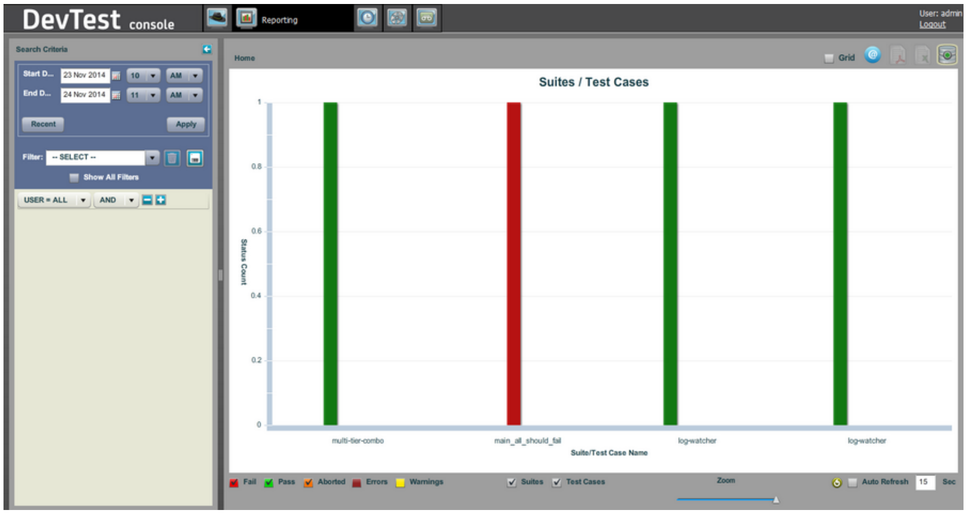
   If the registry is on a remote computer, replace **localhost** with the name or IP address of the computer.

   The DevTest Console opens.
3. Click Reporting Dashboard.

Reporting Portal Layout

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.



Start D... 23 Nov 2014 10 AM

End D... 24 Nov 2014 11 AM

Filter: -- SELECT --

USER = ALL AND

Auto Refresh 15 Sec

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.

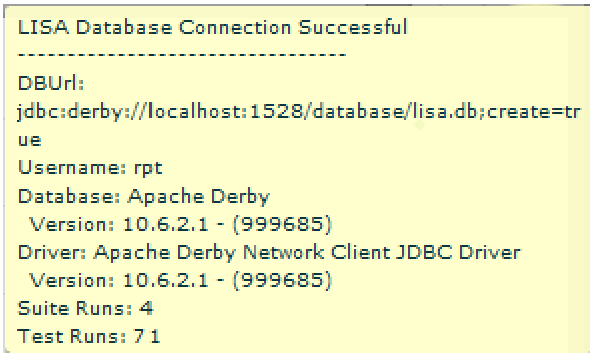
Select from the AND/OR operators for the criteria and click Add ⊞ or Delete ⊟.

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 377).

The Reporting Toolbar appears at the top of the right panel and allows you to:

Icon	Function
□ Grid ☑ Grid	Changes the report view from the default chart view to the grid view, and back. For more information, see Reports - Grid View (see page 393).
@	Copies the URL of the report you are viewing to the operating system clipboard so you can share the report with other users.
	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 406).

	Displays information about the reporting database.  LISA Database Connection Successful -------------------------------- DBUrl: jdbc:derby://localhost:1528/database/lisa.db;create=tr ue Username: rpt Database: Apache Derby Version: 10.6.2.1 - (999685) Driver: Apache Derby Network Client JDBC Driver Version: 10.6.2.1 - (999685) Suite Runs: 4 Test Runs: 71

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 375).

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 372) 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.

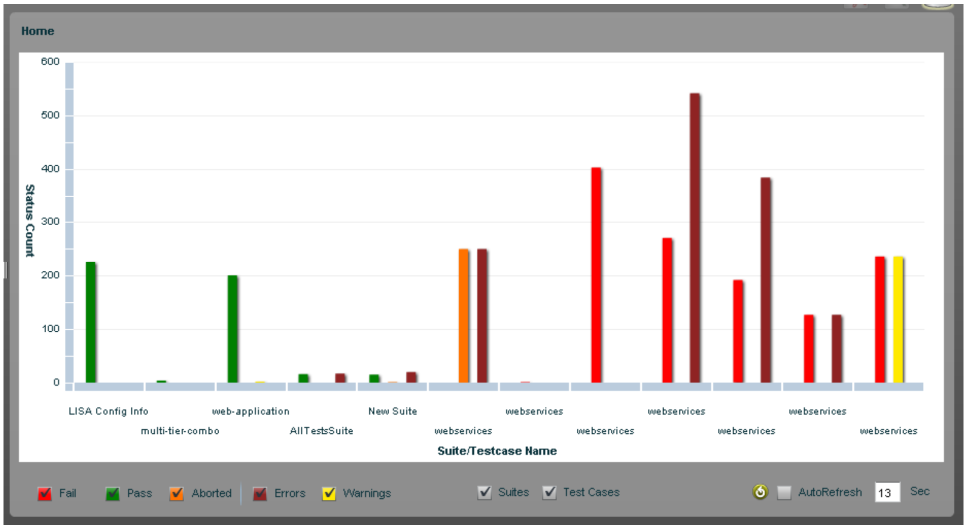
☑ **Suites**

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.



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 377): The default view of the Reporting Portal. You can see all the reports in the graphical format.
- Grid View (see page 393): You can select the grid view to arrange the data in a grid format.

More information:

Standard Reports (see page 395)
Interpreting Reports (see page 405)

Reports - Graphical View

By default, the reports appear as graphs.

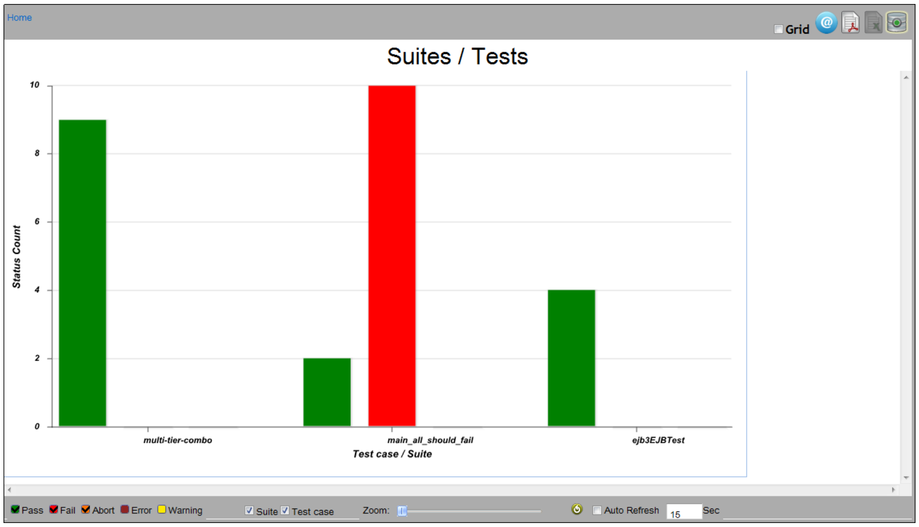
The following sample report contains three test cases:

- 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.

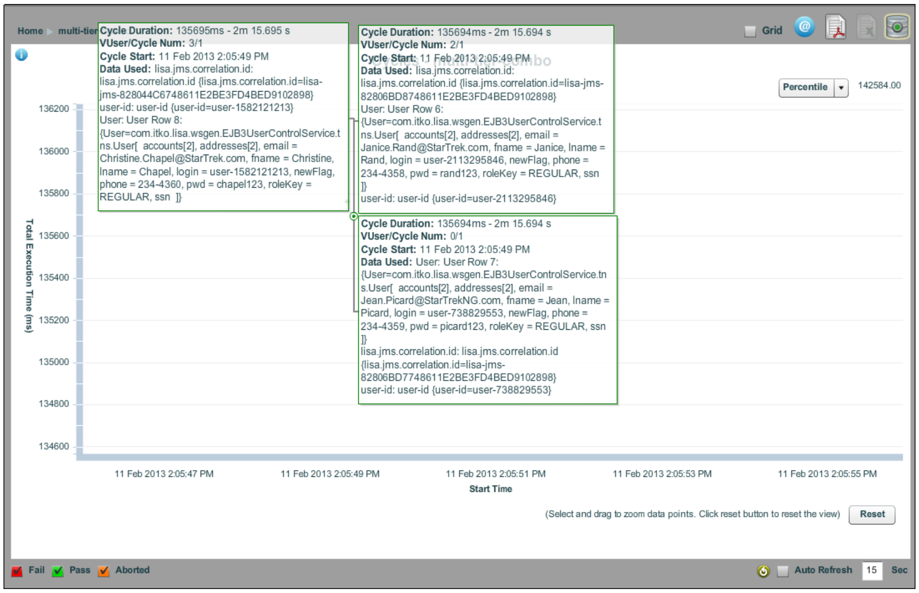


□ Grid

☑ Pass ☑ Fail ☑ Abort ■ Error □ Warning ☑ Suite ☑ Test case □ Auto Refresh 15 Sec

- 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.



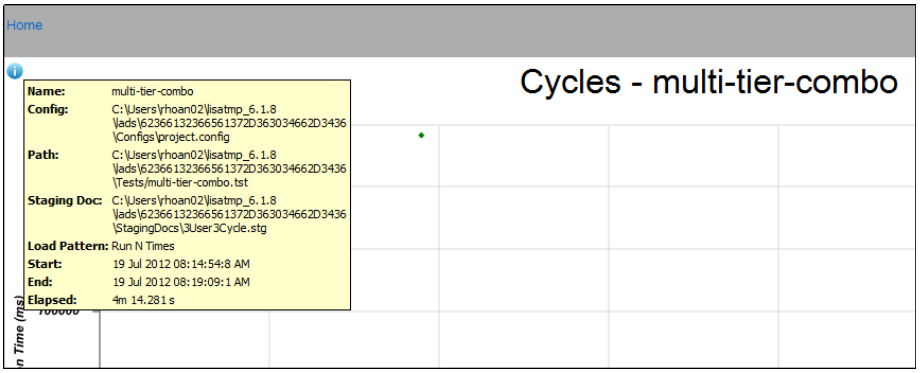
Cycle Duration: 135695ms - 2m 15.695 s
VUser/Cycle Num: 3/1
Cycle Start: 11 Feb 2013 2:05:49 PM
Data Used: lisa.jms.correlation.id: lisa.jms.correlation.id {lisa.jms.correlation.id=lisa-jms-828044C6748611E2BE3FD4BED9102898}
user-id: user-id {user-id=user-1582121213}
User: User Row 8: {User=com.itko.lisa.wsgen.EJB3UserControlService.tns.User[accounts[2], addresses[2], email = Christine.Chapel@StarTrek.com, fname = Christine, lname = Chapel, login = user-1582121213, newFlag, phone = 234-4360, pwd = chapel123, roleKey = REGULAR, ssn]}

Cycle Duration: 135694ms - 2m 15.694 s
VUser/Cycle Num: 2/1
Cycle Start: 11 Feb 2013 2:05:49 PM
Data Used: lisa.jms.correlation.id: lisa.jms.correlation.id {lisa.jms.correlation.id=lisa-jms-82806BD8748611E2BE3FD4BED9102898}
User: User Row 6: {User=com.itko.lisa.wsgen.EJB3UserControlService.tns.User[accounts[2], addresses[2], email = Janice.Rand@StarTrek.com, fname = Janice, lname = Rand, login = user-2113295846, newFlag, phone = 234-4358, pwd = rand123, roleKey = REGULAR, ssn]}
user-id: user-id {user-id=user-2113295846}

Cycle Duration: 135694ms - 2m 15.694 s
VUser/Cycle Num: 0/1
Cycle Start: 11 Feb 2013 2:05:49 PM
Data Used: User: User Row 7: {User=com.itko.lisa.wsgen.EJB3UserControlService.tns.User[accounts[2], addresses[2], email = Jean.Picard@StarTrekNG.com, fname = Jean, lname = Picard, login = user-738829553, newFlag, phone = 234-4359, pwd = picard123, roleKey = REGULAR, ssn]}
lisa.jms.correlation.id: lisa.jms.correlation.id {lisa.jms.correlation.id=lisa-jms-82806BD7748611E2BE3FD4BED9102898}
user-id: user-id {user-id=user-738829553}

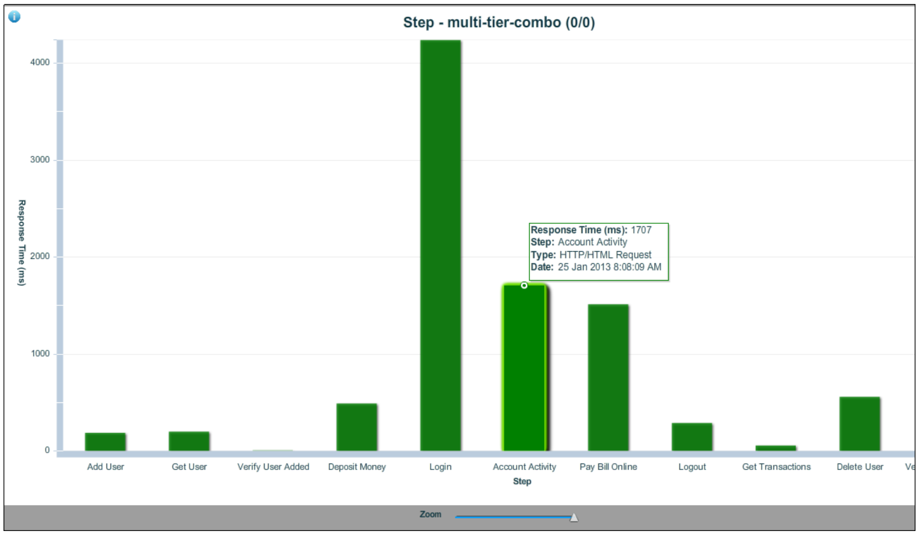
When you move your mouse over these dots, you see that each dot represents a cycle of the test case, and the details for each cycle are shown: response time in milliseconds, the cycle/instance number, the date of the cycle, and the data used. A cycle represents a complete test case from beginning to end.

- Clicking the Information button at the upper left of the window gives you more information about the test case environment.



Name:	multi-tier-combo
Config:	C:\Users\rhoan02\lisatmp_6.1.8\lads\62366132366561372D363034662D3436\Configs\project.config
Path:	C:\Users\rhoan02\lisatmp_6.1.8\lads\62366132366561372D363034662D3436\Tests/multi-tier-combo.tst
Staging Doc:	C:\Users\rhoan02\lisatmp_6.1.8\lads\62366132366561372D363034662D3436\StagingDocs\3User3Cycle.stg
Load Pattern:	Run N Times
Start:	19 Jul 2012 08:14:54:8 AM
End:	19 Jul 2012 08:19:09:1 AM
Elapsed:	4m 14.281 s

- 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 the Reset button.
- Double-clicking a test case shows a view of each step of the case, with information about 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 381).

Analyze	■ **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	■ 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	Shrinks the current graph on to the top half of the window and displays two additional reports: ■ A grid listing of test cases run and their results ■ A line chart showing historic execution times. The history report shows information on previous runs of the same test case.
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 Automatic Reporting Maintenance.
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.

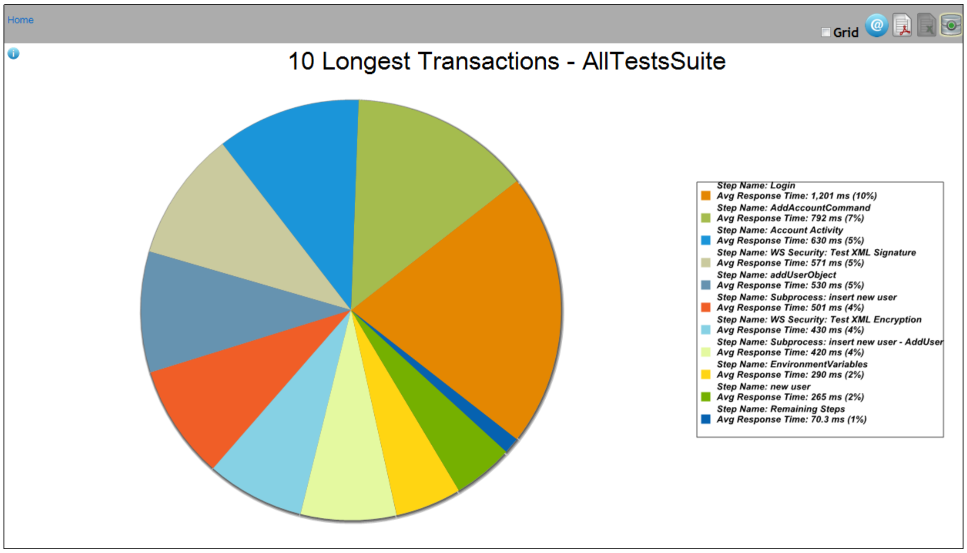
Reports - Graphical View Examples

This section provides examples of the following reporting menu options:

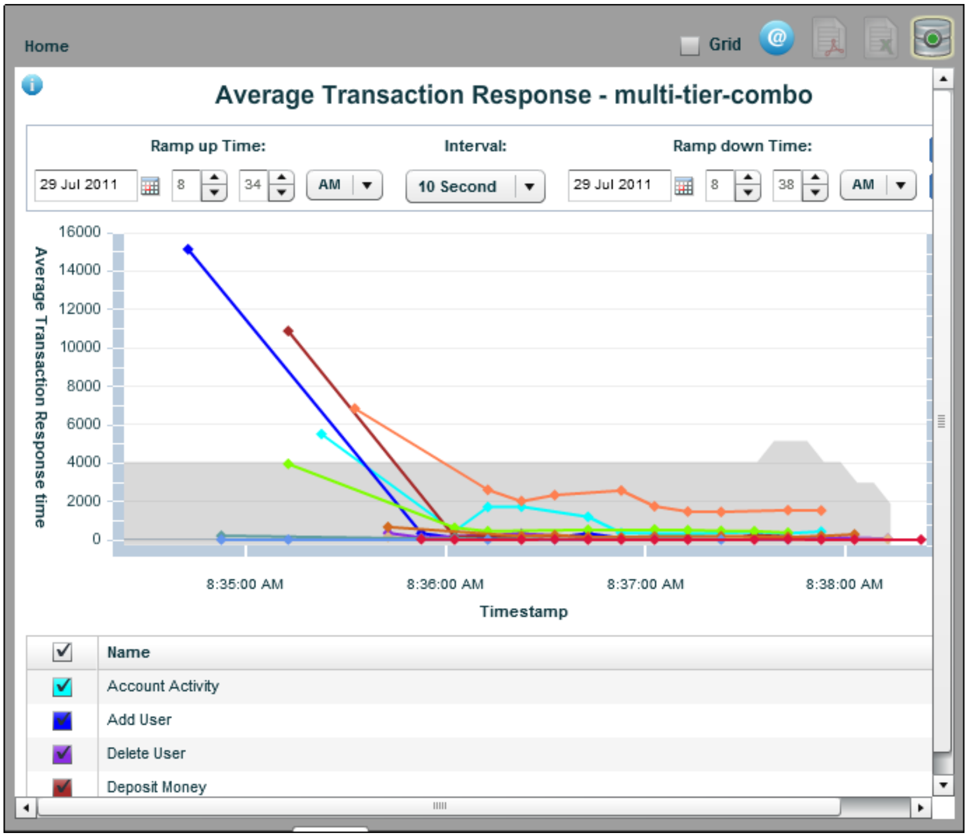
- Analyze (see page 382)
- View Error Reports (see page 388)
- View History (see page 392)

Analyze Report Examples

Top Ten Longest Transactions


Home
□ Grid
10 Longest Transactions - AllTestsSuite
🟧 Step Name: Login
Avg Response Time: 1,201 ms (10%)
🟩 Step Name: AddAccountCommand
Avg Response Time: 792 ms (7%)
🟦 Step Name: Account Activity
Avg Response Time: 630 ms (5%)
■ Step Name: WS Security: Test XML Signature
Avg Response Time: 571 ms (5%)
■ Step Name: addUserObject
Avg Response Time: 530 ms (5%)
🟥 Step Name: Subprocess: insert new user
Avg Response Time: 501 ms (4%)
■ Step Name: WS Security: Test XML Encryption
Avg Response Time: 430 ms (4%)
■ Step Name: Subprocess: insert new user - AddUser
Avg Response Time: 420 ms (4%)
🟨 Step Name: EnvironmentVariables
Avg Response Time: 290 ms (2%)
🟩 Step Name: new user
Avg Response Time: 265 ms (2%)
🟦 Step Name: Remaining Steps
Avg Response Time: 70.3 ms (1%)

Average Transaction Response



Home

□ Grid

Average Transaction Response - multi-tier-combo

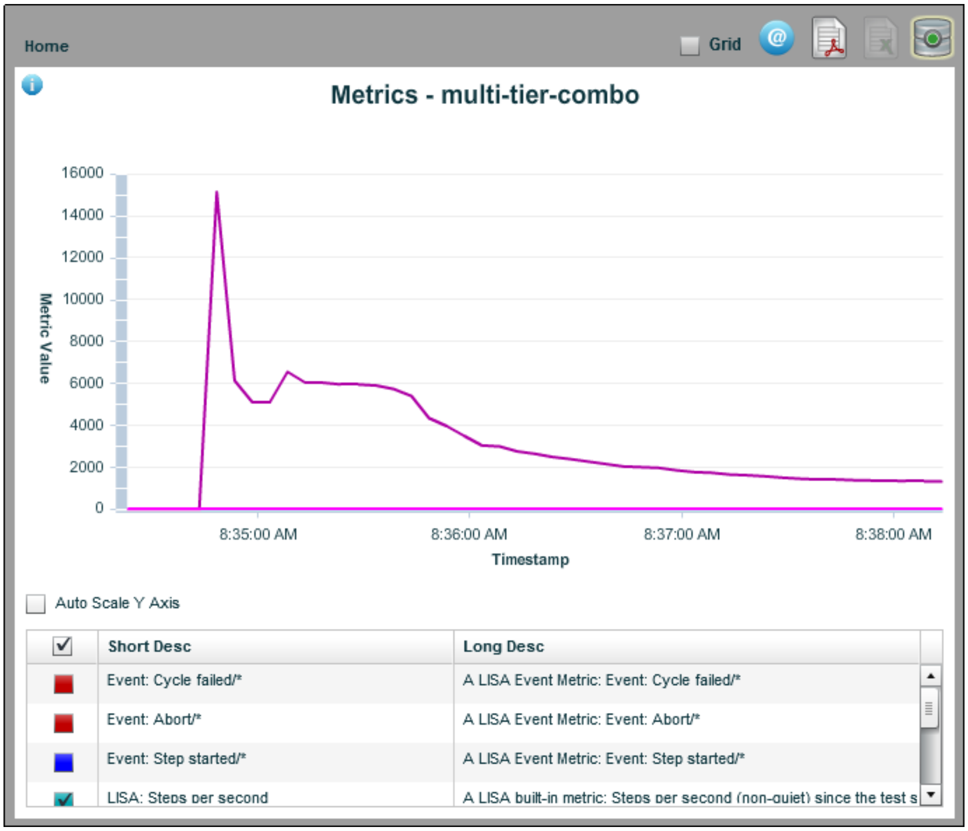
Ramp up Time: 29 Jul 2011 8 34 AM

Interval: 10 Second

Ramp down Time: 29 Jul 2011 8 38 AM

☑	Name
☑	Account Activity
☑	Add User
☑	Delete User
☑	Deposit Money

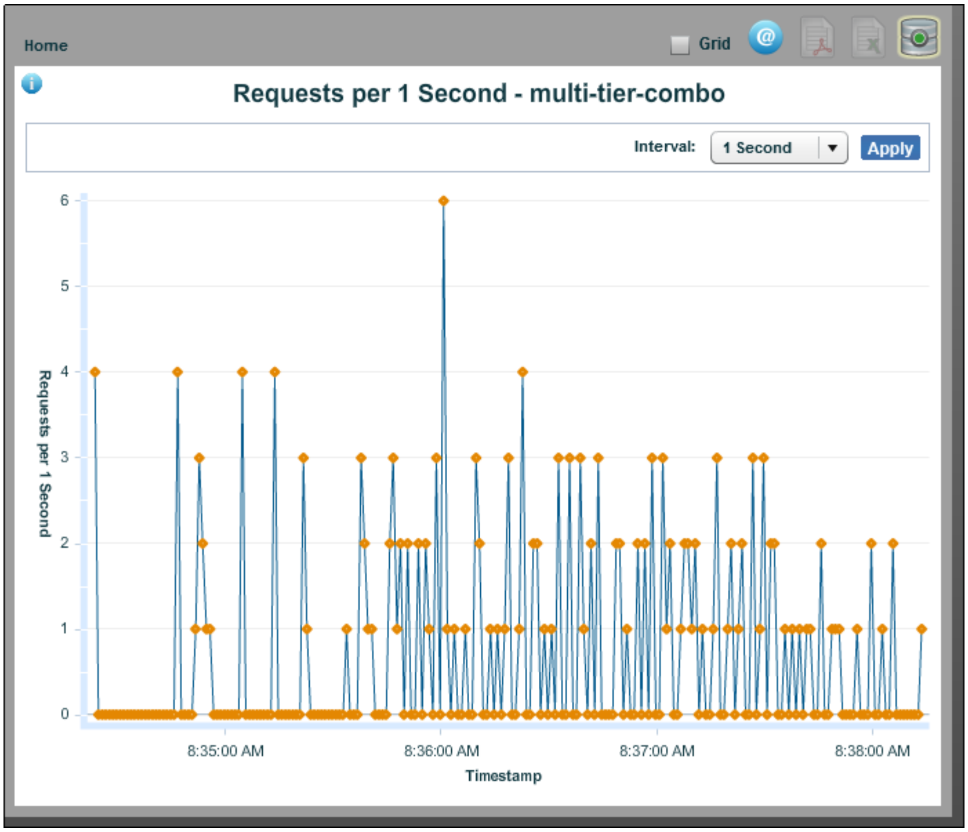
Metrics



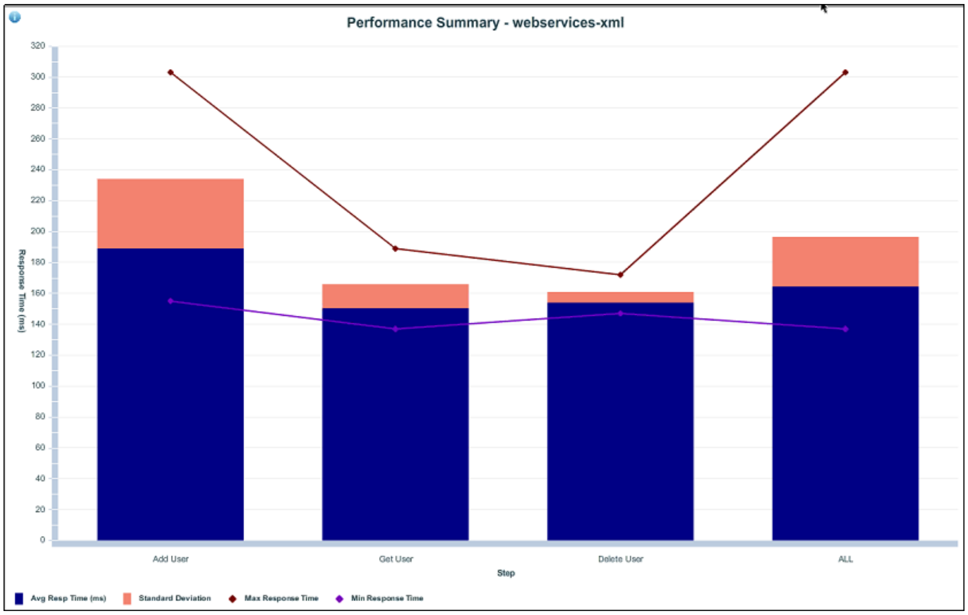
□ Auto Scale Y Axis

☑	Short Desc	Long Desc
🟥	Event: Cycle failed/*	A LISA Event Metric: Event: Cycle failed/*
🟥	Event: Abort/*	A LISA Event Metric: Event: Abort/*
🟦	Event: Step started/*	A LISA Event Metric: Event: Step started/*
☑	LISA: Steps per second	A LISA built-in metric: Steps per second (non-quiet) since the test s

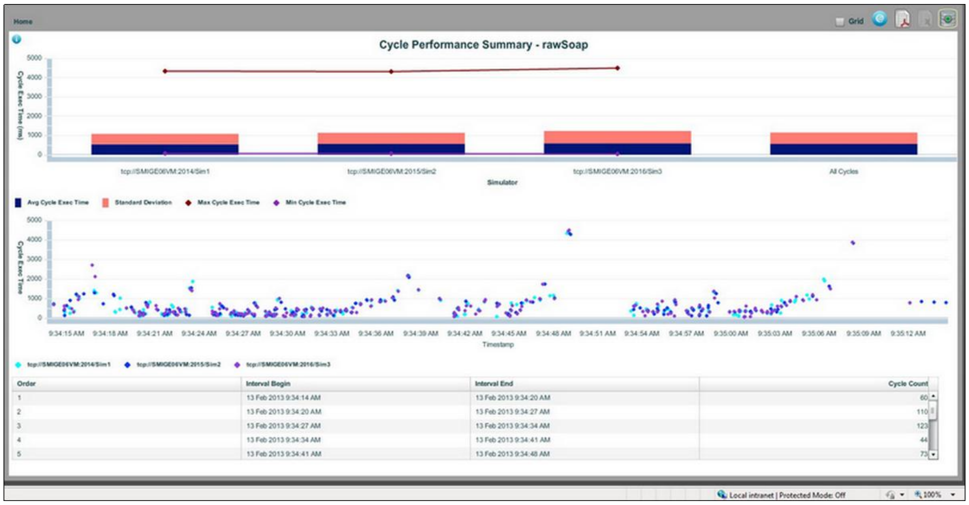
Requests/Second



Performance Summary

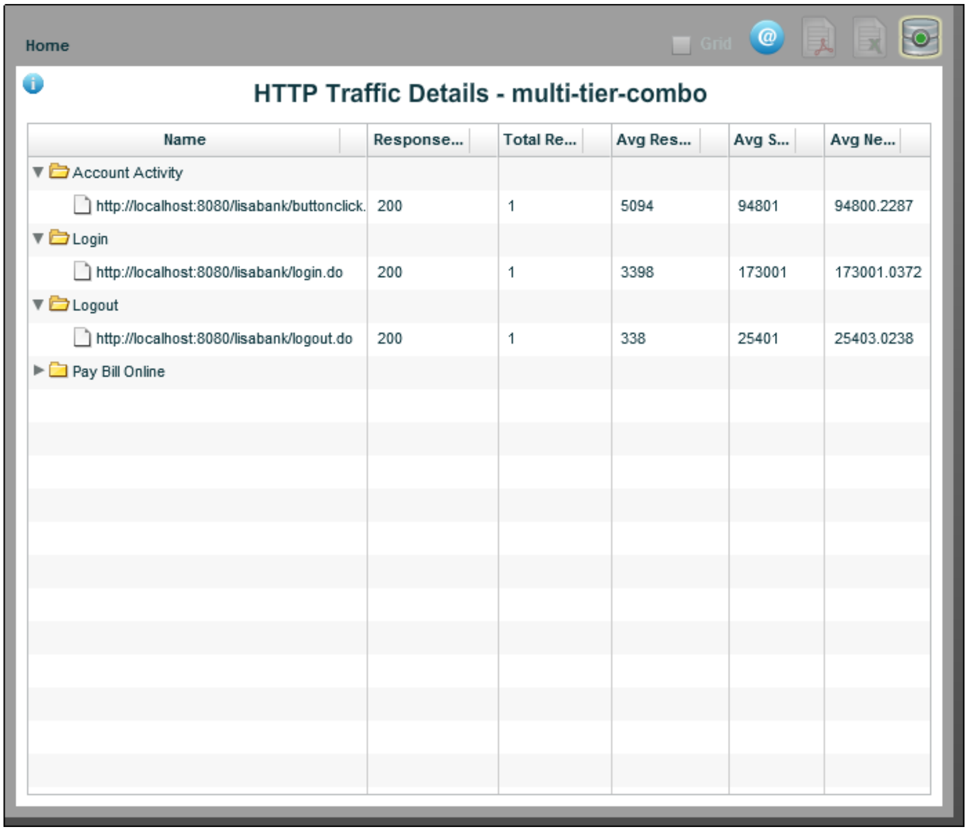


Cycle Performance Summary



Order	Interval Begin	Interval End	Cycle Count
1	13 Feb 2013 9:34:14 AM	13 Feb 2013 9:34:20 AM	60
2	13 Feb 2013 9:34:20 AM	13 Feb 2013 9:34:27 AM	110
3	13 Feb 2013 9:34:27 AM	13 Feb 2013 9:34:34 AM	123
4	13 Feb 2013 9:34:34 AM	13 Feb 2013 9:34:41 AM	44
5	13 Feb 2013 9:34:41 AM	13 Feb 2013 9:34:48 AM	73

HTTP Details

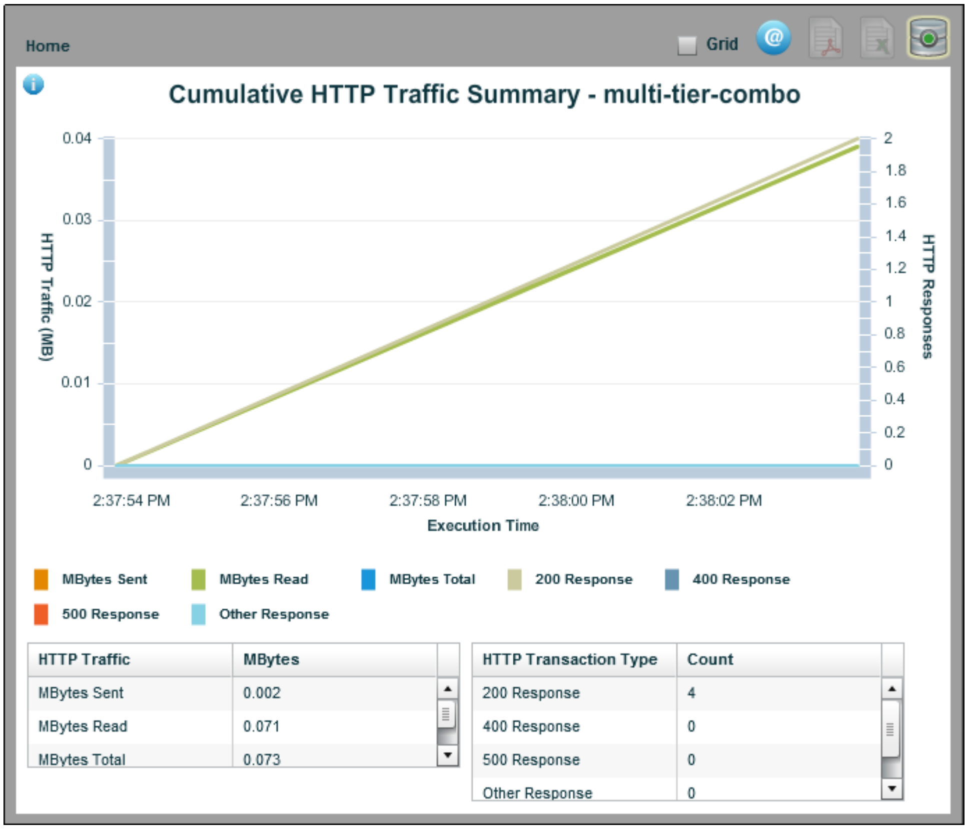


HTTP Traffic Details - multi-tier-combo

Name	Response...	Total Re...	Avg Res...	Avg S...	Avg Ne...
▼ Account Activity					
http://localhost:8080/lisabank/buttonclick.	200	1	5094	94801	94800.2287
▼ Login					
http://localhost:8080/lisabank/login.do	200	1	3398	173001	173001.0372
▼ Logout					
http://localhost:8080/lisabank/logout.do	200	1	338	25401	25403.0238
▶ Pay Bill Online					

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**.



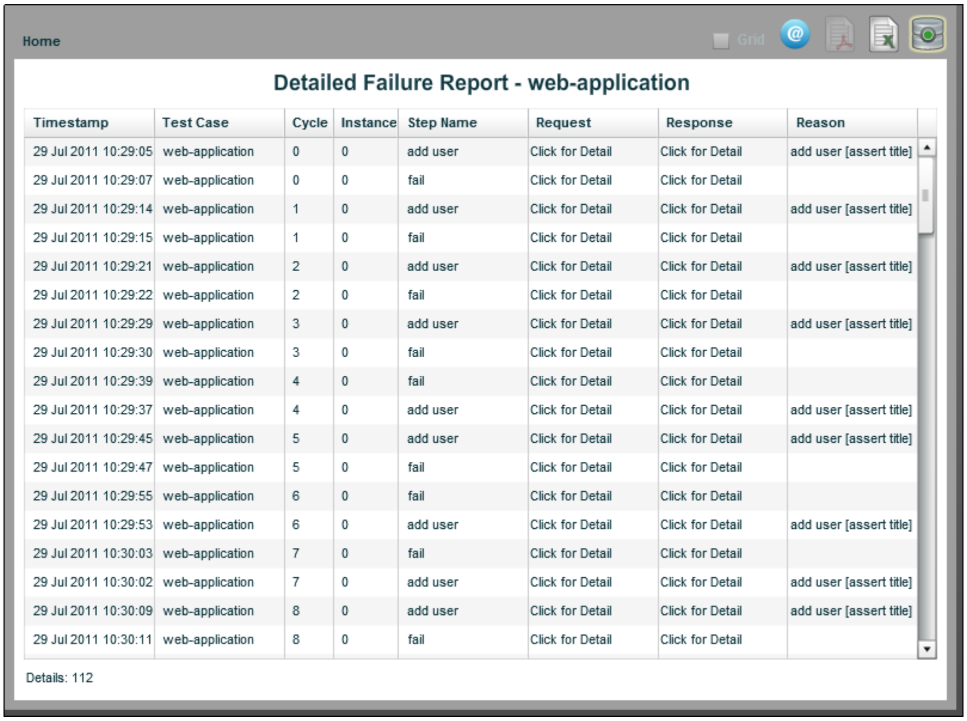
HTTP Traffic	MBytes
MBytes Sent	0.002
MBytes Read	0.071
MBytes Total	0.073

HTTP Transaction Type	Count
200 Response	4
400 Response	0
500 Response	0
Other Response	0

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

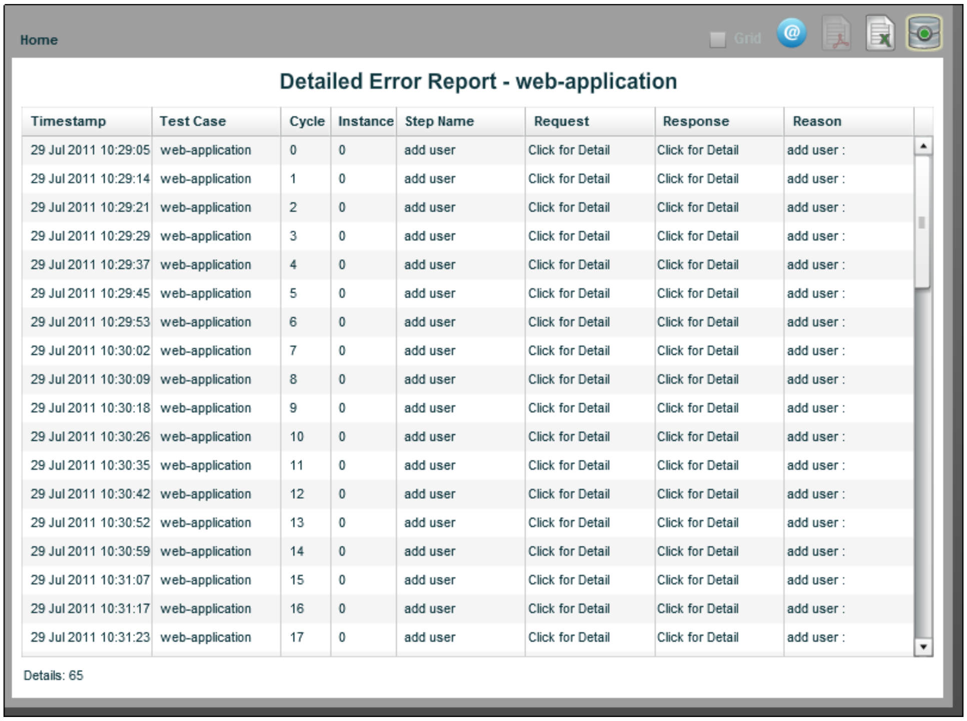


Detailed Failure Report - web-application

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

Detailed Errors

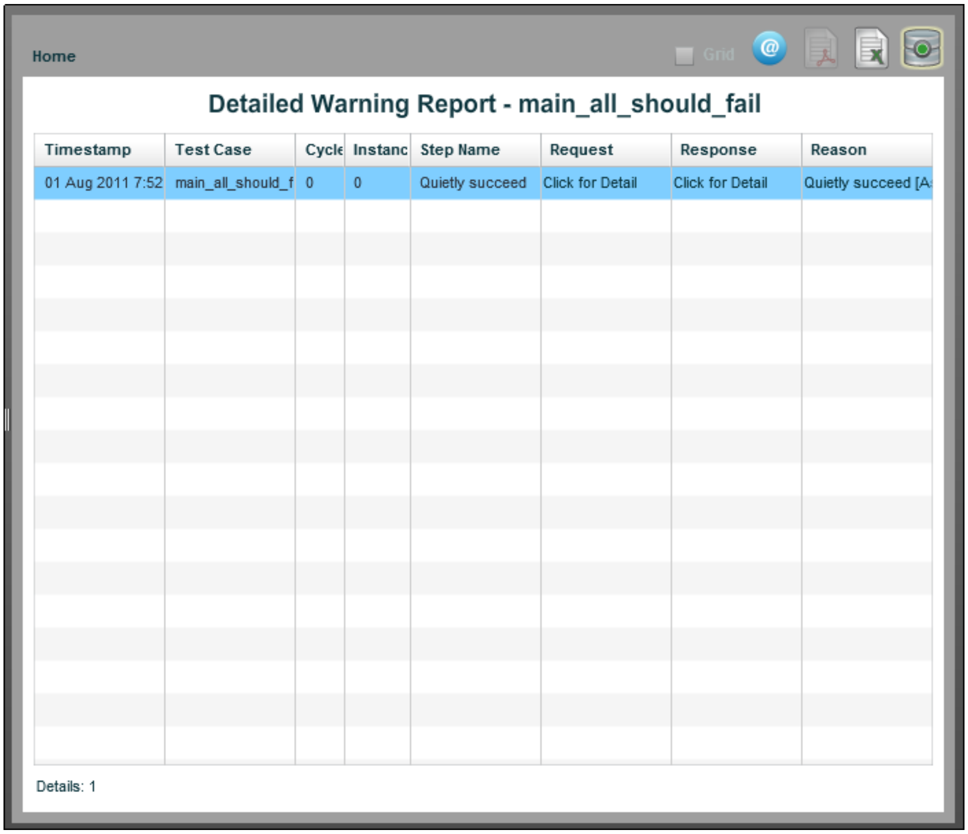


Detailed Error Report - web-application

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

Detailed Warnings

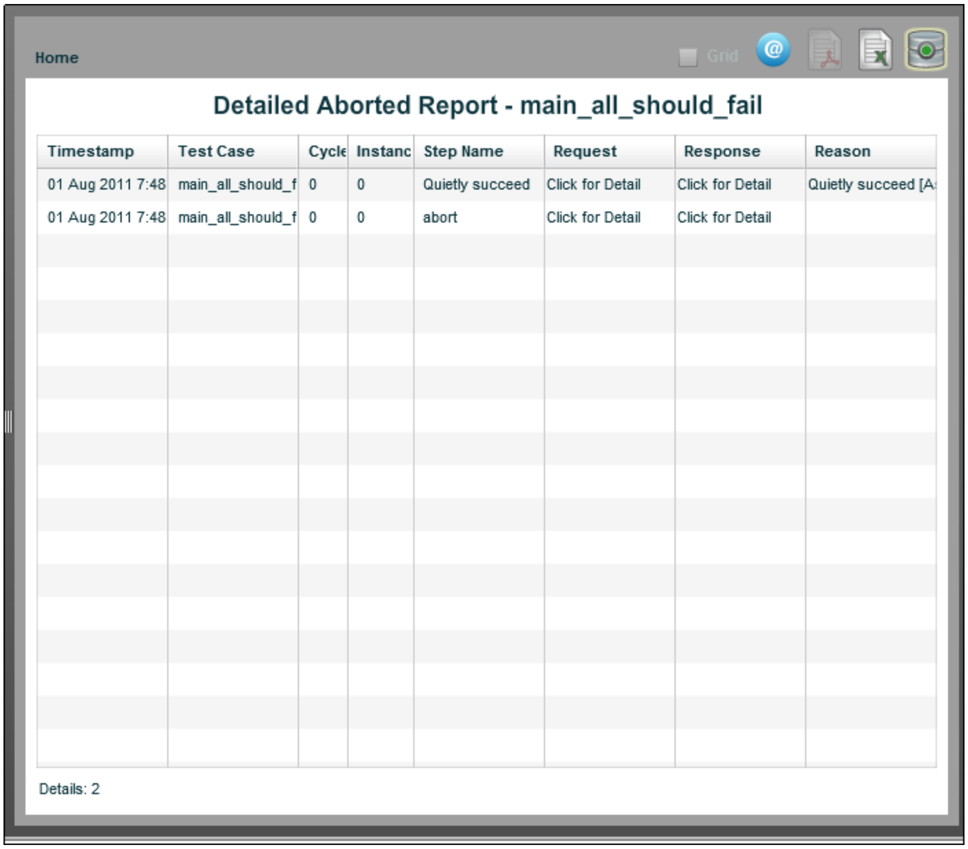


Detailed Warning Report - main_all_should_fail

Timestamp	Test Case	Cycle	Instanc	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

Details: 1

Detailed Aborts



Detailed Aborted Report - main_all_should_fail

Timestamp	Test Case	Cycle	Instanc	Step Name	Request	Response	Reason
01 Aug 2011 7:48	main_all_should_f	0	0	Quietly succeed	Click for Detail	Click for Detail	Quietly succeed [A
01 Aug 2011 7:48	main_all_should_f	0	0	abort	Click for Detail	Click for Detail	

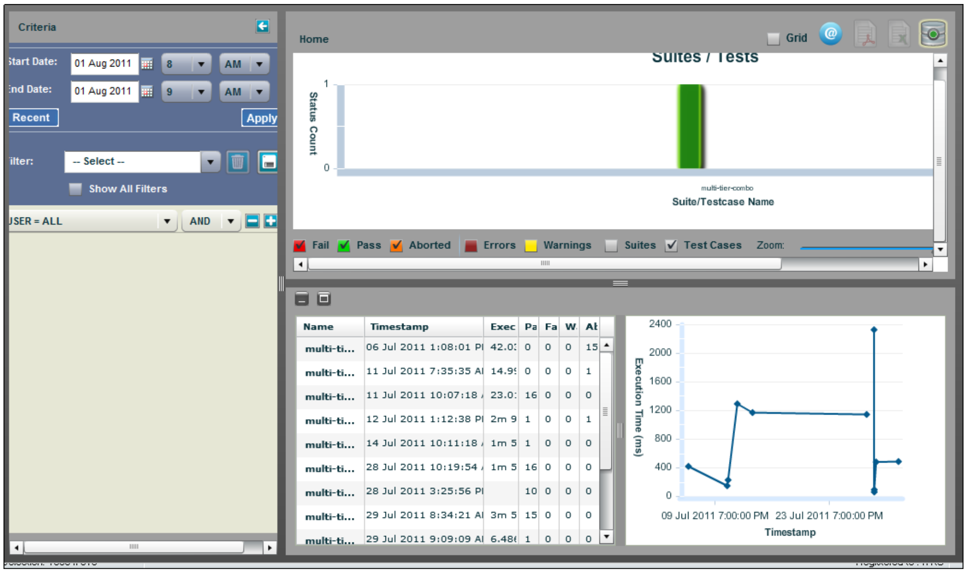
Details: 2

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.



Criteria

Start Date: 01 Aug 2011 8 AM

End Date: 01 Aug 2011 9 AM

Filter: -- Select --

□ Show All Filters

USER = ALL AND

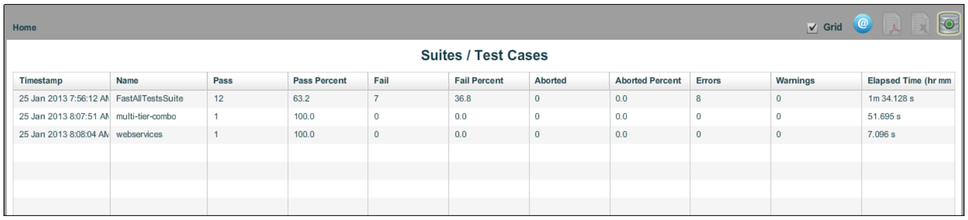
Name	Timestamp	Exec	Pa	Fa	W	Ab
multi-ti...	06 Jul 2011 1:08:01 PI	42.0:	0	0	0	15
multi-ti...	11 Jul 2011 7:35:35 AI	14.9:	0	0	0	1
multi-ti...	11 Jul 2011 10:07:18 /	23.0:	16	0	0	0
multi-ti...	12 Jul 2011 1:12:38 PI	2m 9	1	0	0	1
multi-ti...	14 Jul 2011 10:11:18 /	1m 5	1	0	0	0
multi-ti...	28 Jul 2011 10:19:54 /	1m 5	16	0	0	0
multi-ti...	28 Jul 2011 3:25:56 PI		10	0	0	0
multi-ti...	29 Jul 2011 8:34:21 AI	3m 5	15	0	0	0
multi-ti...	29 Jul 2011 9:09:09 AI	6.48(	1	0	0	0

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 ☑ Grid at the top of the window.

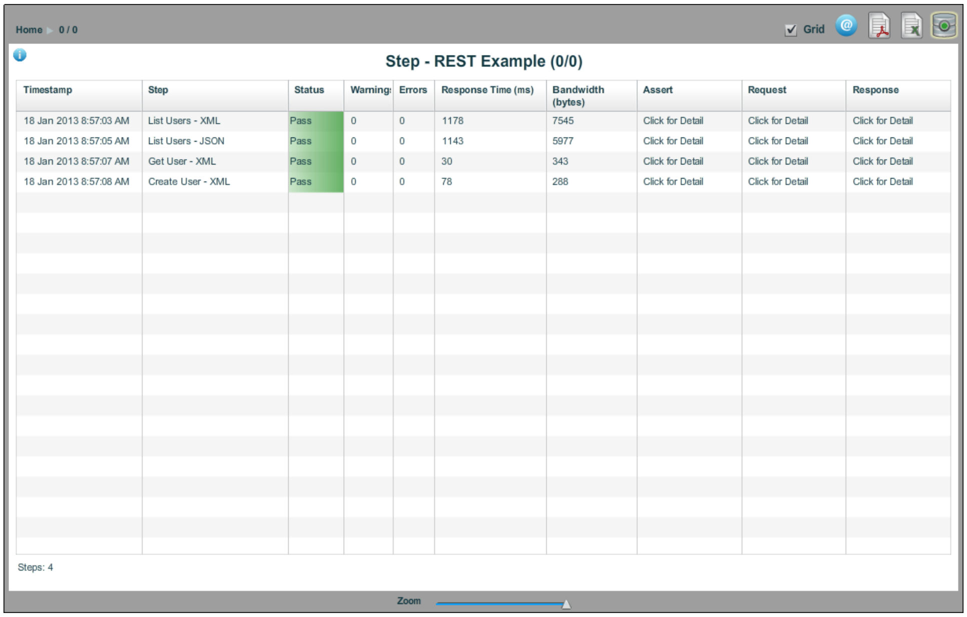


Home ☑ Grid

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

You can select each of the test cases in the report to find more details for the selected test case.



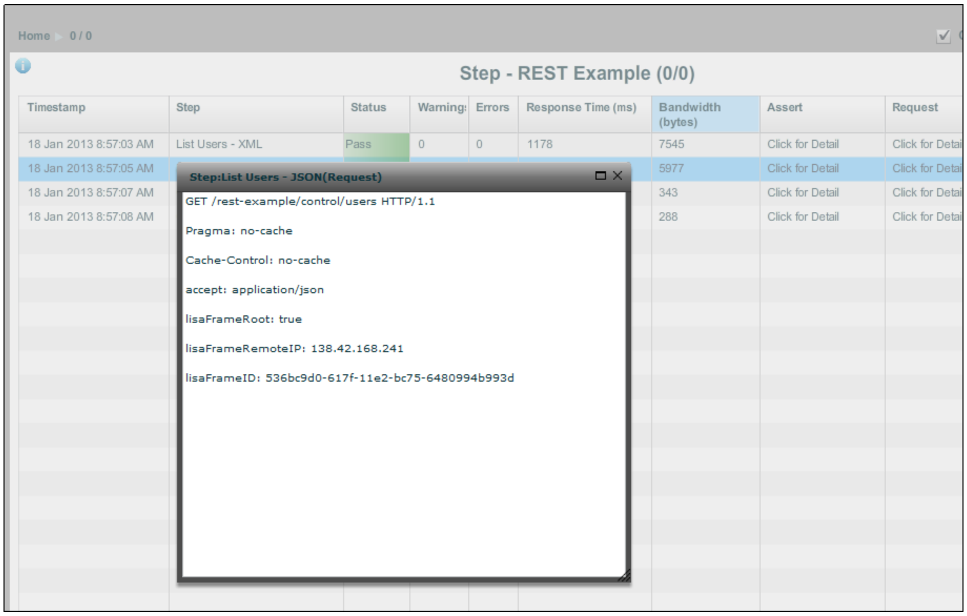
Home ▸ 0 / 0 ☑ Grid

Step - REST Example (0/0)

Timestamp	Step	Status	Warning:	Errors	Response Time (ms)	Bandwidth (bytes)	Assert	Request	Response
18 Jan 2013 8:57:03 AM	List Users - XML	Pass	0	0	1178	7545	Click for Detail	Click for Detail	Click for Detail
18 Jan 2013 8:57:05 AM	List Users - JSON	Pass	0	0	1143	5977	Click for Detail	Click for Detail	Click for Detail
18 Jan 2013 8:57:07 AM	Get User - XML	Pass	0	0	30	343	Click for Detail	Click for Detail	Click for Detail
18 Jan 2013 8:57:08 AM	Create User - XML	Pass	0	0	78	288	Click for Detail	Click for Detail	Click for Detail

Steps: 4

If you click Click for Detail in the Assert, Request, or Response columns, detailed information for each component of the step displays.



Step - REST Example (0/0)

Timestamp	Step	Status	Warning:	Errors	Response Time (ms)	Bandwidth (bytes)	Assert	Request
18 Jan 2013 8:57:03 AM	List Users - XML	Pass	0	0	1178	7545	Click for Detail	Click for Detai
18 Jan 2013 8:57:05 AM						5977	Click for Detail	Click for Detai
18 Jan 2013 8:57:07 AM						343	Click for Detail	Click for Detai
18 Jan 2013 8:57:08 AM						288	Click for Detail	Click for Detai

Step:List Users - JSON(Request)

```
GET /rest-example/control/users HTTP/1.1

Pragma: no-cache

Cache-Control: no-cache

accept: application/json

lisaFrameRoot: true

lisaFrameRemoteIP: 138.42.168.241

lisaFrameID: 536bc9d0-617f-11e2-bc75-6480994b993d
```

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 396)
- Performance Test Report (see page 397)
- Suite Summary Report (see page 399)
- Metrics Report (see page 401)

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).

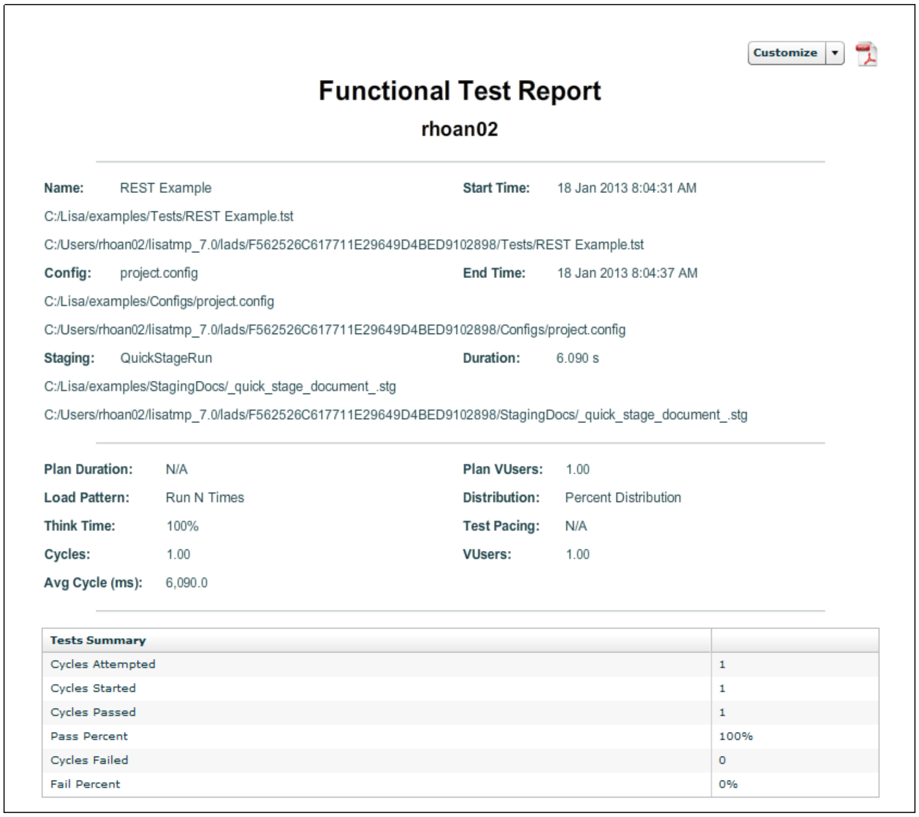
Functional Test Report

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.



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		

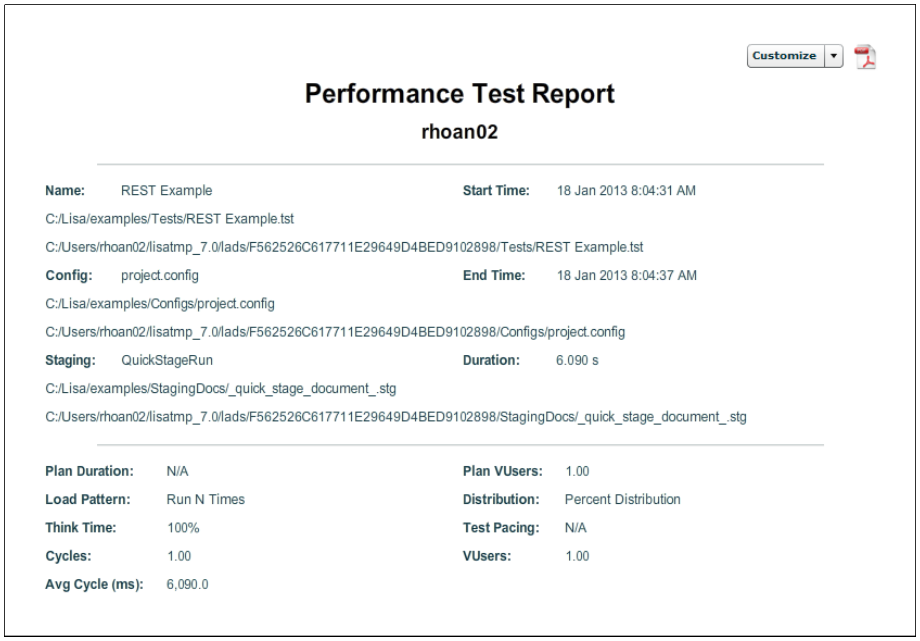
Tests Summary	
Cycles Attempted	1
Cycles Started	1
Cycles Passed	1
Pass Percent	100%
Cycles Failed	0
Fail Percent	0%

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.



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

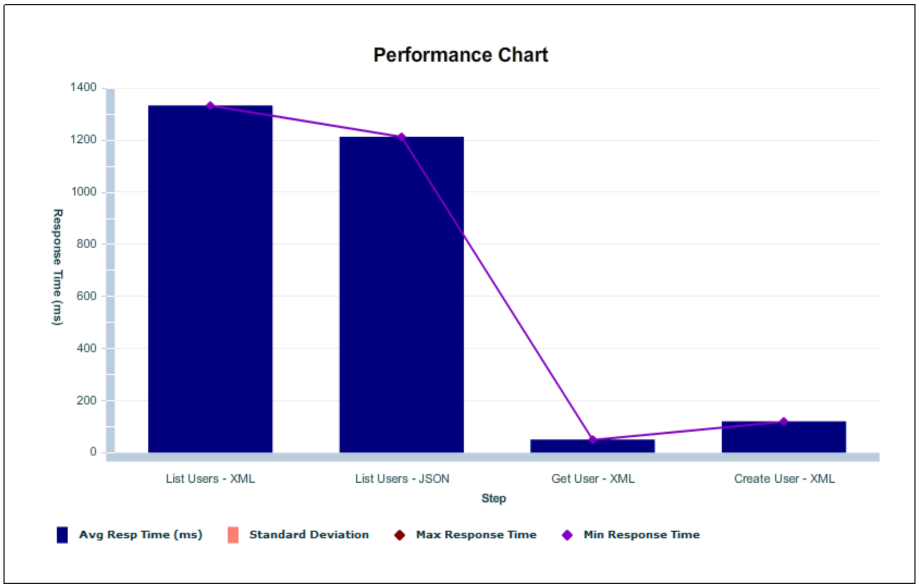
Plan Duration: N/A **Plan VUsers:** 1.00

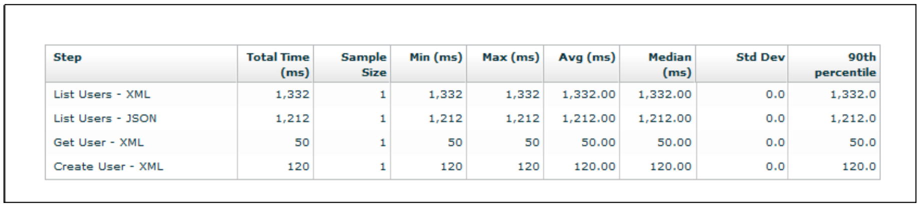
Load Pattern: Run N Times **Distribution:** Percent Distribution

Think Time: 100% **Test Pacing:** N/A

Cycles: 1.00 **VUsers:** 1.00

Avg Cycle (ms): 6,090.0





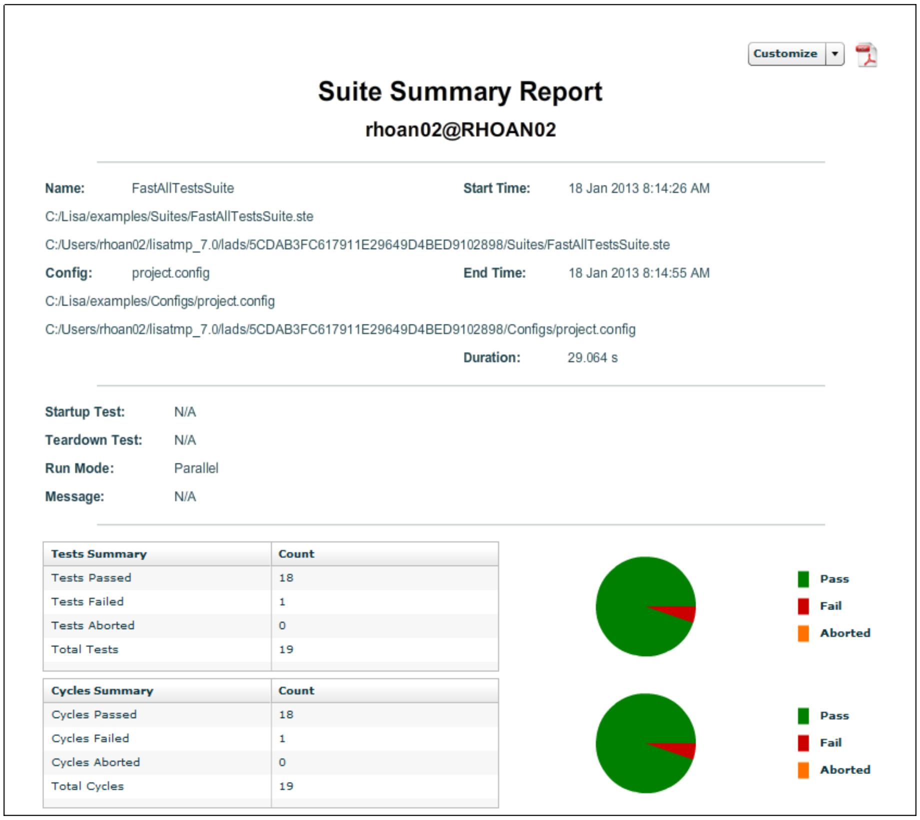
Step	Total Time (ms)	Sample Size	Min (ms)	Max (ms)	Avg (ms)	Median (ms)	Std Dev	90th percentile
List Users - XML	1,332	1	1,332	1,332	1,332.00	1,332.00	0.0	1,332.0
List Users - JSON	1,212	1	1,212	1,212	1,212.00	1,212.00	0.0	1,212.0
Get User - XML	50	1	50	50	50.00	50.00	0.0	50.0
Create User - XML	120	1	120	120	120.00	120.00	0.0	120.0

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.



Suite Summary Report

rhoan02@RHOAN02

Name: FastAllTestsSuite **Start Time:** 18 Jan 2013 8:14:26 AM

C:/Lisa/examples/Suites/FastAllTestsSuite.ste

C:/Users/rhoan02/lisatmp_7.0/lads/5CDAB3FC617911E29649D4BED9102898/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/Configs/project.config

Duration: 29.064 s

Startup Test: N/A

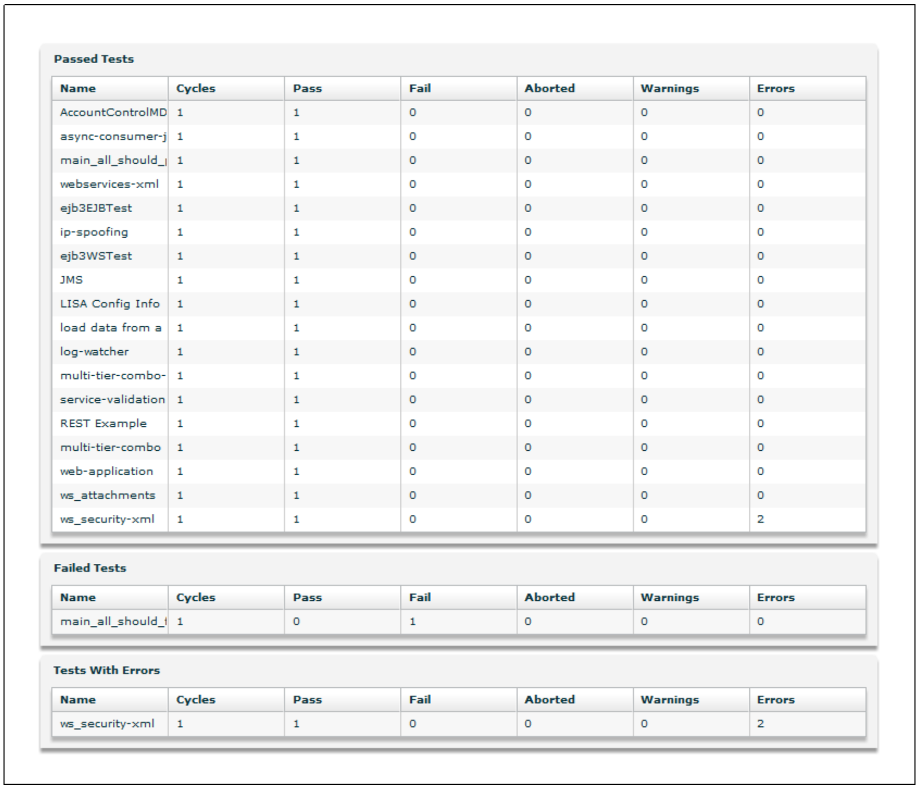
Teardown Test: N/A

Run Mode: Parallel

Message: N/A

Tests Summary	Count
Tests Passed	18
Tests Failed	1
Tests Aborted	0
Total Tests	19

Cycles Summary	Count
Cycles Passed	18
Cycles Failed	1
Cycles Aborted	0
Total Cycles	19



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_	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_	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

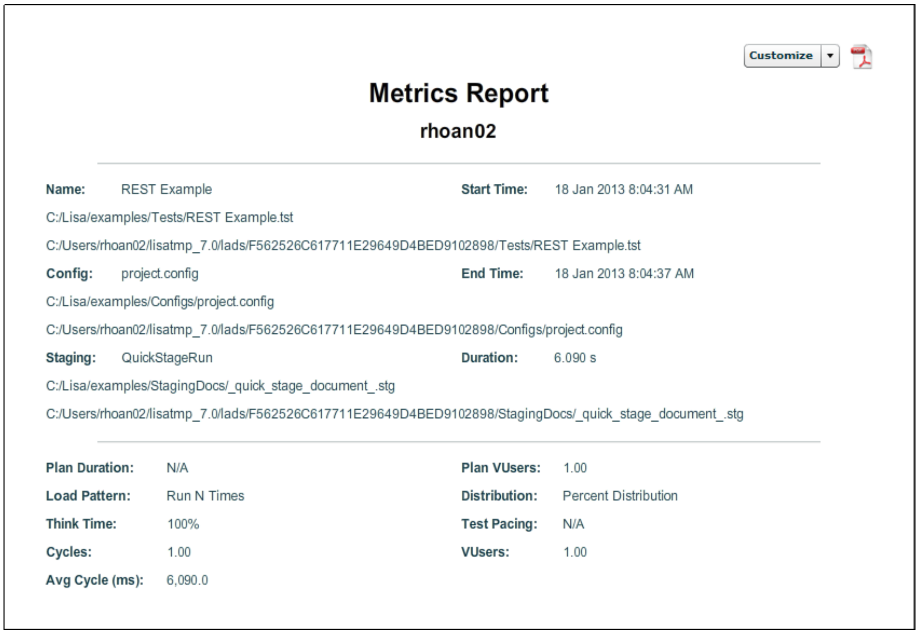
Error Messages

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.



Metrics 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

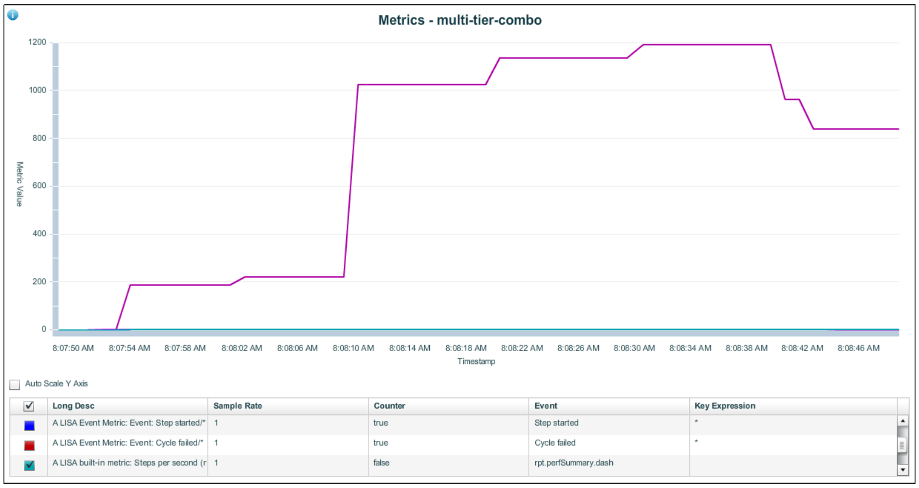
Staging:	QuickStageRun	**Duration:**	6.090 s

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		

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.



☐ Auto Scale Y Axis

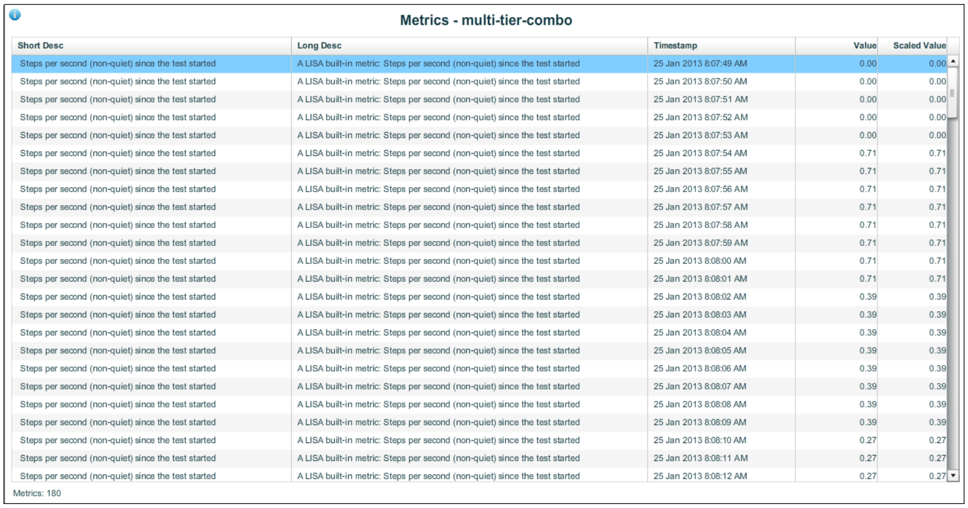
☑	Long Desc	Sample Rate	Counter	Event	Key Expression
🟦	A LISA Event Metric: Event: Step started/*	1	true	Step started	*
🟥	A LISA Event Metric: Event: Cycle failed/*	1	true	Cycle failed	*
☑	A LISA built-in metric: Steps per second (r	1	false	rpt.perfSummary.dash	

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



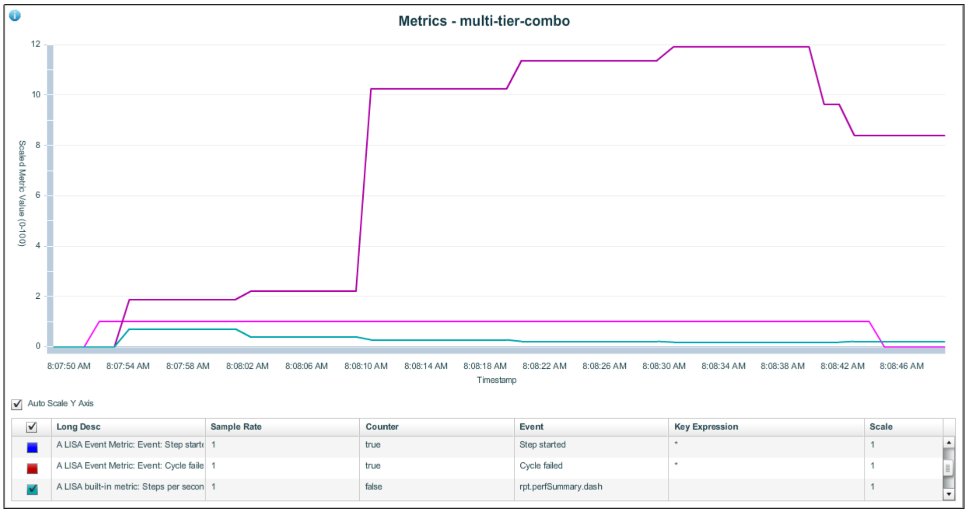
Metrics - multi-tier-combo

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

Metrics: 180

Auto-Scale Y Axis

Compare this report to the same report shown previously to see the auto-scaled Y axis.



☑ Auto Scale Y Axis

☑	Long Desc	Sample Rate	Counter	Event	Key Expression	Scale
🟦	A LISA Event Metric: Event: Step starte	1	true	Step started	*	1
🟥	A LISA Event Metric: Event: Cycle faile	1	true	Cycle failed	*	1
☑	A LISA built-in metric: Steps per secon	1	false	rpt.perfSummary.dash		1

All Metrics

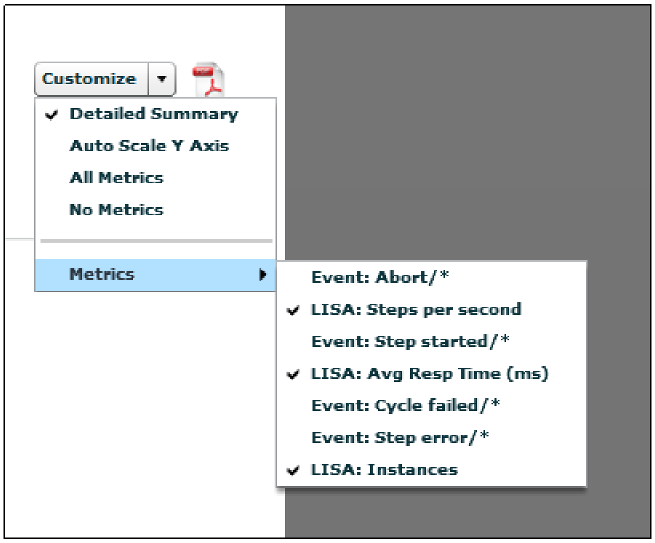
All Metrics is the default selection for the metrics reports. If you have selected No Metrics or if you have selected Metrics in the Customize drop-down, you can select All Metrics to display all metrics available for the report.

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.


Customize
✓ Detailed Summary
Auto Scale Y Axis
All Metrics
No Metrics
Metrics
Event: Abort/*
✓ LISA: Steps per second
Event: Step started/*
✓ LISA: Avg Resp Time (ms)
Event: Cycle failed/*
Event: Step error/*
✓ LISA: Instances

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 377). 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 the Export to PDF icon.

   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.vse.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 clobs 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.

Chapter 18: 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.

This section contains the following topics:

Generate a Web Service (see page 409)
Record a Website (see page 411)
Record a Mobile Test Case (see page 415)

Generate a Web Service

You can create a Web Service (XML) test case. Use the Web Service Execution (XML) step to call web service operations in a test case and test the response and request. These web service operations provide the same functionality as the equivalent method calls in the EJB used in Tutorial 7. For more information about the Web Service Execution step, see Web Service Execution (XML) Step in *Using CA Application Test.*

Ensure you are running the demo server to use this step.

Generating a web service includes the following steps:

1. Create the Web Service (XML) Step (see page 410)
2. Create the Web Service Client (see page 410)
3. Execute the Test Case (see page 410)

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. Click Recording ⚡.
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 topics are available.

- Configure Proxy Settings (see page 413)
- Start Recording (see page 414)
- View Recorded Transactions (see page 414)
- View in ITR (see page 414)

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 200).
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.

   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.

   For more information about adding assertions to the test step, see Add an Assertion to a Mobile Test Step (see page 142).

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 200) 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 269).

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 200) to run.
3. Click the ITR icon ⚙ 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 269).

Chapter 19: Mobile Testing

This section contains the following topics:

Getting Started with Mobile Testing (see page 419)
How to Create and Replay a Mobile Test Case (see page 422)
Remote Testing (see page 422)
Web Browser Testing (see page 425)
The Mobile Lab (see page 426)
Automated Tests by Voyager (see page 428)
Troubleshooting Mobile Test Cases (see page 431)

Getting Started with Mobile Testing

This section contains the following topics:

Overview of Mobile Testing (see page 420)
Supported Mobile Actions and Gestures (see page 421)

Overview of Mobile Testing

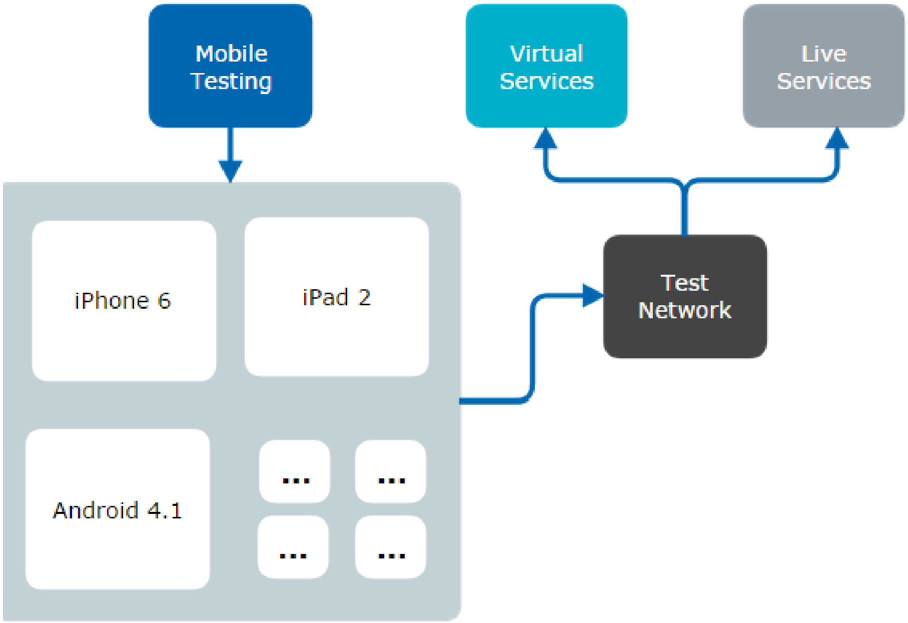
Mobile testing extends basic DevTest functionality for developing, staging, and monitoring tests cases to the testing of mobile iOS, Android, and Hybrid applications. You can easily create test cases by recording interactions with your mobile applications. You can then modify those tests like you would 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:

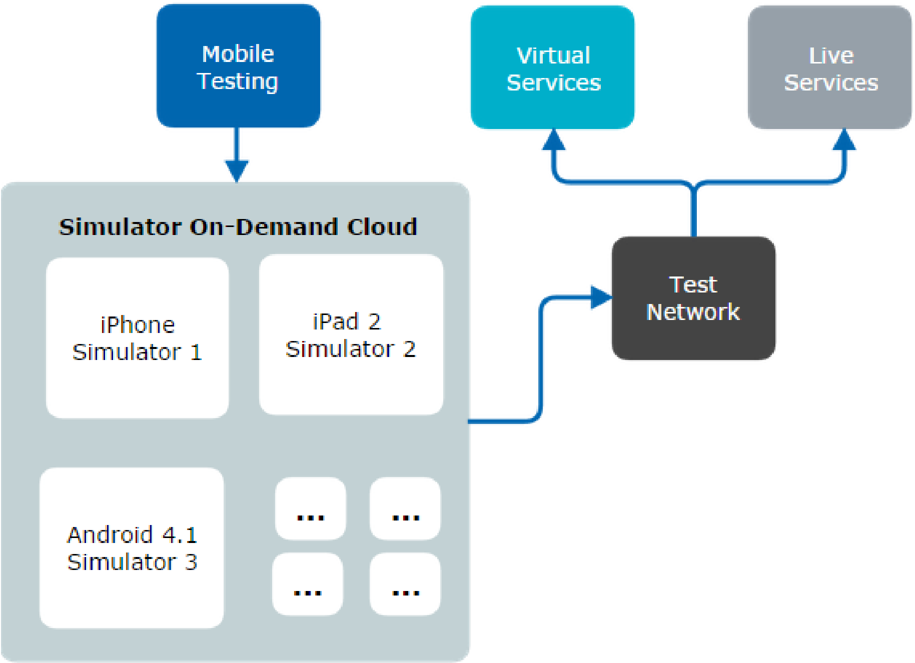
- Record directly from a device that is connected to your machine via the USB port.
- Record using installed Android and iOS simulators.
- Record using cloud-based Android and iOS simulators for a more scalable solution.

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.

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	N/A	N/A	N/A

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:

   SauceLabs Session asset

   Defines the SauceLabs account information for mobile cloud testing.

   Simulator Session asset

   Defines the mobile 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.

   For more information about each of these assets, see Mobile Assets.

   For general information about creating as asset, see Create Assets (see page 111).

2. Record a mobile test case (see page 415).
3. Modify the test steps as appropriate.
4. Run a mobile test case.
   - For general information about running a test case, see Running Test Cases and Suites (see page 269).
   - For specific information about running a mobile test case in the ITR, see Run a Mobile Test Case in the ITR (see page 416).
   - For specific information about running a mobile test case remotely, see Run a Mobile Test Case Remotely (see page 417).

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:

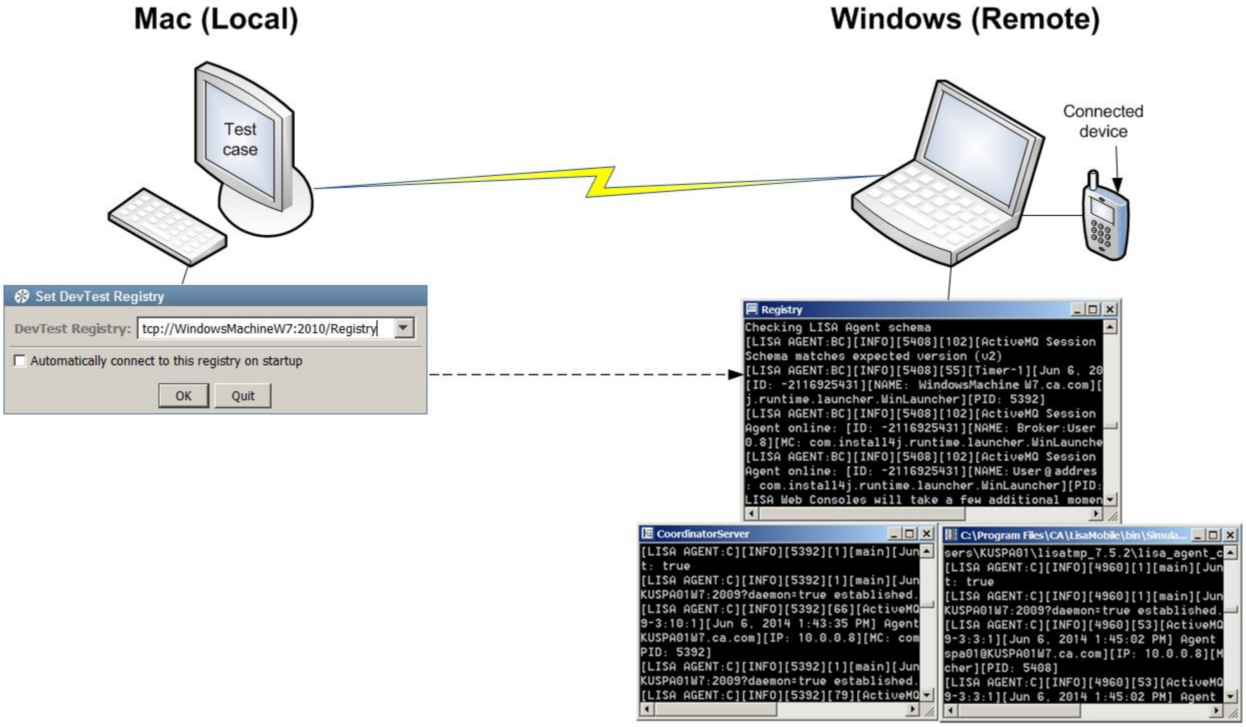
- Mac to Windows
- Windows to Mac
- Windows to Windows
- Mac to Mac

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:



DevTest Registry: tcp://WindowsMachineW7:2010/Registry

□ Automatically connect to this registry on startup

```
Checking LISA Agent schema
[LISA AGENT:BC][INFO][5408][102][ActiveMQ Session
Schema matches expected version (v2)
[LISA AGENT:BC][INFO][5408][55][Timer-1][Jun 6, 20
[ID: -2116925431][NAME: WindowsMachine W7.ca.com][
j.runtime.launcher.WinLauncher][PID: 5392]
[LISA AGENT:BC][INFO][5408][102][ActiveMQ Session
Agent online: [ID: -2116925431][NAME: Broker:User
0.8][MC: com.install4j.runtime.launcher.WinLaunche
[LISA AGENT:BC][INFO][5408][102][ActiveMQ Session
Agent online: [ID: -2116925431][NAME: User@addres
: com.install4j.runtime.launcher.WinLauncher][PID:
LISA Web Consoles will take a few additional momen
```

```
[LISA AGENT:C][INFO][5392][1][main][Jun
t: true
[LISA AGENT:C][INFO][5392][1][main][Jun
KUSPA01W7:2009?daemon=true established.
[LISA AGENT:C][INFO][5392][66][ActiveMQ
9-3:10:1][Jun 6, 2014 1:43:35 PM] Agent
KUSPA01W7.ca.com][IP: 10.0.0.8][MC: com
PID: 5392]
[LISA AGENT:C][INFO][5392][1][main][Jun
KUSPA01W7:2009?daemon=true established.
[LISA AGENT:C][INFO][5392][79][ActiveMQ
```

```
sers\KUSPA01\lisatmp_7.5.2\lisa_agent_c
[LISA AGENT:C][INFO][4960][1][main][Jun
t: true
[LISA AGENT:C][INFO][4960][1][main][Jun
KUSPA01W7:2009?daemon=true established.
[LISA AGENT:C][INFO][4960][53][ActiveMQ
9-3:3:1][Jun 6, 2014 1:45:02 PM] Agent
spa01@KUSPA01W7.ca.com][IP: 10.0.0.8][M
cher][PID: 5408]
[LISA AGENT:C][INFO][4960][53][ActiveMQ
9-3:3:1][Jun 6, 2014 1:45:02 PM] Agent
```

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).

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 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 415) 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 336), 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 346).

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
   ```

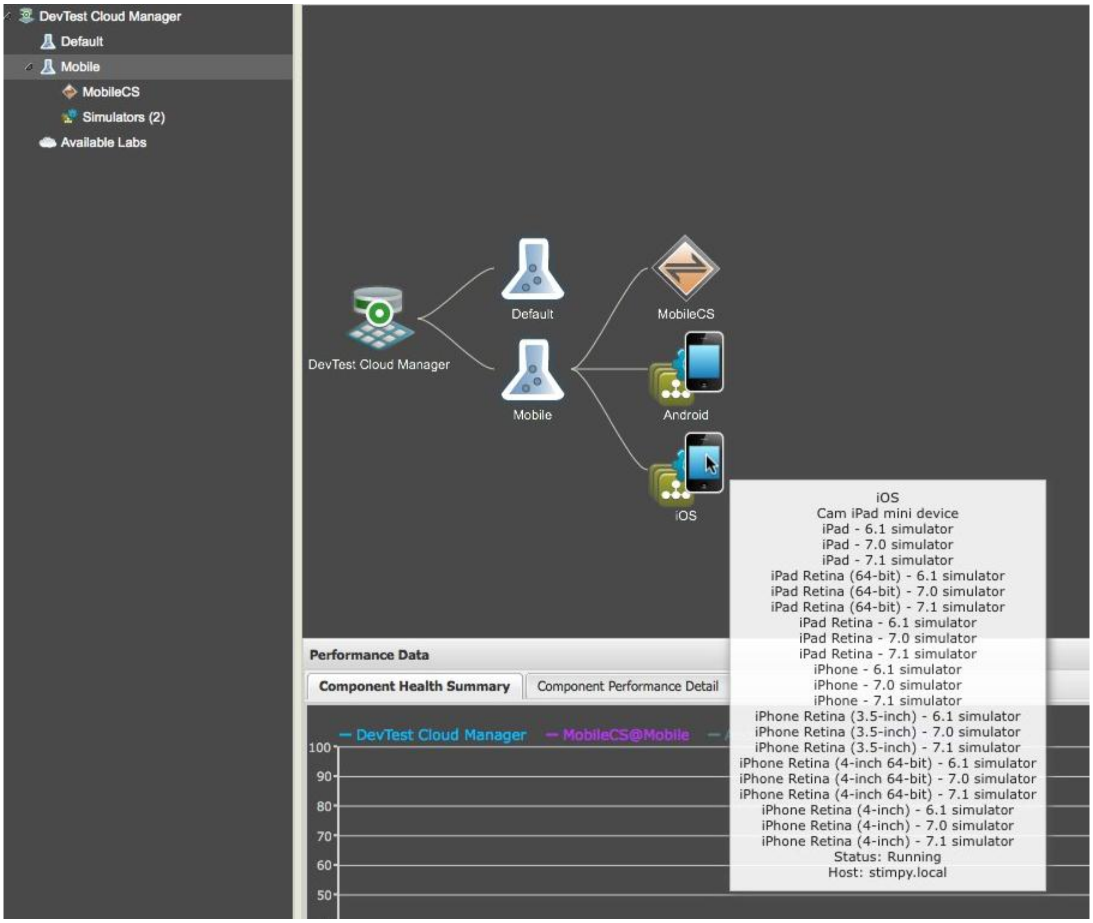
   where *Labname* identifies the user-defined lab name and *LabServerName* is the name of the simulator. One simulator per lab server is recommended.

   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.



iOS
Cam iPad mini device
iPad - 6.1 simulator
iPad - 7.0 simulator
iPad - 7.1 simulator
iPad Retina (64-bit) - 6.1 simulator
iPad Retina (64-bit) - 7.0 simulator
iPad Retina (64-bit) - 7.1 simulator
iPad Retina - 6.1 simulator
iPad Retina - 7.0 simulator
iPad Retina - 7.1 simulator
iPhone - 6.1 simulator
iPhone - 7.0 simulator
iPhone - 7.1 simulator
iPhone Retina (3.5-inch) - 6.1 simulator
iPhone Retina (3.5-inch) - 7.0 simulator
iPhone Retina (3.5-inch) - 7.1 simulator
iPhone Retina (4-inch 64-bit) - 6.1 simulator
iPhone Retina (4-inch 64-bit) - 7.0 simulator
iPhone Retina (4-inch 64-bit) - 7.1 simulator
iPhone Retina (4-inch) - 6.1 simulator
iPhone Retina (4-inch) - 7.0 simulator
iPhone Retina (4-inch) - 7.1 simulator
Status: Running
Host: stimpy.local

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 App field, enter or select the application that you want Voyager to access.
3. 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.

4. 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.

5. In the Output Folder field, enter a location for your .appgraph file, or accept the default location.
6. Click Reconfigure SauceLabs if the SauceLabs account or access key has changed. If SauceLabs is not configured, the SauceLabs dialog opens before starting Voyager.
7. 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.

8. 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 422).
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 Voyager File from an Existing Test Case

You can add mobile test steps to a Voyager file from an existing mobile test case.

Follow these steps:

1. From the project tree in the Project Panel, right-click the .appgraph file that contains your data.
2. Select Merge Test Case.
3. Select the test case that you want to add the new step to.
4. Click OK.

   The test data is merged into the Voyager .appgraph file.

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.

   Note: For this dialog to populate, you must have run Voyager at least once.
3. Select the application file that you have recorded with Voyager.
4. Select the .appgraph file that contains the step you want to add to your test.
5. Click on a recorded page.
6. Verify that the pages were added to the database.

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.

Chapter 20: Advanced Features

This section contains the following topics:

Using BeanShell in DevTest (see page 433)
Class Loader Sandbox Example (see page 438)
In-Container Testing (ICT) (see page 439)
Generating DDLs (see page 444)

Using BeanShell in DevTest

BeanShell (http://www.beanshell.org/) is a free, open source, lightweight Java scripting language. BeanShell is a Java application that uses the Reflection API to execute Java statements and expressions dynamically. By using BeanShell, you avoid the need to compile class files.

BeanShell lets you type standard Java syntax (statements and expressions) on a command line and see the results immediately. A Swing GUI is also available. BeanShell can also be called from a Java class, which is how it is used in the product.

BeanShell is used in several places:

- To interpret property expressions
- As the interpreter framework for the Java Script test step
- As the interpreter framework for the Assert by Script Execution assertion

More information:

Using BeanShell Scripting Language (see page 434)
Using Date Utilities (see page 437)

Using BeanShell Scripting Language

The major difference between BeanShell Java and compiled Java is in the type system. Java is strongly typed, but BeanShell can loosen the typing in its scripting environment. You can, however, impose strict typing in BeanShell if you want.

BeanShell relaxes typing in a natural way that lets you write BeanShell scripts that look like standard Java method code. However, you can also write scripts that look more like a traditional scripting language, such as Perl or JavaScript, while maintaining the Java syntax framework.

If a variable has been typed, then BeanShell honors and checks the type. If a variable is not typed, BeanShell only signals an error if you try to misuse the actual type of the variable.

The following Java fragments are all valid in BeanShell:

```
foo = "Foo";
four = (2+2) * 2 / 2.0;
print(foo + " = " + four);
.
.
hash = new Hashtable();
date = new Date();
hash.put("today", date);
.
.
```

BeanShell lets you declare and then use methods. Arguments and return types can also be loosely typed:

Typed

```
int addTwoNumbers(int a, int b){

return a + b;

}
```

Loosely Typed

```
add (a,b){

return a + b;

}
```

In the second example, the following works correctly:

```
sumI = add (5,7);
sumS = add("DevTest " , "Rocks");
sumM = add ("version ", 2);
```

BeanShell also provides a library of commands that facilitate its use.

A few examples of these commands are:

- **source():** Read a BeanShell (bsh) script.
- **run():** Run a bsh script.
- **exec():** Run a native application.
- **cd()**, **copy()**, and so on: UNIX-like shell commands.
- **print():** Print argument as a string.
- **eval()**: Evaluate the string argument as code.

For more information, see the BeanShell User Guide at http://www.beanshell.org/.

You can also get BeanShell, the source code, and the complete Javadoc at the same place.

Using BeanShell as Stand-alone

BeanShell is available as a stand-alone interpreter so you can try it outside of DevTest. You can download BeanShell from www.beanshell.org. This download is a single small JAR file named bsh-xx.jar (xx is the version number; currently 2.0). Add the JAR file to your classpath.

You can use BeanShell in the following configurations:

- **From a command line:** java bsh.Interpreter [script name] [args]
- **From BeanShell GUI:** java bsh.Console
- **From a Java class:**

```
Import bsh.Interpreter;
.
.
Interpreter I = new Interpreter();
i.set ("x",5);
i.set("today", new Date());
Date d = (Date)i.get("date");
i.eval("myX = x * 10");
System.out.println(i.get("myX"));
```

Using BeanShell in DevTest

The BeanShell interpreter is used in the Java Script Execution step and the Assert by Script Execution assertion. Both of these elements also expose DevTest Java objects and the current DevTest state (properties). This provides a powerful environment for adding custom functionality. The exposed Java objects can be used to both interrogate and to modify the current state of the test. For example, you can read, modify, and create DevTest properties in your scripts.

As a starting point, become familiar with the **TestExec** class in DevTest. Information about TestExec and many other classes can be found in *Using the SDK.*

DevTest also uses BeanShell inside property notation when an equal sign is present. For example:

```
{{= new Date()}}
```

This property expression is interpreted using BeanShell.

Using Date Utilities

The **com.itko.util.DateUtils** class includes a number of date utility functions as static methods. These functions all return the formatted date as a string. You can use these functions in parameter expressions or the Java Script step.

```
com.itko.util.DateUtils.formatDate(Date date, String format)
com.itko.util.DateUtils.formatCurrentDate(String format)
com.itko.util.DateUtils.formatCurrentDate(int offsetInSec, String format)
com.itko.util.DateUtils.rfc3339(Date date)
com.itko.util.DateUtils.rfc3339()
com.itko.util.DateUtils.rfc3339(int offsetInSec)
com.itko.util.DateUtils.samlDate(Date date)
com.itko.util.DateUtils.samlDate()
com.itko.util.DateUtils.samlDate(int offsetInSec)
```

For example, if you have a web service call that takes a formatted date string and the server is 2 minutes slow, you can use:

```
=com.itko.util.DateUtils.formatCurrentDate(-120,"yyyy-MM-dd'T'HH:mm:ss.SSSZ")
```

This generates the string "2007-11-22T13:30:37.545-0500", the current time minus 120 seconds formatted according to these guidelines.

RFC 3339 is slightly different from the date that the default Java date formatter generates. If you need a strict RFC 3339 date, you can use the rcf3339 functions:

```
=com.itko.util.DateUtils.rfc3339()
```

This generates the string "2007-11-22T13:30:37.545-05:00".

SAML dates are formatted using the format "**yyyy-MM-dd'T'HH:mm:ss'Z**'". The samlDate functions are simply helpers so you do not need to remember that format string when using the formatDate APIs.

For more information, see the following:

http://download.oracle.com/javase/1.5.0/docs/api/java/text/SimpleDateFormat.html

http://tools.ietf.org/html/rfc3339#section-5.6

Class Loader Sandbox Example

The following Java class is a simple example of a class that cannot be run in multithreaded or multiuser fashion, because it accesses and modifies a static variable.

```
public class NeedsASandbox \{

static \{

System.out.println("This is my static initializer. You will see
this many times.");

&nbsp;

static String s;

&nbsp;

public NeedsASandbox() \{\};

&nbsp;

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.

In-Container Testing (ICT)

In-Container Testing (ICT) using remote proxies in DevTest is available in the Enterprise JavaBean Execution and Dynamic Java Execution test steps.

This feature allows in-container testing of local EJBs and arbitrary Java objects: any objects that are available in the classpath of the container for the application being tested.

RMI and EJB are both supported as remote object protocols.

In-container testing (ICT) uses two connection modes: EJB and RMI.

This section contains the following topics:

- Access using EJB (J2EE Container Environments) (see page 440)
- Access using RMI (Custom Java Server and Application Environments) (see page 441)
- Testing your ICT Installation from DevTest Workstation (see page 442)

Access using EJB

To test in-container objects in a standard J2EE container, a stateful session Enterprise JavaBean (EJB) is used. This EJB is bundled as an exploded EAR directory named **lisa-remote-object-manager.ear** and a stand-alone JAR file **LISARemoteObjectManagerEJB.jar** with the installer, in LISA_HOME\incontainer\ejb, where LISA_HOME is the directory where DevTest is installed.

This EJB must be deployed with your J2EE application in the J2EE container and be accessible through JNDI using the name "**LISARemoteObjectManagerEJB**". Deploying an EJB varies depending on the J2EE container being used, and vendor-provided documentation may help if there is a problem deploying the ICT EJB.

If your J2EE application uses an isolated classloader, you must incorporate the ICT EJB into your application by modifying the XML deployment descriptors to include the ICT EJB and its dependencies.

JBoss

1. Test that you can successfully deploy the exploded ICT EAR in JBoss by copying the directory: **$LISA\incontainer\ejb\lisa-remote-object-manager.ear to $JBOSS\server\default\deploy** or other appropriate deployment directory.
2. JBoss recognizes the new EAR and deploys it without any errors. Check that you can connect to the EJB from DevTest Workstation.
3. After the EAR is successfully deployed, in a stand-alone configuration, then try to integrate the ICT EJB with your J2EE application. This may be as simple as copying the contents of **$LISA\incontainer\ejb\lisa-remote-object-manager.ear** and including them in your existing application EAR. Or create a new EAR that includes your J2EE application that is combined with these files.

See the application XML deployment descriptor **application.xml** for an idea of how to modify your own application deployment descriptors to include ICT and its dependencies.

The <module> XML elements are used to indicate the presence of the ICT EJB and Java JAR file dependencies.

WebLogic

1. The first test on WebLogic is to deploy the exploded ICT EAR in WebLogic, for example, by using the WebLogic Administrator Console GUI. If your WebLogic Server is in development mode, you can also copy the directory **$LISA\incontainer\ejb\lisa-remote-object-manager.ear** to the **autodeploy** directory on the Server and WebLogic will automatically deploy the EAR. You see that the EAR is deployed without any errors.

2. Check that you can connect to the EJB from DevTest Workstation.
3. After the EAR is successfully deployed in a stand-alone configuration, try to integrate the ICT EJB with your J2EE application. This may be as simple as copying the contents of **$LISA\incontainer\ejb\lisa-remote-object-manager.ear** and including them in your own application EAR. Or create a new EAR that includes your J2EE application combined with these files.

Access using RMI

For custom Java applications, you can modify the source code for your application to integrate with ICT. ICT testing can be performed by binding the **LISARemoteObjectManagerRMIServer** remote Server object to the RMI registry. This object accepts connections from DevTest Workstation to perform ICT.

For example, the following code can be included in your custom application to bind the ICT remote object Server through RMI:

```
LISARemoteObjectManagerRMIServer remoteObjectManagerServer = new
LISARemoteObjectManagerRMIServer();Registry registry =
LocateRegistry.createRegistry(port);registry.bind("LISARemoteObjectManager",
remoteObjectManagerServer);
```

Note: The RMI name "**LISARemoteObjectManager**" must be entered exactly as-is for ICT to work correctly.

DevTest Workstation attempts to connect to your application through the RMI URL **rmi://***hostname***:***port***/LISARemoteObjectManager**.

The hostname and port are variables and can be changed in your test application and configured in DevTest Workstation.

An example server application can be found in $LISA/incontainer/rmi/example. Using a console window, you can run the example server by executing the command "java -jar ExampleServer.jar".

Note: For this command to run successfully, you must be running it from the **$LISA/incontainer/rmi/example** directory. See the sample source code in the **$LISA/incontainer/rmi/example/src** directory.

Testing your ICT Installation

After the server-side portion of ICT is up and running, you should test that you can connect from DevTest Workstation.

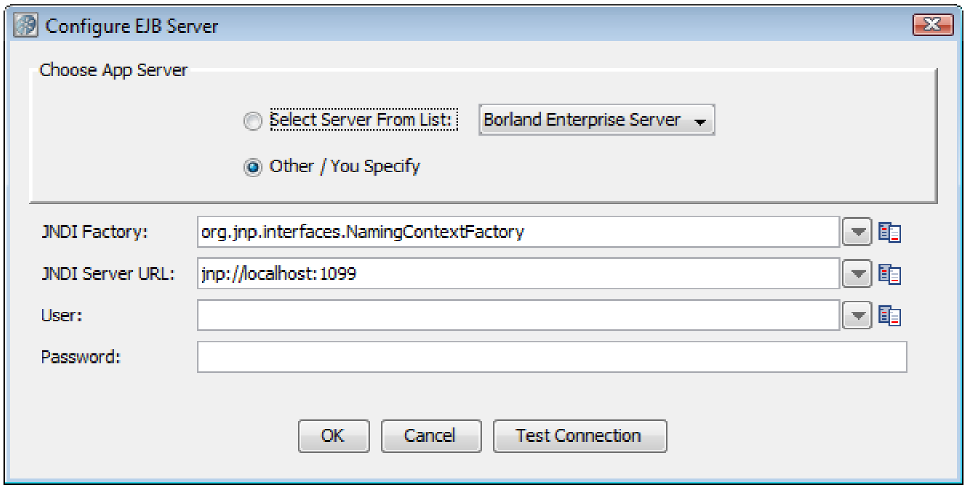
To accomplish this:

1. Open DevTest Workstation.
2. Create a new test case named **ICT Connection Test**.
3. Inside of this test case, create a new Dynamic Java Execution test step.
4. In the editor for the new test step, select the Remote option.
5. Set the Remote Container Type drop-down list to your connection protocol, either EJB or RMI.

Remote Container Type is EJB

If you are using EJB as the connection protocol, click Configure to enter the configuration settings for your protocol.

For example, fill in the Configure EJB Server dialog with host name or IP address and port number of the J2EE container running ICT.



Configure EJB Server

Choose App Server

○ Select Server From List: Borland Enterprise Server

● Other / You Specify

JNDI Factory: org.jnp.interfaces.NamingContextFactory

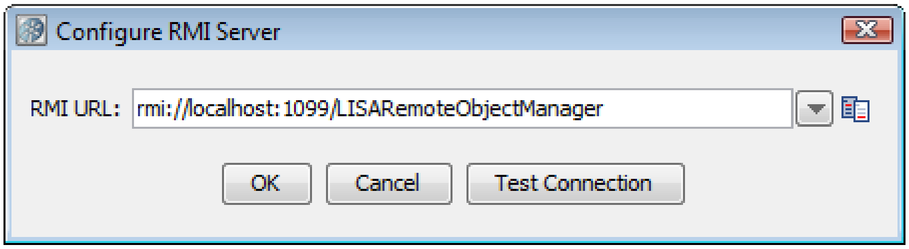
JNDI Server URL: jnp://localhost:1099

User:

Password:

Remote Container Type is RMI

If you are using RMI as the connection protocol, here is an example of the Configure RMI Server dialog that you can use for RMI settings.



RMI URL: rmi://localhost:1099/LISARemoteObjectManager

After you have successfully tested the connection to the ICT Server, enter **java.util.Date** in the Make New Object of Class field and click Construct/Load Object to create a new in-container instance of the java.util.Date class.

Your installation is complete and you are ready to use ICT.

Dependencies

The ICT Server-side classes depend on the following third-party libraries:

- BeanShell 2.0b4 (bsh-2.0b4-lisa-remote-object-manager.jar).

  Note: The original JAR file has been modified so that .bsh scripts are removed. These scripts are known to cause problems with J2EE containers that inspect JAR files and automatically execute them.

- XStream 1.1.2 (xstream-1.1.2.jar)
- Log4j 1.2.13 (log4j-1.2.13.jar)
- Jakarta Commons Logging 1.0.2 (commons-logging.jar)

These dependencies are intentionally distributed as separate files with DevTest ICT JAR files to make ICT integration easier. Because the application that you are testing may already include some or all of these libraries in its classpath, adding these dependencies to the classpath may not be necessary.

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)

Appendix A: Appendix A - 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, as 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 topics:

Comma-Separated List of Paths for Javadoc and Source Code (see page 446)
System Properties (see page 447)
Server Properties (see page 447)
OS X Properties (see page 448)
Update Notifications (see page 448)
Basic Defaults (see page 449)
HTTP Header Keys Properties (see page 451)
HTTP Field Editor Properties (see page 452)
Test Case Execution Parameters (see page 453)
TestEvent Handling Customizations (see page 454)
Test Manager/Editor Properties (see page 455)
J2EE Server Parameters (see page 456)
Native Browser Information to Use for Internal Rendering (see page 458)
Test Manager/Monitor Properties (see page 458)
Built-In String-Generator Patterns (see page 458)
JMX Information (see page 459)
Test Manager/ITR Properties (see page 461)
External Command Shells (see page 462)
Testing Parameters (see page 462)
Properties for Use by StdSchedulerFactory to Create a Quartz Scheduler Instance (see page 463)
VSE Properties (see page 466)
Network Port Properties (see page 476)
CA Continuous Application Insight Properties (see page 478)
Database Properties (see page 483)
Mainframe Properties (see page 485)
Selenium Integration Properties (see page 487)
VSEasy Properties (see page 488)

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

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.testcase=test.xml

Default when running a test from the com.itko.lisa.test.TestCase class directly.

lisa.registry=registry.xml

Default when running a test from the com.itko.lisa.test.TestCase class directly.

lisa.runName=Ad-hoc Run

Default when running a test from the com.itko.lisa.test.TestCase class directly.

lisa.registryName=registry

Default name of the registry to attach to, and the default name of the registry when you start it without a name.

lisa.coordName=coordinator

The coordinator server default name when started without an explicit name AND when the Test Runner needs one and you do not specify a coordinator server on the command line.

lisa.simulatorName=simulator

The default name of a simulator daemon if you do not provide one on the command line.

lisa.vseName=VSE

The virtual environment server default name when started without an explicit name.

lisa.defaultRegistry.pulseInterval=30

Status log interval for the registry. The default value is 30 seconds.

lisa.coordinator.pulseInterval=30

The status log interval for the coordinator. The default value is 30 seconds.

lisa.simulator.pulseInterval=30

The status log interval for the simulator. The default value is 30 seconds.

lisa.vse.pulseInterval=30

The status log interval for VSE. The default value is 30 seconds.

lisa.server.projectmap.refresh.pulseInterval=600

The time interval after which the DevTest servers (coordinator and simulator) refresh the map of project names to file paths.

lisa.defaultRegistryConnectionTimeoutSeconds=90

Timeout value in seconds that coordinators and simulators use when connecting to a DevTest registry. A value of 0 indicates an infinite timeout; that is, we wait forever trying to connect.

lisa.regex.helper.tutorial.url=http://download.oracle.com/javase/tutorial/essential/regex/

For the Regex helper window, the URL to show for the regular expression tutorial.

lisa.hooks=com.itko.lisa.files.SampleHook

To register hooks with DevTest; these values are comma-separated.

lisa.project.ignore=CVS|SCCS|RCS|rcs|\.DS_Store|\.svn|vssver\.scc|vssver2\.scc|\.sbas|\.IJI\..*|.*\.pyc|.*\.pyo|\.git|.*\.hprof|_svn|\.hg|.*\.lib|.*~|__pycache__|\.bundle|.*\.rbc

This property is a Java regular expression that lets you hide the types of files that you do not want to see in the Project tree. By default, a variety of well-known control files for third-party products are hidden. To show files that are hidden by default, modify the regular expression.

HTTP Header Keys Properties

The default values for header keys in the HTTP support are in the following list. To get the benefit of the change, remove or change them for all tests. To change them for a test, or even the execution of one HTTP transaction, use TestNode-specific header directives.

ice.browser.http.agent

Values: compatible, MSIE 6.0, Windows NT 5.0

Default: Mozilla/4.0

lisa.http.header.0.key

Default: Pragma

lisa.http.header.0.value

Default: no-cache

lisa.http.header.1.key

Default: Cache-Control

lisa.http.header.1.value

Default: no-cache

lisa.http.header.0.key

Default: Accept

lisa.http.header.0.value

Default: image/gif, image/x-xbitmap, image/jpeg

lisa.http.header.1.key

Default: Accept-Language

lisa.http.header.1.value

Default: en

lisa.http.header.2.key

Default: Accept-Charset

lisa.http.header.2.value

Default: iso-8859-1,*,utf-8

lisa.http.header.3.key

Default: User-Agent

lisa.http.header.3.value

Default: Mozilla/4.0, MSIE 6.0; Windows NT 5.0

HTTP Field Editor Properties

The following list shows defaults for fields that appear in the HTTP Field editor. Do not include "Authentication" because the editor adds it automatically.

lisa.gui.http.fieldNames

Values: Accept,Accept-Language,User-Agent,Connection

lisa.webrecorder.textMIMEs

Values: html,text,magnus-internal,application/pdf

lisa.webrecorder.notTextMIMEs

Values: css,script

lisa.webrecorder.alwaysIgnore

Values: .gif,.jpg,.jpeg,.css,.js,.ico

lisa.web.ntlm

Enables NTLM authentication in the Test Runner.

Default: true

Test Case Execution Parameters

lisa.hotDeploy=C\Projects\Lisa\custom_classes

This setting tells the ClassLoader built into DevTest where to look for custom classes. The default is $LISA_HOME\hotDeploy.

lisa.overloadThreshold=1000

The TestNode attempts to determine if the simulator is thrashing by checking the actual amount of time slept in think time as opposed to the amount of think time it was supposed to take. This setting is the amount of additional "slip" in think time that is acceptable before sending a warning TestEvent that the simulator is overloaded. The default of 1000 means that if the simulator sleeps 1 second more than it was supposed to, (presumably because the computer is CPU starved), then raise the TestEvent.

lisa.webservices.encode.empty.xmlns=true

Some web service server stacks require empty xmlns strings in the SOAP request (for example, jbossWs). Others, such as Amazon, do not work with empty xmlns strings. Change this property to suit the stack you are calling.

lisa.webservices.encode.version=1.1

Set the encoding version to force for client stub generation. The default is 1.1.

lisa.tm.sys.min.millis=0

lisa.tm.sys.max.millis=0

The default think time for new system steps, in milliseconds. The system steps include subprocesses, continue, continue (quiet), fail and end.

lisa.numFilters.warning=100

lisa.numAsserts.warning=100

Sometimes, filters and assertions are dynamically added to test steps. In a load test, this could mean 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 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 have to be root to use that.

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 drives the formatting of messages to the System Messages window.

Default: %-5p - %m%n

lisa.editor.http.recorderPort

The HTTP recorder binds to this port.

Default: 8010

lisa.editor.proxy.webProxySupport

Default: on

J2EE Server Parameters

lisa.prefill.jndiNames=	JBOSS=org.jnp.interfaces.NamingContextFactory
	Weblogic=weblogic.jndi.WLInitialContextFactory
	Websphere=com.ibm.websphere.naming.WsnInitialContextFactory
	Borland Enterprise Server=com.inprise.j2ee.jndi.CtxFactory
	iPlanet/Sun AS=com.sun.jndi.cosnaming.CNCtxFactory

lisa.prefill.jndiUrlPrefix=	JBOSS=jnp://
	Weblogic=t3://
	Websphere=iiop://
	Borland Enterprise Server=iiop://
	iPlanet/Sun AS=iiop://

lisa.prefill.jndiDefPort=	JBOSS=1099
	Weblogic=7001
	Websphere=2809
	Borland Enterprise Server=1099
	iPlanet/Sun AS=1099

lisa.prefill.jndiNeedsClass=	JBOSS=false
	Weblogic=false
	Websphere=true
	Borland Enterprise Server=true
	iPlanet/Sun AS=true

lisa.prefill.jndiFactories=	org.jnp.interfaces.NamingContextFactory weblogic.jndi.WLInitialContextFactory com.ibm.websphere.naming.WsnInitialContextFactory com.webmethods.jms.naming.WmJmsNamingCtxFactory com.tibco.tibjms.naming.TibjmsInitialContextFactory com.inprise.j2ee.jndi.CtxFactory com.sun.jndi.cosnaming.CNCtxFactory fiorano.jms.runtime.naming.FioranoInitialContextFactory
lisa.prefill.jndiServerURLs=	jnp://SERVER:1099&t3://SERVER:7001 iiop://SERVER:PORT tibjmsnaming://SERVER:7222&iiop://SERVER:PORT iiop://localhost:9010&wmjmsnaming://Broker #1@SERVER:PORT/JmsAdminTest
lisa.editor.URLTransEditor.protos=	http,https
lisa.editor.URLTransEditor.hosts=	
lisa.editor.URLTransEditor.ports=	80,443
lisa.editor.URLTransEditor.files=	

lisa.prefill.jdbc.names=	Oracle SQL Server WLS Oracle JDataStore Sybase DB2 MySQL Derby
lisa.prefill.jdbc.jdbcDrivers=	oracle.jdbc.driver.OracleDriver com.microsoft.sqlserver.jdbc.SQLServerDriver com.microsoft.jdbc.sqlserver.SQLServerDriver weblogic.jdbc.oci.Driver com.borland.datastore.jdbc.DataStoreDriver com.sybase.jdbc.SybDriver com.ibm.db2.jcc.DB2Driver org.gjt.mm.mysql.Driver org.apache.derby.jdbc.ClientDriver
lisa.prefill.jdbc.jdbcConnectionURLs=	jdbc:oracle:thin:@SERVER:1521:SIDNAME jdbc:sqlserver://SERVER:PORT;databasename=DBNAME jdbc:microsoft:sqlserver://SERVER:PORT jdbc:weblogic:oracle:TNSNAME jdbc:borland:dslocal:DBNAME jdbc:sybase:Tds:SERVER:PORT/DBNAME jdbc:db2://SERVER:PORT/DBNAME jdbc:mysql://SERVER:PORT/DBNAME jdbc:derby://DBSERVER:DBPORT/DBNAME

Native Browser Information to Use for Internal Rendering

lisa.internal.browser.on=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.wsdls=http://localhost:8080/itko-examples/services/UserControlService?wsdl

Test Manager/Monitor Properties

monitor.events.maxrows

The maximum number of event rows that are kept in the Test Manager as a test is run. The number of objects that are kept is twice this number.

Default: 500

lisa.tm.sysmess.size

The system messages window maximum size. The memory that is consumed is twice this value.

Default: 10240

Built-In String-Generator Patterns

lisa.patterns.stringgenerator.types=&Phone=(DDD)DDD-DDDD, &SSN=DDD-DD-DDDD, &Date=Lll-DD-DDDD, &Zip=D*(5)

JMX Information

lisa.jmx.types	com.itko.lisa.stats.jmx.JSE5Connection com.itko.lisa.stats.jmx.TomcatConnection com.itko.lisa.stats.jmx.JBossConnection com.itko.lisa.stats.jmx.JSR160RMIConnection com.itko.lisa.stats.jmx.WeblogicConnector com.itko.lisa.stats.jmx.Weblogic9Connector com.itko.lisa.stats.jmx.WebsphereSOAPConnection com.itko.lisa.stats.jmx.ITKOAgentConnection com.itko.lisa.stats.jmx.OracleASConnector

lisa.jmx.typeprops	com.itko.lisa.stats.jmx.JSE5Connection=LISA_JMX_JSE5
	com.itko.lisa.stats.jmx.TomcatConnection=LISA_JMX_TOMCAT5
	com.itko.lisa.stats.jmx.JBossConnection=LISA_JMX_JBOSS3240
	com.itko.lisa.stats.jmx.JSR160RMIConnection=LISA_JMX_JSR160RMI
	com.itko.lisa.stats.jmx.Weblogic9Connector=LISA_JMX_WLS9
	com.itko.lisa.stats.jmx.WeblogicConnector=LISA_JMX_WLS6781
	com.itko.lisa.stats.jmx.OracleASConnector=LISA_JMX_OC4J
	com.itko.lisa.stats.jmx.WebsphereSOAPConnection=LISA_JMX_WASSOAP5X
	com.itko.lisa.stats.jmx.ITKOAgentConnection=LISA_JMX_ITKOAGENT

You do not generally need anything for these parameters.

LISA_JMX_JSR160RMI

Default: LISA_HOME/lib/mx4j.lib

LISA_JMX_ITKOAGENT

Default: LISA_HOME/lib/mx4j.lib

LISA_JMX_JSE5

If you are running JSE 5, you do not need to change this property. We ship a jbossall-client that includes what you need, assuming the version is right.

LISA_JMX_JBOSS3240

Default: LISA_HOME/lib/mx4j.lib{{path.separator}}LISA_HOME/hotDeploy/jbossall-client.jar

LISA_JMX_TOMCAT5

This parameter is for Tomcat.

Default: LISA_HOME/lib/mx4j.lib{{path.separator}}LISA_HOME/hotDeploy/mx4j-tools.jar

LISA_JMX_WLS9

Default: LISA_HOME/hotDeploy/weblogic.jar{{path.separator}}LISA_HOME/hotDeploy/wljmxclient.jar

LISA_JMX_WLS6781

This parameter needs to 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

Runs the test case once with one simulated user

com.itko.lisa.files.1user1cycle0think.stg

Runs the test case once with one simulated user and no think time

com.itko.lisa.files.1user1min.stg

Runs 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 shut down 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), 10 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 or RAML file. If there are more than *lisa.vse.rest.max.optionalqueryparams* optional query parameters specified in a WADL or RAML file, only the first *lisa.vse.rest.max.optionalqueryparams* are used to create transactions.

Default: 5

Date-Checker Properties

The VSE date utilities use the following properties to determine which date patterns are considered valid for date sensitivity conversions. Each of the following entries represents a regular expression that VSE considers as a part of a date.

lisa.vse.datechecker.dayregex

((\[12\]\\d)\|(3\[01\])\|(0?\[1-9\]))

lisa.vse.datechecker.monthnumberregex

((1\[012\])\|(0 \\d)\|0\[1-9\]\|\[1-9\])

lisa.vse.datechecker.monthalpharegex

(\\bJAN\\b\|\\bFEB\\b\|\\bMAR\\b\|\\bAPR\\b\|\\bMAY\\b\|\\bJUN\\b\|\\bJUL\\b\|\\bAUG\\b\|\\bSEP\\b\|\\bOCT\\b\|\\bNOV\\b\|\\bDEC \\b)

lisa.vse.datechecker.yearlongregex

\\d\\d\\d\\d

lisa.vse.datechecker.yearshortregex

\\d\\d

lisa.vse.datechecker.timeregex

(\\s?((\[012\]? \\d)\|(2\[0123\]))\:((\[012345\] \\d)\|(60))\:((\[012345\] \\d)\|(60)))

lisa.vse.datechecker.time.hhmmssregex

((\[012\]? \\d)\|(2\[0123\]))\:((\[012345\] \\d)\|(60))\:((\[012345\] \\ d)\|(60))

lisa.vse.datechecker.time.millisregex

((\[012\]?\\d)\|(2\[0123\]))\:((\[012345\]\\d)\|(60))\:((\[012345\]\\d)\|(60))\\.((\\d \\d \\d)\|0)

lisa.vse.datechecker.time.millis.zoneregex

((\[012\]?\\d)\|(2\[0123\]))\:((\[012345\] \\d)\|(60))\:((\[012345\] \\ d)\|(60)) \\.((\\d \\d \\d)\|0) \\s(\[A-Za-z\]\[A-Za-z\]\[A-Za-z\])

lisa.vse.datechecker.wstimestampregex

\\d\\d\\d\\d-((1\[012\])\|(0\\d)\|0\[1-9\]\|\[1-9\])-((\[12\]-\\-d)\|(3\[01\])\|(0?\[1-9\]))T((\[012\]?-\\-d)\|(2\[0123\]))\:((\[012345\]-\\-d)\|(60))\:((\[012345\]-\\-d)\|(60))-\\-.-\\-d-\\-d-\\-d\[-+\]\\d\\d\\d \\d

lisa.vse.datechecker.ddmmmyyyyregex

((\[12\]\\d)\|(3\[01\])\|(0?\[1-9\]))(JAN\|FEB\|MAR\|APR\|MAY\|JUN\|JUL\|AUG\|SEP\|OCT\|NOV\|DEC)\\d \\d \\d \\d

lisa.vse.datechecker.mmmddyyyyregex

(JAN\|FEB\|MAR\|APR\|MAY\|JUN\|JUL\|AUG\|SEP\|OCT\|NOV\|DEC)((\[12\]\\d)\|(3\[01\])\|(0?\[1-9\]))\\d \\d \\d \\d

lisa.vse.datechecker.yyyyddmmmregex

```
\\d\\d \\d \\d((\[12\]
\\d)\|(3\[01\])\|(0?\[1-9\]))(JAN\|FEB\|MAR\|APR\|MAY\|JUN\|JUL\|AUG\|S
EP\|OCT\|NOV\|DEC)
```

lisa.vse.datechecker.yyyymmmddregex

```
\\d\\d\\d\\d(JAN\|FEB\|MAR\|APR\|MAY\|JUN\|JUL\|AUG\|SEP\|OCT\|NOV
\|DEC)((\[12\] \\d)\|(3\[01\])\|(0?\[1-9\]))
```

lisa.vse.datechecker.ddmmmregex

```
((\[12\]\\d)\|(3\[01\])\|(0?\[1-9\]))(JAN\|FEB\|MAR\|APR\|MAY\|JUN\|JUL\|
AUG\|SEP\|OCT\|NOV\|DEC)
```

lisa.vse.datechecker.mmmddregex

```
(JAN\|FEB\|MAR\|APR\|MAY\|JUN\|JUL\|AUG\|SEP\|OCT\|NOV\|DEC)((\[12
\]\\d)\|(3\[01])|(0?\[1-9\]))
```

lisa.vse.datechecker.ddmmmyyregex

```
((\[12\]\\d)\|(3\[01\])\|(0?\[1-9\]))(JAN\|FEB\|MAR\|APR\|MAY\|JUN\|JUL\|
AUG\|SEP\|OCT\|NOV\|DEC)\\d\\d
```

In VSE, each of the following entries represents a valid pattern for part of date:

lisa.vse.datechecker.dayformat

Format: dd

lisa.vse.datechecker.monthnumberformat

Format: MM

lisa.vse.datechecker.monthalphaformat

Format: MMM

lisa.vse.datechecker.yearlongformat

Format: yyyy

lisa.vse.datechecker.yearshortformat

Format: yy

lisa.vse.datechecker.timeformat

Format: HH:mm:ss

lisa.vse.datechecker.time.hhmmssformat

Format: HH:mm:ss

lisa.vse.datechecker.time.millisformat

Format: HH:mm:ss.SSS

lisa.vse.datechecker.time.millis.zoneformat

Format: HH:mm:ss.SSS z

lisa.vse.datechecker.wstimestampformat

Format: yyyy-MM-dd'T'HH:mm:ss.SSSZ

The following date patterns cannot be constructed by using a combination for the previous patterns that involve at least day, month, and year:

lisa.vse.datechecker.mmmddyyyy.separatorformat

Format: MMM*dd*yyyy

lisa.vse.datechecker.mmddyyyy.separatorformat

Format: MM*dd*yyyy

lisa.vse.datechecker.ddmmmyyyy.separatorformat

Format: dd*MMM*yyyy

lisa.vse.datechecker.ddmmyyyy.separatorformat

Format: dd*MM*yyyy

lisa.vse.datechecker.yyyymmmdd.separatorformat

Format: yyyy*MMM*dd

lisa.vse.datechecker.yyyymmdd.separatorformat

Format: yyyy*MM*dd

lisa.vse.datechecker.ddmmmyyyyformat

Format: ddMMMyyyy

lisa.vse.datechecker.mmmddyyyyformat

MMMddyyyy

lisa.vse.datechecker.yyyyddmmmformat

Format: yyyyddMMM

lisa.vse.datechecker.yyyymmmddformat

Format: yyyyMMMdd

lisa.vse.datechecker.ddmmmyyformat

Format: ddMMMyy

lisa.vse.datechecker.ddmmmformat

Format: ddMMM

lisa.vse.datechecker.mmmddformat

Format: MMMdd

lisa.vse.datechecker.separators

Defines the valid separator characters to use in the date formats.

Values: -/.

Example: lisa.vse.datechecker.separators=/ sets '/' as a separator as in 10/15/2011.

lisa.vse.datechecker.top.priorityorder=lisa.vse.datechecker.wstimestampformat

Defines the order in which to match the date pattern. The separator characters that **lisa.vse.datechecker.separators** defines replace the asterisks.

Example: MM*dd*yy generates four date patterns: "MM-dd-yyyy", "MM dd yyyy", "MM/dd/yyyy", "MM.dd.yyyy".

lisa.vse.datechecker.date.priorityorder

```
lisa.vse.datechecker.mmmddyyyy.separatorformat&\
lisa.vse.datechecker.mmddyyyy.separatorformat&\
lisa.vse.datechecker.ddmmmyyyy.separatorformat&\
lisa.vse.datechecker.ddmmyyyy.separatorformat&\
lisa.vse.datechecker.yyyymmmdd.separatorformat&\
lisa.vse.datechecker.yyyymmdd.separatorformat&\
lisa.vse.datechecker.mmmddyyyyformat&\
lisa.vse.datechecker.ddmmmyyyyformat&\
lisa.vse.datechecker.yyyyddmmmformat&\
lisa.vse.datechecker.yyyymmmddformat&\
lisa.vse.datechecker.ddmmmyyformat
```

lisa.vse.datechecker.time.priorityorder

```
lisa.vse.datechecker.time.millis.zoneformat&\
lisa.vse.datechecker.time.millisformat&\
lisa.vse.datechecker.time.tenthsformat&\
lisa.vse.datechecker.time.hhmmssformat
```

lisa.vse.datechecker.bottom.priorityorder

```
lisa.vse.datechecker.mmmddformat&\
lisa.vse.datechecker.ddmmmformat
```

lisa.vse.datechecker.months

Values: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC

lisa.vse.datechecker.coarse.year.regex

Format:
(?ism).*((\\d[./-](19|20)[0-9]{2})|((JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC)[. /-](19|20)[0-9]{2})|((19|20)[0-9]{2}[. /-][0-9]{2})).*

Defines a coarse level test to determine whether the payload contains anything that resembles a year. To avoid matching too much, it looks nearby for other identifying information such as month information and common date separators (for example, - or / or .).

When the year is difficult to distinguish from any other four-digit number, this pattern may not match. In that case, your magic dates are not found. Try one of the following solutions:

- Tweak this RegEx
- Try the more permissive form that follows

lisa.vse.datechecker.coarse.year.regex

Format:
(?ism).*((\\d[./-]?(19|20)[0-9]{2})|((JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|
OCT|NOV|DEC)[. /-]?(19|20)[0-9]{2})|((19|20)[0-9]{2}[. /-]?[0-9]{2})).*

Defines a more permissive coarse level test than the previous test to determine whether the payload contains anything that resembles a year. If the dates are not recognized, this Regex may allow them to be found. To avoid matching too much, it looks nearby for other identifying information such as month information and common date separators (for example, - or / or .).

However, this Regex can have some performance impact. If this Regex matches too many dates, the patterns above run in situations where they are not applicable. Running large numbers of Regex patterns against large payloads can be time consuming.

VSE JMS Messaging - Custom JMS Properties to Ignore

vse.jms.ignore.proplist

Some JMS platforms include extra custom properties in the JMS messages they deliver. These properties can interfere with VSE creation. **vse.jms.ignore.proplist** contains a list of properties to ignore when recording a JMS service with the VSE messaging recorder. The format is a comma-delimited list of regular expressions. By default, it excludes standard JMS extension properties (JMSX) and JBoss custom bookkeeping properties (JMS_.).

Default: JMSX.*, JMS_.*

VSE Recorder Conversation Batch Size

lisa.vse.recorder.conversation.batch.size

Defines how many conversations we process before committing them to the database. If you have a few large conversations, make this number small. If you have many smaller conversations, make this number large. A batch size of <= 0 is the same as a batch size of 1.

Default: 10

VSE Service Image Database Settings

eclipselink.ddl-generation

Default: create-tables

eclipselink.ddl-generation.output-mode=database

Defines the database to use as the repository for service images. VSE uses the open source EclipseLink JPA provider. By default we use our reports database as the repository for service images.

Default: LISA reports database

Validate the VSE Schema

lisa.eclipselink.query.warn.threshold

If EclipseLink is configured to do query timings (the default), this value defines the maximum interval (in milliseconds) allowed for a database query before the **com.itko.lisa.vse.stateful.model.SqlTimer logger** raises a WARN-level message. If you set the threshold on that logger to DEBUG, timings return for all SELECT statements EclipseLink issues. This logging is the first step in debugging VSE performance issues. If the database is local and all the indexes are created, then you should not see anything over about 20 ms. However, a database on the network that is not under your control and possibly missing indexes could easily take 100ms or more to return a result. In that case, VSE can seem to be slow when in fact it is fast if correctly deployed. Most service images are small enough or have a working set sufficiently small enough to fit into the VSE entity cache. Therefore, the number of SELECTs that VSE issues should dwindle to zero, provided the VSE has a large enough heap. The cache maintains soft references by default.

Default: 100

VSE Delimited Content Detection Properties

lisa.vse.delimited.delimiter.check.maxchars

Defines the maximum number of characters to inspect to determine the content type.

Default: 1000

lisa.vse.delimited.delimiter.check.maxpercent

Defines the maximum percentage of characters to inspect to determine the content type. This property is used if it is higher than **lisa.vse.delimited.delimiter.check.maxchars**.

Default: 2

lisa.vse.delimited.delimiter.list

Defines the characters that the Delimited Content Detector looks for as delimiters.

Default: :,\\,,;,|,\u00A6,\t,\n

lisa.vse.delimited.delimiter.minimum.threshold

Defines the minimum number of delimiters that are counted for the Delimited Content Detector. The payload must have at least this many delimiters.

Default: 3

lisa.vse.delimited.namevalue.separator.list

Defines the Name-Value separators that are used to detect the delimited name-value payload. These values must not be in the **lisa.vse.delimited.delimiter.list**.

REST Data Protocol Properties

lisa.protocol.rest.editor.observedtraffic.max

Defines the maximum number of items of matching traffic that are populated in the lower pane of the URI rules wizard. If the wizard takes a long time to populate the lower pane with matching traffic, try using a lower value in this field.

Default: 100

lisa.protocol.rest.editor.unmatchedtraffic.max

Defines the maximum number of items of unmatched traffic that display in the unmatched traffic dialog. If the wizard takes a long time to populate this wizard, try using a lower value in this field.

Default: 100

lisa.protocol.rest.idPattern

Defines for the REST data protocol a regular expression that detects the parts of an HTTP request that it can consider to be an identifier. The data protocol converts such identifiers to dynamic parameters. When you have dynamic parameters that you know follow a specific format, you can use this property to detect them.

Default: [a-zA-Z]+[0-9]{5,}[a-zA-Z]*

Example: The value **id12345** can denote a stock item and the subsequent rule contains {URLPARAM0}.

lisa.protocol.rest.maxChanges

Defines the maximum number of changes that VSE allows for a token before the variability is considered significant enough to generate a rule.

Default: 1

lisa.protocol.rest.parameterBaseName

Defines the prefix that the REST data protocol uses for the parameters in rules.

Default: URLPARAM

lisa.protocol.rest.startPosition

Defines the position in the URL at which the REST data protocol starts looking for variable tokens.

Default: 3

Example: In the URI **GET /a/b/c/d**, **GET** is at position 0, **a** is at position 1, and **b** is at position 2.

vse.log.trace.truncate.response.at=2048

Limits the size of responses that DevTest writes to the vse.log. This property is only applied when trace-level logging is enabled: specifically, when **com.itko.lisa.vse.http.Transaction**=TRACE or when **com.itko.lisa.vse.stateful.protocol.http.Coordinator**=TRACE.

Values:

- **0**: The entire response is logged.
- **>0**: The logged response is truncated at the character specified.

Default: 2048

Network Port Properties

The network port properties take the form of **lisa.net.APPID.port**.

If you must change these defaults, override them in the **site.properties** file. Clients make assumptions about what port to connect to their server based on these values.

lisa.net.0.port

DevTest Workstation

Default: 2008

lisa.net.2.port

Coordinator

Default: 2011

lisa.net.3.port

Registry

Default: 2010

lisa.net.4.port

JUnit exec

Default: 2012

lisa.net.5.port

Test Runner (command line)

Default: 2005

lisa.net.6.port

Others

Default: 2007

lisa.net.8.port

VSE

Default: 2013

lisa.net.9.port

VSEManager

Default: 2004

lisa.net.11.port

ServiceManager

Default: 2006

lisa.net.1.port

Simulator (we start here so it is easier to have many on the same computer).

Default: 2014

CA Continuous Application Insight Properties

lisa.pathfinder.on

Default: true

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 of the portal app launchers from DevTest Workstation. Currently, the TR host name is used for the Host value so we get that value dynamically. The URL is launched externally in your system's browser.

lisa.portal.url.prefix

Default: http://* (http://*)

lisa.portal.root.base.url

Default: /index.html

lisa.portal.cvsdashboard.base.url

Default: /index.html?lisaPortal=cvsdashboard

lisa.portal.pathfinder.console.base.url

Default: /index.html?lisaPortal=pathfinder

lisa.portal.server.console.base.url

Default: /index.html?lisaPortal=serverconsole

lisa.portal.model.execution.url

Default: /index.html?lisaPortal=serverconsole

lisa.portal.reporting.context

Default: reporting

lisa.portal.reporting.console.base.url

Default: /index.html?lisaPortal=reporting

lisa.portal.defect.capture.url

Default: /pathfinder/lisa_pathfinder_agent/defectcapture

lisa.portal.save.defect.data.url

Default: /pathfinder/lisa_pathfinder_agent/saveDefectData

lisa.portal.invoke.base.url

Default: /lisa-invoke

lisa.portal.invoke.report.url

Default: /reports

lisa.portal.invoke.server.report.directory

Default: lisa.tmpdirlisa.portal.invoke.report.url

devtest.portal.base.url.path

Default: /devtest/#/main

devtest.portal.homepage.url.path

Default: {{devtest.portal.base.url.path}}/dashboard

lisa.portal.pathfinder.explorer.base.url

Default: {{devtest.portal.base.url.path}}/createartifacts

lisa.portal.pathfinder.management.base.url

Default: {{devtest.portal.base.url.path}}/pathManageAgents

lisa.portal.invoke.test.root

Default: LISA_HOME

Reporting Graph View Properties

rpt.lisa.graph.view.threshhold

Default: 100

rpt.lisa.graph.view.infomessage

Results more than maximum threshold. Modify a filter to fetch the details.

rpt.lisa.graph.scatter.view.threshhold

Default: 10000

rpt.lisa.graph.scatter.view.infomessage

Results more than maximum threshold. To fetch the details, modify a filter.

Async Reporting Support

lisa.reporting.useAsync

If you run load tests and you find that the bottleneck in the test cases is writing the events to the reporting database, consider removing everything except metrics from the report generator. If you still see the reporting engine as the bottleneck, consider enabling this property. The property uses JMS to send the reporting event and background threads in the simulators and coordinator write the events to the database asynchronously. This means that your load test finishes before all the events are written to the database, so the report does not appear for some time (how long depends on your test cases and how many events they generate). The simulator queue typically takes the longest to flush; you get a message at the INFO level in the simulator log showing the percentage complete.

This feature is considered as "advanced usage" for now and is disabled by default. Feedback at http://ca.com/support is welcome and encouraged.

Default: false

lisa.reporting.step.max.propsused.buffersize

Default: 100

lisa.reporting.step.max.propsset.buffersize

These properties control the collection of statistics for a test step during the execution of a test case. The values specify the maximum number of occurrences that are recorded for the properties that are used and properties that are set. You should not need to change the default values.

Default: 100

lisa.threadDump.generate

Default: true

lisa.threadDump.interval

Default: 30

lisa.threadDump.loggerName

Enable periodic thread dumps. It is a good idea to leave this enabled. It efficiently checks to see if the threadDumpLogger is at INFO or below and not do anything if it is set to WARN or higher. Properties are only read once at startup; the l**ogging.properties** file is checked every 10 seconds for changes. All you must do is leave this 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.vse.poolName

Default: common

lisadb.acl.poolName

Default: common

lisadb.broker.poolName

Default: common

lisadb.pool.common.driverClass

Default: org.apache.derby.jdbc.ClientDriver

lisadb.pool.common.url

Default: jdbc:derby://localhost:1528/database/lisa.db;create=true

lisadb.pool.common.user

Default: rpt

lisadb.pool.common.password_enc

Set the password by removing the trailing _enc from the property name and adding *=MyPlaintextPassword*. The password is automatically encoded at startup.

Default: 76f271db3661fd50082e68d4b953fbee

The following pool properties keep the number of connections to a minimum when DevTest is idle.

lisadb.pool.common.minPoolSize

Default: 0

lisadb.pool.common.minPoolSize

Default: 0

lisadb.pool.common.maxPoolSize

Default: 10

lisadb.pool.common.acquireIncrement

Default: 1

lisadb.pool.common.maxIdleTime

Default: 45

lisadb.pool.common.idleConnectionTestPeriod

Default: 5

Other common database settings: copy and paste the relevant user, password, and pool size parameters from the common template.

lisadb.pool.POOLNAME.driverClass

Default: oracle.jdbc.OracleDriver

lisadb.pool.POOLNAME.url

Default: jdbc:oracle:thin:@HOST:1521:SID

lisadb.pool.POOLNAME.driverClass

Default: com.ibm.db2.jcc.DB2Driver

lisadb.pool.POOLNAME.url

Default: jdbc:db2://HOST:50000/DBNAME

lisadb.pool.POOLNAME.driverClass

Default: com.microsoft.sqlserver.jdbc.SQLServerDriver

lisadb.pool.POOLNAME.url

Default: jdbc:sqlserver://durry;databaseName=LISA

lisadb.pool.POOLNAME.driverClass

Default: com.mysql.jdbc.Driver

lisadb.pool.POOLNAME.url

Default: jdbc:mysql://HOST:3306/DBNAME

lisa.jdbc.asset.pool.size

When defining a JDBC Connection asset, the connection pool size can be configured by this property.

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 in *Agents.*

WebSphere MQ Properties

lisa.mq.ccsid.default=819

You can use this property to override the default Coded Character Set Identifier (CCSID) for all IBM WebSphere MQ messages sent from DevTest. This value can also be overridden for each case in the Publisher Info's Message Properties of the **IBM WebSphere MQ** step. For more information, see *Using CA Application Test.* For the complete list of CCSIDs, see http://www-01.ibm.com/software/globalization/ccsid/ccsid_registered.html. The default value for United States locales is 819 (ASCII).

Localization Options

lisa.locale.languages=en

You can use this property to indicate the languages to support in the UI. The valid values are **en** and **ja**, indicating English or Japanese.

If you specify both **en** and **ja** here, you can switch the UI between English and Japanese by using the System, Language option off the Main menu.

lisa.supported.html.request.encodings=

Valid values are ISO-8859-1,UTF-8,Shift_JIS,EUC-JP,Windows-31J.

Selenium Integration Properties

selenium.enable.waitfor=true

Specifies whether to add an implicit waitForElementPresent step for each step in the test case that must locate a web element.

Values:

- **True:** Adds the implicit step that executes before the real step to ensure that the element exists and that the page has loaded.
- **False:** Does not add the implicit step. If you set this property to false, scenarios can occur where the script runs successfully in Selenium Builder but fails in DevTest because of delays in page loading. If you have already defined waitForElementPresent steps in Selenium Builder, you can set this property to false.

selenium.browser.type

Defines the type of browser for running a Selenium Integration test case. Use this property in a project configuration file.

Values:

- Chrome
- IE
- Firefox

Note: If this property is not specified, the test runs in Firefox by default.

selenium.ie.driver.path

Defines the full path to the Selenium driver on a local computer that is used to run a Selenium Integration test in a Microsoft Internet Explorer browser. Use this property in a project configuration file.

Example:

```
C:\lisa-se\IEDriverServer.exe
```

selenium.chrome.driver.path

Defines the full path to the Selenium driver on a local computer that is used to run a Selenium Integration test in a Chrome browser. Use this property in a project configuration file.

Example:

```
C:\lisa-se\ChromeDriverServer.exe
```

selenium.remote.url

Defines the URL of a remote Selenium Server hub to run Selenium Integration test cases on a remote browser. Use this property in a project configuration file.

Example:

```
http://your_remote_hostname:4444/wd/hub
```

Note: You can define **selenium.chrome.driver.path** and **selenium.ie.driver.path** in the same project configuration file. A configuration file that contains **selenium.chrome.driver.path** or **selenium.ie.driver.path** cannot also contain **selenium.remote.url**.

selenium.WebDriver.DesiredCapabilities.filePath

Specifies the location of the parameter file used to set advanced options in the Selenium web driver.

Default: {{LISA_PROJ_ROOT}}/Data/selenium-capabilities.conf

VSEasy Properties

The port properties control the dynamic assignment of ports when using auto configuration with the HTTP protocol in VSEasy.

If the minimum port is less than 1024, the value is set to 1024. If the maximum port is greater than 65535, the value is set to 65535. If the maximum port is less than or equal to the minimum port, both values are reverted to the defaults.

lisa.vseasy.http.min.dynamic.port

Default: 8000

lisa.vseasy.http.max.dynamic.port

Default: 65535

lisa.vseasy.default.group.tag

Specifies the name of the default group in the VSEasy console.

Default: VSEasy

Appendix B: 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).

This section contains the following topics:

Local Properties File (see page 490)
Site Properties File (see page 515)
logging.properties (see page 519)

Local Properties File

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.conversational.back.navigation.allowed

VSE supports unrecorded back navigation during playback in a conversational transaction. Set this property to *true* to enable back navigation for all services. The conversational session then tracks received requests and allows any next request to be one of the previous ones.

Default: false

lisa.vse.tracking.delete.data.age

Defines how long to keep session tracking information, in hours.

Default: 8

lisa.vse.protocol.ims.response.includes.llll

Indicates the existence of the 4-byte IMS Connect LLLL length field in the request payload. A value of true includes the LLLL field. A value of false excludes it.

Default: false

lisa.vse.ims.connect.llzz.request

Indicates the existence of the 4-byte IMS Connect LLZZ length field in the request payload. A value of **true** includes the LLZZ field. A value of **false** excludes it.

Default: false

lisa.vse.ims.connect.llzz.response

Instructs the protocol to generate the 4-byte IMS Connect LLZZ length field in the response messages. A value of **true** creates the LLZZ field. A value of **false** does not create it.

Default: true

Enterprise Dashboard Properties

The Enterprise Dashboard database is used for the DevTest 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".

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\webreckeys.ks**). For a stronger certificate, specify the **lisa.net.keyStore** property and the plaintext password in **lisa.net.keyStore.password**.

The next time DevTest starts, an encrypted string (**lisa.net.keyStore.password_enc**) replaces the plaintext password.

lisa.net.keyStore

Default: {{LISA_HOME}}lisa.ks

lisa.net.keyStore.password

This is where we keep our identification certificate. If we are a server installation, this is the certificate that clients 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.trustStore

Default: {{LISA_HOME}}lisa.ts

lisa.net.trustStore.password_enc

Default: 079f6a3d304a978146e547802ed3f3a4

This is where we keep trusted certificates. If we are primarily a client installation, this holds a list of trusted servers. If we are a server installation and we are doing mutual (client) authentication, this is where we put trusted client certificates.

lisa.net.clientAuth

Indicates whether to use mutual authentication.

Default: false

lisa.net.default.protocol

Values: SSL or TCP

Default: SSL

Default: ssl

License Properties if Using an HTTP Proxy Server

laf.usehttpproxy.server

Default: true

laf.httpproxy.server

Defines the proxy server, in the format *my_proxyserver.com*.

laf.httpproxy.port

If your proxy server requires credentials, leave blank to use the native NTLM authentication.

Default: 3128

laf.httpproxy.domain

Defines the proxy domain, if needed for NTLM.

laf.httpproxy.username

Optional

laf.httpproxy.password

Optional

laf.email.alert

If there is a license error, send an email. Refer to the following example. Replace *mailhost* with the hostname of your mail host. Replace *recipient@mycompany.com* with the target email address. Replace *from@mycompany.com* with the email address from which the email is sent.

Example: *mailhost,recipient@mycompany.com,from@mycompany.com*

SDK Properties

The DevTest SDK examples require:

asserts

Format: com.mycompany.lisa.AssertFileStartsWith

com.mycompany.lisa.AssertFileStartsWith

Format:
com.itko.lisa.editor.DefaultAssertController,com.itko.lisa.editor.DefaultAssertEditor

filters

Format: com.mycompany.lisa.FilterFileFirstLine

com.mycompany.lisa.FilterFileFirstLine

Format: com.itko.lisa.editor.FilterController,com.itko.lisa.editor.DefaultFilterEditor

nodes

Format: com.mycompany.lisa.node.FTPTestNode

com.mycompany.lisa.node.FTPTestNode

Format:
com.mycompany.lisa.node.FTPTestNodeController,com.mycompany.lisa.node.FTPTestNodeEditor

SSL Properties

Change the default behavior for validating the SSL certificates.

ssl.checkexpiry

A value of "true" says to validate the validity dates for the certificate.

Default: false

ssl.checkcrl

A value of "true" says to validate the cert revocation list that is specified in the certificate.

Default: false

Enable a client cert and password for SSL (used by HTTP step 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.

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=NOREFERENCES

NOREFERENCES means a simple tree is rendered. Circular references are not allowed. XPATHREFERENCES means the XPATH notation is used for references. Circular references can be used. If there is a circular reference, we fall back to XPATHREFERENCES. However, any serialization before version 3.6 that is re-read fails and could require you to set a default of XPATHREFERENCES.

Workstation Management

lisa.ui.admin.tr.control=no

lisa.ws.jms.SoapAction.quoted=false

> SOAP over JMS with TIBCO BusinessWorks/Active Matrix: BW/AM requires that the SOAPAction header is quoted.

When DevTest serializes a date, time, or dateTime, it uses the following formats by default. To change the default formats/time zone, use these properties. You can also dynamically set them during the test case. You can set the format to one that does not comply with the XML schema specification. However, doing so is not recommended (except for a negative test case).

lisa.ws.ser.dateFormat=yyyy-MM-dd

lisa.ws.ser.timeFormat=HH:mm:ss.SSS'Z'

lisa.ws.ser.timeFormat.timeZone=GMT

lisa.ws.ser.dateTimeFormat=yyyy-MM-dd'T'HH:mm:ss.SSS'Z'

lisa.ws.ser.dateTimeFormat.timeZone=GMT

ws.raw.format=true

> To format the SOAP response for RAW Soap Steps, set this property to **true. T**his formatting can also be done dynamically at run time.

lisa.ws.endpoint.fullautoprop=false

> The entire WS Endpoint URL is automatically converted into a single DevTest property. To convert the URL to use the WSSERVER and the WSPORT properties, set this property to **false**.

stats.unix.xml.folder={ {LISA_HOME } }/umetrics

IP Spoofing

lisa.ipspoofing.interfaces=2-34-56-78-90-AB, eth0, Realtek PCIe GBE Family Controller

Interfaces: a comma-separated list of interfaces that is used for the IP spoofing. These interfaces can be named using the MAC address (JDK 1.6+), interface name, or interface display name.

lisa.ui.useNativeFileDialog=true

Force the use of a native file dialog.

Quick Start Recent Items Properties

lisa.prefill.recent

Specify the maximum number of recent items to show in the Prefill combo boxes (no less than 10).

Default: 10

lisa.quickstart.recent

Specify the maximum number of recent items to show in the Open Recent Quick Start tab (no less than 5).

Default: 5

lisa-invoke Properties

lisa.portal.invoke.base.url

Default: /lisa-invoke

lisa.portal.invoke.report.url

Default: /reports

lisa.portal.invoke.server.report.directory

Default: lisa.tmpdirlisa.portal.invoke.report.url

lisa.portal.invoke.test.root

Default: d:/lisatests/

Server Host Name Properties

lisa.net.externalIP

This value must be the external IP address or DNS name for the registry to be accessible remotely. This parameter is referenced in the relevant connection properties. If the value is left at "localhost," it is automatically overridden in the external IP address of the registry server when sent to connecting applications.

Default: localhost

DevTest to DevTest Communication Encryption Properties

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\webreckeys.ks is a default internal self-signed certificate. For a stronger certificate, specify the **lisa.net.keyStore** property and the plaintext password in **lisa.net.keyStore.password**. The next time DevTest starts, an encrypted string **lisa.net.keyStore.password_enc** replaces the plain text password.

lisa.default.keystore={{LISA_HOME}}webreckeys.ks

lisa.default.keystore.pass=passphrase

> Where the identification certificate is kept. For a server installation, this value is the certificate to add to the list of trusted servers. For a client installation doing mutual (client) authentication, this value is the certificate to add to the truststore on the server side. For a client installation that is not using mutual authentication, you do not need to specify this parameter.

lisa.net.keyStore={{LISA_HOME}}lisa.ks

lisa.net.keyStore.password=PlainTextPasswordWilBeConvertedToEncrypted

> Where trusted certificates are kept. For a primarily client installation, this field holds a list of trusted servers. For a server installation doing mutual (client) authentication, this field is where to put trusted client certificates.

lisa.net.trustStore={{LISA_HOME}}lisa.ts

lisa.net.trustStore.password_enc=079f6a3d304a978146e547802ed3f3a4

> Whether or not to use mutual authentication.

lisa.net.clientAuth=false

> Defines the default protocol for ActiveMQ connections. The default is **tcp**.

lisa.net.default.protocol=tcp

DevTest Agent and CAI Properties

lisa.pathfinder.on

If you are experiencing agent errors and you are not licensed for the DevTest agent, consider turning it off here.

Default: false

IBM WebSphere MQ Properties

The string-value is supplied by a data set such as the Unique Code Generator.

lisa.mq.correlation.id

A run-time value that sets the correlation ID for an outgoing MQ message.

Default: string-value

lisa.mq.correlation.id.bytes

A run-time value that sets the correlation ID as a byte[] for nonprintable values.

Default: byte[]

lisa.mq.message.id

A run-time value that sets the message ID for an outgoing MQ message.

Default: string-value

lisa.mq.message.id.bytes

A run-time value that sets the message ID as a byte[] for nonprintable values.

Default: byte[]

JMS Properties

The string-value is often supplied by a data set such as the Unique Code Generator.

lisa.jms.correlation.id

A run-time value that sets the JMSCorrelationID for an outgoing JMS message.

Value: string-value

lisa.jms.ttl.milliseconds

A run-time value that sets the time-to-live for an outgoing JMS message.

Value: num-value

jms.always.close.jndi

A value of true auses 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

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

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.

user.session.check.interval.seconds=60

Site Properties File

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=tcp://somehost:2003/EnterpriseDashboard

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.

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.vse.poolName

Default: common

lisadb.acl.poolName

Default: common

lisadb.broker.poolName

Default: common

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. If you configured all the previous components to use an external database, you can save resources by setting this property to *false*.

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]

lisab.pool.common.user

Default: none

lisadb.pool.common.password

Default: none

MySql Properties

lisadb.pool.common.driverClass

Default: com.mysql.jdbc.Driver

lisadb.pool.common.url

Default: jdbc:mysql://[DBHOST]:[DBPORT]/[DBNAME]

lisadb.pool.common.user

Default: none

lisadb.pool.common.password

Default: none

Derby Properties

lisadb.pool.common.driverClass

Default: org.apache.derby.jdbc.ClientDriver

lisadb.pool.common.url

Default: jdbc:derby://[SERVER]:[PORT]/[DATABASE];create=true

lisadb.pool.common.user

Default: none

lisadb.pool.common.password

Default: none

Miscellaneous Properties

lisa.pathfinder.on

Default: true

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.appender.A1

Default: com.itko.util.log4j.TimedRollingFileAppender

log4j.appender.A1.File

Default: ${lisa.tmpdir}/${LISA_LOG}

log4j.appender.A1.MaxFileSize

Default: 10MB

log4j.appender.A1.MaxBackupIndex

Default: 5

log4j.appender.A1.layout

Default: org.apache.log4j.EnhancedPatternLayout

log4j.appender.A1.layout.ConversionPattern

Default: %d{ISO8601}{UTC}Z (%d{HH:mm}) [%t] %-5p %-30c - %m%n

log4j.logger.VSE

Keep a separate log for VSE transaction match/no-match events, Having a separate log makes debugging much easier.

Change INFO to WARN for production systems. The logging can slow down systems with high transaction rates. Do not simply comment out the following line. Explicitly set the log level to OFF or WARN instead of INFO.

Default: INFO, VSEAPP

log4j.additivity.VSE

To add VSE logging to other log destinations, comment out this line.

Default: false

log4j.appender.VSEAPP

Default: com.itko.util.log4j.TimedRollingFileAppender

log4j.appender.VSEAPP.File

Default: ${lisa.tmpdir}/vse_matches.log

log4j.appender.VSEAPP.MaxFileSize

Default: 10MB

log4j.appender.VSEAPP.MaxBackupIndex

Default: 20

log4j.appender.VSEAPP.layout

Default: org.apache.log4j.EnhancedPatternLayout

log4j.appender.VSEAPP.layout.ConversionPattern

Default: %d{ISO8601}{UTC}Z (%d{HH:mm})[%t] %-5p - %m%n

Keep a separate log for advisory events. This logger warns of potential configuration issues such as potential memory leaks. The log is deliberately kept separate from the application log to minimize noise.

log4j.logger.ADVICE

Default: INFO, ADVICE_APP

log4j.additivity.ADVICE

Default: false

log4j.appender.ADVICE_APP

Default: org.apache.log4j.RollingFileAppender

log4j.appender.ADVICE_APP.File

Default: ${lisa.tmpdir}/advice.log

log4j.appender.ADVICE_APP.MaxFileSize

Default: 10MB

log4j.appender.ADVICE_APP.MaxBackupIndex

Default: 20

log4j.appender.ADVICE_APP.layout

Default: org.apache.log4j.EnhancedPatternLayout

log4j.appender.ADVICE_APP.layout.ConversionPattern

Default: %d{ISO8601}{UTC}Z (%d{HH:mm}) %-5p - %m%n

If enabled, periodic thread dumps are sent here. DevTest writes logs at the INFO level. Therefore, to get the thread dumps, change WARN in the next line to INFO, even with DevTest servers or DevTest Workstation running. This action results in a thread dump in the named file in 30 seconds. For more information, search for "threadDump" in **lisa.properties**. This action also simplifies getting thread dumps to debug performance issues. Change WARN in the next line to INFO, wait for 1 minute or 2 minutes, then revert the setting to WARN.

You can also generate a point-in-time thread dump with the LISA_HOME/bin/ServiceManager application. For example, use standard Java tools such as jstack, or issue the following command:

```
ServiceManager -threadDump tcp://hostname:2014/Simulator
```

log4j.logger.threadDumpLogger

Default: WARN, THREAD_DUMPS

log4j.additivity.threadDumpLogger

Default: false

log4j.appender.THREAD_DUMPS

Default: org.apache.log4j.RollingFileAppender

log4j.appender.THREAD_DUMPS.File

Default: ${lisa.tmpdir}/threadDumps/TD_${LISA_LOG}

log4j.appender.THREAD_DUMPS.MaxFileSize

Default: 10MB

log4j.appender.THREAD_DUMPS.MaxBackupIndex

Default: 20

log4j.appender.THREAD_DUMPS.layout

Default: org.apache.log4j.EnhancedPatternLayout

log4j.appender.THREAD_DUMPS.layout.ConversionPattern

Default: %d{ISO8601}{UTC}Z (%d{HH:mm}) [%t] %-5p - %m%n

log4j.appender.registry

Mirrors DevTest logging to the (remote) registry.

Default: com.itko.lisa.net.LoggingToRegistryAppender

log4j.appender.registry.layout

Default: org.apache.log4j.EnhancedPatternLayout

log4j.appender.registry.layout.ConversionPattern

Default: %d{ISO8601}{UTC}Z [%t] %-5p - %m%n

The following properties display the requests and responses for HTTP-based traffic in the vse.log. This information is useful for debugging a virtual service that returns a "no match found" response. Such a result can indicate that the VS did not receive the expected request.

These log messages reveal the same requests and response that TCPMON shows. The requests and responses display in the vse.log, without having to inject TCPMON between the virtual service and its client application. These options only log HTTP requests and responses. These options can help to debug virtual services, especially when a virtual service responds with an unexpected message and you want to match a raw request with the raw response.

log4j.logger.com.itko.lisa.vse.http.Transaction

Prints the requests that travel through this class in the vse.log. These log messages begin with "Raw playback response."

Values: TRACE, null

DEFAULT: null

log4j.logger.com.itko.lisa.vse.stateful.protocol.http.Coordinator

Prints the requests that travel through this class in the vse.log. These log messages begin with "Raw request start."

Values: TRACE, null

DEFAULT: null

log4j.logger.com.itko.lisa.vse.stateful.protocol.http.HttpListenStep

Prints the requests that travel through this class in the vse.log. These log messages begin with "Raw request start."

Values: TRACE, null

DEFAULT: null

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 129) in *Using CA Application Test.*

asset

An *asset* is a set of configuration properties that are grouped into a logical unit. For more information, see Assets (see page 106) in *Using CA Application Test*.

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 237) in *Using CA Application Test.*

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 155) in *Using CA Application Test.*

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 100) in *Using CA Application Test.*

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 353) in *Using CA Application Test.*

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.*

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 13) in *Using CA Application Test.*

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 in *Using CA Service Virtualization.*

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 145) in *Using CA Application Test.*

desensitize

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 Desensitizing Data in *Using CA Service Virtualization.*

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 239) in *Using CA Application Test.*

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 114) in *Using CA Application Test.*

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 270) in *Using CA Application Test.*

lab

A *lab* is a logical container for one or more lab members. For more information, see Labs and Lab Members (see page 334) in *Using CA Application Test.*

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 in *Using CA Service Virtualization.*

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 in *Using CA Service Virtualization.*

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 in *Using CA Service Virtualization.*

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 243) in *Using CA Application Test.*

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 255) in *Using CA Application Test.*

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 255) in *Using CA Application Test.*

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 in *Using CA Service Virtualization.*

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 346) in *Using CA Application Test*.

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.*

path graph

A *path graph* contains a graphical representation of a path and its frames. For more information, see Path Graph in *Using CA Continuous Application Insight.*

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 46) in *Using CA Application Test.*

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 86) in *Using CA Application Test.*

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 281) in *Using CA Application Test.*

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 11) in *Using CA Application Test.*

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 in *Using CA Service Virtualization.*

simulator

A *simulator* runs the tests under the supervision of the coordinator server. For more information, see Simulator Server (see page 15) in *Using CA Application Test.*

staging document

A *staging document* contains information about how to run a test case. For more information, see Building Staging Documents (see page 215) in *Using CA Application Test.*

subprocess

A *subprocess* is a test case that another test case calls. For more information, see Building Subprocesses (see page 209) in *Using CA Application Test.*

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 73) in *Using CA Application Test.*

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 188) in *Using CA Application Test.*

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 243) in *Using CA Application Test.*

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 190) and Staging Document Editor (see page 217) - Base Tab in *Using CA Application Test.*

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 in *Using CA Continuous Application Insight.*

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.*

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) in *Using CA Service Virtualization.*