

# DevTest Solutions

Reference

Version 8.0



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.

Copyright © 2014 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

# 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](mailto: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: Test Case Reference	9
Assertion Descriptions .....	9
HTTP Assertions .....	9
Database Assertions.....	17
XML Assertions.....	19
JSON Assertions.....	32
Virtual Service Environment Assertion .....	36
Mobile Assertions .....	37
Other Assertions .....	40
Asset Descriptions.....	57
JDBC Connection Assets .....	58
JMS Client Assets.....	60
JNDI Initial Context Asset .....	63
SAP JCo Destination Assets .....	64
Email Connection Asset.....	67
Mobile Assets.....	69
Companion Descriptions .....	71
Web Browser Simulation Companion .....	72
Browser Bandwidth Simulation Companion .....	73
HTTP Connection Pool Companion .....	74
Configure DevTest to Use a Web Proxy Companion .....	76
Set Up a Synchronization Point Companion .....	77
Set Up an Aggregate Step Companion .....	78
Observed System VSE Companion .....	79
VSE Think Scale Companion .....	88
Batch Response Think Time Companion.....	90
Recurring Period Think Time Companion .....	92
Create a Sandbox Class Loader for Each Test Companion .....	93
Set Final Step to Execute Companion .....	94
Negative Testing Companion .....	94
Fail Test Case Companion .....	94
XML Diff Ignored Nodes Companion.....	95
Data Set Descriptions .....	95
Read Rows from a Delimited Data File Data Set .....	96
Create Your Own Data Sheet Data Set.....	98
Create Your Own Set of Large Data Data Set .....	101
Read Rows from a JDBC Table Data Set .....	103

---

Create a Numeric Counting Data Set .....	104
Read Rows from Excel File Data Set .....	105
Read DTOs from Excel File Data Set .....	107
Unique Code Generator Data Set .....	113
Random Code Generator Data Set .....	114
Message-Correlation ID Generator Data Set .....	115
Load a Set of File Names Data Set .....	116
XML Data Set .....	118
Filter Descriptions .....	122
Utility Filters .....	123
Database Filters .....	130
Messaging/ESB Filters .....	137
HTTP/HTML Filters .....	140
XML Filters .....	158
JSON Filters .....	165
Java Filters .....	167
VSE Filters .....	169
CAI Filters .....	170
Copybook Filters .....	172
Test Step Descriptions .....	174
Test Step Information .....	174
Web-Web Services Steps .....	175
Java-J2EE Steps .....	238
Other Transaction Steps .....	251
Utilities Steps .....	260
External-Subprocess Steps .....	276
JMS Messaging Steps .....	285
BEA Steps .....	308
Sun JCAPS Steps .....	316
Oracle Steps .....	320
TIBCO Steps .....	333
Sonic Steps .....	342
webMethods Steps .....	345
IBM Steps .....	356
SAP Steps .....	362
Selenium Integration Steps .....	372
LISA Virtual Service Environment Steps .....	379
CAI Steps .....	379
Mobile Steps .....	382
Custom Extension Steps .....	385

---

Chapter 2: Test Document Reference	391
Events.....	391
Metrics .....	397
DevTest Whole Test Metrics .....	397
DevTest Test Event Metrics.....	398
SNMP Metrics .....	400
JMX Metrics.....	402
TIBCO Hawk Metrics.....	405
Windows Perfmon Metrics .....	407
UNIX Metrics Via SSH .....	409
Glossary	411





# Chapter 1: Test Case Reference

---

This section contains the following topics:

[Assertion Descriptions](#) (see page 9)  
[Asset Descriptions](#) (see page 57)  
[Companion Descriptions](#) (see page 71)  
[Data Set Descriptions](#) (see page 95)  
[Filter Descriptions](#) (see page 122)  
[Test Step Descriptions](#) (see page 174)

## Assertion Descriptions

This section describes each of the assertions that are available in DevTest.

Regular expressions are used for comparison purposes in many assertions. For more information about regular expressions, see [Regular Expressions](#).

**This section contains descriptions of the following assertions:**

[HTTP Assertions](#) (see page 9)  
[Database Assertions](#) (see page 17)  
[XML Assertions](#) (see page 19)  
[JSON Assertions](#) (see page 32)  
[Virtual Service Environment Assertion](#) (see page 36)  
[Mobile Assertions](#) (see page 37)  
[Other Assertions](#) (see page 40)

## HTTP Assertions

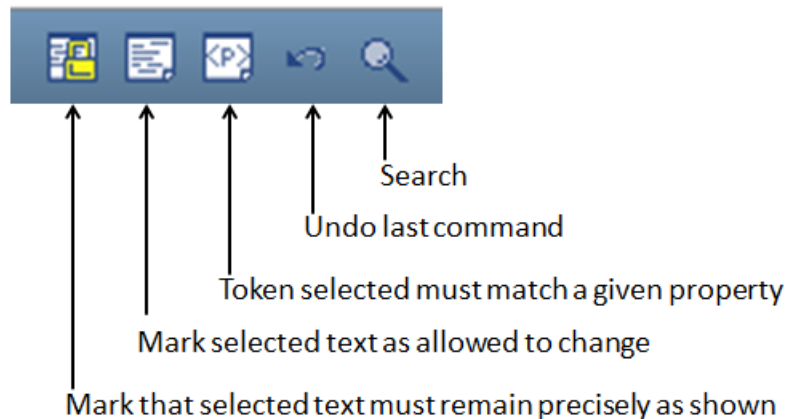
**The following assertions are available in the HTTP assertions list for any test step.**

[Highlight HTML Content for Comparison](#) (see page 10)  
[Check HTML for Properties in Page](#) (see page 12)  
[Ensure HTTP Header Contains Expression](#) (see page 14)  
[Check HTTP Response Code](#) (see page 15)  
[Simple Web Assertion](#) (see page 16)  
[Check Links on Web Responses](#) (see page 17)

## Highlight HTML Content for Comparison

The Highlight HTML Content for Comparison assertion lets you base a comparison on the contents of an HTML page. This assertion uses the "paint the screen" technique that is designed to work with HTML pages. For example, if there is a large HTML document, then you can identify the data before and after the "content of interest". Then, you simply identify what to compare the "content of interest" against (typically an expected value that is supplied in a data set).

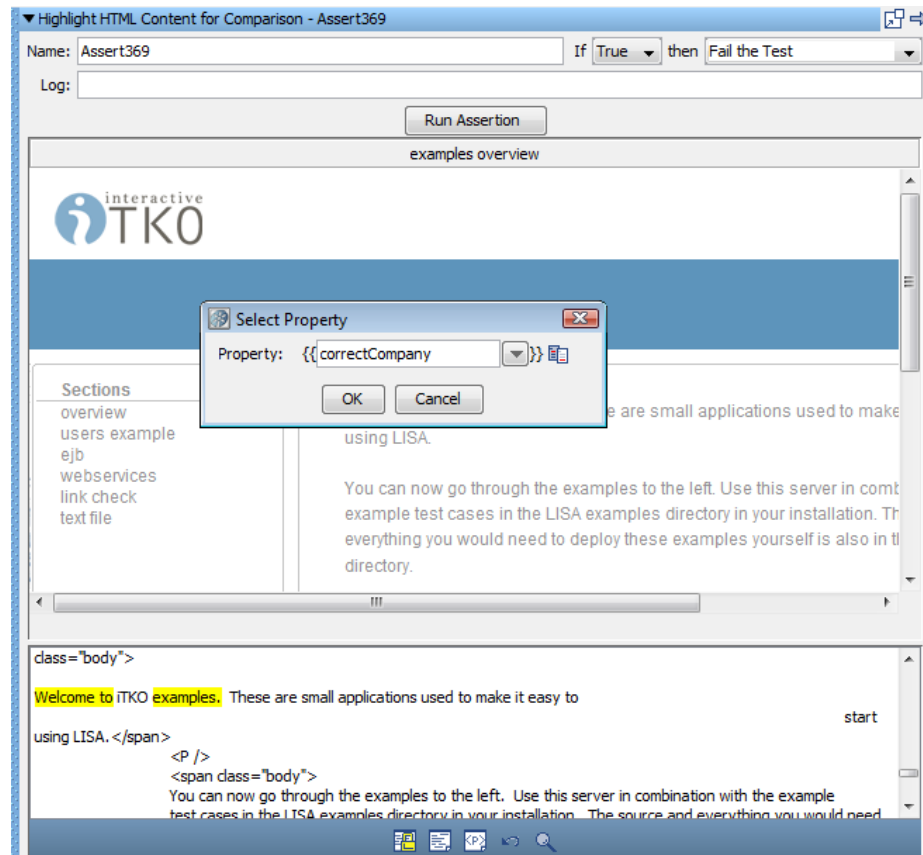
The text is marked using the icons at the bottom of the editor:





This technique is best explained with an example.

In the following example, we want to ensure that the company name, currently ITKO, that appears in the phrase "Welcome to ITKO examples" matches the value in a specified property. We have marked the text, using the buttons that were shown previously, by selecting text and then clicking the appropriate icon.

- Yellow background indicates text that must appear exactly as shown.
- White background indicates text that does not need to be present, or can change.
- Red background identifies the text that must match the property that was entered into the dialog.



This screen shows the HTML that is rendered in a browser in the top panel, and the actual HTML text in the bottom panel. We want to make the phrases "Welcome to" and "examples" required. We have set the boundaries around those phrases, and clicked the Must  icon. Then we selected the company name text, "ITKO", inside the highlighted content, and clicked the Property  icon. We entered the property name correctCompany into the dialog. This property is compared to the text that appears between the two bounding phrases. The company name text has been replaced with the name of the property.

To execute an assertion, click the Run Assertion button.

When this assertion is run, the value of the property correctCompany is inserted between the phrases "Welcome to" and "examples". The resulting phrase is compared to the corresponding phrase in the HTML response. The phrase "Welcome to correctCompany examples" can change its location in the HTML and it is still found.

## Check HTML for Properties in Page

Use the Check HTML for Properties in Page assertion when the web page contains property data that might be used the assertion can use. The property data is made available for assertion by parsing the web page for the following items:

- Meta tags
- Title tags
- Hidden form fields
- Other tags that the product can automatically parse, including <isaprop> tags and the DevTest Integration API.

Here is a sample of the available properties table.

▼ Check HTML for Properties in page - Check HTML for Properties in page~1

Name: Check HTML for Properties in page~1 If True then Fail the Test

Log:

Run Assertion

Properties Referenced on Page

Property	Observed Value	Expression
user_profile.userid	user-1398721607	
title	LisaBank - Account Activity	
user_profile.action	Withdraw	
user_profile.accountid	-2d085326:1157caf4ae9:-7fff	

Find:

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

Click Run Assertion to execute the assertion.

**Note:** You may be prompted to install the Parse HTML for Tag filter.

## Ensure HTTP Header Contains Expression

The Ensure HTTP Header Contains Expression assertion lets you check that a specific HTTP result header contains a field that matches a specified regular expression.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

**Header Field**

The name of the header field.

**RegularExpression**

The regular expression that must appear in the header field.

Click Run Assertion to execute the assertion.

## Check HTTP Response Code

The Check HTTP Response Code assertion lets you verify that the HTTP response code matches a specified regular expression.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

**RegularExpression**

The regular expression that must appear in the response code. For example, to verify that the HTTP response code is in the 400-499 range, set the RegularExpression to 4\d\d.

Click Run Assertion to execute the assertion.

## Simple Web Assertion

The Simple Web Assertion reads the return code from the web application.

If the application returns code 404 (page not found), 500 (server error) or any other error then this assertion returns true.

The multi-tier-combo test case in the examples project has this type of assertion.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

Click Run Assertion to execute the assertion.



## Check Links on Web Responses

The Check Links on Web Responses assertion verifies that every link on the returned web page is valid and does not return an HTTP error like a 404 error, or others. This assertion is commonly used to ensure that the links are working properly across the application and there are no inactive links on the page.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

The following criteria can be checked for the links:

**Check only links in the same domain**

Checks only links in the current domain of the returned web page.

**Include query strings**

If any query strings are present on the returned web page, then they are checked.

**Include anchors (<\_a>)**

Any anchor links in the current web page are checked.

**Include images**

All the images on the returned web page are checked.

**Include assets (<\_link> & <\_script>)**

The current web page is checked for script and links.

**Skip Links Matching RegEx**

Enter a RegEx expression for any links you want to skip.

## Database Assertions

**The following assertions are available in the Database Assertions list for any test step.**

[Ensure Result Set Size](#) (see page 18)

[Ensure Result Set Contains Expression](#) (see page 19)

## Ensure Result Set Size

The Ensure Result Set Size assertion counts the rows in a result set and verifies that the size falls between an upper and lower value.

An example of this assertion could be verifying that the number of rows in an HTML table matches a specified expected value from a data set.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

**Result set has warnings**

If selected, the database could return warnings in the result set. To determine whether your database supports warnings in the result set, consult your system administrator.

**Row Count >=**

The minimum number of rows in the result set. -1 indicates no minimum.

**Row Count <\_ =**

The maximum number of rows in the result set. -1 indicates no maximum.

Click Run Assertion to execute the assertion.

For example, to ensure that a Database Assertion step returns one, and only one, row, set the Row Count >= field to "1" and the Row Count <= field to "1".

## Ensure Result Set Contains Expression

The Ensure Result Set Contains Expression assertion verifies that a supplied expression matches at least one value in a specific column in a result set.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

**Column**

Defines a column that contains the text to verify. This value can be a column name or an index.

**Regular Expression**

The regular expression to match in the column.

Click Run Assertion to execute the assertion.

For example, to verify that at least one row that a query returned has a login value that starts with "wp", set the Column field to "login" and the Regular Expression field to "wp.\*".

## XML Assertions

**The following assertions are available in the XML assertions list for any test step.**

[Highlight Text Content for Comparison](#) (see page 20)

[Ensure Result Contains String](#) (see page 22)

[Ensure Step Response Time](#) (see page 23)

[Graphical XML Side-by-Side Comparison](#) (see page 24)

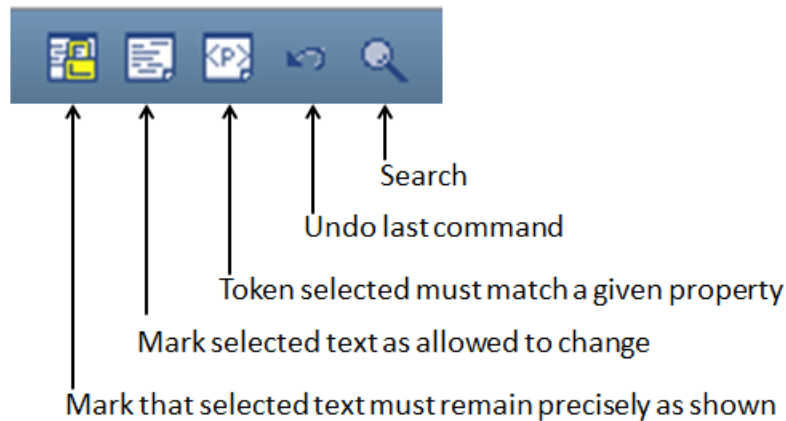
[XML XPath Assertion](#) (see page 29)

[Ensure XML Validation](#) (see page 31)

## Highlight Text Content for Comparison

The Highlight Text Content for Comparison assertion uses the "paint the screen" technique that is designed to work with HTML pages. For example, if there is a large HTML document, then you identify the data before and after the content of interest. Then, you identify what to compare the content of interest against (usually this content is an expected value that is supplied in a data set).

Mark the text with the icons at the bottom of the editor:

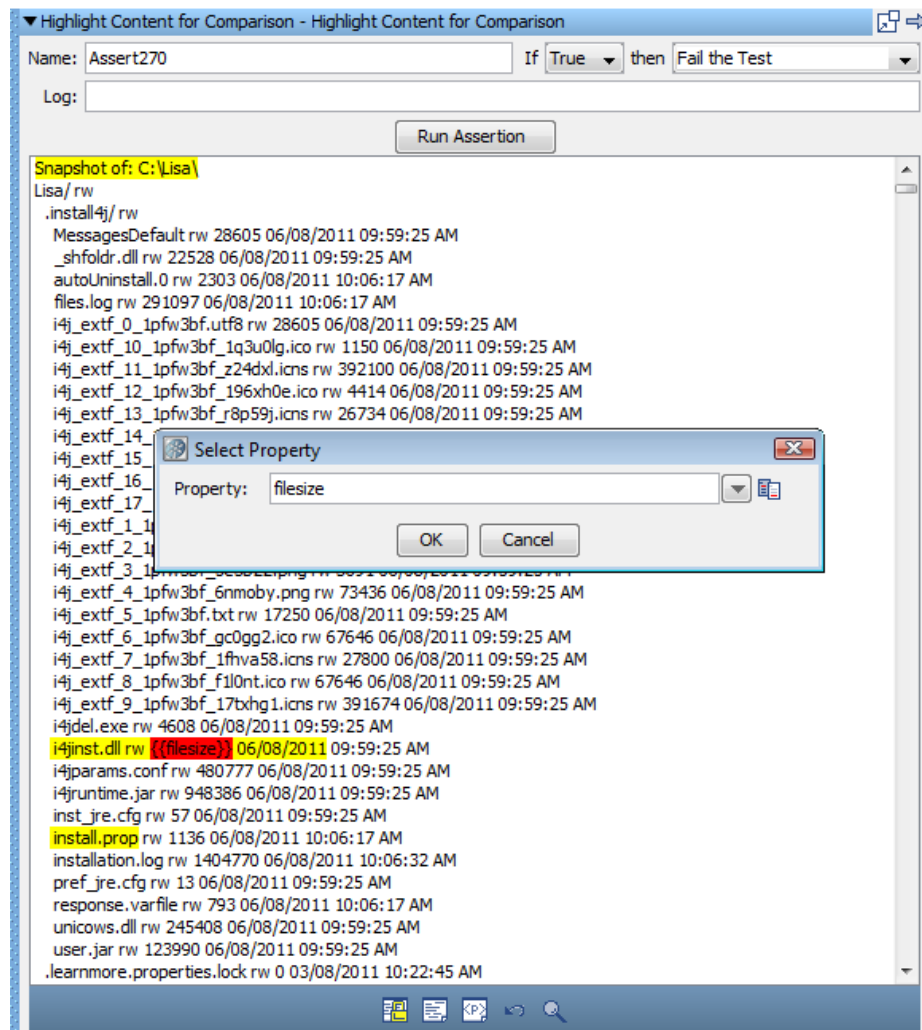


In the following example, we want to:

- Ensure that the buffer contains specific files
- Compare the size of one of the files to the value of a property

We have marked the text using the three icons that were shown in the previous graphic, by selecting text, and then clicking the appropriate icon.

- Yellow background indicates text that must appear exactly as shown.
- White background indicates text that does not need to be present, or can change.
- Red background identifies the text that must match the property that is entered into the dialog.



The set of tokens that is shown in the previous graphic can be read this way:

- The buffer must start with the phrase in yellow: "Snapshot of: C:\Lisa\".
- A number of files may or may not be in the buffer in the next token. Because the next token is an "Any" token, the variance is immaterial.
- The file "i4jinst.dll" and "rw" attributes must appear.

The red filesize means that the value associated with the property key filesize is swapped into the expression, and then the comparison made.

- The text "06/08/2011" must appear.
- The file "install.prop" must appear.
- The buffer can have any amount of content afterward.

After you have finished the markup, enter the following parameters:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

Click Run Assertion to execute the assertion.

**Note:** "Must" blocks must always appear on both sides of "Property" blocks.

## Ensure Result Contains String

The Ensure Result Contains String assertion lets you search the response (as text) for a string.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

**Contains String**

The string to search for in the step result - this string can contain a property.

## Ensure Step Response Time

The Ensure Step Response Time assertion lets you define the minimum and maximum bounds on the response time and assert that the response time is in those bounds.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

**Time must be at least (millis)**

Enter the lower bound in milliseconds.

**Time must not be more than (millis)**

Enter the upper bound in milliseconds. This value is ignored if set to -1.

**Note:** Parameters can contain properties.


## Graphical XML Side-by-Side Comparison

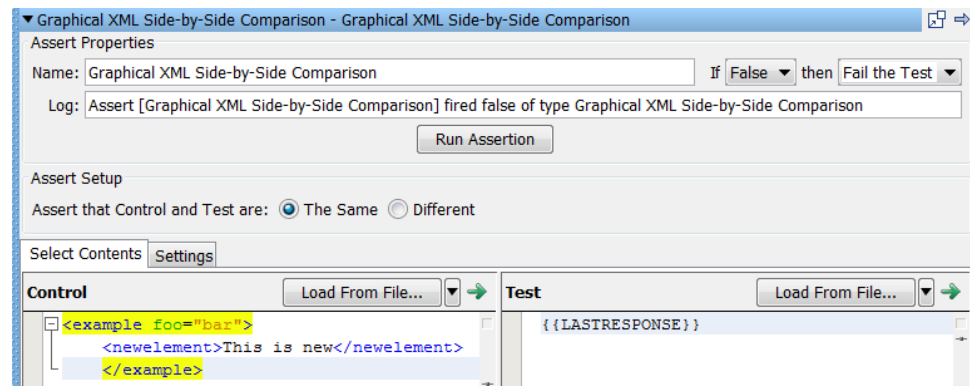
The Graphical XML Side-by-Side Comparison assertion lets you compare a test XML value received from a test with a control XML value. If the responses are the same or different, the assertion can return true. This assertion provides the flexibility to compare XML documents at various steps in a business process to ensure they match expected criteria. This approach is known as "exclusive" testing, where an entire response is compared except for a few values that are known to change.

The assertion editor works by comparing a left and right side of XML to each other. The left side is known as Control Content. The right side is known as Test Content. For example, Control Content is the expected XML returned from a web service in the application under test, while the Test Content is actual content. By default, Test Content is loaded from the last response of the test step that is associated with the assertion, signified by the empty property key. Otherwise, any valid property key can be used and the Test Content is loaded from it.

Alternatively, you can complete one of the following actions in test case authoring mode so that a quick graphical diff can be performed:

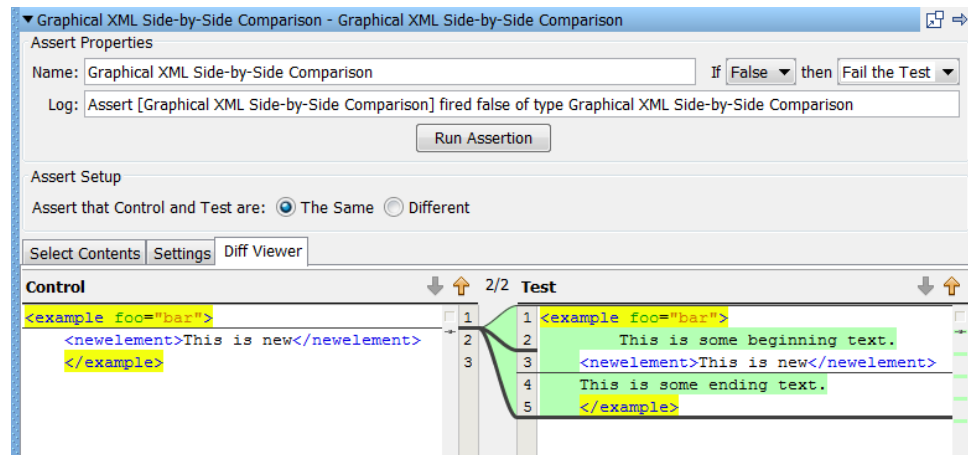
- Load XML from a file
- Enter XML manually for Control and Test Content

To perform the diff, click the  green arrow.



After a diff is executed, the results appear in the visualizer in the Diff Viewer tab in the assertion editor.





### Output During Execution

When an assertion is executed, DevTest logs the diff results as test events.

An Info message EventID containing the XML diff results is always logged.

If the assertion fires, an Assertion fired EventID containing the XML diff results is logged.

The diff results are reported in a format resembling the original UNIX diff utility. An example of a text diff report is:

```
Assert [Assert1] fired false of type Graphical XML Diff Assertion
XML is [Different]
=====
1,2[ELEMENT_NAME_CHANGED]1,2
<! <test2>
<! </test2>
---
>! <test>
>! </test>
```

Each difference is displayed with a heading of the format:

```
<First Start Line>, <First End Line>['<Diff Type>']<Second Start Line>,<Second
End Line>
```

Then the difference in the first content is displayed, followed by the separator '---', followed by the difference in the second content.

The + character indicates addition, the - character indicates a deletion, and the ! character indicates a change. When these characters are present, they indicate that an actual change occurred on the line of content (instead of to a context line).

### XML Compare Options

The following comparison options are available for use by the diff engine:

### **General**

#### **Case sensitive**

Whether case sensitivity is used during the comparison (enabled by default).

### **Whitespace**

#### **Trim whitespace**

During a comparison, all leading and trailing whitespace is removed from element text and attribute values (enabled by default).

#### **Collapse whitespace**

In addition to trimming whitespace, any sequence of one or more whitespace characters inside text is converted to a single space character.

#### **Normalize whitespace**

Any sequence of one or more whitespace characters is converted to a single space character.

#### **Ignore all whitespace**

All whitespace is ignored during the comparison.

### **Namespaces**

#### **Ignore namespaces**

The namespace value of an element or attribute is ignored.

#### **Ignore namespace prefixes**

The namespace prefix of an element or attribute is ignored (enabled by default).

### **Ordering**

#### **Ignore child element ordering**

Ignore the order of child elements in the XML document.

#### **Ignore attribute ordering**

Ignore the order of attributes in the XML document (enabled by default).

### **Node Types**

#### **Ignore element text**

Ignore all element text.

#### **Ignore attribute values**

Compare attribute names but ignore attribute values.

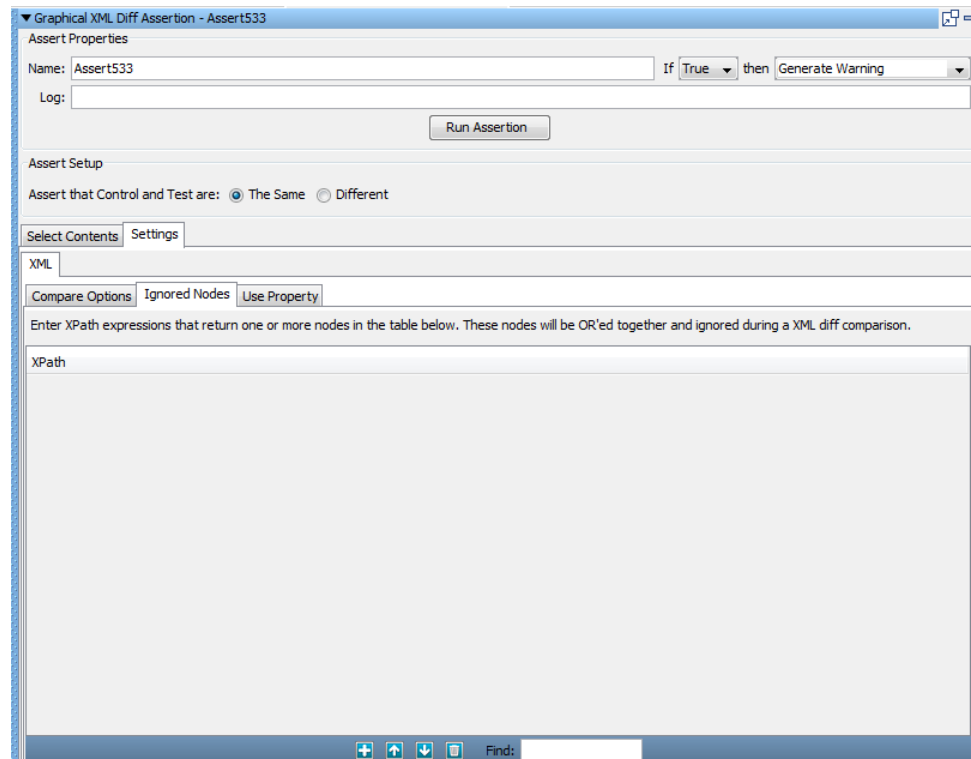
#### **Ignore attributes**

Ignore attribute names and values.

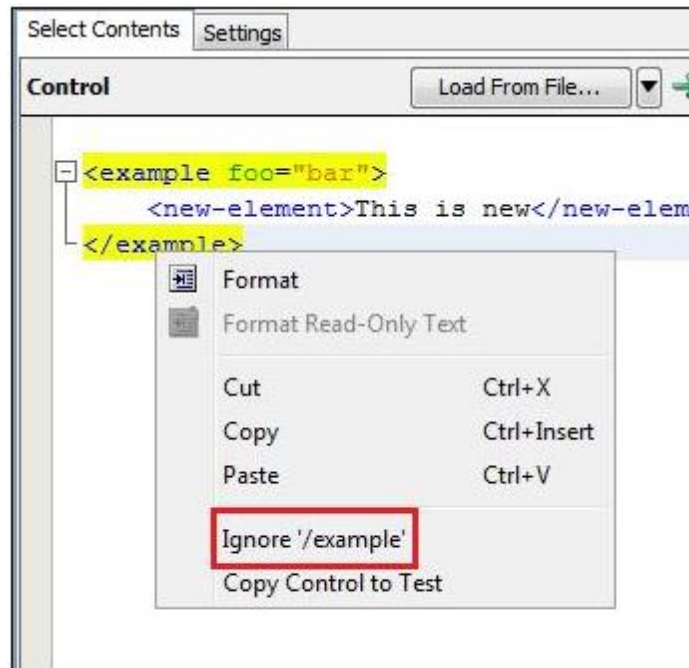
### Ignored Nodes

Ignored nodes are created from a list of XPath expressions that are run against the left and right documents. Each evaluated XPath that returns a node set is aggregated. When the diff occurs, any node that is found in the aggregate set is ignored.

Ignored node XPath expressions can be any arbitrary query that returns a node set. For example, the XPath `//*` excludes all nodes in an XML document. `/example/text()` excludes the first text node child of the `example` element in an XML document. `/example/@myattr` is the XPath to ignore the `myattr` attribute, including the attribute text value, belonging to the `example` element in an XML document.



A right-click menu item also lets a node be selected directly inside an XML document and its XPath is added to the Ignored Nodes list.



## XML XPath Assertion

The XML XPath assertion lets you use an XPath query that runs on the response. When this assertion is selected, the last response is loaded into the content panel.

Think of XPath as the "SQL for XML." XPath is a powerful query language that makes parsing the XML simple. XPath assertions are useful for validating a web service response in a more sophisticated way than simply parsing the entire result for a specific string. For example, you can ensure the second and third order item contains "ITKO" and the value of those line items is greater than ten.

You can view the response as an XML document or as a DOM Tree. However, you can only make the XPath selection from the DOM Tree view.

You can construct the XPath query in the following ways:

- Manually enter the XPath expression in the XPath Query text box.
- Select an element from the DOM tree and let DevTest construct the XPath expression.
- Select an element from the DOM tree, and then edit the XPath that DevTest constructs. For example, you can modify it to use a property, or a counter data set.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

Now construct the XPath query using one of the methods that was described previously.

After an XPath query has been constructed, test it by clicking the Run Assertion button on the top of the panel. The results of the query are displayed in the Query Results panel.

The previous example uses the fourth occurrence of the <wsdl:part> tag.

A common use case is to select an XPath node in a web service result and compare the node to a text value. You can then assert that a response contains the expected value. For this common use case, you can add an equality operator to the end of the initial expression that is provided. For example, if we select the new password that is returned in the response (BobPass):

DevTest builds the following XPath expression:

```
string(/env:Envelope/env:Body/ns2:updatePasswordResponse/[name(
)='return']/[name()='pwd'])=
```

Adding 'NewPassword' (as in the following example) compares the string of the result to the value of the property that is used to set the new password. If the equality test does not match, the assertion fails.

```
string(/env:Envelope/env:Body/ns2:updatePasswordResponse/[name(
)='return']/[name()='pwd'])='NewPassword'
```

This expression checks the web service response, looking for the new password and making sure it matches as expected.

## Ensure XML Validation

The Ensure XML Validation assertion lets you validate an XML document. You can verify whether the XML document is well-formed, you can validate against a Document Type Definition (DTD), or you can validate against one or more schemas. If you have an XML fragment, you can specify to have DevTest add the XML declaration tag. You can also specify that DevTest reports warnings as errors. You enter the XML to validate as a property.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

**Source**

The property that contains the XML. If this field is left blank, the last response is used.

**Validate**

Multiple validation options can be selected:

**Well Formed XML**

Check that XML is well-formed.

**DTD Conformance**

Check conformance with a DTD.

**Schema(s)**

Check conformance with one or more schemas.

**XML Fragment**

If XML is a fragment, an XML declaration is added to the top of the XML fragment.

**Treat Warnings As Errors**

Warnings are reported as errors.

**Honor All Schema Locations**

If you have multiple imports for the same namespace, this option opens each schema location instead of only the first one.

Click Run Assertion to execute the assertion.

### Validation Tab

You can run the validation by clicking the Run Validation button. Any resulting validation errors are displayed in the Validation Error List. You can use the Validation Type option buttons to select how to handle the errors:

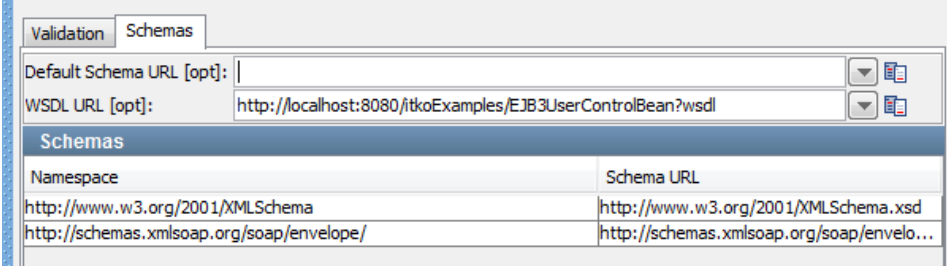
#### No Errors Allowed

The validation fails on any error.

#### Error Message Expressions

Errors can be marked to be ignored in the validation. If you select this option, you can select the errors to ignore in the Validation Error List.

### Schemas Tab



The screenshot shows the 'Schemas' tab of a validation interface. At the top, there are two tabs: 'Validation' and 'Schemas'. Below the tabs, there are two input fields: 'Default Schema URL [opt]:' and 'WSDL URL [opt]:'. The 'WSDL URL' field contains the text 'http://localhost:8080/itkoExamples/EJB3UserControlBean?wsdl'. Below these fields is a table titled 'Schemas'.

Namespace	Schema URL
http://www.w3.org/2001/XMLSchema	http://www.w3.org/2001/XMLSchema.xsd
http://schemas.xmlsoap.org/soap/envelope/	http://schemas.xmlsoap.org/soap/envelo...

Enter the information for each schema you want to use in the validation. You can also specify the default schema:

#### Default Schema URL

Optionally, specify the default URL of the schema.

#### WSDL URL

Optionally, specify a URL of a WSDL.

## JSON Assertions

The following assertions are available in the JSON assertions list for any test step.

[Ensure Result Equals](#) (see page 33)

[Ensure Result Contains](#) (see page 34)

[Ensure JSON Schema](#) (see page 35)



## Ensure Result Equals

The Ensure Result Equals assertion lets you ensure that a JSON Path result is equal to an expected value.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

**JSON Path**

Designates an expression that consists of a sequence of JSON properties in a JSON document. The JSON Path represents a path to a destination JSON property.

**Expected value**

Defines the expected value of the JSON Path result. Two arrays are considered equal when the order of the elements in both arrays is equal, because an array is an ordered list.

**Run Assertion**

To run and execute the filter, click Run Assertion.

## Ensure Result Contains

The Ensure Result Contains assertion lets you ensure that a JSON Path result that is either a JSON object or a JSON array contains any or all of the values in a specified list.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

**JSON Path**

Designates an expression that consists of a sequence of JSON properties in a JSON document. The JSON Path represents a path to a destination JSON property.

**Contains all expected values or Contains any expected values**

Defines the parameters with which to match the JSON Path and the Expected value. If you select the Contains all expected values check box, the JSON Path result must include all values in the Expected value list to pass the assertion. If you select the Contains any expected values check box, the JSON Path result must include at least one value in the Expected value list to pass the assertion.

**Expected value**

Defines a list of values that the JSON Path result must include. To stop editing values in the Expected value field, hold down Cntl and click Enter. For an OS X system, hold down Command and click Enter.

**Run Assertion**

To run and execute the filter, click Run Assertion.

## Ensure JSON Schema

The Ensure JSON Schema assertion lets you ensure that a JSON schema is valid and that the payload of a JSON response is valid for that schema.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

**Validate Payload Against Schema**

Indicates whether to validate both the JSON schema and the payload (the response). When cleared, only the schema is validated.

**Default:** selected




### Select Contents Tab

The Select Contents tab lets you select the origin of the JSON schema and the payload, and inspect their contents.

Complete the following fields:




**Schema**

Indicates the source of the JSON schema for this assertion. From the drop-down list, select from:

- From Content: Loads the JSON schema from a flat file on your file system. To select the file, click Browse File Selection.
- From URL: Loads the JSON schema from a URL. Enter the URL in the URL Path field. To browse the file system, click Browse . To reload the URL, click Refresh .
- From Property: Loads the JSON schema from a property. To refresh the property value, click Refresh .

**Payload**

Indicates the source of the payload, or JSON text. From the drop-down list, select from:

- From Content: Loads the JSON schema from a flat file on your file system. To select the file, click Browse File Selection.
- From URL: Loads the JSON schema from a URL. Enter the URL in the URL Path field. To browse the file system, click Browse . To reload the URL, click Refresh .
- From Property: Loads the JSON schema from a property. To refresh the property value, click Refresh .

### Settings Tab

The Settings tab lets you set advanced options for the Ensure JSON Schema assertion.

Complete the following fields:

#### Use Subschema

Instructs the application to look for a specific subschema. Enter the subschema in the Subschema Filter field.

#### Referencing Mode

Indicates the dereferencing mode to use.

To dereference all resolved URIs, select Canonical. To dereference URIs within the schema, select Inline.

**Default:** Canonical

#### Run Assertion

To run and execute the filter, click Run Assertion.

## Virtual Service Environment Assertion

**The following assertion is available in the Virtual Service Environment assertions list for any test step.**

[Assert on Execution Mode](#) (see page 37)

## Assert on Execution Mode

The Assert on Execution Mode assertion checks the current execution mode to its reference and fires if they match. This assertion is primarily used to control step flow for virtual service models.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

**Execution Mode**

Select the execution mode from the available options in the drop-down. For more information about execution modes, see the section Specify How the Selected Model Should Behave in *Using CA Service Virtualization*.

Click Run Assertion to execute the assertion.

## Mobile Assertions

**The following assertions are available in the Mobile assertions list for test steps.**

[Ensure Same Mobile Screen](#) (see page 38)

[Ensure Screen Element Matches Expression](#) (see page 40)

[Ensure Screen Element is Removed](#) (see page 40)

## Ensure Same Mobile Screen

The Mobile Ensure Same Mobile Screen assertion verifies that all screen elements on the mobile device match the original recording. By default, if any element on the screen is different, the test step fails.

Complete the following fields:

### Name

The name of the assertion. The default value is **Assert same screen**.

### If

Select one of the following values.

- **True:** The action that you define in the Then field is executed when all screen elements on the mobile device match the original recording.
- **False:** The action that you define in the Then field is executed when the screen elements on the mobile device do not match the original recording. **False** is the default value.

### Then

Select the action to take when the conditions of the selected If statement are met:

- Generate a warning or error.
- End, fail, or abort the test.
- Select the next test step to run.

### Log

The text that appears as log event text when the assertion runs.

Click Run Assertion to execute the assertion.

### Difference Threshold

The difference threshold lets you define the level of precision to use when comparing screens. Subtle differences can exist between the application and the recorded screen shots, even when the application has not changed. For example, your test could compare a table in your application to a screenshot of that same table. If the screenshot included a highlighted row, but no rows were highlighted in the application, a precise pixel-by-pixel comparison shows these tables as nonmatching.

The default difference threshold is **1000**, but you can adjust this number if you find that your test case is returning incorrect matches. The following values provide some guidance for your settings:

- **0:** Indicates an exact match where every pixel on the screen must match every pixel on the recording.
- **1000:** Indicates that the screen and the recording are close enough to each other to be considered a match.

- **2000+:** Indicates that the screen and the recording can almost certainly be classified as nonmatching.

**Start of Step**

Selecting this check box indicates that the screen comparison occurs at the beginning of the step.

**End of Step**

Selecting this check box indicates that the screen comparison occurs at the end of the step.

**Consider image pixels**

Selecting this check box indicates that pixel comparisons are used to determine the screen matches.

**Consider screen structure**

Selecting this check box indicates that the structure of the screen is used instead of a pixel comparison to determine screen matches. For example, you might have a screen that displays the current date and time. The date and time in a recording that was performed two days ago do not match a test that you run today. This option lets you ensure that the same screen structure is in place, even though the specific values on the screen do not match.

**Consider component bounds**

## Ensure Screen Element Matches Expression

The Mobile Ensure Screen Matches Expression assertion verifies that the data entered for a screen element matches your specified regular expression. By default, if the value entered for the screen element does not match the specified expression, the test step fails.

When you select the Ensure Screen Element Matches Expression assertion, the system prompts you to enter an expression that you want the screen element to match. Once entered, the name and expression appear as a Key/Value pair in the Actions section of the tab. You can double-click these fields to modify them.

For example, the regular expression "ame" would match the names "Cameron" and "Pamela". If you want to ensure that a value entered for a screen element starts with a capital letter, and then three lower-case letters, enter this regular expression:

```
'[A-Z][a-z]{3}'
```

For more information about regular expressions, see [Assertion Descriptions](#) (see page 9).

Complete the following fields:

**Expression to Match (Value)**

The regular expression to match for the specified screen element.

**If no match, execute step**

Select a step from the drop-down list to execute if a match is not made.

## Ensure Screen Element is Removed

The Ensure Screen Element is Removed assertion verifies that a specified screen element on the mobile device has been removed before proceeding to the next step. By default, if the specified element is present, the test step fails.

When you add this assertion, a pop-up dialog lets you select the screen to test that the element has been removed.

For example, you could have a test case that includes three steps. The last step involves clicking the Back button. To ensure the Back button is *not* included on the first step, you would add the Ensure Screen Element is Removed assertion for the Back button on the last step, and select that first step in the drop-down list.

## Other Assertions



**The following assertions are available in the Other Assertions list for any test step.**

[Highlight Text Content for Comparison Assertion](#) (see page 42)

[Ensure Non-Empty Result Assertion](#) (see page 44)

[Ensure Result Contains String Assertion](#) (see page 45)

[Ensure Result Contains Expression Assertion](#) (see page 45)

[Ensure Property Matches Expression Assertion](#) (see page 46)

[Ensure Step Response Time Assertion](#) (see page 46)

[Scripted Assertion](#) (see page 47)

[Ensure Properties Are Equal Assertion](#) (see page 49)

[Assert on Invocation Exception Assertion](#) (see page 50)

[File Watcher Assertion](#) (see page 51)

[Check Content of Collection Object Assertion](#) (see page 52)

[WS-I Basic Profile 1.1 Assertion](#) (see page 53)

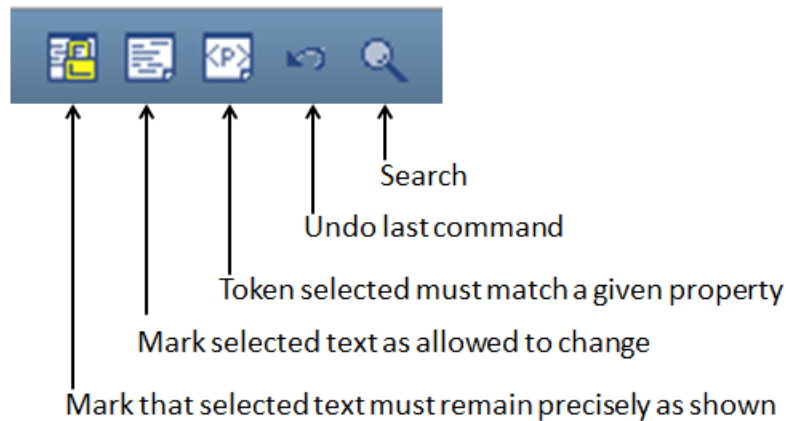
[Messaging VSE Workflow Assertion](#) (see page 54)

[Validate SWIFT Message Assertion](#) (see page 55)

## Highlight Text Content for Comparison Assertion

The Highlight Text Content for Comparison assertion uses the "paint the screen" technique that was designed to work with HTML pages. For example, if there is a large HTML document, you identify the data before and after the content of interest. Then, you simply identify what to compare the "content of interest" against (typically an expected value that is supplied in a data set).

The text is marked using the icons at the bottom of the editor:



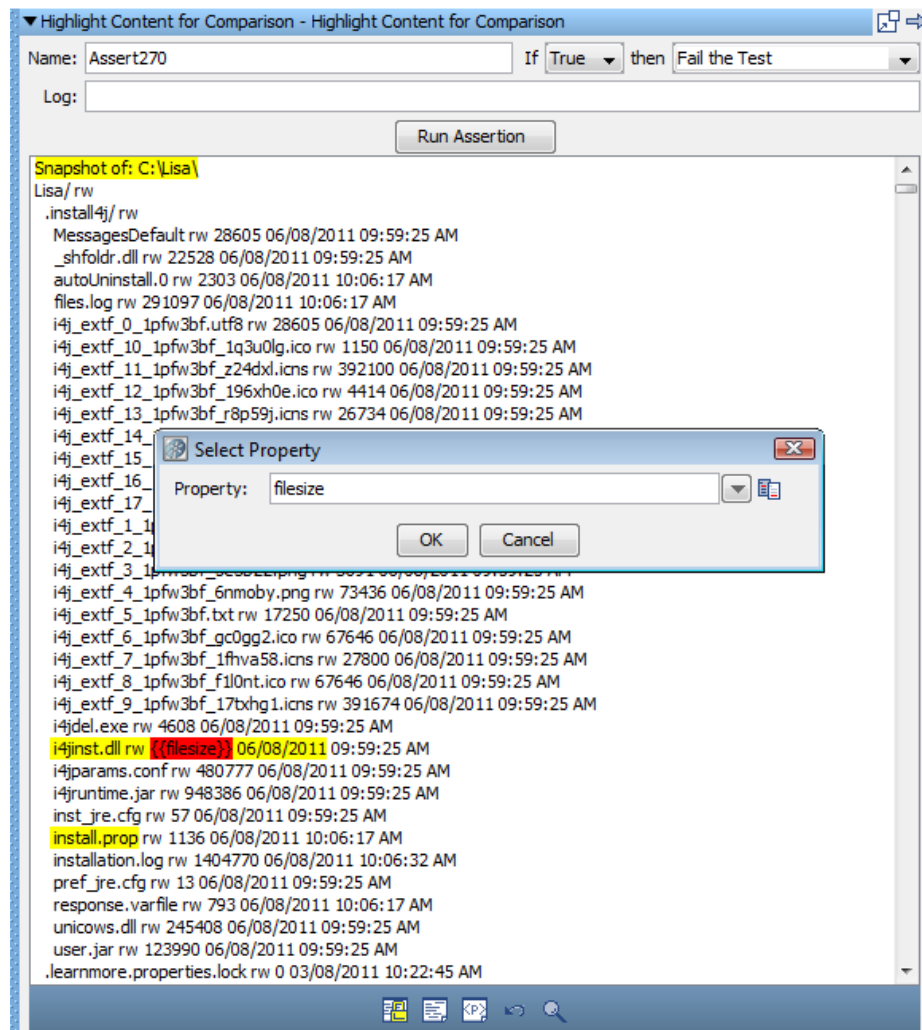
This technique is best explained by using an example.

In the following example, we want to complete the following actions:

- Ensure that specific files appear in the buffer
- Compare one of the file sizes to the value of a property

The text is marked using the three icons that are shown in the previous graphic, by selecting text and clicking the appropriate icon.

- Yellow background indicates text that must appear exactly as shown.
- White background indicates text that does not need to be present, or can change.
- Red background identifies the text that must match the property that was entered into the dialog.



The set of tokens that is shown here can be read this way:

- The buffer must start with the phrase in yellow: "Snapshot of: C:\Lisa\".
- The number of files in the buffer in the next token can vary. Because the token is an "Any" token, the variance is immaterial.
- The file "i4jinst.dll" and "rw" attributes must appear.
- The red filesize means that the value associated with the property key filesize are swapped into the expression, then the comparison made.
- The text "06/08/2011" must appear.
- The file "install.prop" must appear.
- The buffer can have any amount of content afterward.

After you have finished the markup, enter the following parameters:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

Click Run Assertion to execute the assertion.

**Note:** "Must" blocks must always appear on both sides of "Property" blocks.

## Ensure Non-Empty Result Assertion

The Ensure Non-Empty Result assertion checks the return value from the step to verify that some value has been returned. If there is no response (timeout) or the return value has a length of 0, the return value is considered empty. When the If field is set to True, this assertion fails the test unless it receives an empty response. When the If field is set to False, this assertion fails the test when it receives an empty response.

If the result is NULL, a Fail event is raised and the assertion's evaluate() method returns. Therefore, the If/then logic of the assertion is never reached.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

Click Run Assertion to execute the assertion.

No other attributes are required.

**Note:** Use this assertion carefully, because it does not have any content validation.

## Ensure Result Contains String Assertion

If it finds the value being searched anywhere in the response, the Ensure Result Contains String assertion returns true. This assertion is typically used to ensure that the response contains a required value such as a unique ID that was supplied during the request.

For more information, see [Ensure Result Contains String](#) (see page 22).

## Ensure Result Contains Expression Assertion

The Ensure Result Contains Expression assertion lets you verify that a specified regular expression occurs somewhere in the result, as text.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

**Regular Expression**

Defines the regular expression for which to search in the step result. For example, to verify that the result contains a number in the 400s, set this parameter to "4/d/d".

Click Run Assertion to execute the assertion.

## Ensure Property Matches Expression Assertion

The Ensure Property Matches Expression assertion lets you verify that the current value of a property matches a specified regular expression.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

**Property Key**

The name of the property to be checked. Enter the property name, select from the drop-down list of properties, select an existing string pattern, or create a string pattern.

**RegularExpression**

The regular expression that must appear in the current value of the property.

Click Run Assertion to execute the assertion.

## Ensure Step Response Time Assertion

The Ensure Step Response Time assertion lets you define an upper and lower threshold for application response time. If the performance is either too fast or too slow, the test case can be failed by using this assertion. Sometimes an application that returns a response quickly can be a sign that the transaction was not properly processed.

For more information, see [Ensure Step Response Time](#) (see page 23).

## Scripted Assertion

The Scripted Assertion assertion lets you write and run scripts. The result must be a Boolean, or false is returned.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

Click Run Assertion to execute the assertion.

**Language**

Designates the scripting language to use.

**Values:**

- Applescript (for OS X)
- Beanshell
- Freemarker
- Groovy
- JavaScript
- Velocity

**Default:** Beanshell

To use additional scripting languages, see "Enabling Additional Scripting Languages."

**Copy properties into scope**

Allows you to specify which properties to download for use in the step.

**Values:**

- Test state and system properties: all properties for the test case and system
- Test state properties: properties that provide information about the test case
- TestExec and logger only: only the TestExec and logger properties

**Default:** Test state properties

Enter your script into the script editor on the left.

All the objects available for use in the script editor are listed in the Available Objects panel on the right. The list includes primitive types of data (strings and numbers) and objects such as EJB response objects that were executed in the test case. Double-click an entry in the Available Objects table to paste that variable name into the editor area.

Click Test to open a window with the result of the script execution or a description of the errors that occurred.

When you save the test case, the assertion is checked for syntax errors.

Some things to remember:

- If you use a property - **{{someprop}}** - in a script, it is substituted for the property value at run time before the script is executed.
- If you need access to a property with a "." in the name, such properties are imported into the script environment replacing "." with "\_". So **{{foo.bar}}** in a script is the same as **foo\_bar**.
- You can produce a DevTest log event inside a script step or assertion by using the **testExec** object. To produce a DevTest log event, code the following line, as opposed to using the log4j logger. The **testExec.log()** method causes an actual DevTest event to be raised. You can see the event in the ITR.

```
testExec.log("Got here");
```



## Ensure Properties Are Equal Assertion

The Ensure Properties Are Equal assertion lets you compare the values of two properties to ensure that they are same. Typically this assertion is used with a data set and supplied "expected value" to ensure that the application functionality is correct.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

**First Property**

The first property in the comparison. Enter the property name, select from the drop-down list of properties, select an existing string pattern, or create a string pattern.

**Second Property**

The second property in the comparison. Enter the property name, select from the drop-down list of properties, select an existing string pattern, or create a string pattern.

Click Run Assertion to execute the assertion.

## Assert on Invocation Exception Assertion

The Assert on Invocation Exception assertion lets you alter the test flow, in regard to the occurrence of a Java exception. The assertion asserts true if a specific Java exception is returned in the response.

Complete the following fields:

**Name**

Defines the name of the assertion.

**Log**

Identifies event text to print if the assertion fires.

**Assert**

Select the behavior of the assertion using the option buttons.

**Execute**

Select the step to redirect to if the assertion fires.

**Expression**

The expression to search for in the invocation exception. The expression can be a regular expression. The expression `'.*'` is common.

## File Watcher Assertion

The File Watcher assertion lets you monitor a file for specific content, and react to the presence (or absence) of a specific expression. This assertion runs in the background while your test case is executing.

Complete the following fields:

**Name**

Defines the name of the assertion.

**Log**

Identifies event text to print if the assertion fires.

**The amount of time (in seconds) to delay before checking the file contents**

The number of seconds to wait before checking the file at the beginning of the step that contains this assertion.

**The amount of time (in seconds) to wait between checks on the file contents**

The number of seconds to wait between each check.

**The time (in seconds) the File Watcher will give up watching for the expression**

The total number of seconds this assertion checks for the expression.

**The url of the file to watch**

The URL or path to the file being watched.

**The expression to watch for in the file**

The regular expression being watched for in the response.

**Note:** The times are in seconds and must be integers. The times default to 0.

## Check Content of Collection Object Assertion

The Check Content of Collection Object assertion lets you make simple assertions on the contents of a collection. This assertion is a useful way to find out if specific tokens are in the collection, with the option of adding some simple constraints. For example, if data that a bank web service returns includes an account list, this assertion can verify whether the account IDs match expected values.

Complete the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

**Property to Check (blank for whole response)**

The name of the property holding the object that you want to use in the assertion. Leave this field blank to use the last response. The object must be of type Java Collection or Array.

**Field to Check (blank for "toString")**

Enter the name of a field and DevTest calls its get method.

**Tokens To Find (value1, value2)**

A comma-delimited string of tokens for which to check.

**Exact Match Only**

The token names must match exactly.

**Must be in this order**

Select this check box if the tokens must be found in the same order as in the Tokens To Find string.

**Must only contain these tokens (no extra objects)**

The tokens in the Tokens To Find string must be the only tokens found.

Click Run Assertion to execute the assertion.

## WS-I Basic Profile 1.1 Assertion

The WS-I Basic Profile 1.1 assertion lets you get a WS-I Basic Profile compliance report for a specific web service. This report is delivered in the standard format that the WS-I Basic Profile specifies.

The screenshot shows a configuration window for the 'WS-I Basic Profile 1.1 Assertion'. The window has a title bar with a minus, maximize, and close button. Below the title bar, there is a 'Name' field set to 'Basic Profile 1.1 Assertion', an 'If' dropdown set to 'True', and a 'then' dropdown set to 'Fail the Test'. A 'Log' field is empty. Below these are several dropdown menus: 'Report Type' set to 'Display Only Not Passed Assertions', 'Auto-Select Port' set to 'Don't Auto-select', 'Service Name' set to 'EJB3AccountControlBeanService' (with a 'Select...' button), 'Service Namespace' set to 'http://ejb3.examples.itko.com/', and 'Port Name' set to 'EJB3AccountControlBeanPort' (with a 'Test' button). Below the configuration fields is a section titled 'WS-I Profile Conformance Report'. This section contains the 'WS-I Profile Conformance Report' logo, the text 'Report: WS-I Basic Profile Conformance Report.', the 'Timestamp: 2012-05-08T13:26:05.808', and a copyright notice: 'Copyright (c) 2002-2004 by The Web Services-Interoperability Organization (WS-I) and Certain of its Members. All Rights Reserved.' Below this is a blue header for 'Analyzer Tool Information' and a table with one row: 'Version' | '1.0.0'. At the bottom of the window is a 'Done' button.

Complete the following fields:

### Name

Defines the name of the assertion.

### Log

Identifies event text to print if the assertion fires.

### **Report Type**

Select one of the following levels of (WS-I) assertions to include in the report:

- Display All Assertions
- Display All But Info Assertions
- Display Only Failed Assertions
- Display Only Not Passed Assertions

### **Auto-Select Port**

Determine how the port is selected – a specific port or "Don't Auto-select".

### **Service Name**

Select a service name from the list. This name can auto-populate from the step.

### **Service Namespace**

Is automatically populated based on the service name.

### **Port Name**

Is automatically populated based on the service name.

Click Test to run the assertion.

## Messaging VSE Workflow Assertion

The Messaging VSE Workflow Assertion assertion is automatically added from the VSE recorder. The assertion serves a specific purpose that makes VSE recordings work properly. Use it with care. If the assertion was added to a step in a VSE model, do not remove or edit it.

Click Run Assertion to execute the assertion.

## Validate SWIFT Message Assertion

The Validate SWIFT Message assertion lets you validate the syntax and semantics of SWIFT messages.

The assertion includes the following fields:

**Name**

Defines the name of the assertion.

**If**

Specifies the behavior of the assertion from the drop-down list.

**then**

Specifies the step to which to redirect if the assertion fires.

**Log**

Identifies event text to print if the assertion fires.

The assertion also includes the following SWIFT-specific fields:

**SWIFT Message Type**

Specifies the message type with which to validate the message. The available message types are MT, MX, and SEPA. The default message type is MT.

For the message type MT, DevTest supports the following categories:

**MT1nn**

Customer Payments

**MT2nn**

Financial Institution Transfers

**MT3nn**

FX, Money Market, and Derivatives

**MT4nn**

Collections and cash letters

**MT5nn**

Securities Markets

**MT7nn**

Documentary Credits and Guarantees

**MT9nn**

Cash Management and Customer Status

For the message type SEPA, DevTest supports the following categories:

- Resolution of Investigation (camt.029.001.03)
- Payment Cancellation Request (camt.056.001.01)
- Customer Credit Transfer Initiation(pain.001.001.03)
- Customer Payment Status Report (pain.002.001.03)
- Financial Institution Payment Status Report (pacs.002.001.03S2)
- Payment Return (pacs.004.001.02)
- Financial Institution Customer Credit Transfer (pacs.008.001.02)

For the message type MX, DevTest supports the latest versions (as of February 2014) of the full catalog of [ISO20022 messages](http://www.iso20022.org/full_catalogue.page), which are listed at [http://www.iso20022.org/full\\_catalogue.page](http://www.iso20022.org/full_catalogue.page).

#### Validation Syntax only

Specifies whether the assertion validates syntax and semantics.

##### Values:

- **Selected:** The assertion validates only the syntax.
- **Cleared:** The assertion validates both syntax and semantics.

**Default:** Cleared

If you click Run Assertion, any validation errors appear in the System Messages window.

#### Notes:

- Ensure that lines in MT messages end with the required Carriage Return/Line Feed characters (0D0A in ASCII hex; 0D25 in EBCDIC hex). Otherwise, the following error is reported.

"The input Swift message cannot be parsed because of invalid syntax. Please check the message structure. Take notice of block separators, carriage-return line-feed characters and the presence of mandatory blocks."

- Ensure that an MT message has no invalid trailing spaces before the end of a line. The reported errors are sometimes unclear.

For example, if the following line

:32A:071119EUR50000,

contains an invalid trailing space before the end of the line, DevTest reports the error message:

"T43 - The integer part of Rate must contain at least one digit. A decimal comma is mandatory and is included in the maximum length tag:32A."

- Ensure that the MX messages conform to the supported versions.



For example, **camt.052.001.04** is supported but the older **camt.052.001.01** is not.

## Asset Descriptions

**This section contains descriptions of the following assets:**

[JDBC Connection Assets](#) (see page 58)

[JMS Client Assets](#) (see page 60)

[JNDI Initial Context Asset](#) (see page 63)

[SAP JCo Destination Assets](#) (see page 64)

[Email Connection Asset](#) (see page 67)

[Mobile Assets](#) (see page 69)



## JDBC Connection Assets

To define the connection information to your JDBC system, use a destination asset. The asset class for JDBC connection assets is named JDBC Connection Assets.

**Prerequisites:** The JDBC driver appropriate for your database must be on the DevTest classpath. You can place the driver JAR file in the hot deploy directory. The Derby client driver is included in the DevTest classpath, so you do not need to add it again.

**Parameter Requirements:** Have the name of the JDBC driver class, the JDBC URL for your database, and a user ID and password for the database. You also must know the schemas for the tables in the database to construct your SQL queries.

If you have predefined destination assets, you can select one from the Destination drop-down field in the step editor. To create an asset from the step editor, select the

Add Asset  icon. To edit an asset from the step editor, select the Edit Asset  icon.



### To create an asset:

1. Define the following fields for this asset. You can use properties for connection parameters.

#### Name

The name of the asset. This name appears in the Destination field in the step editor. Use a name that is meaningful for your JDBC system.

#### Description

Information that provides more details about the system that the asset targets.

#### JDBC Driver

Enter or select the full package name of the appropriate driver class. Standard driver classes are available in the drop-down list. You can also use the Browse button to browse the DevTest class path for the driver class.

#### Connect String

The connect string is the standard JDBC URL for your database. Enter or select the URL. The drop-down list contains JDBC URL templates for the common database managers.

#### User ID

Enter a user ID (if the database requires it).

#### Password

Enter a password (if the database requires it).

#### Use Connection Pool

When you select Use Connection Pool, you can configure the connection pool size can be configured with the **`lisa.jdbc.asset.pool.size`** property in the **`lisa.properties`** file.

The screenshot shows a dialog box titled "JDBC (localhost:1528)" with a "JDBC Connection" icon and a "PRO" icon in the top right corner. The dialog contains the following fields and options:

- Name:** A dropdown menu showing "JDBC (localhost:1528)".
- Description:** A large empty text area.
- Scope:** A dropdown menu showing "Staged".
- JDBC Driver:** A dropdown menu showing "org.apache.derby.jdbc.ClientDriver" with a small icon to its right.
- Connect String:** A dropdown menu showing "jdbc:derby://localhost:1528/database/lisa.db;create=true".
- User ID:** A dropdown menu showing "rpt".
- Password:** A text field with three dots indicating a masked password.
- Use Connection Pool:** A checked checkbox.
- Buttons:** "OK" and "Cancel" buttons at the bottom.

2. To display the advanced mode, select the PRO icon in the upper right corner of the asset editor. The advanced mode allows you to specify the run-time scope of the asset.

## JMS Client Assets

The Java Message Service (JMS) is a specification that allows Java programs to interact with enterprise messaging systems. The original version is 1.0. The latest version is 1.1.

JMS includes the following terminology:

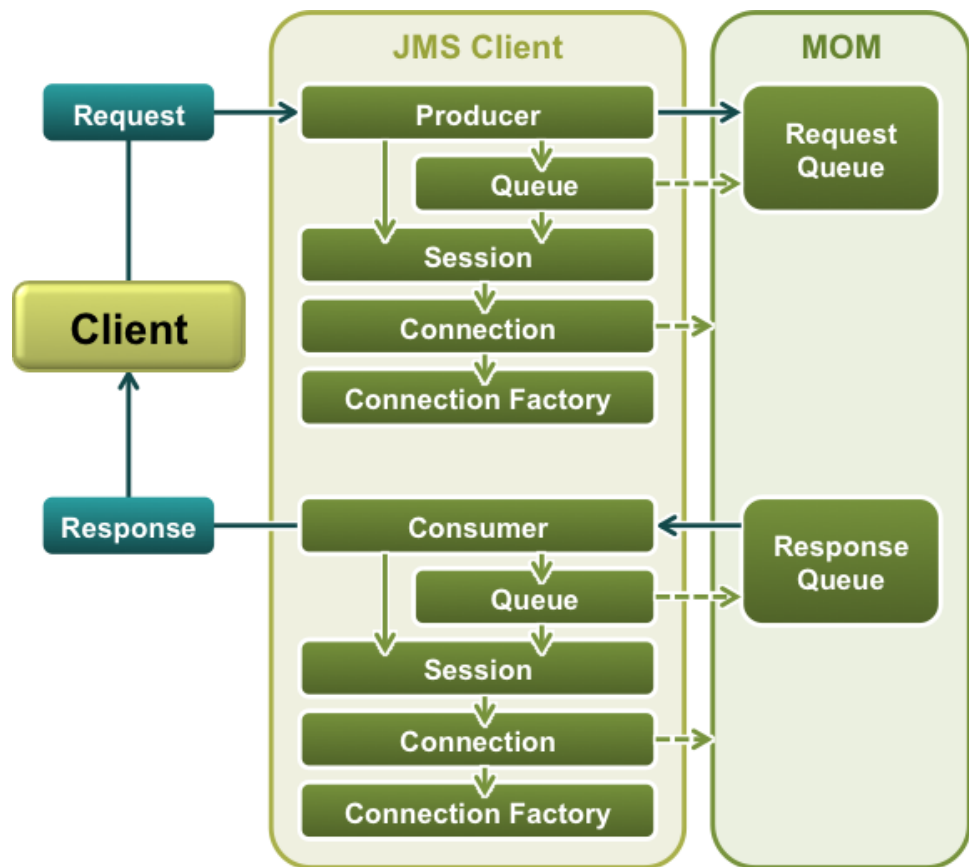
- **JMS client:** A Java program that uses JMS to communicate with a messaging system.
- **JMS provider:** A messaging system that implements JMS.

You can create assets for the following types of objects that the JMS clients use:

- Connection factory
- Connection
- Session
- Producer
- Consumer
- Destination

In the editor for each JMS client asset, each parameter has a tooltip that describes the purpose of the parameter.

The following graphic shows how a JMS client handles request messages and response messages.



### Connection Factories

A connection factory is used to create connections.

The connection factories can be characterized in terms of how they are initialized:

- **Generic connection factory:** Initialized by using the Java Naming and Directory Interface (JNDI).
- **Direct connection factory:** Initialized in a way that is specific to a JMS provider. This type of connection factory has its own, often large, set of parameters.

The connection factories can also be characterized in terms of the destinations that they support:

- **Queue connection factory:** Supports queues only. This type is from JMS version 1.0.
- **Topic connection factory:** Supports topics only. This type is from JMS version 1.0.
- **Connection factory:** If a connection factory is not specified as either of the preceding types, then it supports both queues and topics.

Many connection factory assets are a combination of these two categories. For example, the Direct JMS 1.0 Topic Connection Factory for TIBCO EMS asset is a direct connection factory that supports topics only.

### Connections

A connection represents an active connection with a JMS provider. Connections are used to create sessions.

Connections can be characterized in terms of the destinations that they support:

- **Queue connection:** Supports queues only. This type is from JMS version 1.0.
- **Topic connection:** Supports topics only. This type is from JMS version 1.0.
- **Connection:** If a connection is not specified as either of the preceding types, then it supports both queues and topics.

A connection must remain open while its sessions are open.

### Sessions

A session is used to create producers and consumers.

Sessions can be characterized in terms of the destinations that they support:

- **Queue session:** Supports queues only. This type is from JMS version 1.0.
- **Topic session:** Supports topics only. This type is from JMS version 1.0.
- **Session:** If a session is not specified as either of the preceding types, then it supports both queues and topics.

A session must remain open while its producers and consumers are active.

### Producers

A producer is used to send a message to a destination.

DevTest has only one type of producer.

### Consumers

A consumer is used to receive a message from a destination.

DevTest has only one type of consumer. However, you can use a consumer in either of two ways:

- **Synchronously:** While a client is waiting to receive a message, the client cannot do anything else.
- **Asynchronously:** While a client is waiting to receive a message, the client can perform other tasks.

### Destinations

A destination represents a location on the JMS platform where messages are placed.

Destinations are divided into queues and topics:

- **Queue:** A destination that supports the point-to-point messaging model. When a message is sent to a queue, only one receiver can receive the message.
- **Topic:** A destination that supports the publish/subscribe messaging model. When a message is sent to a topic, multiple receivers can receive the message.

Destinations can also be characterized in terms of how they are created:

- **JNDI destination:** Initialized by using the Java Naming and Directory Interface (JNDI).
- **Temp JNDI destination:** A JMS session can be used to create a destination temporarily. This destination exists as long as the session is open, and is deleted when the session is closed.
- **Destination:** A static destination is not specified as either of the preceding types, and is obtained through the JMS session.

Destination assets are a combination of these two categories. For example, the JMS JNDI Queue asset is a point-to-point destination that is initialized through JNDI.

## JNDI Initial Context Asset

The Java Naming and Directory Interface (JNDI) is a specification that allows Java programs to interact with naming and directory services.



DevTest includes a JNDI Initial Context asset. The [JMS client assets](#) (see page 60) use the JNDI Initial Context asset to locate JMS connection factories and destinations.

In the editor for the JNDI Initial Context asset, each parameter has a tooltip that describes the purpose of the parameter.

## SAP JCo Destination Assets

To define the connection information to your SAP system, use a destination asset. One asset class is used for all three SAP steps. The class is named SAP JCo Destination Assets.

If you have predefined destination assets, you can select one from the Destination drop-down field in the step editor. To create an asset from the step editor, click Add

Asset . To edit an asset from the step editor, click Edit Asset .

### To create an asset:

1. Define the following fields for this asset. You can use properties for SAP connection parameters.

#### **Name**

The name of the asset. This name appears in the Destination field in the step editor. Use a name that is meaningful for your SAP system.

#### **Description**

Information that provides more details about the system that the asset targets.

#### **Server Type**

Select Application Server or Message Server.

#### **Host**

Hostname or IP address of SAP system

#### **System Number**

SAP system number (for Application Server)

#### **R/3 Name**

R/3 name (for Message Server)

#### **Group**

Group of SAP application servers (for Message Server)

#### **User**

SAP username

#### **Password**

SAP user password

#### **Client**

SAP client

The following graphics illustrate destination asset definitions for both application and message servers.



The screenshot shows a dialog box titled "Message Server Destination" with a subtitle "SAP JCo Destination" and a "PRO" badge. The dialog contains the following fields and controls:

- Name:** A dropdown menu with the value "Message Server Destination".
- Description:** A text input field containing "SAP Message Server Destination".
- Scope:** A dropdown menu with the value "Global".
- Server Type:** Two radio buttons: "Application Server" (unselected) and "Message Server" (selected).
- Host:** A dropdown menu with the value "{{SAP\_HOST}}".
- R/3 Name:** A dropdown menu with the value "{{SAP\_R3NAME}}".
- Group:** A dropdown menu with the value "{{SAP\_GROUP}}".
- User:** A dropdown menu with the value "{{SAP\_USER}}".
- Password:** A text input field filled with 15 dots.
- Client:** A dropdown menu with the value "{{SAP\_CLIENT}}".
- Advanced Connection Settings:** A button located below the main fields.
- OK** and **Cancel** buttons at the bottom of the dialog.

**Application Server Destination** SAP JCo Destination **PRO**

**Name:** Application Server Destination ▼

**Description:** SAP Application Server Destination

**Scope:** PRO Global ▼

**Server Type:** ☒ Application Server ☐ Message Server

**Host:** {{SAP\_HOST}} ▼

**System Number:** {{SAP\_SYS\_NUMBER}} ▼

**User:** {{SAP\_USER}} ▼

**Password:** ●●●●●●●●●●●●●●

**Client:** {{SAP\_CLIENT}} ▼

**Advanced Connection Settings**

**OK** **Cancel**

2. To display the advanced mode, select PRO in the upper right corner of the asset editor. The advanced mode allows you to specify the runtime scope of the asset.
3. To select advanced SAP connection properties, click Advanced Connection Settings. You can override the default value of a setting by entering a value for the setting. Select the settings that you want, and click OK.

## Email Connection Asset

To define the connection information to the SMTP mail server, use a connection asset.

If you have predefined connection assets, you can select one from the Connection drop-down list in the step editor. To create an asset from the step editor, click Add Asset



. To edit an asset from the step editor, click Edit Asset



### To create an asset:

1. Define the following fields for this asset. You can use properties for SMTP connection parameters.

#### Name

Defines the name of the asset. This name appears in the Connection field in the step editor. Use a name that is meaningful for your SMTP mail system.

#### Description

Specifies information that provides more details about the system that the asset targets.

#### Server

Specifies the name of the SMTP mail server.

#### Security

Specifies which type of encryption to use to secure a communication channel.

##### Values:

- **None:** The communication uses no encryption.
- **SSL/TLS:** The communication uses SSL/TLS encryption.

#### Port

(Optional) Specifies the port on which the SMTP mail server connects.

##### Defaults:

- **25:** The default when **None** is specified for Security.
- **465:** The default when **SSL/TLS** is specified for Security.

#### Authentication

Specifies whether to use authentication to connect to an email server.

##### Values:

- **Off:** The email server requires no authentication.
- **Password Authentication:** The email server requires user and password authentication.

#### User

(Optional) Specifies the user ID that the SMTP mail server uses to authenticate the connection. This field is disabled when Authentication is **Off**.

**Password**


(Optional) Specifies the password that the SMTP mail server validates. This field is disabled when Authentication is **Off**.

2. To display the advanced mode, click PRO in the upper right corner of the asset editor. The advanced mode allows you to specify the runtime scope of the asset.

## Mobile Assets

You can create a mobile asset with various devices using the Mobile Session dialog.

**Follow these steps:**

1. Open the configuration file where you want to create the asset.
2. In the Asset Browser, click Add  at the bottom of the pane.
3. Click Mobile Session.  
The Mobile Session dialog displays.
4. Enter a Name for the asset. This name appears in the Destination field in the step editor. Use a name that is meaningful.
5. Enter a Description that helps you identify the asset.
6. Select a Platform: iOS or Android.
7. In the Application field, enter the full path to your application package file (.apk or .app). Or, click the Folder icon to locate and select the file on your computer.
8. (iOS only) In the Family field, select the iOS device type: iPhone or iPad, iPhone only, or iPad only.
9. Select a Target:
  - Emulator (Android)
  - Simulator (iOS)
  - Attached Device
  - SauceLabs

Fields then vary depending upon the target you select.

10. Define the following fields:

**For an Emulator (Android only)**

Emulator assets specify the mobile emulator that is used for testing on an Android device.

**AVD**

Click the folder icon to locate the Android AVD that you defined when configuring mobile testing.

**SDK Version**

Select the version of the SDK to use with your test case.

**For a Simulator (iOS only)**

Simulator assets specify the mobile simulator that is used for testing on an iOS device.

**Simulator**

Click the folder icon to locate the simulator on your computer.

#### **iOS Version**

Select the version of the iOS to use with your test case.

#### **For an Attached Device**

Attached Device assets specify the mobile device that is used for your test cases. This asset is used for physical iOS and Android mobile devices that are connected to your computer.

#### **Attached**

Click the folder icon to locate the simulator of your choice on your computer.

**Note:** If you connect or disconnect a device, click Refresh in the Attached Device dialog to view the latest devices.

#### **SDK Version (Android only)**

Select the version of the SDK to use with your test case.

#### **iOS Version (iOS only)**

Select the version of the iOS to use with your test case.

#### **For SauceLabs**

SauceLabs is a cloud provider of scalable iOS and Android simulators. SauceLabs assets specify the SauceLabs account information for mobile cloud testing.

**Note:** Before you can create a SauceLabs asset, you must have a SauceLabs account with a unique access key.

#### **User Name**

Defines the user name with which to access SauceLabs.

#### **Access Key**

Defines the key for accessing SauceLabs.

#### **SDK Version (Android only)**

Select the version of the SDK to use with your test case.


#### **iOS Version (iOS only)**

Select the version of the iOS to use with your test case.

11. To define advanced options, click **PRO**.

The Scope field is now available. For more information, see Runtime Scope.

12. Once you enter an application in the Application field, specific information about the application displays in the Details window.
13. If your application is built using both native and browser elements, select the Mixed Mode check box.

14. To verify the asset before saving, click .

15. Click OK.

The new asset appears in the Asset Browser.

**Note:** If you defined multiple mobile assets, you must select one before you can record a test case.

**Important!** Android testing requires the correct version of Android SDK Build-tools. If you encounter an error message regarding zipalign or aapt, see Android SDK Build-tools before you continue.

## Companion Descriptions

**This section contains descriptions of the following companions:**

[Web Browser Simulation Companion](#) (see page 72)

[Browser Bandwidth Simulation Companion](#) (see page 73)

[HTTP Connection Pool Companion](#) (see page 74)

[Configure DevTest to Use a Web Proxy Companion](#) (see page 76)

[Set Up a Synchronization Point Companion](#) (see page 77)

[Set Up an Aggregate Step Companion](#) (see page 78)

[Observed System VSE Companion](#) (see page 79)

[VSE Think Scale Companion](#) (see page 88)

[Batch Response Think Time Companion](#) (see page 90)

[Recurring Period Think Time Companion](#) (see page 92)

[Create a Sandbox Class Loader for Each Test Companion](#) (see page 93)

[Set Final Step to Execute Companion](#) (see page 94)

[Negative Testing Companion](#) (see page 94)

[Fail Test Case Companion](#) (see page 94)

[XML Diff Ignored Nodes Companion](#) (see page 95)

## Web Browser Simulation Companion

The Web Browser Simulation companion lets you simulate various web browsers. The web browsers identify themselves to a web server using the User-Agent HTTP header. You configure DevTest to simulate several user-agents when you are running several virtual users in a staged test. Each user-agent string is assigned a relative weight, allowing one browser to appear more often than others.

To specify the weights, use the default Browser Selection Companion Editor.

To configure the Web Browser Selection companion, enter or edit the list of browser agents and the weights:

Complete the following fields:

**User-Agent**


The browser to simulate.


**Weight**

The weight for this browser. For example, assume you want to assign weights of 25 percent, 25 percent, and 50 percent for three browsers. Enter the weights as 1, 1, and 2 for the three rows, and 0 for the others (or delete the extra rows).

When the test case is run, DevTest selects one of the browsers to emulate: all HTTP transactions sent from DevTest include the User-Agent string for that browser. The selection criteria is random, with each browser weight representing one “chance” that the browser will be selected.

DevTest only selects a browser to emulate when the test case initializes. The browser agent string that DevTest selects remains in effect during the execution of the test case. To simulate a distribution of browsers, run the test case multiple times inside a suite.

To add another user-agent, click Add .

To delete a line, click Delete .



## Browser Bandwidth Simulation Companion

The Browser Bandwidth Simulation companion lets you simulate varied bandwidths for the virtual users. Some testing scenarios call for the simulations of different types of internet connections.

### To configure the Browser Bandwidth Simulation companion:

Complete the following fields:

#### **BytesPerSec**

The connection speed. For example, to simulate a connection speed of 56K, enter a BytesPerSec of 7000 (56000 bits / 8 bits per byte = 7000 bytes per second).

#### **Weight**

The weight that is given to this row. For example, to assign weights of 25 percent, 25 percent, and 50percent to three rows, set the Weight columns of the rows to 1, 1, and 2. In the example that is shown, half the virtual users connect at 6000 bytes per second and half at 100,000 bytes per second.

To add a line, use the Add button.

To delete a line, use the Delete button.

## HTTP Connection Pool Companion

The HTTP Connection Pool companion enables you to limit the number of HTTP connections for each target server. This companion applies only to the HTTP/HTML Request, REST, and Raw SOAP Request test steps.

DevTest typically uses one HTTP connection for each virtual user. For example, if you run a test with 100 virtual users, then the client and the server each have 100 sockets open.

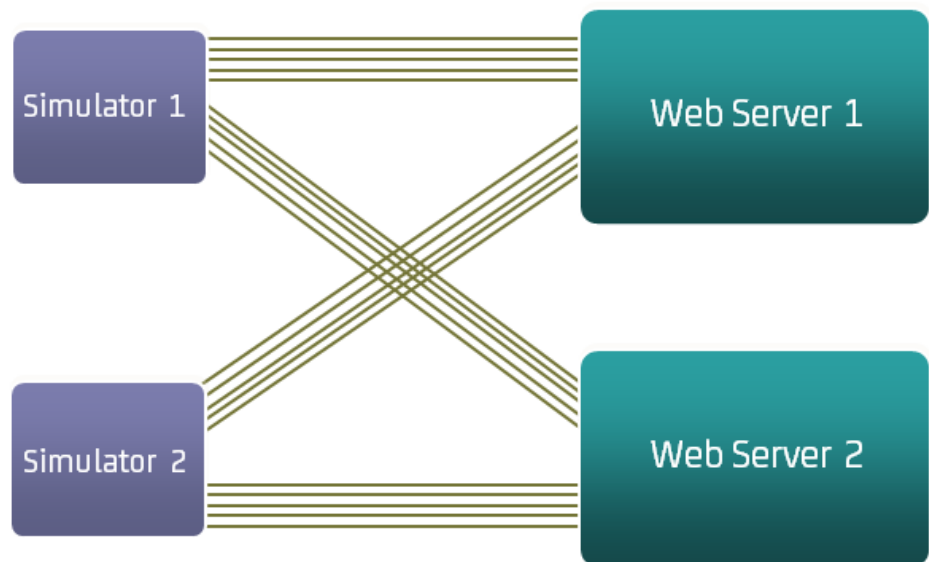
If you run a load test with thousands of virtual users for each simulator, the underlying operating system can run out of available sockets. In this scenario, use the HTTP Connection Pool companion.

The `ConnectionsPerTargetHost` parameter specifies the number of connections to allocate to each unique endpoint.

Assume that a test case has two steps: an HTTP step to go to web server 1, and a second HTTP step to go to web server 2. The test case has the HTTP Connection Pool companion with a setting of five connections for each target host. The staging document is configured to run 100 virtual users. Because there are two simulator servers, they get 50 virtual users each by default.

Simulator 1 creates five connections to web server 1, and five connections to web server 2. Simulator 2 does the same thing. Each web server now has 10 client connections. At the first HTTP step, a virtual user must wait for one of the five connections to web server 1 to become available. The virtual user uses the connection to make the HTTP call, and the connection goes back into the pool.

The following graphic illustrates this scenario. Simulator 1 has five connections to web server 1 and five connections to web server 2. Simulator 2 has five connections to web server 1 and five connections to web server 2.



## Configure DevTest to Use a Web Proxy Companion

The Configure DevTest to use a Web Proxy companion lets you set up a proxy for all web testing steps. If your environment dictates the use of a proxy, use this companion. Proxy information is specific to your organization. Consult your operations team for the proxy settings for your company.

To configure the Web Proxy Setup companion, enter the following parameters in the Web Proxy companion editor.

### **Web Proxy Server (Host & Port)**

The name or IP address of the proxy server in the first field, and the port number in the second field.

### **Bypass web proxy for these Hosts & Domains**

Names of the domains and hosts for which you want proxy to be bypassed.

### **Secure Web Proxy Server (SSL Proxy Host & Port)**

The name or IP address of the SSL proxy server in the first field, and the port number in the second field.

### **Bypass secure web proxy for these Hosts & Domains**

Names of the domains and hosts for which you want the secure proxy to be bypassed.

### **Exclude Simple Hostnames**

Select to exclude hostnames like localhost or servername.company.com.

### **Proxy Server Authentication**

The domain name with the user name and password, if necessary, for authenticating to the proxy server.

### **Send Preemptively**

Select Wait for Challenge, Send BASIC, or Send NTLM.

DevTest can also use its local.properties file to assign a web proxy for all test cases. This file is in the DevTest home directory. Update the **lisa.http.webProxy.host** and **lisa.http.webProxy.port** properties as appropriate and restart DevTest. If you do not already have a local.properties file in the DevTest home directory, rename the existing \_local.properties to local.properties, and use it.

## Set Up a Synchronization Point Companion

Use the Create a Synchronization Point companion to select a test step to use as a synchronization point in a test case or a test suite. At a synchronization point, virtual users pause and wait until each one has reached this step. Then all virtual users are released to execute the step simultaneously. This capability is useful when setting up load tests for concurrent testing or peak resource utilization.

For example, you could set up a 100-user test so all users log in to your application and order the same seat in a theater simultaneously.

Synchronization points apply to a single test or you can apply them to all tests in a test suite. In test suites, the synchronization point name must be the same across the suite, but the At Step can be different. Multiple test scenarios (and test suites) must be set to run in parallel, because serial tests cannot access the synchronization point simultaneously by definition. Multiple synchronization points can be defined in a test case or test suite.

To configure the Create a Synchronization Point companion, enter the following parameters:

**Sync Point Name**

The name that you specify for the synchronization point.

**At Step**

Select the step for the synchronization point from the drop-down list. The virtual users pause before executing this step.

**Time out secs (0 for none)**

The number of seconds to wait for the synchronization to occur. All virtual users must reach the At Step before the timeout period elapses.

## Set Up an Aggregate Step Companion

The Set Up an Aggregate Step companion lets you aggregate and report several physical test steps as one logical step for metrics collection and reporting. The companion automatically sets all included steps to Quiet.

Enter the following parameters in the Set Up an Aggregate Step editor:

**Aggregate name**

Defines the name for the aggregation step. Spaces are allowed.

**Starting step**

Specifies the start (first) step of the aggregation.

**Participants**

Select the steps to include in the aggregate (exclude the Starting and Ending steps).

**Ending step**

Specifies the end (last) step of the aggregation from the pull-down menu.

All reports show the aggregate step.

A test case can have only one Aggregate Step companion.

## Observed System VSE Companion

Before LISA 6.0.6, the response time for an inbound request came from the think time specification on the specific response that was determined for that request. Sometimes, primarily during load and performance testing scenarios, the response times should be modeled after the times from a live system. For example, a live system can show a drop in performance equivalent to twice the nominal response times during peak load. In this case, it is helpful to have VSE emulate this response time curve. It is also helpful to let VSE cover (or play back) this curve over an arbitrary interval to allow, for example, a 12-hour period of observed response time metrics to be played out over a 3-hour test.

The VSE Observed System companion supports these requirements. If a virtual service will provide this behavior, add and configure this companion.

### Configuration Information

The Observed System companion requires the following parameters:

#### Start date/time and End date/time

Defines a time "window" in which to read observed response times. These timestamps are inclusive. Timestamp data from your data set or data provider that is outside this window is ignored.

#### Assumed run length

Defines a duration that represents the interval over which the virtual service "fits" the response time curve. For the previous example, this value would be set to three hours.

For example, with the following values:

- The assumed run length is 30 minutes
- The Start date/time window is 30 minutes
- The End date/time window is 30 minutes
- The buffer size is 10 minutes

we get the buffer for the first 10 minutes, then for 10-20 minutes, then for 20-30 minutes, then 45-60 minutes.

With the following values:

- The assumed run length is 15 minutes
- The Start date/time window is 30 minutes
- The End date/time window is 30 minutes
- The buffer size is 10 minutes

The first 10 minutes plays back in the first 5 minutes, the second 10 minutes plays back in the second 5 minutes, and the third 10 minutes plays back in the third 5 minutes.

### **Buffer size**

Defines a duration in minutes and can be used to control how much of the response time data is acquired at once. The parameter defaults to one hour of data at a time.

For example, an assumed run length of 1 hour and a buffer size of 15 minutes provides the following results:

- We get the buffer for the first 15 minutes
- Then for 15-30 minutes
- Then for 30-45 minutes
- Then 45-60 minutes
- Then refetch the data from the first 15 minutes

### **Observed System data provider**

This parameter tells the companion where to get the data. We currently provide a DevTest data set data provider and a CA Application Performance Management (Wily) data provider.

Any data provider must provide three pieces of information:

- An ID (which is a string)
- A timestamp
- The response time at the timestamp

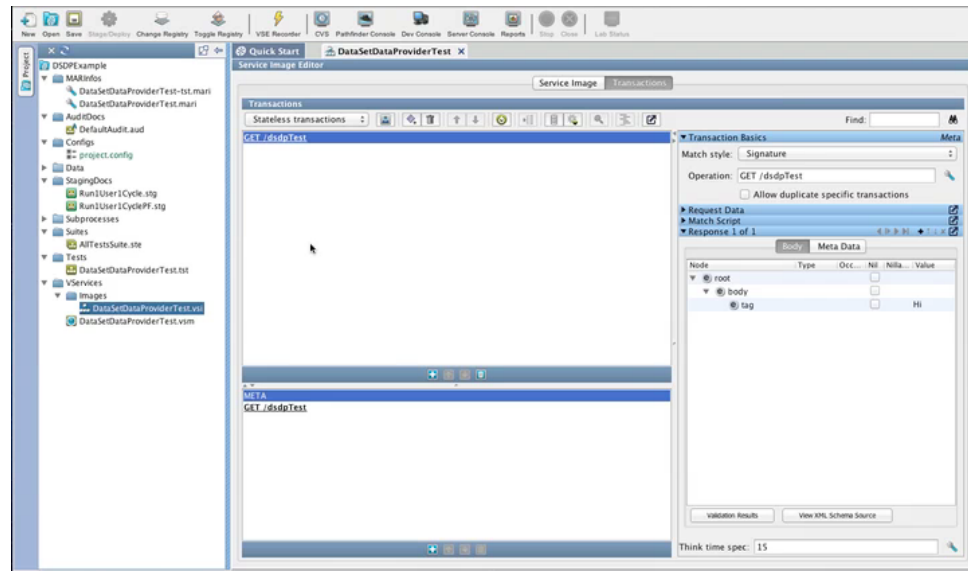
The DevTest data set data provider requires that the data set provide this information in fields labeled "id", "timestamp" and "responseTime", respectively. The timestamp value, if not an actual Date object, must be a string in the form "yyyy-MM-dd HH:mm:ss.SSS". How the ID maps to any specified inbound request is data-provider specific. For the data set provider, it must match the operation of the request. The CA Application Performance Management (Wily) provider uses a regular expression-based approach.

### **Data Set Source Example**

In this example, a data set provides the input for the Observed System VSE companion. The definition of the data that this companion provides lets you change the response time for a transaction, or for multiple transactions over time.

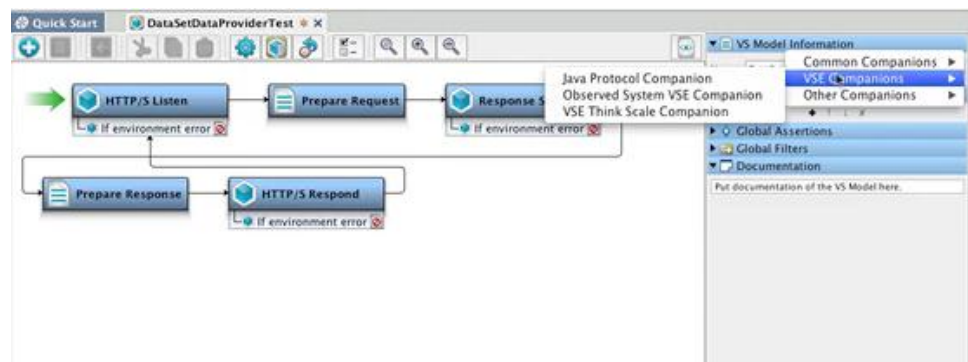
The service image that is shown contains one transaction, and the Think time spec is set to 15 milliseconds.





The virtual service model that is associated with this service image has a few simple steps.

1. To add a companion, click  Add under the Companions panel.
2. Select VSE Companions, Observed System VSE Companion.





The top part of the panel provides general information about the parameters for the companion.

In this example, the Start date/time and End date/time define a two-hour window, but the Assumed run length is set to one hour. Therefore, for every one hour of VSE run time, the companion goes through two hours of data from the data set data provider. The buffer size of 30 minutes means that VSE accesses the data set every 30 minutes to retrieve data. The buffer size is before scaling. Therefore, with a 30-minute buffer:

- VSE retrieves data from the data set between 10:22 and 10:52.
- VSE scales (in this case) by half so it can calculate correctly.

To disable the companion temporarily, clear the Enabled check box.

Any data providers, including DevTest data sets and the Wily Observed System Data Provider, can back the Observed System Companion. Click (click here to select), select Data Set Source, and the Select Data Set Type button appears. Click Select Data Set Type, select Common DataSets, then select Create your own Data Sheet.

▼ Observed System VSE Companion - Observed System VSE Compan...  

Start date/time: Jul 1, 2012 10:22

End date/time: Jul 1, 2012 12:22

Assumed run length: 1 hours / 2h

Buffer size: 30 minutes

☒ Enabled

Observed System Data Provider Data Set Source

Select Data Set Type

Common DataSets ▾

- Read rows from a delimited data file
- Create your own Data Sheet
- Create your own set of Large Data
- Read rows from a JDBC table
- Create a numeric counting data set
- Read rows from Excel file
- Read data objects from Excel file
- Unique Code Generator
- Random Code Generator
- Message/Correlation ID Generator
- Load a set of File names
- XML Data Set

When the table for Create your own Data Sheet opens, the columns are prepopulated with:

**id**

Matched against the Operation Name when the request is being processed.

**timestamp:**

**responseTime**

If the standardDeviation value is 0, the responseTime defines the interval to use during playback. Use the mathematical formula that is described for the standardDeviation column for non-zero values. This response time calculation overrides the think time spec of 15 milliseconds that was set in the service image.

**standardDeviation**

Defines the statistical data that represents the standard deviation from the mean value of the response time. The response time that is used during playback is within +/- 3sigma of the average response time. If **x** is the responseTime and **y** is the standardDeviation, the response time during playback is a random value in the range (**x -3 y**) and (**x +3 y**).

These columns are required for any type of data set provider that you use to provide information for this companion.

In this example, copy the Operation Name of GET / dsdpTest from the service image and enter it in the id field. In the responseTime field, enter 150.

▼ Observed System VSE Companion - Observed System VSE Companion

Start date/time: Jul 1, 2012 10:22

End date/time: Jul 1, 2012 12:22

Assumed run length: 1 hours / 2h

Buffer size: 30 minutes

☒ Enabled

Observed System Data Provider

Data Set Source

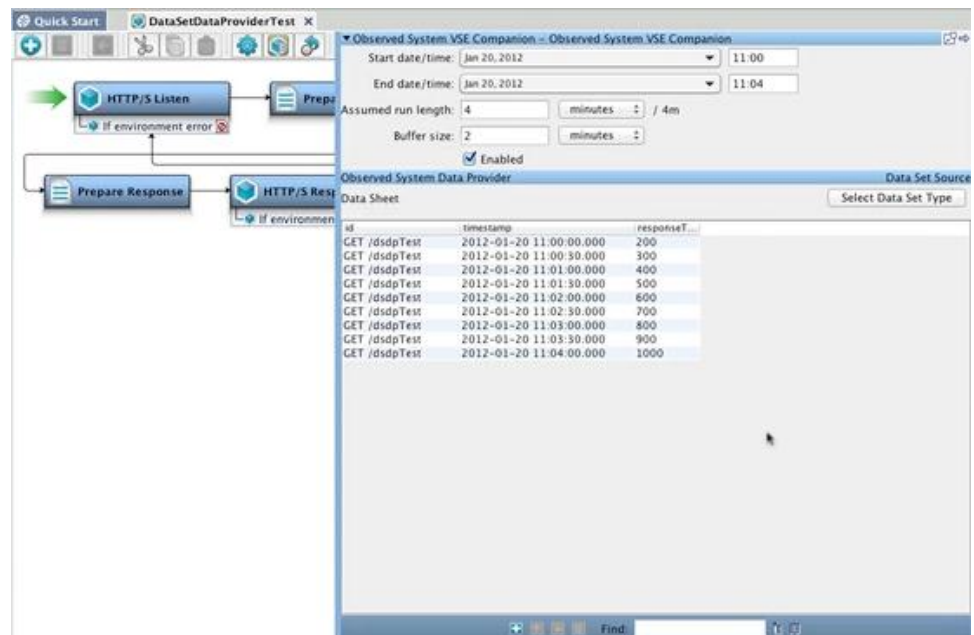
Data Sheet

Select Data Set Type

id	timestamp	responseTi...
GET / dsdpTest	2012-07-01 10:30:00:00.000	150

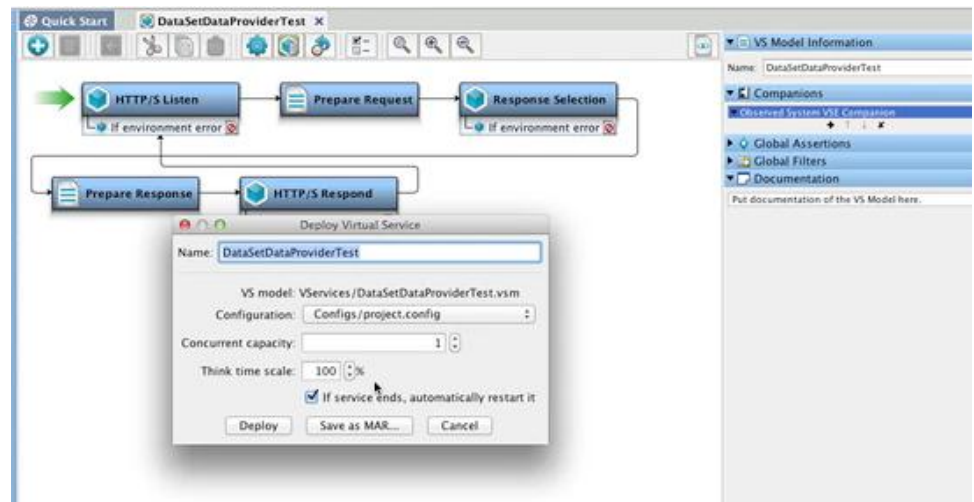
In the timestamp field, enter a date and time that falls into the time window that Start date/time and End date/time define.

To continue the example, we use an existing project with a prepopulated data set.



The data set response times define an increase of 100 milliseconds every 30 seconds. Therefore, this companion should always override the Think time spec of 15 milliseconds that is set in the virtual service model.

**Note:** When you deploy the virtual service, do not enter a Think time scale of 0 percent.



### Observed System Data Provider (Wily) Source

The Observed System VSE Companion can also receive input from the CA Application Performance Management (Wily) application. To configure the companion to work with Wily, enter the configuration information in the top part of the panel. Then click (click here to select), and select Data Set Source of Wily Source.

Parameters for the Wily Observed System companion are:

**Web Service URL**

Specifies the URL for the Wily web service.

**Agent Regex**

Specifies a regular expression that identifies the agent.

**Metric Regex**

Specifies a regular expression that identifies the metric. This metric RegEx is wrapped as the middle part of the metric RegEx that is sent to the Wily server. The metric RegEx sent to the server is:

```
".*WebServices\|Server\|<user entered pattern>:Average Response Time.*"
```

For example, if you want to query the metric:

```
"WebServices\|Server\|http_//ejb3\.examples\.itko\.com/\|(.*)Average Response Time \ (ms\)"
```

you can input

```
"http_//ejb3\.examples\.itko\.com/"
```

This example fetches all the Average Response Time metrics for the web service. To retrieve the metrics for specific operations, use a custom RegEx.

**Default:** ".\*"

**Service Username**

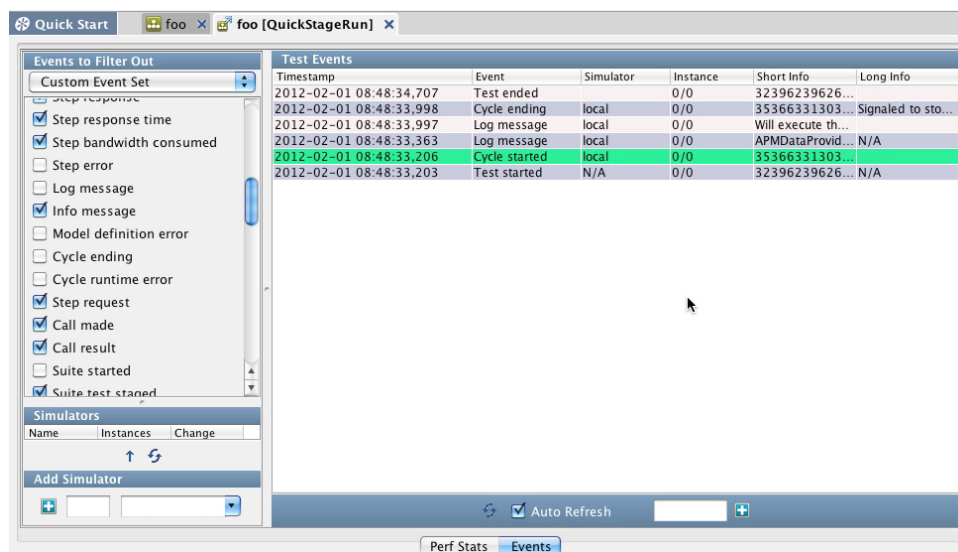
Specifies the user name used to access the Wily service.

**Service Password**

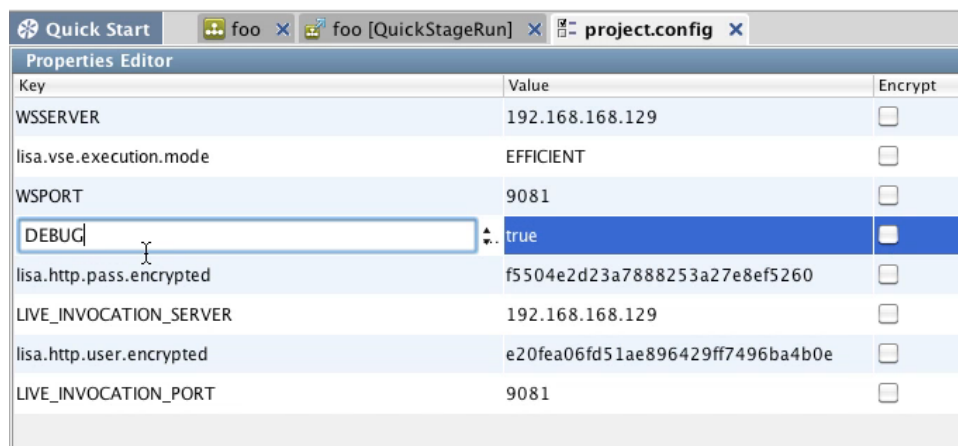
Specifies the password that is associated with the Service Username, if necessary.

To test the behavior of the companion, stage a quick test.

The test case in this example had one step, an output log message step. You can see the output in the Test Events tab of the Quick Stage Run window.



In the config file that is used for this test case, setting the property debug to true means that the log message appears in the output.



### Run Time Priorities

The process that is followed at run time is:

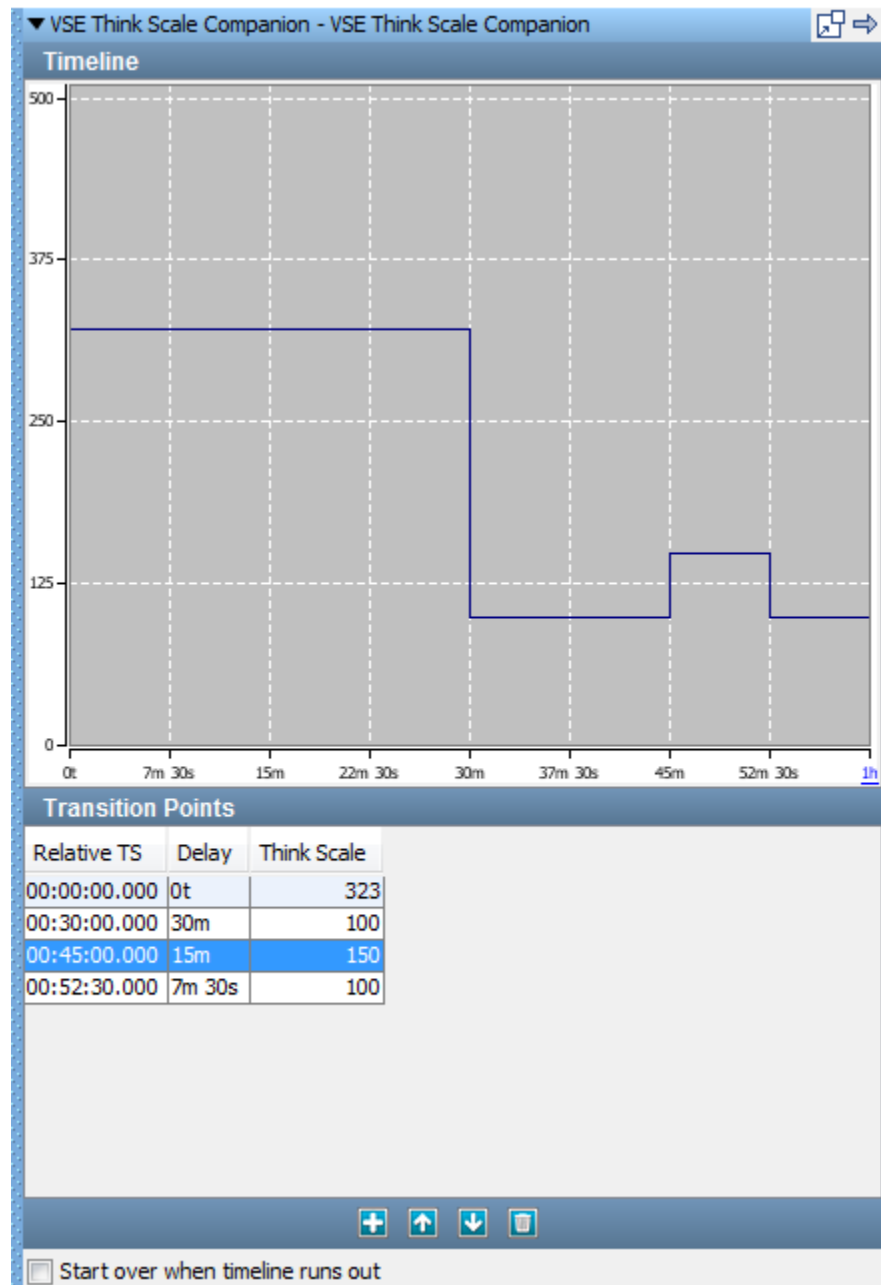
- The think time specification from the VSE response is examined when the delay factor is being determined.
- If the think time specification contains state (that is, a double-brace expression), it is evaluated directly and the result is used as the response time for the response.
- If the think time specification does not contain state and the virtual service contains the Observed System companion, the companion is asked to determine the response time for the response.
- If the companion is not present or cannot determine a response time, the think time specification is used as the response time.

For example, an assumed run length of 1 hour and a buffer size of 15 provides the following results:

- We get the buffer for the first 15 minutes
- Then for 15-30 minutes
- Then for 30-45 minutes
- Then 45-60 minutes
- Then refetch the data from the first 15 minutes

## VSE Think Scale Companion

The VSE Think Scale Companion can be added to a virtual service model. This companion allows the think scale percent for the service to change over time by specifying the graph of think scale changes over time.





Click Add, Delete, and Move to add, delete, and reorder Transition Points. You can directly edit the Delay and Think Scale entries. You can also click and drag the lines in the Timeline display to indicate the delay and scale that you want. The Transition Points table is updated to correspond.

**Relative TS**

Calculated based on the delay you enter, in format hh:mm:ss.ms.

**Delay**

The interval, in minutes and seconds, to wait before the specified think scale is applied.

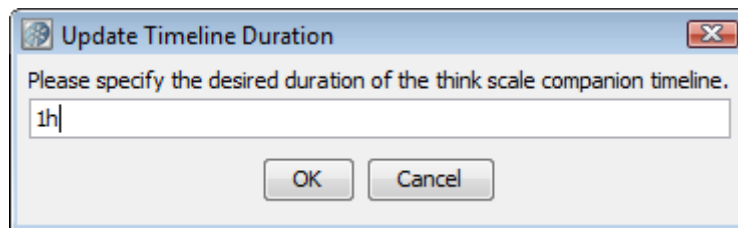
**Think Scale**

The think scale is a percentage that is applied to the think times in the responses.

**Start over when timeline runs out**

If the end of the timeline is reached, start from the first Relative TS.

If you click the label for the right-most tick mark on the horizontal axis, you can adjust the total timeline of the companion. This label is the "1h" label in the previous window. The Update Timeline Duration dialog opens. If you update the timeline duration to be shorter than your transition points, you receive a warning that some transition points are removed.



As you move your cursor over the graph, tooltips display that show time on vertical lines and think scale percentages on horizontal lines.

## Batch Response Think Time Companion

The Batch Response Think Time companion varies the think scale so that a virtual service model sends a defined number of responses at specified times.

This companion lets you batch responses for a virtual service model. You can specify a schedule for sending responses. For example, you can send 100 responses every day at 8:15. The requests can arrive at any time, but VSE only responds at 8:15. If there are fewer than 100 pending responses, VSE sends all the responses. If there are more than 100 pending responses, VSE sends 100 responses and saves the rest until the next batch.

VSE only responds to incoming requests; it does not generate unsolicited responses. If VSE receives 12 requests, it sends 12 responses, even if the Quantity is greater than 12.

This companion is useful only for the SAP transport protocol.

Enter the following parameters in the companion editor:

**Enabled**

Specifies whether to enable the companion.

**Time**

Defines when to send the responses.

**Values:** 00:00 to 23:59.

**Quantity**

Defines the maximum number of responses to send.

Click Add, Up, Down, and Delete to add, move, and delete rows in the Response Schedule table.

**Repeat daily**

Specifies whether to send responses each day.

**Send all at once**

Specifies how to distribute the responses.

**Values:**

- **Selected:** Sends the responses all at once or as the requests arrive for the specified interval until the size limit is reached.
- **Cleared:** Spreads the responses evenly between the time for the current batch until the time for the next batch or the scheduled end time.

**Schedule End Time**

If the Send all at once check box is cleared, this field defines the end time for the last-scheduled time in the Response Schedule table.

**Values:** 00:00 to 23:59.

**Limits:** The value must be later than the last-scheduled time in the Response Schedule table.

**Example:**

Assume that the Response Schedule table contains two entries:

- The first entry has the values 08:00 and 10.
- The second entry has the values 08:10 and 10.

Also assume that the Send all at once check box is cleared and the Schedule End Time field is set to 08:15.

When you deploy the virtual service model, the results of this configuration are:

- One response each minute between 08:00 and 08:10
- One response every 30 seconds between 08:10 and 08:15.

These results assume that the service receives enough requests with matching responses during those periods.

## Recurring Period Think Time Companion

The Recurring Period Think Time companion adjusts response think time to send a batch of responses for every time period that you specify.

This companion lets you batch responses for a virtual service model. You can specify a schedule for sending responses. For example, you can send 100 responses every hour. The requests can arrive at any time, but VSE only responds on the hour. If there are fewer than 100 pending responses, VSE sends all the responses. If there are more than 100 pending responses, VSE sends 100 responses and saves the rest until the next batch.

VSE only responds to incoming requests; it does not generate unsolicited responses. If VSE receives 12 requests, it sends 12 responses, even if the Quantity is greater than 12.

This companion is useful only for the SAP transport protocol.

Enter the following parameters in the companion editor:

**Enabled**

Specifies whether to enable the companion.

**Start**

Defines when the timer starts.

**Values:**

- Immediately
- On the Quarter Hour
- On the Half Hour
- On the Hour

**Every**

Defines the number of minutes or hours to wait between sending responses.

**Quantity**

Defines the maximum number of responses to send.

**Repeat daily**

Specifies whether to send responses each day.

**Send all at once**

Specifies how to distribute the responses.

**Values:**

- **Selected:** Sends the responses all at once or as the requests arrive until the size limit is reached.
- **Cleared:** Spreads the responses evenly over the configured period.

**Example:**

Assume that the Start field is set to Immediately and the Every field is set to 5 minutes. Also assume that the Quantity field is set to 10 and the Send all at once check box is cleared.


When you deploy the virtual service model, the result of this configuration is one response every 30 seconds. This result assumes that the service receives enough requests that have matching responses during those periods.

## Create a Sandbox Class Loader for Each Test Companion

The Create a Sandbox Class Loader companion lets you verify that every test run executes in its own Java Class loader (JVM). This companion is valuable when testing local Java objects that are not designed for multithreaded or multiuser access. Most testing does not require this companion. It is usually only necessary when testing local Java objects with the Dynamic Java Execution step.

To configure the Class Loader Sandbox companion, use the Class Path Sandbox companion editor.

- If the hotDeploy directory contains the class you want to run, select the Add Hot Deploy Path Entries check box.

If the hotDeploy directory does not contain the class you want to run, click Add  to add a line in the Class Path Directories list, then add the appropriate class path.

**Note:** Any Java objects that you want to edit or run **must** be in the class path or the hotDeploy directory. They **must not** be in the DevTest lib or bin directories. The Class Loader Sandbox will not work because of the way the Java VM loads classes.

## Set Final Step to Execute Companion

Use the Set Final Step to Execute companion to verify that the system under test is left in a consistent state regardless of the test result. You specify which step is always run last when the normal test flow is circumvented.

A common use is to ensure that resources are released at the end of the test. Specifying the first step is not commonly needed, but there are many scenarios where specifying the final step is important.

The final step is executed even if the test case reaches an End step or the test case is instructed to end abruptly.

The final step is not shown in the Execution History list in the ITR. The results can be seen in the Events tab for the last step that was executed.

The Set Final Step companion Editor is used to set the final step.

To configure the Set Final Step companion, enter the following parameter:

### **Final Step**

Select the final step to execute from the drop-down list.

A step that is designated as the final step to execute is noted with a green flag icon.

## Negative Testing Companion

The Negative Testing companion is useful when you want all your steps to fail. To cause a normal-ending test case to fail if any contained test step passes, use this companion.

This companion has no configuration parameters.

## Fail Test Case Companion

If a step in a test case has an error, the Fail Test Case companion marks a test case as failed, even if it ends successfully. The companion registers an event listener for the EVENT\_TRANSFAILED event.

**Example:** A WSDL validation assertion on a web service step has an error. You want to be informed that the validation failed, but you also want the test case to continue.

This companion has no configuration parameters.

## XML Diff Ignored Nodes Companion

The XML Diff Ignored Nodes companion lets you enter XPath expressions that return one or more nodes in the following table. The companion uses an OR operation to collect these nodes, then ignores them during all XML diff comparisons for this test case.

## Data Set Descriptions

**This section contains descriptions of the following data sets:**

[Read Rows from a Delimited Data File Data Set](#) (see page 96)

[Create Your Own Data Sheet Data Set](#) (see page 98)

[Create Your Own Set of Large Data Data Set](#) (see page 101)

[Read Rows from a JDBC Table Data Set](#) (see page 103)

[Create a Numeric Counting Data Set](#) (see page 104)

[Read Rows from Excel File Data Set](#) (see page 105)

[Read DTOs from Excel File Data Set](#) (see page 107)

[Unique Code Generator Data Set](#) (see page 113)

[Random Code Generator Data Set](#) (see page 114)

[Message-Correlation ID Generator Data Set](#) (see page 115)

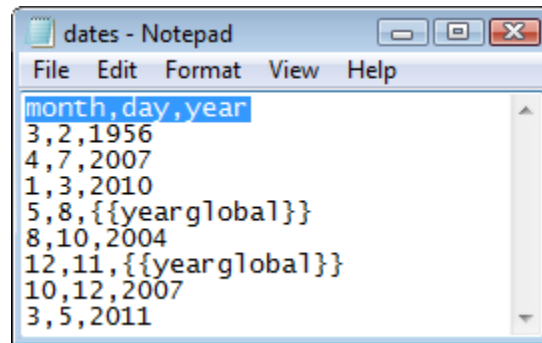
[Load a Set of File Names Data Set](#) (see page 116)

[XML Data Set](#) (see page 118)

## Read Rows from a Delimited Data File Data Set

The Read Rows from a Delimited Data File data set assigns values to properties based on the contents of a text file. It is the most commonly used type of data set in DevTest Solutions. The first line of the text file specifies the names of the properties into which the data values are stored. The subsequent lines list the data values to be used for these properties. The text file is created using a simple text editor.

The following example shows a comma-delimited data file. The first row shows the property names in this data set.



The Data Set Editor is used to define the data set.

Complete the following fields:

**Name**

Enter the name of the data set.

**Local**

Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

**Random**

Whether the record after the current record (sequential access) is read, or a random record is read. Sequential reading is the default.

**Max Records to Fetch**

The upper bound on the number of records to fetch for random access. This text field is disabled if the Random check box is not selected.

**At End Of Data**

Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

**File Location**

The full path name of the text file, or browse to file with the Browse button.



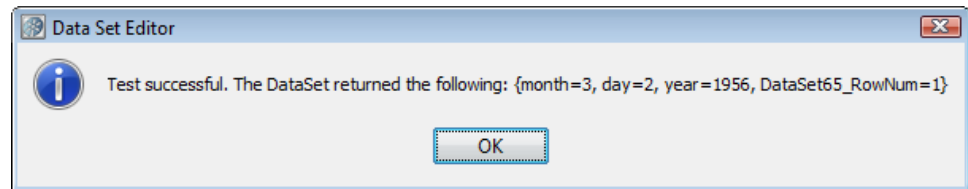
**File Encoding**

Accept the default encoding of UTF-8 or select an alternate encoding from the drop-down list. You can also select Auto-detect and click the Detect button to have an encoding type selected for you.

**Delimiter**

The delimiter being used. Any character can be a delimiter. The drop-down list contains common delimiters.

Click the Test and Keep button to test and load the data. You get a message that confirms that the data set can be read, and shows the first set of data.



## Create Your Own Data Sheet Data Set

The Data Sheet data set lets you generate your data set data in DevTest, without requiring any reference to an external file.

The Data Sheet consists of a data table. The column headings specify the property names and the table rows specify the data values for those properties. The table skeleton is built by specifying the number of rows and the column names (properties). The default name is **Create your own Data Sheet**. Subsequent data sheets that are created have "~number" appended to the data sheet name.

Complete the following fields:

### Name

Enter the name of the data set.

### Local

Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

### Random

Whether the record after the current record (sequential access) is read, or a random record is read. Sequential reading is the default.

### Max Records to Fetch

The upper bound on the number of records to fetch for random access. This text field is disabled if the Random check box is not selected.

### At End Of Data

Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

### Number of Rows

The initial estimate of the number of rows. This value can be modified later.

### Column Names

Comma delimited list of column names. These names are also the property names.

Click the Create Data Sheet Skeleton button and enter your data into the table.

To sort ascending or descending on a column, click the column labels. Right-click the column label for a column menu.

### Encrypt Column

To encrypt all values in the column, select this option. An encrypted column has a lock icon and all values appear as a row of asterisks. Exporting shows that the column is <name>\_enc and the data is encrypted.

**Change column name:**

To rename the column, select this option.

**Sort Ascending**

To sort the column values ascending, select this option. To change the sort to descending, click again.

**Reset Sort**

To sort the column values as they were originally, select this option.

**Column Resize Mode:**

Select Automatic, Subsequent, Next, Last, or Manual.

**Maximize All Columns**

Selecting this option changes the Column Resize Mode to Manual. The columns are all displayed, and scroll bars are available.

To create and modify the table, use the functions in the bottom toolbar.

**Add**

Add rows to the table.

**Up and Down**

Select a row and move it up or down in the table.

**Delete**

Delete the selected row.

**Add Column**

Add a column to your table.

**Delete Column**

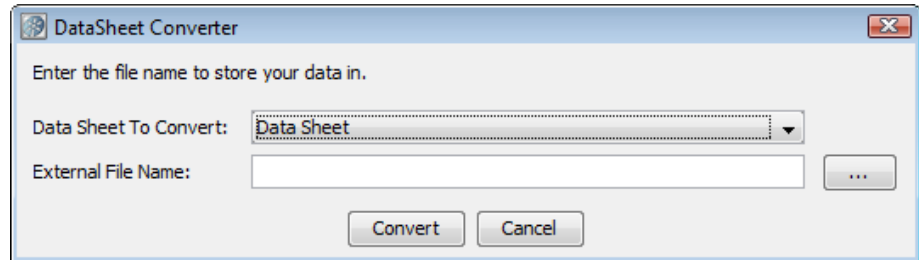
Delete a column. Select a cell in the column you want deleted, making sure that the cell is selected for editing, and click this button.

Click the Test and Keep button to test and load the data. You get a message that confirms that the data set can be read, and shows the first set of data.



You can also complete the following actions:

- To sort the rows, double-click the column header.
- To change the column name, right-click on the column name
- To select the toolbar functions and Launch Extended View to edit the cell, select and then right-click a cell.
- To convert your data sheet to a data set or a file, click Convert to Data Set at the bottom of the panel.



#### Data Sheet to Convert

By default, the data sheet you created, or select from the pull-down list.

#### External File Name

The name and path of the file you want to create from the data sheet.

**Note:** Moving the rows up or down in the table can affect the outcome of a test. The order of columns does not affect the outcome of the test.

Select the steps that use the data sheet.

## Create Your Own Set of Large Data Data Set

The Create Your Own Set of Large Data data set lets you define a custom data table that can be arbitrarily large. The data can have any number of rows and columns. A backing file name is the file in which all the data is stored.

Complete the following fields:

**Name**

Enter the name of the data set.

**Local**

Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

**Random**

Whether the record after the current record (sequential access) is read, or a random record is read. Sequential reading is the default.

**Max Records to Fetch**

The upper bound on the number of records to fetch for random access. This text field is disabled if the Random check box is not selected.

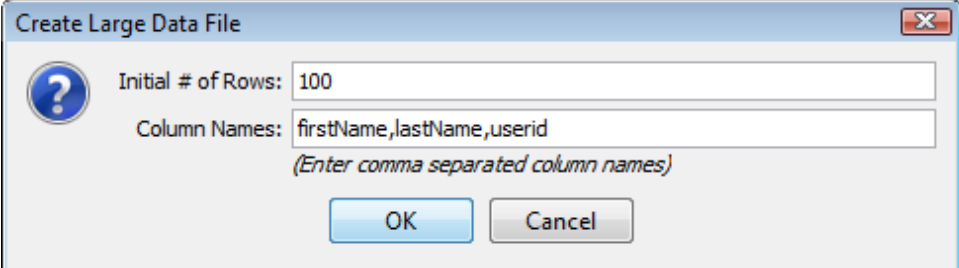
**At End Of Data**

Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

**Backing file name**

The name of the file in which data is stored. This file is created automatically and data that you supply is inserted in it.

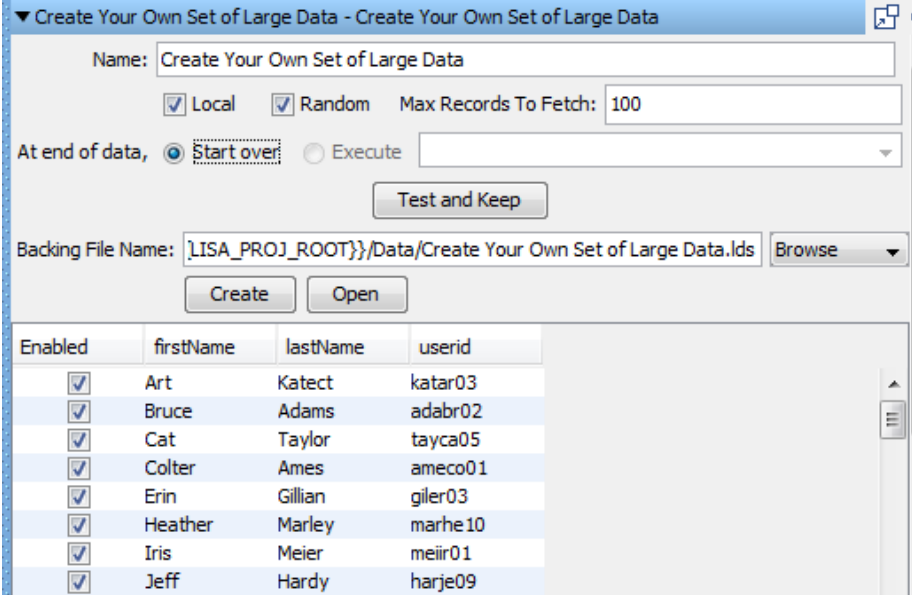
The Create button opens a panel that lets you specify more parameters for file creation.

**Initial # of Rows**

Defines the initial number of rows to create. You can use the editor to add more rows.

**Column Names**

Defines comma-separated column names that go in the data set.



Enabled	firstName	lastName	userid
<input checked="" type="checkbox"/>	Art	Katect	katar03
<input checked="" type="checkbox"/>	Bruce	Adams	adabr02
<input checked="" type="checkbox"/>	Cat	Taylor	tayca05
<input checked="" type="checkbox"/>	Colter	Ames	ameco01
<input checked="" type="checkbox"/>	Erin	Gillian	giler03
<input checked="" type="checkbox"/>	Heather	Marley	marhe10
<input checked="" type="checkbox"/>	Iris	Meier	meir01
<input checked="" type="checkbox"/>	Jeff	Hardy	harje09

Press the Test and Keep button after entering data in the columns and data is copied in the backing file created.

## Read Rows from a JDBC Table Data Set

The Read Rows from a JDBC Table data set is used to read source test case data from a database. The data is read using a JDBC driver (which must be supplied by the user). Each column in the table of data is represented as a property. The data set then loops across the rows that were returned from the SQL query.

Complete the following fields:

**Name**

Enter the name of the data set.

**Local**

Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

**Random**

Whether the record after the current record (sequential access) is read, or a random record is read. Sequential reading is the default.

**Max Records to Fetch**

The upper bound on the number of records to fetch for random access. This text field is disabled if the Random check box is not selected.

**At End Of Data**

Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

**Driver Class**

Enter or select the full package name of the appropriate driver class. Standard driver classes are available in the pull-down menu.

**Connect String**

The connect string is the standard JDBC URL for your database. Enter or select the URL. The pull-down menu contains JDBC URL templates for common database managers.

**User ID**

Enter a user ID (if the database requires it).

**Password**

Enter a password (if the database requires it).

**SQL Query**

The SQL query that is used to create the data set.

Click the Test and Keep button to test and load the data. You get a dialog window that confirms that the data set can be read, and shows the first set of data.

## Create a Numeric Counting Data Set

The Create a Numeric Counting data set assigns a number to a property. The number assigned starts at a specific value and changes by a fixed amount every time the data set is used until it exceeds a known limit. This data set is used to simulate a "for" loop, or to set the number of times something occurs. The following example shows how to use the Create a Numeric Counting data set to make 100 calls to the same step.

Complete the following fields:

**Name**

Enter the name of the data set.

**Local**

Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

**Random**

Whether the record after the current record (sequential access) is read, or a random record is read. Sequential reading is the default.

**Max Records to Fetch**

The upper bound on the number of records to fetch for random access. This text field is disabled if the Random check box is not selected.

**At End Of Data**

Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

**Property Key**

The name of the property into which the counter value is stored.

**From**

The initial counter value.

**To**

The final counter value.

**Increment**

The counter step amount. To use counter data set to count backwards, set a negative Increment value.

Click the Test and Keep button to test and load the data. A message confirms that the data set can be read, and shows the first set of data.

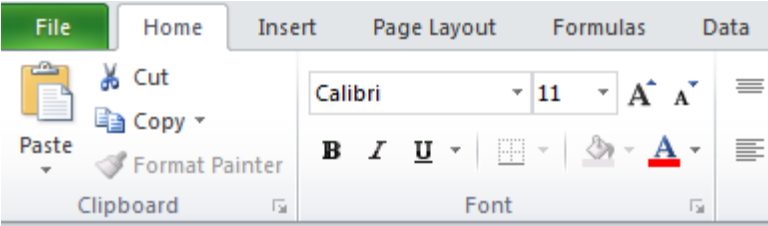


## Read Rows from Excel File Data Set

The Read Rows from Excel File data set assigns values to properties based on the contents of an Excel spreadsheet.

The first nonblank row of the Excel spreadsheet specifies the names of the properties to which the data values are assigned. The subsequent rows list the data values to be used for these properties. The first full row of empty cells is treated as the end of data.

For example, you can test a set of first name, last name, user ID, and password combinations.



	A	B	C	D	E	F
1						
2		firstName	lastName	id	password	
3		Lynn	Parker	parly03	parkpass	
4		Paul	Martin	marpa02	martpass	
5		Jason	Sauer	sauja01	saupass	
6						

Complete the following fields:

### Name

Enter the name of the data set.

### Local

Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

### Random

Whether the record after the current record (sequential access) is read, or a random record is read. Sequential reading is the default.

### Max Records to Fetch

The upper bound on the number of records to fetch for random access. This text field is disabled if the Random check box is not selected.

### At End Of Data

Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

**File Location**

Enter the full path name of the Excel file, or browse to file with the browse button. You can use a property in the path name (for example, LISA\_HOME).

**Sheet Name**

Enter the name of the sheet in the Excel spreadsheet.

Click the Test and Keep button to test and load the data. You get a message that confirms that the data set can be read, and shows the first set of data.

Select the steps that use the data set.

You can click the Open XLS File button to open and edit the Excel spreadsheet.

**Note:** Excel File data sets and Excel DTO data sets can use Excel 2007 or later spreadsheets. Excel DTO data sets are still created using the XLS format.

## Read DTOs from Excel File Data Set

The Read DTOs from Excel File data set lets you parameterize Java data transfer objects (DTOs) in your test steps. The data set gives you an easy way, through Excel spreadsheets, to provide data values for those parameters.

The Read DTOs from Excel file data set assigns values to the properties of a DTO and stores the object in a property. This property can then be used whenever the DTO is required as a parameter. The data in the data set can be simple data types like numbers or strings, or complex data types such as DTOs, arrays, and collections. The data that is represented in Excel is converted into the proper data types automatically when needed. The only complex part of using this data set is the initial creation of the Excel spreadsheet. Fortunately, the creation is done for you. Given the package name of the DTO, a template is created, using one or more Excel sheets that represent the object. Data types such as primitives, strings, arrays of primitives and simple individual DTOs can be represented on a single sheet. More complex data types, such as arrays of objects, require more Excel sheets to represent the full DTO.

Often, a web service endpoint expects complex DTOs. The Excel data set simplifies creating objects to use as parameters to the web service. When the web service is first referenced, it is given a name and a URL for the WSDL. Java DTO classes in the form **com.lisa.wsgen.SERVICENAME.OBJECTNAME** are automatically generated and made available on the classpath. You can browse to that generated class in the DTO class browser, generate an Excel file, and simply fill in the template. See the following example.

Building the data set is a two-step process. First, let DevTest build the template in Excel. Then open the Excel spreadsheet and fill in the data fields in all the sheets that are produced.

### To build the template:

1. Enter the following parameters in the Data Set Editor:

#### **Name**

The name of the data set. This name becomes the property that is used to store the current DTO object.

#### **Local**

Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

#### **Random**

Whether the record after the current record (sequential access) is read, or a random record is read. Sequential reading is the default.

#### **Max Records to Fetch**

The upper bound on the number of records to fetch for random access. This text field is disabled if the Random check box is not selected.

#### **At End Of Data**

Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

#### **File**

The fully qualified path name, or browse to the Excel file using the browse pull-down menu.

#### **DTO Class Name**

The full package name, or browse to, the DTO object. The class file must be on Test Manager. Your class can be copied to the hotDeploy directory to put it on the Test Manager.

Advanced Settings let you specify:

#### **Use flattened child property notation during generation**

Select to override child property flattening during Excel DTO data set generation. If cleared, child properties are generated as references with their own worksheets.

#### **Use new empty cell semantics for flattened properties**

Empty cell semantics for flattened properties means how empty cells are interpreted in a DTO spreadsheet. For example, given the following flattened properties:

```
{ "prop1.subprop1", "prop1.subprop2" }
```

if both subprop1 and subprop2 have empty cell values, then under the new semantics the reference to "prop1" is set to null. Under the old semantics, prop1 would be not null, but references to subprop1 and subprop2 would both be null. The new semantics should be used by default, especially by web services with WSDLs that use nonnull types. If not used in the case of nonnull types, the intermediate not null references for containing properties that are automatically created when reading a DTO spreadsheet could result in the generation of invalid XML according to the schema (because cell values are empty).

1. Click the Generate Template button. DevTest builds the template and the system messages report when the file is built.
2. Click the Open XLS File button.

The spreadsheet contains everything that is necessary to construct the object. We will show how to add data in the next section.

3. Close the XLS file.
4. Click the Test and Keep button to test and load the data. You see a window that confirms that the data set can be read, and shows the first set of data.

#### **Building the Excel spreadsheet**

To facilitate this explanation, we use an actual DTO object: `com.itko.example.dto.Customer`. This class is included with the DevTest examples and can be found by browsing the Test Manager using the Browse button.

The Customer DTO has the following properties:

Property Name	Type
balance	Double
id	int
name	String
poAddr	Address
since	Date
types	int[]
locations	Address[]

The Address DTO has the following properties:

Property Name	Type
city	String
line1	String
line2	String
state	String
zip	String

The first six Customer DTO properties can appear on one Excel spreadsheet. However, the locations property, an array of Address objects, requires a second Excel spreadsheet.

Looking at the first spreadsheet, at the top, DevTest lists the DTO spec (Customer) and the current DTO object (Customer). The spreadsheet would also list the Java doc location, if available. The following graphic shows the data sheet, with a row specifying property names, followed by a row specifying the data types. The first field (column) is not a DTO property, but a special field (Primary key), that holds a unique value for each row.

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2		Spec:	com.itko.examples.dto.Customer									
3		DTO:	com.itko.examples.dto.Customer									
4		Docs:	No docs available									
5		Static Constructor Methods:	No methods available									
6		Static Constructor Fields:	No fields available									
7												
8		Primary Key	balance	id	name	poAddr.city	poAddr.line1	poAddr.line2	poAddr.state	poAddr.zip	since	types
9		(unique per row)	double	int	String	String	String	String	String	String	Date	int[]
10												
11												
12												

Looking at the data sheet we can see:

- Each row contains the data for a single Customer DTO.
- The poAddr property, of type Address, has been "flattened" and its properties are listed on this sheet. These properties are prefixed with poAddr and then the property name in the address object (that is, poAddr.city).
- The since property, of type date, is prefilled with the current date. This entry is to show you the required format for the date mm/dd/yyyy. All dates must have this format in the Excel template.
- The types property, of type int[], is a single cell that can contain the array elements as a comma-separated list. This list is only possible for arrays of primitives or strings.

The location property, of type Address[], does not appear on this sheet. Because it is an array of objects, the location property is on the second sheet in the Excel file. This sheet contains the data for an Address object in each row. There are two special fields in this sheet: "Primary Key", and the "reference the containing DTO" field that is used to link the rows in this sheet to the rows in the primary sheet.

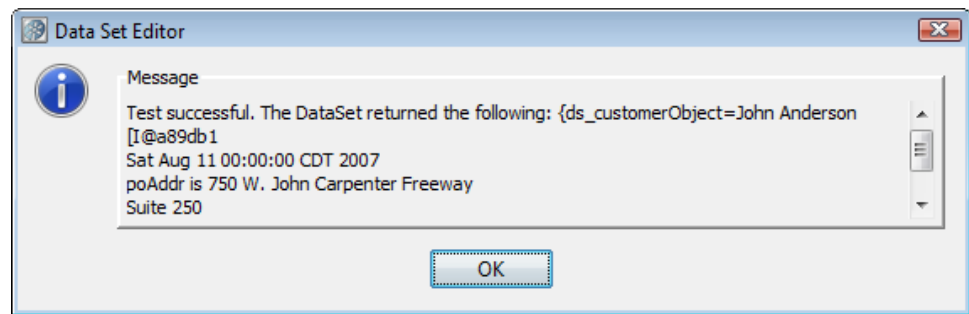
	A	B	C	D	E	F	G	H	I	J
1										
2		Spec:	com.itko.examples.dto.Customer.locations							
3		DTO:	com.itko.examples.dto.Address							
4		Docs:	No docs available							
5		Static Con	No methods available							
6		Static Con	No fields available							
7										
8		Primary Key	com.itko.e city	line1	line2	state	zip			
9		(unique per row)	(reference the containing DTO)	String	String	String	String	String		
10										
11										
12										
13										

Because each Customer object can have several locations, several rows in the locations sheet belong to an object specified in a single row of the Customer sheet. The second sheet manifests this by listing the primary key of the parent Customer object in the "reference the containing DTO" field of each location that belongs to the Customer. This is similar to primary/foreign key relationships in databases.

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2		Spec:	com.itko.examples.dto.Customer									
3		DTO:	com.itko.examples.dto.Customer									
4		Docs:	No docs available									
5		Static Constructor Methods:	No methods available									
6		Static Constructor Fields:	No fields available									
7												
8		Primary Key	balance	id	name	poAddr.city	poAddr.line1	poAddr.line2	poAddr.state	poAddr.zip	since	types
9		(unique per row)	double	int	String	String	String	String	String	String	Date	int[]
10			1	75	101 John Ani Dallas	750 W. John Ce Suite 250	TX	75024	8/11/2007	1,5,10		
11			2	250	102 Dirk Mar Plano	5800 Granite Pz Apt. 3455	TX	75031	6/29/2010	2,4,9		
12			3	800	103 Francis I Southlake	1376 Lakeview I Suite 809	TX	76092	5/10/2011	3,7,9		
13												

	A	B	C	D	E	F	G	H
1								
2		Spec:	com.itko.examples.dto.Customer.locations					
3		DTO:	com.itko.examples.dto.Address					
4		Docs:	No docs available					
5		Static Con	No methods available					
6		Static Con	No fields available					
7								
8		Primary Key	com.itko.e city	line1	line2	state	zip	
9		(unique per row)	(reference the containing DTO)	String	String	String	String	String
10			1	1 Dallas	12 Main St #45	TX	75201	
11			2	1 Plano	3354 Bilglade Ave.	TX	76133	
12			3	1 Frisco	2109 Tanglewood Driv	TX	75061	
13			4	2 Dallas	6788 Woodbrook Driv	TX	76092	
14			5	2 Fort Worth	443 Church Suite 87	TX	75925	
15			6	3 Dallas	7789 17th Blvd.	TX	74552	
16			7	3 Dallas	122 Hwy. 26	TX	71655	
17								

When you save the spreadsheet and click Test and Keep in DevTest, you see the first Customer DTO in the message.



Depending on the complexity of your DTO, you could have several more Excel sheets in the Excel workbook. However, the process is the same as in the previous example.

To see how you could use this Customer DTO as a property, see the sections on testing Java objects in Test Steps in *Using CA Application Test*.



## Unique Code Generator Data Set

The Unique Code Generator data set provides a unique token (or code) each time DevTest calls it. The token can be numeric or alphanumeric, and can have a user-defined prefix prepended to it. The Unique Code Generator is commonly used in testing to create new users, accounts, and so on. By using the generator, you can ensure that the generated token or code does not use a value that is already inside a system.

Complete the following fields:

**Name**

Enter the name of the data set.

**Local**

Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

**Random**

Whether the record after the current record (sequential access) is read, or a random record is read. Sequential reading is the default.

**Max Records to Fetch**

The upper bound on the number of records to fetch for random access. This text field is disabled if the Random check box is not selected.

**At End Of Data**

Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

**Type**

The type of token to return. The choices are number or alphanumeric.

**Prefix (opt)**

Prefix to be prepended (optional).

Click the Test and Keep button to test and load the data. A dialog opens to verify that the data set is readable. This dialog shows the first set of data.

This data set always returns a token. The At End of Data section of the Data Set Editor has no meaning for this data set type.

## Random Code Generator Data Set

The Random Code Generator data set generates numeric or alphanumeric data randomly for use in a test case. This data set is similar to the Unique Code Generator Data Set data set, but it lets you set a length for the result. This data set can be used to create a specific type of unique value such as a telephone number, or Social Security number.

Complete the following fields:

**Name**

Enter the name of the data set.

**Local**

Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

**Random**

Whether the record after the current record (sequential access) is read, or a random record is read. Sequential reading is the default.

**Max Records to Fetch**

The upper bound on the number of records to fetch for random access. This text field is disabled if the Random check box is not selected.

**At End Of Data**

Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

**Prefix (opt)**

Adds a prefix to the generated data. This field is optional.

**Type**

This field allows either alphanumeric or number type of data set to be generated.

**Length**

This field restricts the length of the random data that is generated to the value set here.

Click Test and Keep to test and load the data. You get a message that confirms that the data set can be read, and shows the first set of data.

## Message-Correlation ID Generator Data Set

The Message/Correlation ID Generator data set is a specialized unique code generator useful for messaging. The data set generates a 24-byte unique code. The data set is designed specifically for an IBM MQ Series correlation ID, but can also be used for any JMS provider. This data set creates or updates two special properties:

**`lisa.jms.correlation.id`** and **`lisa.mq.correlation.id`**. The messaging steps recognize these properties and set the message correlation ID appropriately.

Complete the following fields:

**Name**

Enter the name of the data set.

**Local**

Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

**Random**

Whether the record after the current record (sequential access) is read, or a random record is read. Sequential reading is the default.

**Max Records to Fetch**

The upper bound on the number of records to fetch for random access. This text field is disabled if the Random check box is not selected.

**At End Of Data**

Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

**Prop to set**

DevTest property to set. The default is **`lisa.jms.correlation.id`**.

## Load a Set of File Names Data Set

The Load a Set of File Names data set assigns a value to a property based on a filtered set of file names from the file system.

The set of file names can include all the files in a specific directory, or a set that a "file pattern" filters. Another option lets you include subdirectories in the set, recursively. The files are returned in a case-sensitive, alphabetical order, top directories first, followed by subdirectories (using depth first ordering). Each time the data set is used, it returns the next file name (full path name) in the data set. This file name is stored in a property with the same name as the data set.

Complete the following fields:

### **Name**

Enter the name of the data set.

### **Local**

Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

### **Random**

Whether the record after the current record (sequential access) is read, or a random record is read. Sequential reading is the default.

### **Max Records to Fetch**

The upper bound on the number of records to fetch for random access. This text field is disabled if the Random check box is not selected.

### **At End Of Data**

Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

### **Directory Location**

The full path name of the directory to scan, or you can browse using the Browse button.

### **File Pattern**

A filter pattern string, using an asterisk (\*) as a wild card if appropriate.

### **Include Files from Subdirectories**

If the files in subdirectories are listed, select this option.

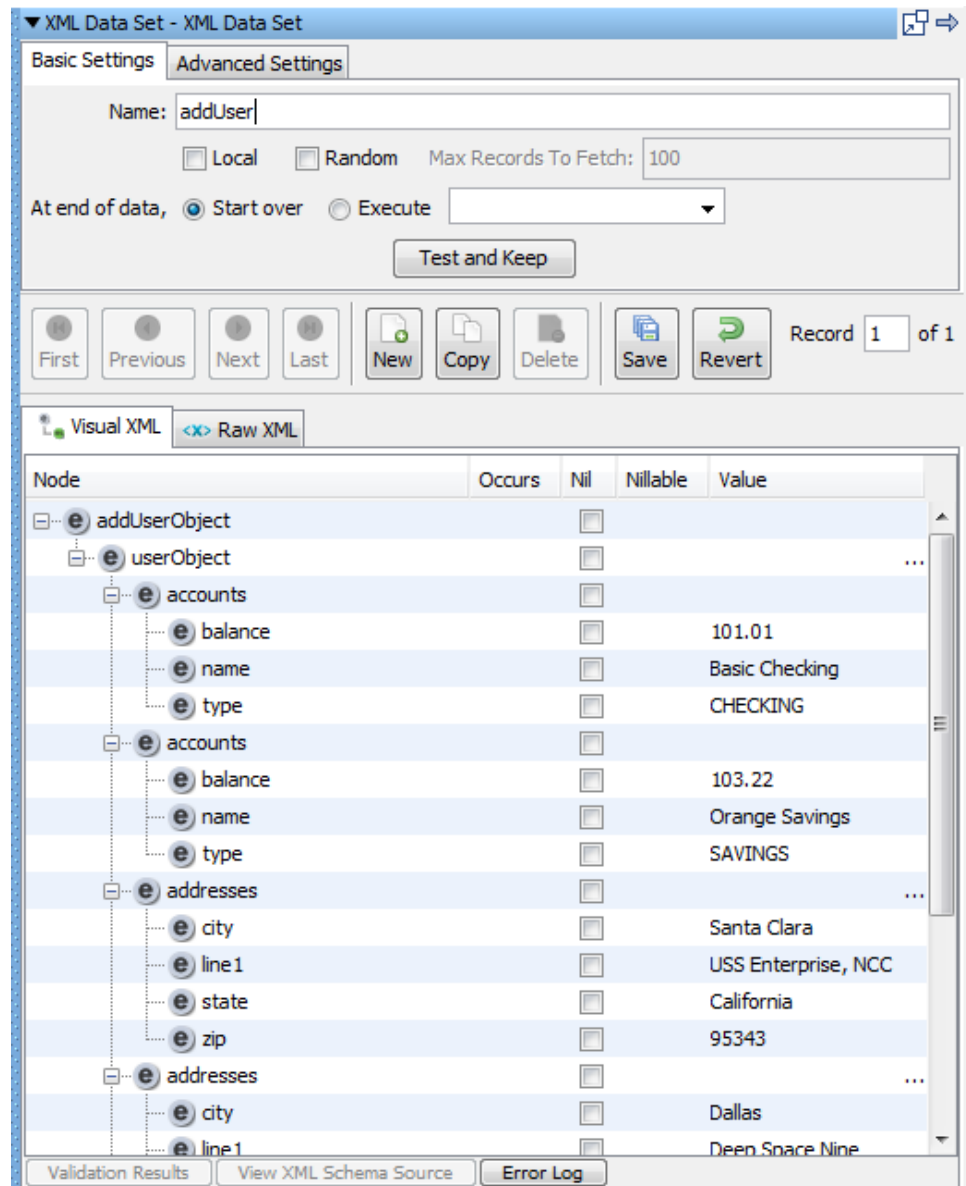
Click Test and Keep to test and load the data. You get a dialog that confirms that the data set can be read, and shows the first set of data.

**Note:** A large amount of data could take longer than you want; use the Cancel button to stop the operation.



## XML Data Set

Like other data sets in DevTest, the XML data set allows user-specified content to be added to distinct records. However, the XML data set specializes in handling XML content.



At design-time, the first record of an XML Data Set populates the value for a property in test state by clicking Test and Keep. The property name is the same as the name of the data set. For example, if the data set has the name **DataSet1**, then the property that is populated in the test case is "{{DataSet1}}". At run time, all of the records can be accessed sequentially to create a data-driven test case.

## Basic Settings Tab

**Name**

The name of this data set. This value is used as the name of the property in test step. For example, an XML Data Set with the name **DataSet1** populates the property `{{DataSet1}}`.

**Local**

Designates whether the data set is global or local. Global is the default. Local data sets are created with one for each simulator. Global data sets are created once and shared by all simulators.

**Random**

Whether the record after the current record (sequential access) is read, or a random record is read. Sequential reading is the default.

**Max Records to Fetch**

The upper bound on the number of records to fetch for random access. This text field is disabled if the Random check box is not selected.

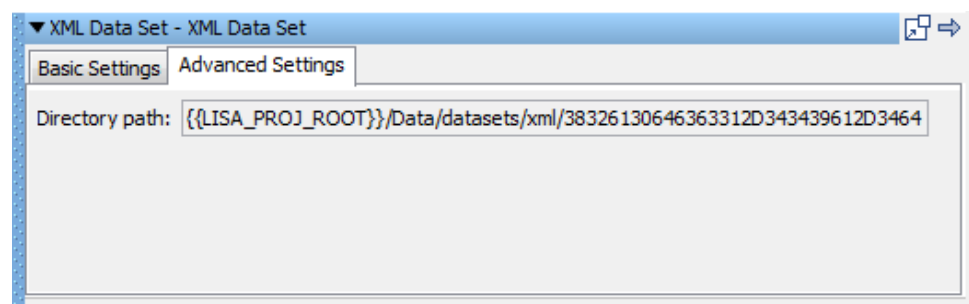
**At End Of Data**

Select the action for the end of the data set. You can start over, reading values from the start of the data set, or you can select the step to execute.

**Test and Keep**

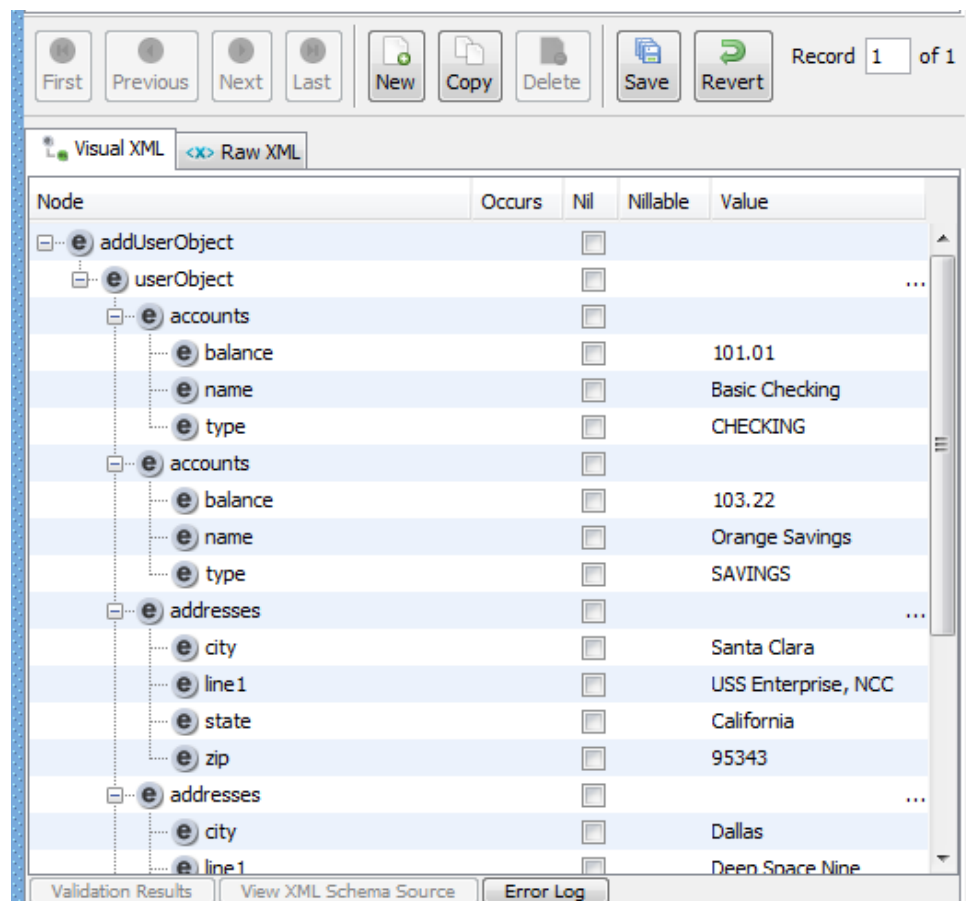
At design-time, populate the property that is associated with the data set in test state with the value of the first record.

## Advanced Settings Tab

**Directory path**

This read-only value represents the directory on the file system where records are saved. The mapping between a record and a file in the directory path is a one-to-one mapping. If an XML data set is created within a project, the directory path starts with "`{{LISA_PROJ_ROOT}}/Data/datasets/xml/`" or "`{{LISA_RELATIVE_PROJ_ROOT}}/Data/datasets/xml/`". If no project is used, the directory path starts with "`{{LISA_HOME}}/datasets/xml/`".

## Record Editing Panel



### Action Buttons

#### First

Move to the first record in the XML data set.

#### Previous

Move to the previous record.

#### Next

Move to the next record.

#### Last

Move to the last record.

#### New

Create a record.

#### Copy

Create a copy of the current record at the end of the data set and move to it.

#### Delete



Delete the current record.

**Save**

Save all pending changes.

**Revert**

Revert all pending changes.

**Record Number Selector**

**Record X of X**

To jump to a specific record, enter the record number.

**Visual XML Tab**

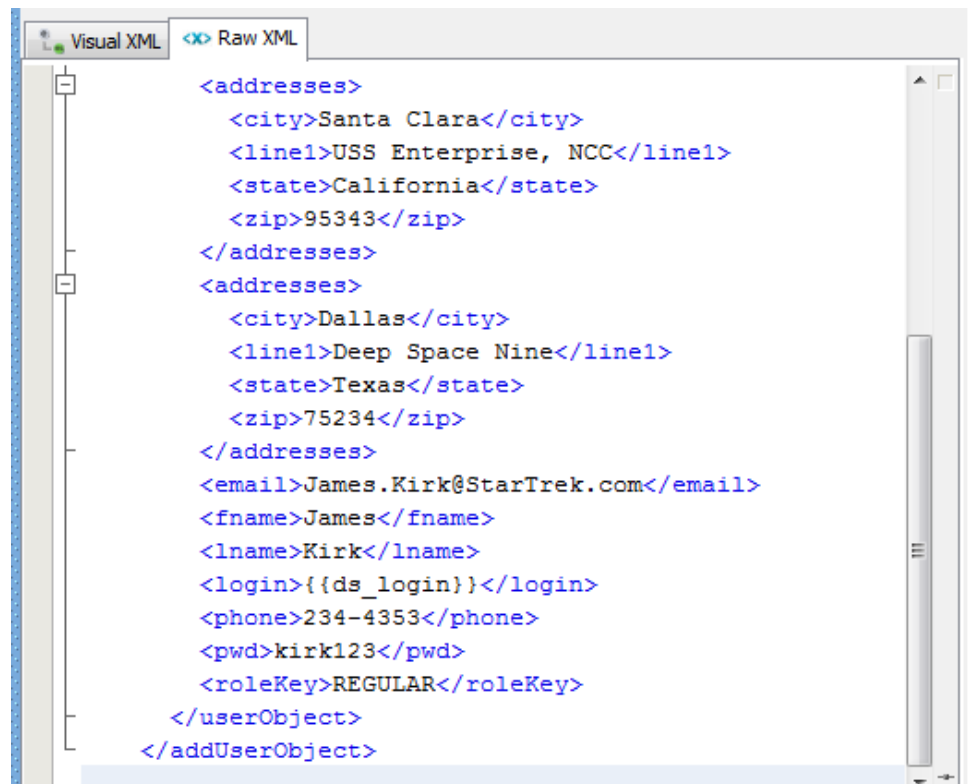
The screenshot shows the Visual XML Editor interface. At the top, there are two tabs: 'Visual XML' (selected) and '<x> Raw XML'. Below the tabs is a table with columns: 'Node', 'Occurs', 'Nil', 'Nillable', and 'Value'. The table displays a hierarchical structure of XML nodes. The 'Node' column shows a tree view with expandable nodes. The 'Occurs', 'Nil', and 'Nillable' columns contain checkboxes. The 'Value' column contains the values of the selected nodes. Some values are truncated with ellipses (...). At the bottom of the table, there are three buttons: 'Validation Results', 'View XML Schema Source', and 'Error Log'.

Node	Occurs	Nil	Nillable	Value
addUserObject		<input type="checkbox"/>		
userObject		<input type="checkbox"/>		...
accounts		<input type="checkbox"/>		
balance		<input type="checkbox"/>		101.01
name		<input type="checkbox"/>		Basic Checking
type		<input type="checkbox"/>		CHECKING
accounts		<input type="checkbox"/>		
balance		<input type="checkbox"/>		103.22
name		<input type="checkbox"/>		Orange Savings
type		<input type="checkbox"/>		SAVINGS
addresses		<input type="checkbox"/>		...
city		<input type="checkbox"/>		Santa Clara
line 1		<input type="checkbox"/>		USS Enterprise, NCC
state		<input type="checkbox"/>		California
zip		<input type="checkbox"/>		95343
addresses		<input type="checkbox"/>		...
city		<input type="checkbox"/>		Dallas
line 1		<input type="checkbox"/>		Deen Space Nine

**Visual XML Editor**

Moving the mouse over the Node and Value columns displays a tooltip. The tooltip can help if the value in the table is too long to be fully displayed.

**Raw XML Tab**



This tab contains a raw text view of the XML contained in a record.

Double-clicking the left border toggles the visibility of a top toolbar, line numbers bar, and bottom editing info bar in the editor.

## Filter Descriptions

This section describes each of the filters that are available.

Regular expressions are used for comparisons in several filters. For more information about regular expressions, see [Regular Expressions](#).

**This section contains descriptions of the following filters:**

[Utility Filters](#) (see page 123)  
[Database Filters](#) (see page 130)  
[Messaging/ESB Filters](#) (see page 137)  
[HTTP/HTML Filters](#) (see page 140)  
[XML Filters](#) (see page 158)  
[JSON Filters](#) (see page 165)  
[Java Filters](#) (see page 167)  
[VSE Filters](#) (see page 169)  
[CAI Filters](#) (see page 170)  
[Copybook Filters](#) (see page 172)

## Utility Filters

**The following filters are available in the Utility Filters list for any test step.**

[Create Property Based on Surrounding Values](#) (see page 124)  
[Store Step Response](#) (see page 126)  
[Override Last Response Property](#) (see page 127)  
[Save Property Value to File](#) (see page 127)  
[Parse Property Value As Argument String](#) (see page 128)  
[Save Property from One Key to Another](#) (see page 129)  
[Time Stamp Filter](#) (see page 130)

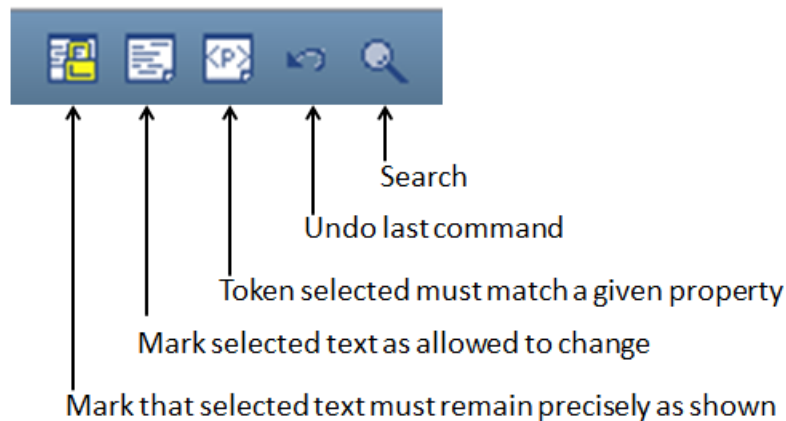
## Create Property Based on Surrounding Values

The Create Property Based on Surrounding Values filter lets you read textual content and filter it for information to be stored in DevTest properties. The filter can be used on text, and on XML and HTML treated as text. The filter uses a "paint the screen" technique.

"Paint the screen" gives you great flexibility to define what in the HTML you want to parse as properties. Mark the text in one of the following ways:



- Text that must appear in the response exactly as shown: a Must block.
- Text that is not required to appear in the response, or can change: an Any block.
- Text that is stored in a property: a Property block.

The text is marked using the icons at the bottom of the editor.

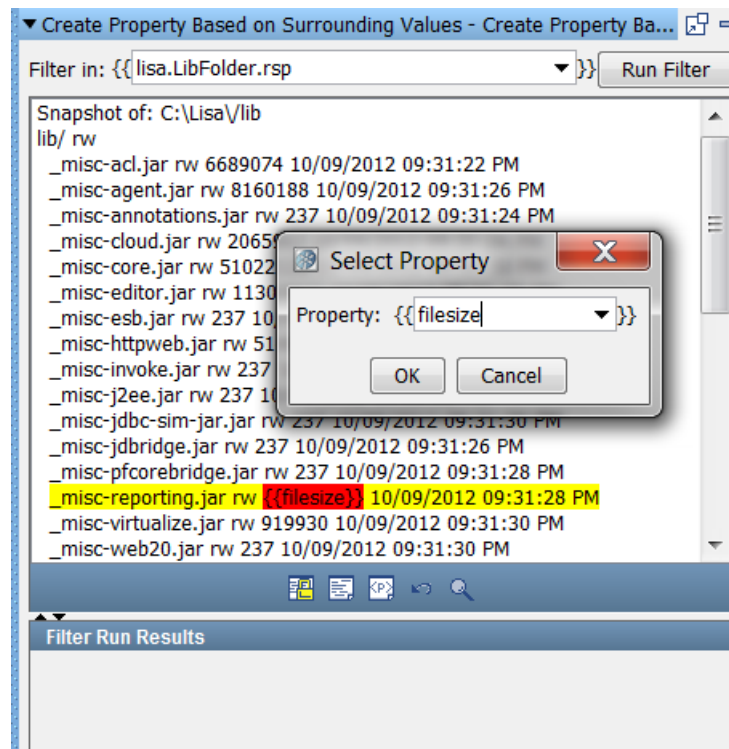


In the following example, the goal is to store the size of a specific file in a property.



The text is marked using the editor icons, by selecting the text, and then clicking the appropriate icon.

- Yellow background indicates text that must appear as shown (indicated by the arrows from "Text uses ....."). This is a Must block and is marked using the Must  icon.
- Red background identifies the text that is stored in the property that is entered in the dialog (indicated by a single arrow). This is a Property block and is marked using the Property  icon.

**Note:** Property blocks must always be bounded by Must blocks.



This screen shows the contents of the text buffer. The goal is to parse the file size of the `_misc-reporting.jar` file. The file size is the number that appears after `_misc-reporting.jar`.

The boundaries are set around the file size, and Must  has been clicked. The actual file size text inside the selected content was selected, and Property  was clicked.

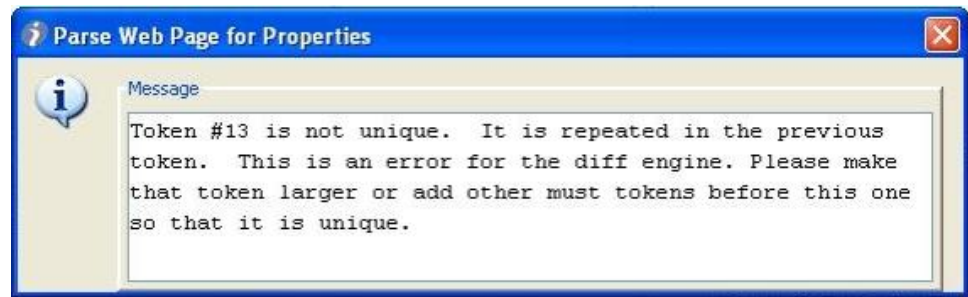
The property name was then entered into the dialog. The actual value of the file size has been replaced with the name of the property.

When this filter is run, the property `filesize` is assigned the size of the `_misc-reporting.jar` file.

You can repeat this process on this text buffer to define as many properties as necessary.

### Handling Nonunique Tokens

If you see the following error message, your selected token is not unique; the selection you made is repeated in the token before it.



To solve this issue in most cases, simply create another token to make the prior token a Must token also. In other cases, when this technique does not work, a judicious placement of another Must block between the two duplicate tokens avoids the error.

This solution works because DevTest can distinguish between the two duplicate tokens, which are based on their relative location.

## Store Step Response

The Store Step Response filter lets you save the last response as a property for future use.

Complete the following fields:

**Filter in**

Where to apply the filter. You cannot change this value for this filter.

**Property**

The name of the property to store the last response.

**Filter Run Results**

Displays the property and values that result from running the filter.

**Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## Override Last Response Property

The Override "Last Response" Property filter lets you replace the current value of the last response with the value of an existing property. For example, assume that you execute an EJB, but you eventually receive a value after making some method calls on the EJB. Using the result of a method call (instead of the EJB object) as the last response could improve the test case. You can use a filter to save the return value from that call as a property, and then use that property in this filter.

Complete the following fields:

**Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

**Convert to XML**

Select this option if you want the response to be converted to valid XML.

**Filter Run Results**

Displays the property and values that result from running the filter.

**Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## Save Property Value to File

The Save Property Value to File filter lets you save the value of an existing property to a file in your file system.

Complete the following fields:

**Filter in**

The name of the property whose value is written to the file.

**Location**

The path name of the file to write the value to. You can browse to the file. You can use properties in the location.

**Append Mode**

To append the information to an existing file, select this check box.

**Filter Run Results**

Displays the property and values that result from running the filter.

**Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## Parse Property Value As Argument String

The Parse Property Value As Argument String filter lets you store the text of a specific attribute in a property. This filter is useful as a second filter, where you parse a filtered value for information.

Complete the following fields:

**Filter in**

The name of the existing property to parse. For example, to parse the "lisa.deleteUser.cookies.rsp" property to return the value of the SESSIONID attribute, enter lisa.deleteUser.cookies.rsp.

**IsURL**

Select this check box if the property value is a URL.

**Attribute**

The attribute to retrieve. The example shows the JSESSIONID attribute.

**Property**

The name of the property to store the text of the attribute. The example shows sessionId.

**Default (if not found)**

If the attribute is not found, the default value to use.

**Filter Run Results**

Displays the property and values that result from running the filter.

**Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.



## Save Property from One Key to Another

The Save Property from one key to another filter copies values from one key to another by reference where normal Java rules apply.

This filter simply optimizes BeanShell overhead for simple property copy.

Complete the following fields:

**Filter in**

This field accepts the property content of which is copied in some other property.  
This field is the input source property.

**To Property**

This field is the property name where input property content is copied.

**Run Filter**

Lets you test the filter immediately while developing test steps rather than waiting until the test case is completed.

**Pre Process**

Run the filter before the step.

**Post Process**

Run the filter after the step.

## Time Stamp Filter

Use the Time Stamp filter to assign the current time and date to the property so that you can use it in the following steps.

Complete the following fields:

**Filter in**

The name of the existing step.

**Date Pattern**

Select the date pattern to display.

**Offset**

Used to offset the date to an appropriate (future or past) date that is based on the current date.

**Pre Process**

When selected, generates a timestamp before the step runs.

**Property for Pre Process**

The property to store the preprocess timestamp.

**Post Process**

When selected, generates a timestamp after the step runs.

**Property for Post Process**

The property to store the post timestamp.

**Filter Run Results**

Displays the property and values that result from running the filter.

**Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## Database Filters

**The following filters are available in the Database Filters list for any test step.**

[Extract Value from JDBC Result Set](#) (see page 131)

[Simple Result Set](#) (see page 133)

[Size of JDBC Result Set](#) (see page 133)

[Set Size of a Result Set to a Property](#) (see page 134)

[Get Value For Another Value in a ResultSet Row](#) (see page 135)

## Extract Value from JDBC Result Set

The Extract Value from JDBC Result Set filter lets you store the text of a specific JDBC result set value in a property. You can create this filter either manually from the filter list or by using the embedded filter commands on a result set response.

### To create the filter manually:

Complete the following fields:

#### Filter in

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

#### Column (1-based or name)

The index or name of the column (field).

#### Row (0-based)

The row to retrieve the value. This field is a zero-based index.

#### Property

The name of the property where the value in the cell at the row/column intersection is stored.

#### Filter Run Results

Displays the property and values that result from running the filter.

#### Run Filter

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

### To create a filter from a result set response:

Follow these steps:


1. Display the step response that contains the result set.
2. From the result set, select the cell of the value to store in the filter.

▼ SQL Database Execution (JDBC) - Verify User Added


Result Set

LOGIN	PWD	NEWFLAG	FNAME	LNAME	EMAIL	PHONE	ROLEKEY	SSN
dmxxx-009	tIDAdNa3...	1	first-9	last-9	test@test...		1	
Testuser	IGyAQTu...	0	Test	User	anne@itk...	817-433-...	1	433-87-3...
TEST1	nU4eI71b...	1					1	
First1	8FePHnF0...	1	Firstname1	Lastname1	first1@itk...	555-555-...	1	555-55-8...
Last1	i+UhJqb9...	1	Firstname2	Lastname2	last1@itko...	333-448-...	1	533-88-9...
test1	nU4eI71b...	1	First1	Last1	test1@itk...	214-111-...	1	111-11-1...
test2	nU4eI71b...	1	First2	Last2	test2@itk...	215-222-...	1	222-22-2...
demo	89yJPVNn...	0	DEMOFIRST	DEMOLAST	demo@itk...	817-433-...	1	677234567
admin	0DPiKuNIr...	1	ITKO	Admin	lisabank-a...	123-4567	2	434-47-5...
<b>sbellum</b>	26yJsXNp...	1	Sara	Bellum	sbellum@...	232-4345	1	614-40-1...
wpiece	/UuJ0Me...	1	Warren	Piece	wpiece@...	455-3232	1	546-71-4...
areck	AHRRRjD...	1	Amanda	Reckonwith	areck@my...	555-2244	1	350-02-1...
boaty	RQii0Ldp...	1	Boaty	Rabbit	boaty@ra...	333-4521	1	616-51-0...
itko	qUqP5cyy...	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...

Base Result Set Generate Filter for Current Col/Row Value

- Click Generate Filter , as the arrow in the previous graphic shows.
- In the dialog, enter the property key:

Please enter the property key for this value. ✕



- Click OK.

The filter to store the value **sbellum** in the property **theLogin** is created.

## Simple Result Set

The Simple Result Set filter is used to count the number of rows in a Result Set Response.

For more information, see [Size of JDBC Result Set](#) (see page 133) in *Using CA Application Test*.

## Size of JDBC Result Set

The Size of JDBC Result Set filter lets you check that the result set returned in each JDBC-based step matches the criteria that are specified. The filter is a simple filter to handle most common database errors automatically.

This filter does not affect non-JDBC steps, and is often used as a global filter in a test case.

Complete the following fields:

### **Result Set Has Warnings**

Some databases return warnings in the result set. If your database supports this feature and you want a warning to fire the On error step for this filter, select this check box.

### **Row Count >=**

The minimum number of rows in the result set. If the result set contains less than this value, the filter sets the next step to the value specified in On Error step.

### **Row Count <\_ =**

The maximum number of rows in the result set. If the result set contains more than this value, the filter sets the next step to the value specified in On Error step.

### **On Error**

The step to execute if the conditions for this filter are not met.

**Note:** This filter can serve the purpose of a general global assertion because you can select a next step based on the presence of an error.

## Set Size of a Result Set to a Property

The Set Size of a Result Set to a Property filter lets you store the count of a result set to a property provided.

Complete the following fields:

### **Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

### **Property to Store Row Count**

User-provided property name to store the row count. The default property name is PROP.

### **Filter Run Results**

Displays the property and values that result from running the filter.

### **Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## Get Value For Another Value in a ResultSet Row

The Get Value for Another Value in a ResultSet Row filter lets you search a column (field) in a result set for a specific value. If the value is found the value in another column (field) and the same row is placed in a property.

You can create this filter either manually from the filter list or by using the embedded filter commands on a result set response.

### **To create the filter manually:**

Complete the following fields:

#### **Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

#### **Search Text (Regular Expression)**

The search string.

#### **Search Column (1-based or Name)**

The index or name of the column to search.

#### **Value Column (1-based or Name)**

The index or name of the column to extract the property value.

#### **Property**

The name of the property to store the value.

#### **Filter Run Results**

Displays the property and values that result from running the filter.

#### **Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

### **To create the filter from a result set response:**

1. Display the step response that contains the result set.
2. From the result set select the two values in different columns, using the Ctrl key.

▼ SQL Database Execution (JDBC) - Verify User Added

Result Set

LOGIN	PWD	NEWFLAG	FNAME	LNAME	EMAIL	PHONE	ROLEKEY	SSN
dmxxx-009	tIDAdNa3...	1	first-9	last-9	test@test...		1	
Testuser	IGyAQTu...	0	Test	User	anne@tk...	817-433-...	1	433-87-3...
TEST1	nU4eI71b...	1					1	
First1	8FePHnF0...	1	Firstname1	Lastname1	first1@tk...	555-555-...	1	555-55-8...
Last1	i+UhJqb9...	1	Firstname2	Lastname2	last1@tko...	333-448-...	1	533-88-9...
test1	nU4eI71b...	1	First1	Last1	test1@tk...	214-111-...	1	111-11-1...
test2	nU4eI71b...	1	First2	Last2	test2@tk...	215-222-...	1	222-22-2...
demo	89yJPVNn...	0	DEMOFIRST	DEMOLAST	demo@tk...	817-433-...	1	677234567
admin	ODPIKuNIr...	1	ITKO	Admin	lisabank-a...	123-4567	2	434-47-5...
sbellum	26yJsXNp...	1	Sara	Bellum	sbellum@...	232-4345	1	614-40-1...
wpiece	/UuJ0Me...	1	Warren	Piece	wpiece@...	455-3232	1	546-71-4...
areck	AHDRRjD...	1	Amanda	Reckonwith	areck@my...	555-2244	1	350-02-1...
boaty	RQIi0Ldp...	1	Boaty	Rabbit	boaty@ra...	333-4521	1	616-51-0...
itko	qUqP5cyx...	1	itko	test	itko.test@...	650-234-...	1	140-72-2...
lisa_simpson	60fAFoq+...	1	lisa	simpson	lisa.simps...	123-456-...	1	295-20-0...
virtuser	nU4eI71b...	1	Virtual	User	virtuser@i...	123-456-...	1	297-55-9...

Base Result Set Filter for a value and then get another column value

3. Select Filter for a value and then get another column value filter using the Filter



icon.

4. In the the Generate Value for Value Filter dialog, select or reassign the columns for the search and the value, then enter the Property Key.

Generate Value for Value Filter

Search Column (1-based or Name): LOGIN

Value Column (1-based or Name): EMAIL

Property Key to save value into: theEmail

OK Cancel

5. Click OK.

The filter that is created in this example is the same as the filter you created manually in the previous example.



In the example, **sbellum** is searched for, and if found, the value in the EMAIL column of that row is placed in the property **theEmail**.

## Messaging/ESB Filters

**The following filters are available in the Messaging/ESB Filters list for any test step.**

[Extract Payload and Properties from Messages](#) (see page 138)

[Convert an MQ Message to a VSE Request](#) (see page 139)

[Convert a JMS Message to a VSE Request](#) (see page 140)

## Extract Payload and Properties from Messages

DevTest auto extracts several internal properties of messages into properties in the test step using the Extract Payload and Properties from Messages filter. You can also select to auto extract the payload into a property. This method is a fast way to get data from a message.

Different messaging platforms impose various restrictions and can be seen as warnings at run time.

The property names can default to **lisa.stepName.message**, or you can specify the prefix. You can specify an exact name for the payload.

Complete the following fields:

### **Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

### **Get Payload**

Select this option if you require a payload.

### **Property key to store the Payload**

Enter or select the property key to use as the payload.

### **Prefix for extracted details**

Enter the prefix to be attached to the property name in the result.

### **Get Message ID**

Select to get the message ID.

### **Get Correlation ID where appropriate**

Select to get the correlation ID.

### **Additional extended properties**

Select to get any additional extended properties.

### **Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## Convert an MQ Message to a VSE Request

The Convert an MQ Message to a VSE Request filter is automatically added from the VSE Recorder. This filter serves the specific purpose that enables proper functioning with recordings. Use it carefully. If this filter is added to a step in a VSE model, do not remove or edit it.

Complete the following fields:

**Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

**Object form**

Select to get the Object Form.

**Track Correlation ID**

Select to track the correlation ID.

**Track Message ID**

Select to track the message ID.

**Transaction Tracking Type**

Select the appropriate tracking type from - Sequential, Correlation ID, Message ID, or Message ID to Correlation ID.

**Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## Convert a JMS Message to a VSE Request

The Convert a JMS Message to a VSE Request filter is automatically added from the VSE Recorder. This filter serves the specific purpose that enables proper functioning with recordings. Use it carefully. If this filter is added to a step in a VSE model, do not remove or edit it.

Complete the following fields:

### **Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

### **Object form**

Select to get the Object Form.

### **Track Correlation ID**

Select to track the correlation ID.

### **Track Message ID**

Select to track the message ID.

### **Transaction Tracking Type**

Select appropriate tracking type from: Sequential, Correlation ID, Message ID, or Message ID to Correlation ID.

### **Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## HTTP/HTML Filters

**The following filters are available in the HTTP/HTML Filters list for any test step.**

[Create Resultset from HTML Table Rows](#) (see page 142)

[Parse Web Page for Properties](#) (see page 144)

[Parse HTML/XML Result for the Value of a Specific Tag or Attribute](#) (see page 147)

[Parse HTML Result for Specific Value and Parse It](#) (see page 149)

[Parse HTML Result for the Child Text of a Tag](#) (see page 150)

[Parse HTML Result for HTTP Header Value](#) (see page 151)

[Parse HTML Result for the Value of an Attribute](#) (see page 152)

[Parse HTML Result for DevTest Tags](#) (see page 153)

[Choose Random HTML Attribute](#) (see page 154)

[Parse HTML into List of Attributes](#) (see page 155)

[Parse HTTP Header Cookies](#) (see page 156)

[Dynamic Form Filter](#) (see page 157)

[Parse HTML Result by Searching Tag Attribute Values](#) (see page 158)

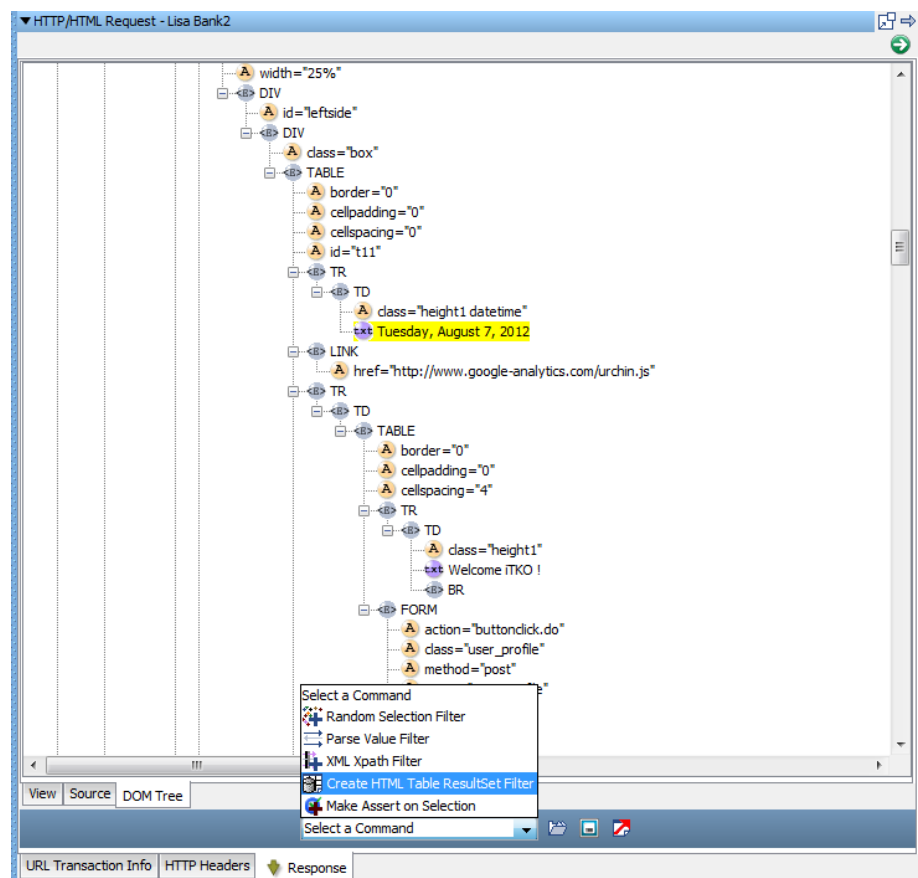
## Create Resultset from HTML Table Rows

To create a result set (for example, a JDBC result set) from an HTML table that the response returns, use the Create Resultset from HTML Table Rows filter. You can select the columns and rows of an HTML table, then create a result set from them. You can then use the result set to generate assertions exactly as in a database step.

You can create this filter by selecting it from the filter list and defining the parameters. However, it is easier to create the filter directly from the HTTP/HTML Request step response using one of the filter commands for that step. This approach is used here. The parameters that are produced in the following procedure (the parameters you would calculate to create this filter manually) are shown later in this section.

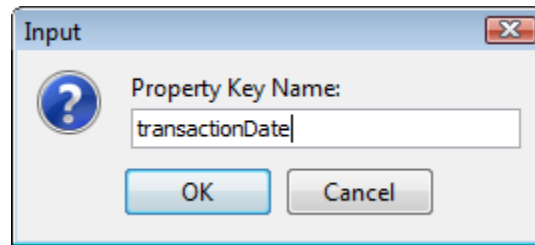
### To create a filter on a table:

1. Record a web page that contains the table.
2. Go to the appropriate HTML step, and view it from the DOM tree.
3. Select the values to place in the table, using the Ctrl key to select multiple fields. Select one example value from each column in the table that you want to use in the result set.



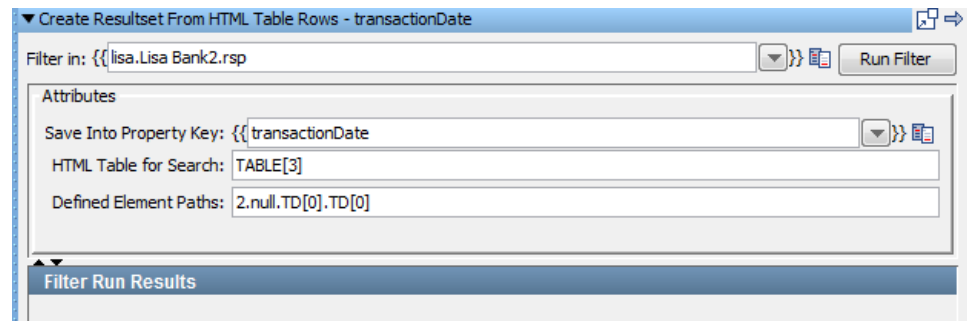
4. When it is highlighted, select Create HTML Table Results Filter.

5. Enter the property name in the window.

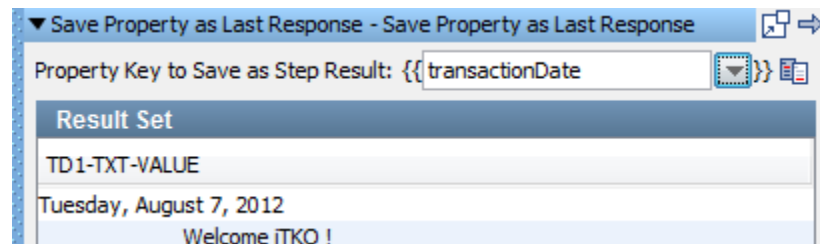


The property is now available in the test case.


The property is added to the current step. The following graphic shows the parameters that were calculated for this step. These parameters are required when you create this filter manually.



To show the filter results, we added a step of type Save Property as Last Response and added the property that the filter creates. The result set panel displays the results.



If you are editing a test case, you may need to replay the test case to generate the property from the filter using the Replay test case to a specific point command. To do so, use the Replay test case to a specific point command. The Replay test case to a

specific point command is activated by clicking Replay  on the toolbar. You can now use the embedded filters and assertions that are available at the bottom of the result set window of this step.

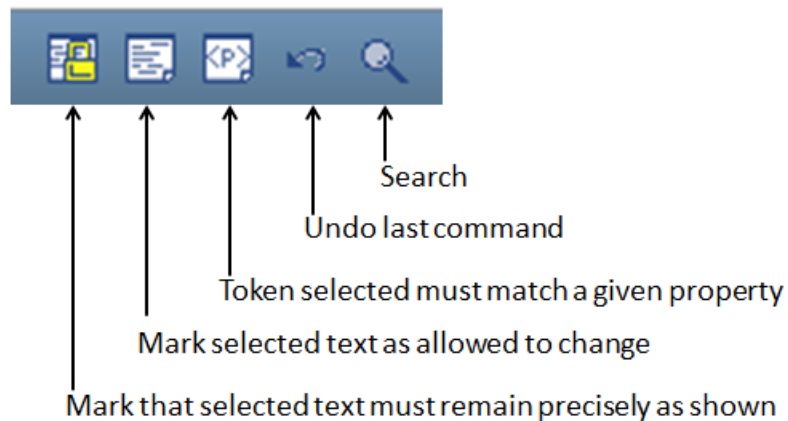
## Parse Web Page for Properties

The Parse Web Page for Properties filter lets you view a rendered web page to create properties from the HTML content. This filter uses the "paint the screen" technique.

"Paint the screen" gives you great flexibility to define what in the HTML you want to parse as properties. Mark the text in one of the following ways:

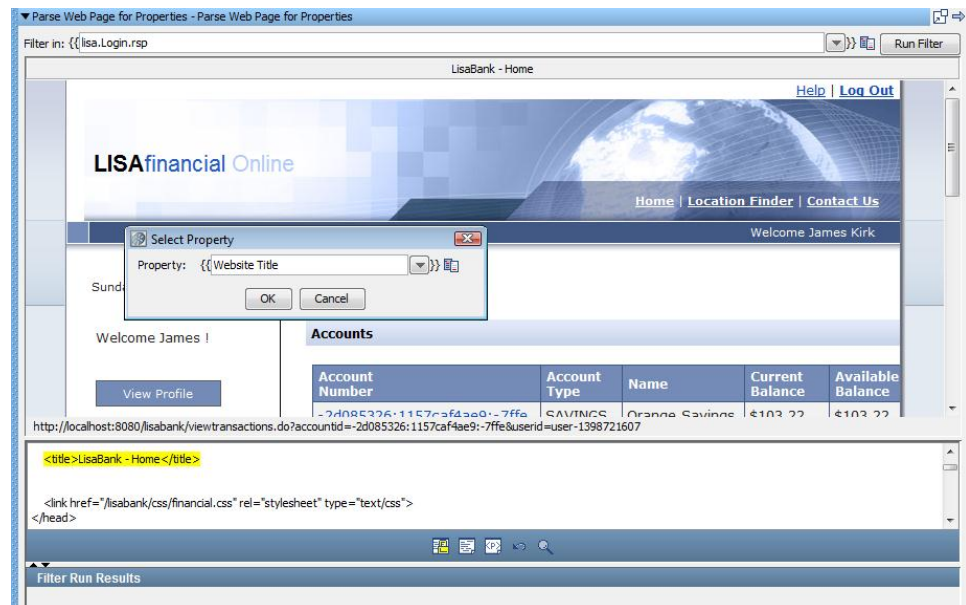
- Text that must appear in the response exactly as shown: a Must block.
- Text that is not required to appear in the response, or can change: an Any block.
- Text that is stored in a property: a Property block.

The text is marked using the icons at the bottom of the editor.




In the following example, assume that the website title "LisaBank - Home" changes from user to user, and therefore must be stored as a property.







This window shows the HTML that is rendered in a browser in the top panel, and the actual HTML text in the bottom panel.

#### To create properties from fields on a website:

1. Navigate to the field identified in the HTML text in the bottom panel.
2. Select the text and click Must .
 

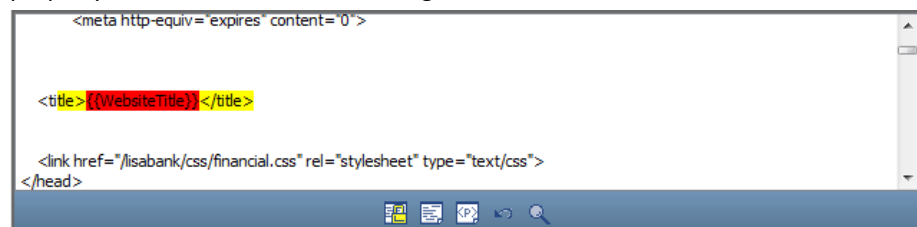
In this example, the selected field is the website title. The yellow highlights indicate text that must appear exactly as shown.
3. Select the name text inside the highlighted content, and click the Property  icon.
 

In this example, the website name text, "LisaBank - Home," is selected.
4. Click Property .

The Select Property dialog is displayed.

5. Enter the property name into the dialog.

In the HTML text in the bottom panel, the name of the property replaces the website name text. The red background identifies the text that is stored in the property that is entered into the dialog.



**Note:** All Property blocks must always be bounded by Must blocks.

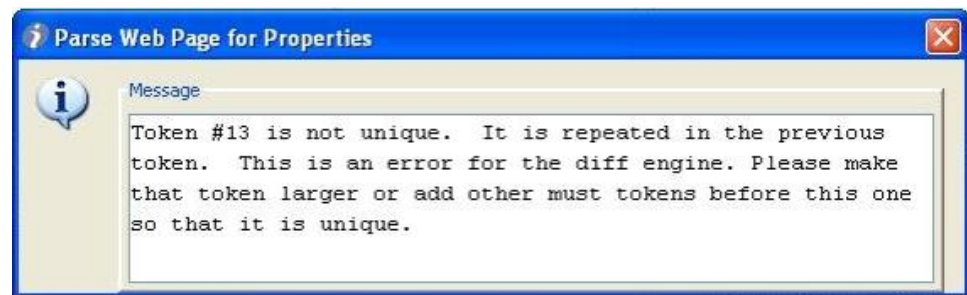
Frequently you can do this action purely from the web page view by selecting the content in the web browser. It can be easier to click the web browser in the area that you want to select, and then make your selection in the HTML panel.

The **Website Title** property is assigned the current value that appears on the HTML page when the filter runs. The website title can change its location in the text buffer and it is still found and parsed for the property.

To define more properties, repeat this process on this text buffer.

### Handling Nonunique Tokens

If you see the following error message, your selected token is not unique; the selection you made is repeated in the token before it.



To solve this issue in most cases, simply create another token to make the prior token a Must token also. In other cases, when this technique does not work, a judicious placement of another Must block between the two duplicate tokens avoids the error.

This solution works because DevTest can distinguish between the two duplicate tokens, which are based on their relative location.

## Parse HTML/XML Result for the Value of a Specific Tag or Attribute

You can create this filter either manually from the filter list or by using the embedded filter commands on a result set response.

### **To create the filter manually:**

Complete the following fields:

#### **Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

#### **Tag**

The name of the HTML tag. For an image tag, enter "IMG".

#### **Tag Count**

The occurrence of the tag from the top of response. For the first image tag, enter "1".

#### **Attribute**

The name of the attribute to filter. For the source attribute, enter "src".

#### **Property**

The property in which to store the value.

#### **Default (if not found)**

The value to use if the attribute value is not found.

#### **URLEncode**

When selected, property value is URLEncoded.

#### **Filter Run Results**

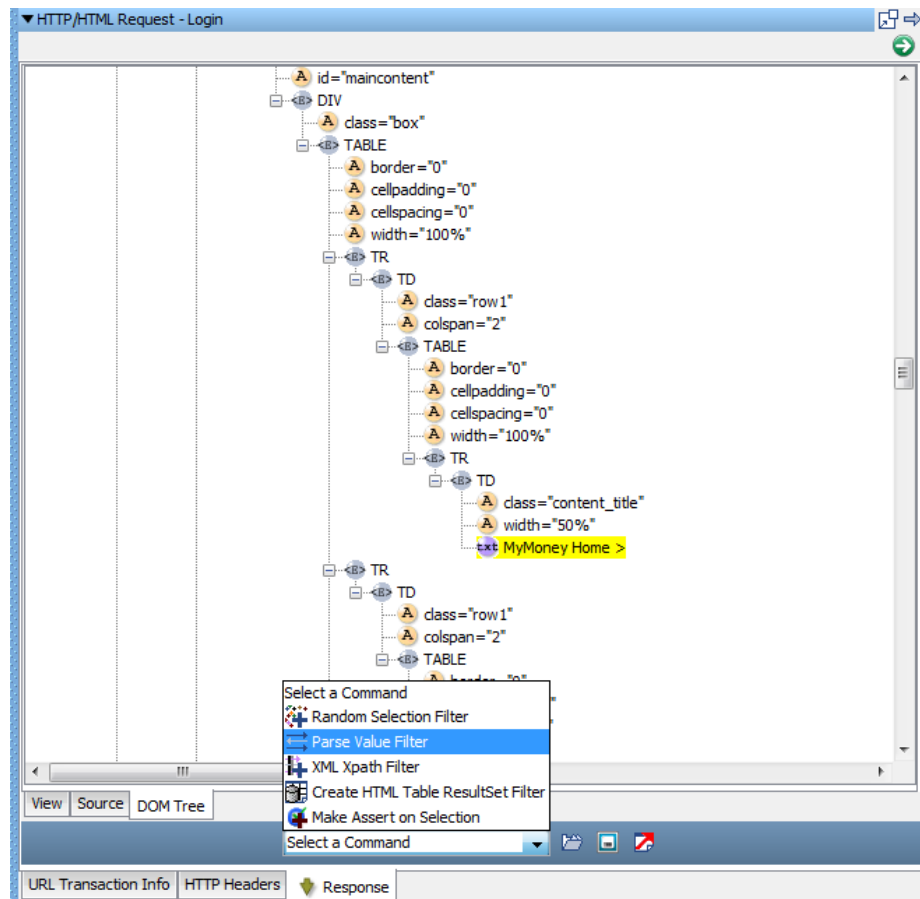
Displays the property and values that result from running the filter.

#### **Run Filter**

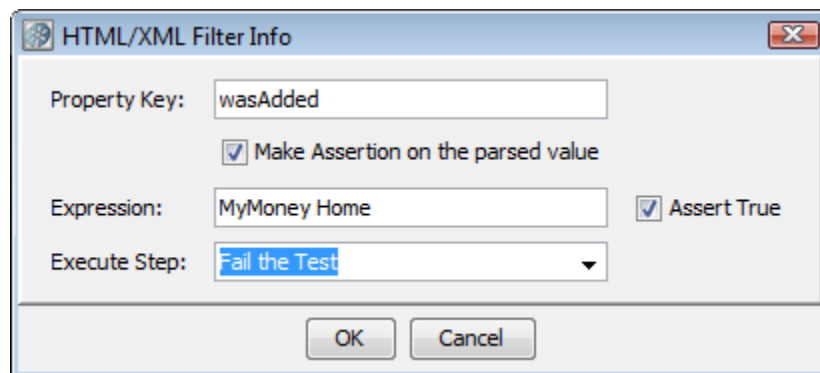
To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

### **To create the filter from the HTTP/HTML request step response page:**

1. Display the step response that contains the HTML response.



2. From the DOM Tree view, select the attribute whose value you want to store in a property.
3. When it is highlighted, select Parse Value Filter.
4. Enter the property name in the window.



You can also add assertions here.

## Parse HTML Result for Specific Value and Parse It

The Parse HTML Result for the Value of a Specific Tag or Attribute Value and Parse It filter combines the following filters:

- Parse HTML Result for the Value of an Attribute
- Parse Property Value as Argument String

This filter is designed to find a specific attribute in a web page, and then further parse that attribute. If the attribute is a URL, and not only a name-value pair, there is a function for handling that information.

In this example, the filter finds the "href" attribute for the seventh anchor tag, which is a URL. The filter takes the "cmd" parameter and stores that value in the **cmdlistusers\_KEY**.

Complete the following fields:

**Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

**Tag**

The name of the HTML tag. For an anchor tag, enter "a".

**Tag Count**

The occurrence of the tag from the top of response. For the seventh anchor tag, enter "7".

**Attribute**

The name of the attribute to filter. For the href attribute, enter "href".

**IsURL**

Select the check box if the attribute value is a URL.

**Argument to Parse**

The name of the argument to parse for its value; in this example, "cmd".

**Property**

The property in which to store the value.

In this example, "cmdlistusers\_KEY".

**URLEncode**

When selected, property value is URLEncoded.

**Default (if not found)**

The value to use if the attribute value is not found.

**Filter Run Results**

Displays the property and values that result from running the filter.

**Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## Parse HTML Result for the Child Text of a Tag

The Parse HTML Result for the Child Text of a Tag filter lets you store the child text of a tag in a property.

Complete the following fields:

**Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

**Tag**

The type of the tag. For example, for an h1 tag, enter "h1".

**Tag Count**

The occurrence of the tag. For the child text of the third h1 tag, enter "3".

**Property**

The property in which to store the value.

**Filter Run Results**

Displays the property and values that result from running the filter.

**Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## Parse HTML Result for HTTP Header Value

The Parse HTML Result to HTTP Header Value filter is commonly used to save the HTTP header Server in a property with the name **SERVER\_NAME**.

Complete the following fields:

**Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

**HTTP Header Key**

The name of the HTTP header; for example, "Server".

**Property**

The property in which to store the value.

**Filter Run Results**

Displays the property and values that result from running the filter.

**Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## Parse HTML Result for the Value of an Attribute

The Parse HTML Result for the Value of an Attribute filter lets you store the text of a specific attribute in a property. The attribute can occur anywhere in the result, including scripting code.

Complete the following fields:

**Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

**Attribute**

The type of attribute to retrieve. For example, for the URL of an anchor tag, enter "href".

**Count**

The occurrence of the tag. For example, for the URL of the third anchor tag on the page, enter "3".

**Property**

The property in which to store the value. In this example, "anchor3".

**Filter Run Results**

Displays the property and values that result from running the filter.

**Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.



## Parse HTML Result for DevTest Tags

The Parse HTML Result for DevTest Tags filter provides a way for developers to test-enable their web applications. For an in-depth study on test-enabling, see *Using the SDK*.

This filter lets you insert "LISAPROP" tags into your web page. The LISAPROP tag has two attributes: name and value. The LISAPROP tags do not show up in your web pages. They function only to provide valuable information about your web page to a tester. An example of a LISAPROP is:

```
<LISAPROP name="FIRST_USER" value="sbellum">.
```

If a tester has installed this type of filter, the property **FIRST\_USER** is automatically assigned the value **sbellum**. This filter removes any need for the tester to parse for this value. This type of filter helps a developer make the testing easier.

Frequently a web page does not contain the information that is necessary for proper validation, or that information is difficult to parse. Even when the information exists, subtle changes in the generated HTML can result in incorrect parsing. This filter can resolve many parsing issues for web testing.

No parameters are required.

## Choose Random HTML Attribute

The Choose Random HTML Attribute filter lets you store the text of a random selection from a set in a property. The attribute can occur anywhere in the result, including scripting code.

Complete the following fields:

**Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

**Outer Tag**

The outer element that contains the list from which to pick. For example, to select a drop-down list, enter the text "select".

**Tag Count**

The occurrence of the outer tag. For example, to select the second drop-down list, enter the text "2".

**Inner Tag**

The tag to pick the attribute from, randomly. To pick a random item in the drop-down list, enter the text "option".

**Filter Attribute**

An optional field to specify attribute names to exclude from the pick list.

**Filter Value**

An optional field to specify attribute values to exclude from the pick list.

**Attribute**

The attribute from which to retrieve text. If this field is blank, the child text of the inner tag is returned.

**Property Key**

The property to store the text of the attribute.

**Filter Run Results**

Displays the property and values that result from running the filter.

**Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## Parse HTML into List of Attributes

The Parse HTML into List of Attributes filter lets you store the text of a set of attributes as a list in a property.

Complete the following fields:

**Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

**Outer Tag**

The outer element that contains the list of tags to parse. For example, to store all the links from all the anchor tags in a table, enter "table".

**Outer Tag Count**

The occurrence of the outer tag. For the second table, enter "2".

**Inner Tag**

The tag to retrieve the values from. For example, for all the anchor tags in the table enter "a".

**Filter Attribute**

An optional field to specify attribute names that must not appear in the pick list.

**Filter Value**

An optional field to specify attribute values that must not appear in the pick list.

**Attribute**

The attribute of the inner tag to retrieve the text from. If this field is blank, the child text of the inner tag is returned. To store all the links from all of the anchor tags in a table, enter "href".

**Property Key**

The name of the property to store the text of the attribute.

**Filter Run Results**

Displays the property and values that result from running the filter.

**Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## Parse HTTP Header Cookies

The Parse HTTP Header Cookies filter lets you parse the HTTP header and store cookie values in a property that starts with a specific prefix.

Complete the following fields:

### **Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

### **Property Prefix**

A text string that is prefixed to the cookie name to provide the property name to use. The full names of these properties are therefore dependent on the names of the cookies that were returned. The cookie names can be identified in the Property tab of the Interactive Test Run (ITR).

### **Filter Run Results**

Displays the property and values that result from running the filter.

### **Run Filter**

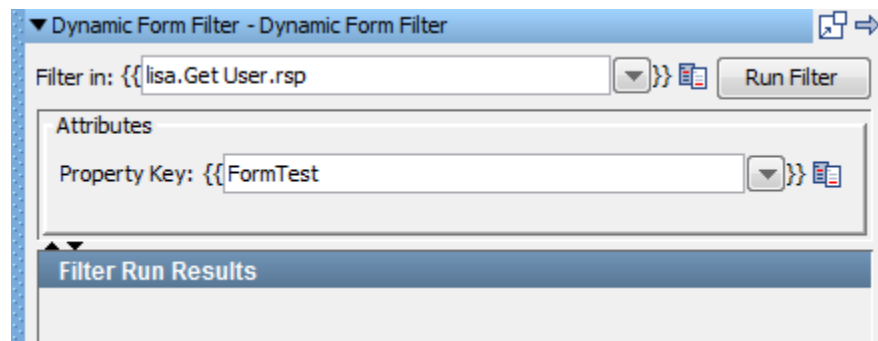
To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## Dynamic Form Filter

The Dynamic Form filter identifies dynamically generated forms in HTML responses and parses them into a set of properties. The property key that you enter becomes part of the property name for each form element in each form. This behavior is easier to understand by examining the example that follows.

You could test an HTML page with two dynamically generated forms:

```
<form name="F001" action="index.jsp"> <input type="text"
name="0001A" value="default" /> <input type="text" name="0001B"
value="" /></form>
<form name="F002" action="orders.jsp"> <input type="text"
name="0002A" value="Key" /> <input type="text" name="0002B"
value="" /></form>
```



Using a property key of FormTest in the filter panel creates the following key/value pairs:

Key	Value
FormTest.Form1.text1.name	0001A
FormTest.Form1.text1.value	default
FormTest.Form1.text2.name	0001B
FormTest.Form1.text2.value	
FormTest.Form2.text1.name	0002A
FormTest.Form2.text1.value	
FormTest.Form2.text2.name	0002B
FormTest.Form2.text2.value	

## Parse HTML Result by Searching Tag Attribute Values

To filter an attribute value by searching the tag for the name and value of another attribute, use the Parse HTML Result by Searching Tag/Attribute Values filter. If multiple tags fit the criteria, you can specify which tag to use.

Complete the following fields:

**Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

**Tag**

The name of the tag to search.

**Search Criteria Attribute**

The attribute to search for.

**Search Criteria Value Expression**

The attribute expression to search for.

**Tag Count**

The specific tag to use from those tags that satisfy the search criteria.

**Attribute**

The attribute whose value you want.

**Property**

The property in which to store the value.

**Default (if not found)**

The value to use if the attribute value is not found.

**URLEncode**

When selected, property value is URLEncoded.

**Filter Run Results**

Displays the property and values that result from running the filter.

**Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## XML Filters

**The following filters are available in the XML Filters list for any test step.**

[Parse Text from XML](#) (see page 160)

[Read Attribute from XML Tag](#) (see page 162)

[Parse XML Result for DevTest Tags](#) (see page 164)

[Choose Random XML Attribute](#) (see page 164)

[XML XPath Filter](#) (see page 165)

## Parse Text from XML

The Parse Text from XML filter stores the child text of a tag in a property. To define a Parse text from XML filter, set the type of the filter and set the three attributes.

You can create this filter either manually from the filter list or by using the embedded filter commands on an XML response.

### **To create the filter manually:**

Complete the following fields:

#### **Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist. You can edit this value for this filter.

#### **Tag**

The type of the tag. For example, if you want the child text of the multiRef tag, enter "multiRef".

#### **Tag Count**

The occurrence of the tag. For example, if you want the child text of the first multiRef tag, set the count to "1".

#### **Property**

The property in which to store the value.

#### **Filter Run Results**

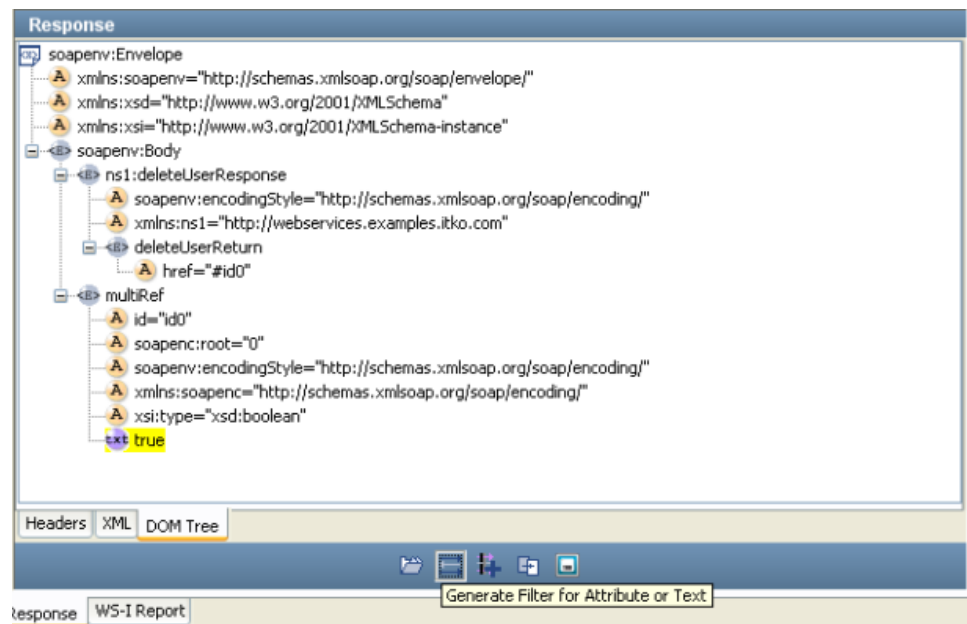
Displays the property and values that result from running the filter.

#### **Run Filter**

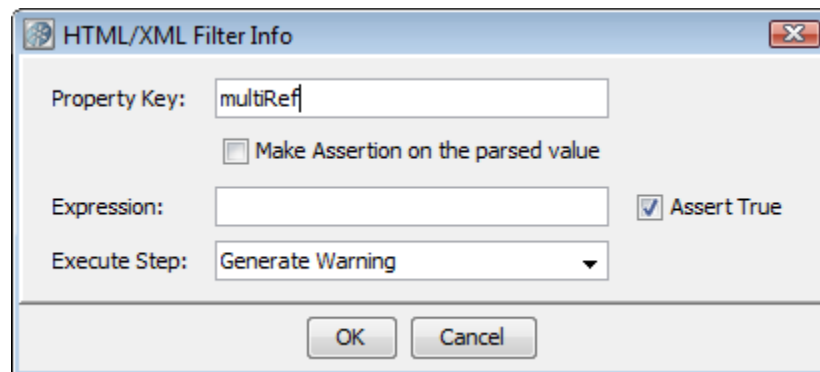
To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

### **To create the filter directly from the response page:**





1. From the DOM Tree view, select the attribute whose value you want to store in a property.
2. After it is selected, select `Generate Filter for Attribute or Text`.
3. Enter the property name in the dialog.



Assertions can also be added here.

## Read Attribute from XML Tag

The Read Attribute from XML Tag filter lets you store the text of a specific attribute in a property. The attribute can occur anywhere in the result.

You can create this filter either manually from the filter list, or by using the embedded filter commands on an XML response.

### To create the filter manually:

Complete the following fields:

#### Filter in

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

#### Tag

The name of the XML tag; for example, "target".

#### Tag Count

The occurrence of the tag from the top of response; for the first tag enter "1".

#### Attribute

The name of the attribute to filter; for the href attribute enter "href".

#### Property

The property in which to store the value.

#### Default (if not found)

The value to use if the attribute value is not found.

#### URLEncode

When selected, property value is URLEncoded.

#### Filter Run Results

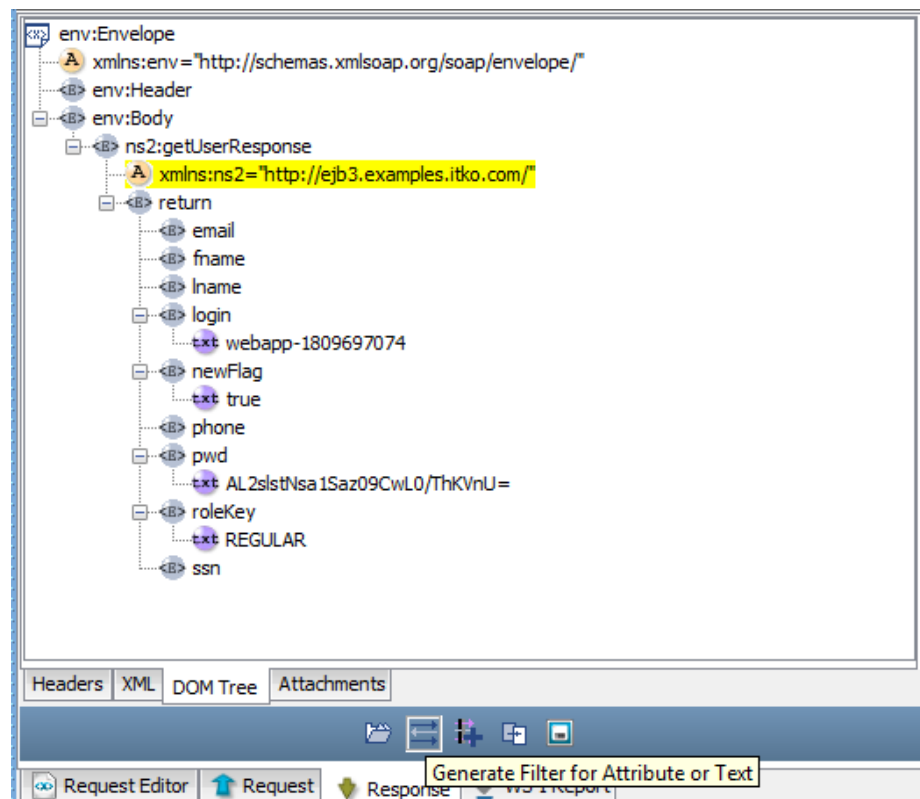
Displays the property and values that result from running the filter.

#### Run Filter

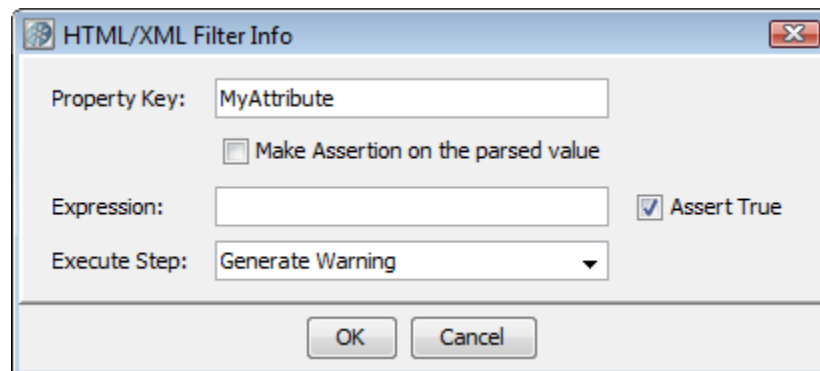
To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

### To create the filter from the response page:

1. Display the step response that contains the XML.



2. From the DOM Tree view, select the attribute whose value you want to store in a property.
3. When it is highlighted, select Generate Filter for Attribute or Text.
4. Enter the property name in the window.



You can also add an assertion here. A Property Value Expression Assertion can be added to this step.

## Parse XML Result for DevTest Tags

The Parse XML Result for DevTest Tags filter provides a way for developers to test-enable their XML applications. For an in-depth study on test-enabling, see *Using the SDK*.

This filter lets you insert LISAPROP tags into your XML page. The LISAPROP tag has two attributes: name and value. The LISAPROP tags function only to provide valuable information about your XML to a tester. An example of a LISAPROP is:

```
<LISAPROP name="FIRST_USER" value="sbellum">.
```

If a tester has installed this type of filter, the property "FIRST\_USER" is automatically assigned the value **sbellum**. This filter removes any need on behalf of the tester to parse for this value. This type of filter helps a developer make the testing easier.

Sometimes the XML does not contain the information that is necessary to validate properly, or that information is difficult to parse. Even when the information exists, subtle changes in the generated XML can result in incorrect parsing. This LISAPROP filter can resolve many parsing issues.

No parameters are required.

## Choose Random XML Attribute

The Choose Random XML Attribute filter lets you store the text of a random selection from a set in a property. The attribute can occur anywhere in the result. This filter works exactly like [Choose Random HTML Attribute](#) (see page 154).

## XML XPath Filter

The XML XPath filter lets you use an XPath query that runs on a property or the last response and stores it in a property. When this filter is selected, the last response is loaded into the content panel.

You can view the response as an XML document or as a DOM tree. However, the XPath selection can be made only from the DOM tree.

Construct the XPath query by using one of the following methods:

- Manually enter the XPath expression in the XPath Query text box.
- Select an element from the DOM tree and let DevTest construct the XPath expression.
- Select an element from the DOM tree, and then edit the XPath that is constructed. For example, you can modify it to use a property, or a counter data set.

Complete the following fields:

### **Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

### **Save To Property**

The property in which to store the result of the XPath query.

Now construct the XPath query using one of the methods described earlier.

After an XPath query has been constructed, test it by clicking Run Filter. The results of the query appear in the Filter Run Results pane.

The **`lisa.xml.xpath.computeXPath.alwaysUseLocalName`** property controls whether the XPath `local-name()` function is always used during the XPath generation. The default value is `false`, which means that the `local-name()` function is used only when necessary. To generate an XPath that works regardless of an XML node namespace, set the value to `true`.

## JSON Filters

**The following filter is available in the JSON filters list for any test step.**

[JSON Path Filter](#) (see page 166)

## JSON Path Filter

The JSON Path filter lets you extract a JSON property value from a JSON object and save it to a property. To open its editor, click the filter.

Complete the following fields:

### Filter in

Specifies the name of the property to filter for the step. If the property is not in the pull-down menu, you can enter it. The property must exist. You can edit this value for this filter.

### JSON Path

Designates an expression that consists of a sequence of JSON properties in a JSON document. The JSON Path represents a path to a destination JSON property.

An array filter, denoted by a `?()` expression, can be applied to an array as a select criteria to select certain array elements. For example, `?(@.age > 20)` can be used to select array members whose age is greater than 20. Another example is `?(@.name in ('Mary', 'John'))`, which selects array members whose name is either Mary or John.

The following table lists supported filter operators that are available for JSON data types.

Operator	String	Number	Boolean
<code>==</code> (equal)	supported	supported	supported
<code>!=</code> (not equal)	supported	supported	supported
<code>&gt;=</code>		supported	
<code>&lt;=</code>		supported	
<code>&gt;</code>		supported	
<code>&lt;</code>		supported	
<code>in</code>	supported	supported	
<code>not in</code>	supported	supported	

**Save Value to Property**

Designates the name of the property where the property value of the JSON path is saved.

**Save Length to Property**

Defines the name of the property where the number of components in the attribute value of the JSON path is saved. If the property value is an array, the number of components is the number of elements in the array. If the property value is a JSON object, the number of components is the number of properties in the JSON object. For simple data types such as String or Number, the number of components is 1.

**Filter Run Results**

Displays the property and values that result from running the filter.

**Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## Java Filters

**The following filters are available in the Java Filters list for any test step.**

[Java Override "Last Response" Property Filter](#) (see page 168)

[Java Store Step Response Filter](#) (see page 168)

[Java Save Property Value to File Filter](#) (see page 169)

## Java Override "Last Response" Property Filter

A special property (Last Response) contains the response from the previous step. For example, if the previous response was an HTTP step, the last response is the web page that was returned.

If you want the last response to be something other than the default value, use the Override "Last Response" Property filter. This filter lets you replace the current value of the last response with the value of an existing property.

Complete the following fields:

### **Filter in**

The name of the property to consider as the last response for the step. If the property is not in the pull-down menu, you can enter it. The property must exist.

### **Convert to XML**

If you want the response to be converted to valid XML, select this option.

### **Filter Run Results**

Displays the property and values that result from running the filter.

### **Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

For more information, see [Override Last Response Property](#) (see page 127).

## Java Store Step Response Filter

The Store Step Response filter lets you save the last response as a property for future use.

Complete the following fields:

### **Filter in**

Enter the response to which to apply the filter. The previous graphic shows **lisa.Add User.rsp**, which means that the filter is applied to the response of Add User. You cannot change this value for this filter.

### **Property**

The property in which to store the value.

### **Filter Run Results**

Displays the property and values that result from running the filter.

### **Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.



## Java Save Property Value to File Filter

The Save Property Value to File filter lets you save the value of an existing property to a file in your file system.

Complete the following fields:

**Filter in**

The name of the property whose value you want written to the file.

**Location**

The path name of the file to write the value to. You can browse to the file. You can use properties in the location.

**Append Mode**

To append the information to an existing file, select this check box.

**Filter Run Results**

Displays the property and values that result from running the filter.

**Run Filter**

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## VSE Filters

**The following filter is available in the VSE Filters list for any test step:**

[Data Protocol Filter](#) (see page 170)

## Data Protocol Filter

The Data Protocol filter is used on protocol-specific listen steps for virtual models. It provides the necessary wrapper for a data protocol to act as a filter, which is the appropriate functionality for the VSE run-time side.

To open its editor, click the filter.

Complete the following fields:

### Filter in

Enter the response to which to apply the filter. The previous graphic shows **lisa.GetUser.rsp**, which means that the filter is applied to the response of Get User. You cannot change this value for this filter.

### Data Protocol

Select the appropriate data protocol to be used from the drop-down list.

### Process Requests

Select to see the process request.

### Process Responses

Select to see the process response.

### Run Filter

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

## CAI Filters

**The following filters are available in the CAI filters list for any test step.**

[Integration for CA Continuous Application Insight](#) (see page 171)

[Integration for webMethods Integration Server](#) (see page 172)

## Integration for CA Continuous Application Insight

The DevTest Integration Support for CA CAI filter is a common filter to enable CAI for all the technologies that DevTest supports. This filter collects extra information from a CAI application.

Currently DevTest supports integration with web services, JMS, servlets, EJB, and Java objects.

Complete the following fields:

**Error if Max Build Time (millis) Exceeds**

Enter build time in milliseconds. An error is generated if the build time exceeds the specified interval.

**On Transaction Error Step**

Select the step to redirect to on transaction error after filter is set to run.

**On CAI Warnings Step**

Select the step to redirect to on CAI warnings after filter is set to run.

**Report Component Content**

Generate a report of the component content.

**Force a Garbage Collection on the server at the start & end of the request**

Forces a garbage collection on the server at the start and end of the request.

**Fail test if server-side exception is logged**

Fail the test case if an exception is thrown at the server side.

**Log4J Level to capture in the test events**

Select the log4j level that is captured in the test events.

**Log4J Logger to temporarily change (blank is Root Logger)**

Enter the name of the logger.

## Integration for webMethods Integration Server

The DevTest Integration Support for webMethods Integration Server filter collects more information from CAI-enabled webMethods Integration Server.

Complete the following fields:

**Error if Max Build Time (millis) Exceeds**

Enter build time in milliseconds. An error is generated if the build time exceeds the specified interval.

**On Transaction Error Step**

Select the step to redirect to on transaction error after filter is set to run.

**On CAI Warnings Step**

Select the step to redirect to on CAI warnings after filter is set to run.

## Copybook Filters

**The following filter is available in the Copybook filters list for any test step.**

[Copybook Filter](#) (see page 173)

## Copybook Filter

The Copybook Filter converts copybook payloads to XML at run time. You can review and maintain your data by displaying these payloads in XML.

To open its editor, click the filter.

Complete the following fields:

### **Filter in**

Specifies the name of the property to filter for the step. If the property is not in the pull-down menu, you can enter it. The property must exist. You can edit this value for this filter.

### **Copybook definition file**

Designates the location where you store your copybook file definition.

### **Encoding**

Select a valid Java charset. This value is used to try to convert the bytes in the payload into text for use in the output XML.

**Default:** UTF-8

### **Copybook parser column start**

Copybooks frequently start each line with a line number. This parameter defines the column on which the parser starts when trying to parse a copybook file definition.

**Value:** A zero-based inclusive index. However, you can think of it as a "normal" one-based exclusive index.

**Default:** 6

**Example:** If you set this value to 6, the parser skips the first six characters in a line and starts with the seventh character.

### **Copybook parser column end**

Occasionally, copybooks contain other reference data at the end of each line. When that happens, the parser must know on which column to stop. If there is no "extra" data at the end of the lines in the file, set this number to something greater than the length of the longest line in the file. If this number is greater than the length of a line, the parser stops at the end of the line.

**Value:** A zero-based exclusive index. However, you can think of it as a "normal" one-based inclusive index.

**Example:** If you set this value to 72, the parser reads the 72nd character in the line and then it stops.

### **Filter Run Results**

Displays the property and values that result from running the filter.

### Run Filter

To run and execute the filter, click Run Filter. The results appear in the Filter Run Results section.

For performance purposes, the Copybook filter caches the copybook definition in memory for 86400 seconds. When the time expires, DevTest removes the converted copybook definition from the cache. If the file is needed again, DevTest reads and reconverts it.

## Test Step Descriptions

**This section contains descriptions of the following test steps:**

- [Test Step Information](#) (see page 174)
- [Web-Web Services Steps](#) (see page 175)
- [Java-J2EE Steps](#) (see page 238)
- [Other Transaction Steps](#) (see page 251)
- [Utilities Steps](#) (see page 260)
- [External-Subprocess Steps](#) (see page 276)
- [JMS Messaging Steps](#) (see page 285)
- [BEA Steps](#) (see page 308)
- [Sun JCAPS Steps](#) (see page 316)
- [Oracle Steps](#) (see page 320)
- [TIBCO Steps](#) (see page 333)
- [Sonic Steps](#) (see page 342)
- [webMethods Steps](#) (see page 345)
- [IBM Steps](#) (see page 356)
- [SAP Steps](#) (see page 362)
- [Selenium Integration Steps](#) (see page 372)
- [LISA Virtual Service Environment Steps](#) (see page 379)
- [CAI Steps](#) (see page 379)
- [Mobile Steps](#) (see page 382)
- [Custom Extension Steps](#) (see page 385)

## Test Step Information

**The following test steps are standard:**

- [Abort the Test](#) (see page 175)
- [End the Test](#) (see page 175)
- [Fail the Test](#) (see page 175)

## Abort the Test

**Follow these steps:**

1. Right-click the test step after which you want to abort the test case.
2. Select For Next Step, Abort the Test.

The Abort the Test step quits the test case and marks the step as having aborted.

## End the Test

**Follow these steps:**

1. Right-click the test step after which you want to end the test case.
2. Select For Next Step, End the Test.

The step completes the test and marks the test as having ended successfully.

## Fail the Test

**Follow these steps:**

1. Right-click the test step after which you want to fail the test case.
2. Select For Next Step, Fail the Test.

The Fail the Test step fails the test case and marks the test as having failed.

## Web-Web Services Steps

**The following steps are available:**

[HTTP-HTML Request Step](#) (see page 176)

[REST Step](#) (see page 186)

[Web Service Execution \(XML\) Step](#) (see page 186)

[WSDL Validation](#) (see page 228)

[Web-Raw SOAP Request](#) (see page 229)

[Base64 Encoder](#) (see page 230)

[Multipart MIME Step](#) (see page 231)

[SAML Assertion Query](#) (see page 233)

## HTTP-HTML Request Step

This step is used while testing a traditional web application to send and receive HTTP(S) requests. Requests can include GET parameters and POST parameters and optionally, embedded images as a response. You can also record the HTTP steps using the Website Proxy Recorder.

The HTTP/HTML Request step has a default name using this convention: *HTTP(s) ("GET" or "POST") (leaf of URL)*. An example is **HTTP GET rejectCard.jsp**. You can change step names at any time.

You can manually execute the HTTP/HTML step at design time (similar to the WS step). When you select Actions, Replay through here, DevTest saves the step responses so the step editors can display the response values.

When you add this step to a test case, the step editor includes the following tabs:

- [URL Transaction Info Tab](#) (see page 177)
- [HTTP Headers Tab](#) (see page 181)
- [Response Tab](#) (see page 181)
- [SSL Tab](#) (see page 182)



## URL Transaction Info Tab

Specify the information that is used to construct the URL on the URL Transaction Info tab.

You can set up the URL transaction information with either of the following options:

- Specify URL in parts
- Use property

### Specify URL in parts

Select the Specify URL in parts option (default) to specify the URL in its essential pieces.

#### Protocol

The protocol that is used to communicate with the web server. The default is **http**.

#### Host Name

The host name of the web server. Use the property SERVER or enter hostname or IP address of your application server. The host name can be a domain name, such as **www.mycompany.com** or an IP address, such as **123.4.5.6**. For a local web server, use the host name **localhost** or the IP address **127.0.0.1**.

#### Port

(Optional) If necessary, use the PORT property or the port on the web server that is used to access the web server. For example, the port that is required to access the Apache Tomcat web server by default is 8080.

#### Path

The path to the file to access. For example, if the URL to access is **http://localhost:8080/mysite/index.jsp**, enter **mysite/index.jsp** in the Path field.

#### User

Enter if a user ID is required for the application server.

#### Password

Enter if a password is required for the application server.

#### Encoding

The property governs the Encoding drop-down:  
**lisa.supported.html.request.encodings=ISO-8859-1, UTF-8, Shift\_JIS, EUC-JP, Windows-31J**

You can change the comma-separated list to include the encodings to support. The underlying JVM must also support all encodings in this list. If a web page uses an encoding that is not supported in the list, then the drop-down entry is blank. If you save in that situation, DevTest replaces the encoding with the DevTest default (file.encoding key in lisa.properties). Also, if you do not select an encoding when creating a HTTP/HTML Request step, then the DevTest default encoding is assumed.

#### URL Parameters

GET (or URL) Request Parameters: these request parameters are passed as part of the URL, and so they are exposed to the user in address bar of the web browser.

#### POST Parameters

POST Request Parameters. The request parameters are passed as part of the body of the page request. They are not exposed to the user in the address bar of the web browser.

#### Form Encoding

During a step execution, parameters are URL encoded as they are sent. The MIME type that is used is application/form-urlencoded.





#### All Known State




All known properties, such as test case properties, data sets, and filters, are listed.

#### Download images referenced (test bandwidth)

If this element is selected, the step downloads web page images into the test environment. If you do not select this check box, no images are downloaded.

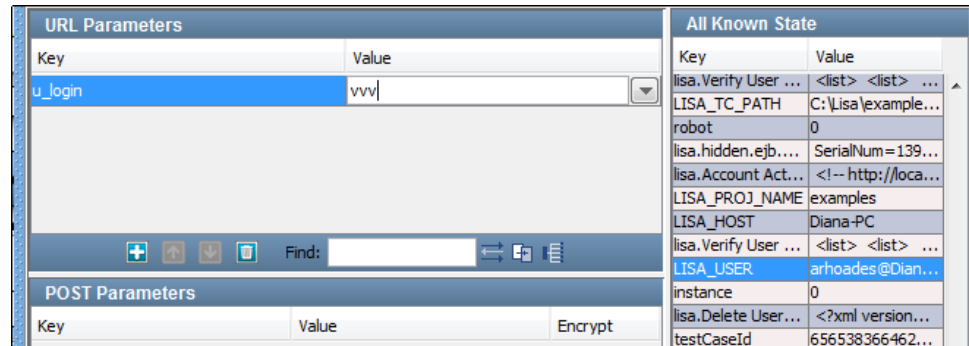
The following table describes other functions that are part of the toolbar that is available on the URL Parameters and POST Parameters sections.

Field	Icon	Description
Add		Add a request parameter
Up		Move an existing parameter up in the list of parameters
Down		Move an existing parameter down in the list of parameters
Delete		Delete an existing parameter
Find		Find text

Auto-Generate a Filter		From the referring step to make this parameter dynamic. Create a new filter to auto-populate this property at run time. For more information on filters, see the Filters section.
Apply selected All Known State property		Apply state to the parameter. For more information on applying state, see the All Known State section that follows.
Auto Apply all Known State properties		Apply all state to all properties possible by patterns. For more information on applying state, see the All Known State section that follows.


### All Known State

All known properties, such as test case properties, data sets, and filters, are displayed in the All Known State panel.



You can assign the values of properties to URL request parameters.

For example, to assign the value of the **LISA\_USER** data set key to the **u\_login** request parameter in the previous example:

1. Select the **u\_login** key in the URL Parameters pane.
2. Select the **LISA\_USER** key in the All Known State panel.
3. Click Apply selected All Known State property to current parameter .
 

A warning message opens asking you to confirm the impending change.
4. Click OK.

The new property is displayed in the URL Parameters pane.

If all the names of the URL Parameter keys equal the names of the All Known State keys, you can click Apply to All to assign all the properties to the associated parameters quickly.

### Use Property Option

If the Use property option button is selected, you can specify the following parameters:

#### Property Key

Specify a property that contains the connection information.


#### Download images referenced (test bandwidth)

If this element is selected, the step downloads web page images into the test environment. If you do not select this check box, no images are downloaded.

## HTTP Headers Tab

On the HTTP Headers tab, create any custom HTTP headers.

- The top section, Custom HTTP Headers (Current Only), is for headers that are only sent to the server for this request.
- The bottom section, Custom HTTP Headers (Persist), is for headers that are sent on this transaction and every other transaction in the test.

To create a request parameter in either section, click Add  and change the key and value to the target values.

## Response Tab

On the Response tab, view the HTTP response that the server returns when this test was recorded. You can view:

- The source of the response.
- The DOM Tree of the response.

## SSL Tab

The SSL tab lets you enter information for either single or multiple SSL certificates.

### Using a Single SSL Certificate

Enter the following information:

#### SSL Keystore File

The name of the keystore file where the client identity certificate is stored. The file can be in JKS or PKCS format.

#### SSL Keystore password

Password for the keystore file.

#### SSL Key alias

The keystore attribute that defines the alias that is used to store and retrieve the private key for the server.

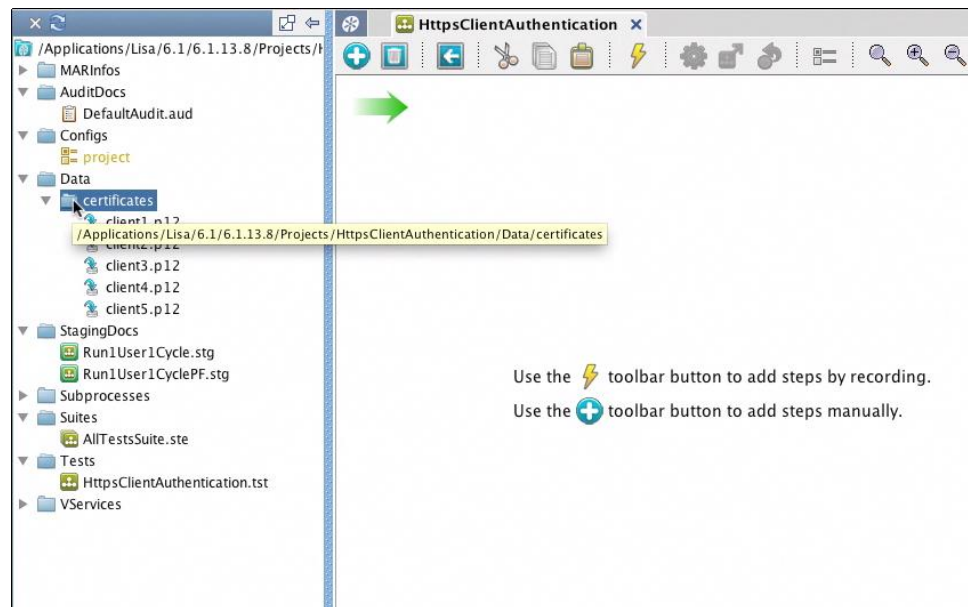
#### SSL Key password

An optional password for the key entry if using a JKS keystore, and key has a different password from keystore.

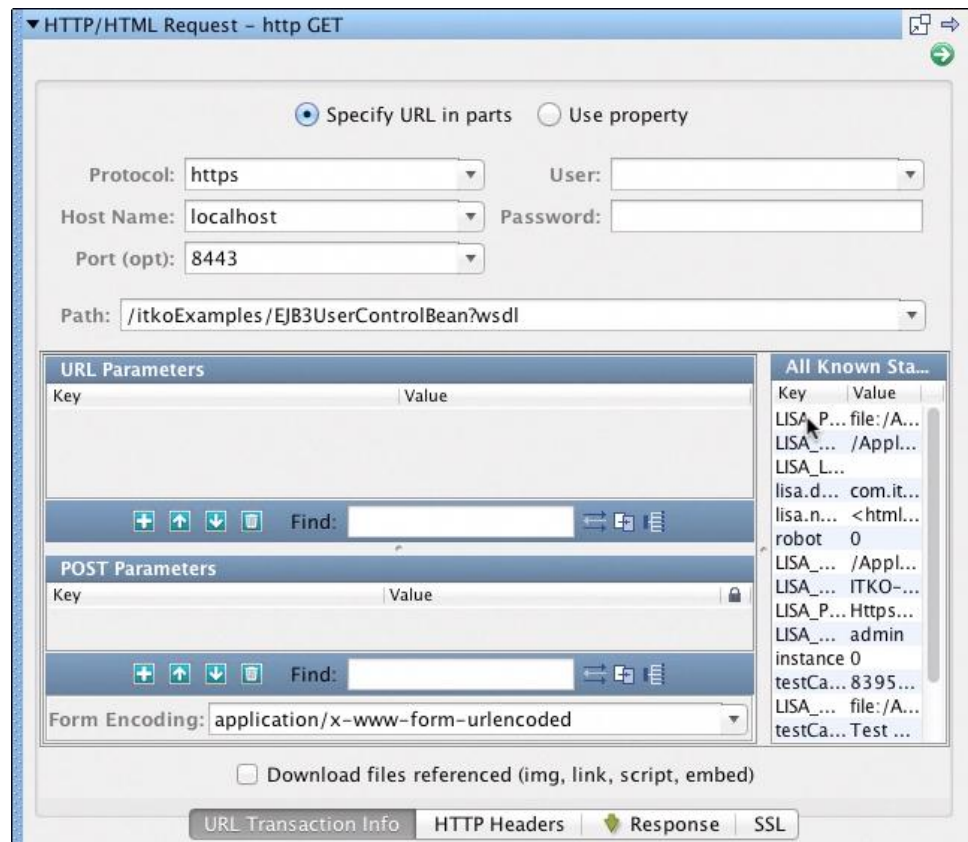
### Using Multiple SSL Certificates

To use multiple certificates on one step, store your SSL information in a data set. The following example shows how you can use a data sheet to store information about multiple certificates.

In this example, the demo server is running in the background and it has been configured to require client-side authentication. The demo server accepts five certificates. The keystore files for those certificates are in the project structure in a Data with the name **certificates**. These keystore files are pkcs12 keystores.



Next, an HTTPS Request step is added and https is specified as the protocol and the port and path are entered.



To ensure this process works with all the certificates and only run the test once, a data set is created for this test step. Use the Create your own Data Sheet data set to create the data set. The data set has the name **Certificate Information** and it is configured to run through the data set rows once, and then exit. Each certificate has five certificates and four parameters. When all the information is complete, use the Create Data Sheet Skeleton button to open the editor.

▼ Create your own Data Sheet – Create your own Data Sheet

Name:

☐ Local ☐ Random Max Records To Fetch:

At end of data, ☐ Start over ☒ Execute

Enter Data Sheet Params

Number of Rows:

Column Names:   
(Enter comma separated column names)

For keystoreFile, we use the short file name of the keystore so that relative paths can be used later. To encrypt the password columns, right-click the column label and select Encrypt.

▼ Create your own Data Sheet – Create your own Data Sheet

Name:

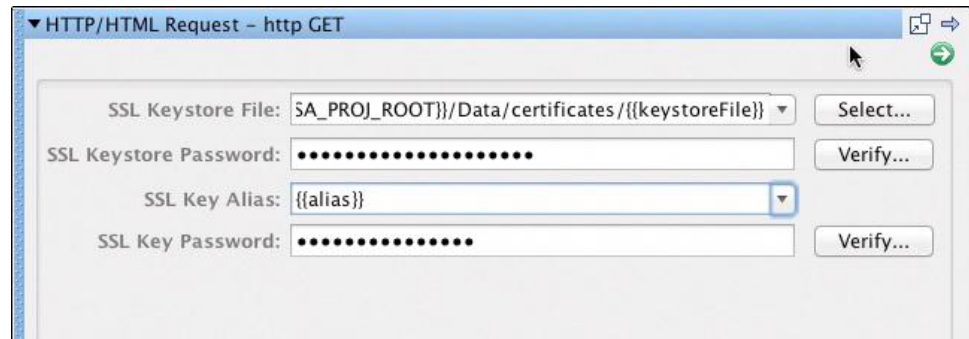
☐ Local ☐ Random Max Records To Fetch:

At end of data, ☐ Start over ☒ Execute

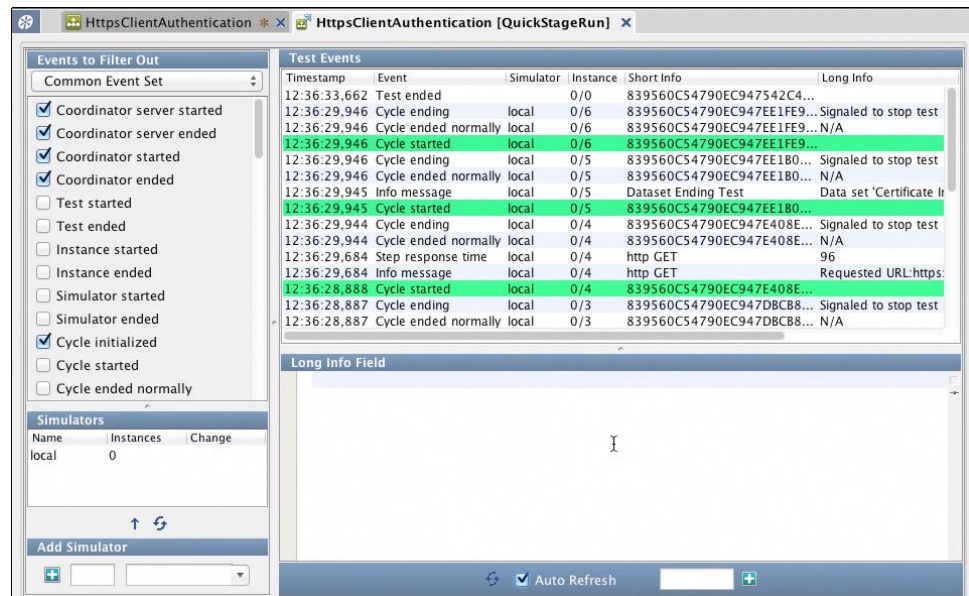
keystoreFile	keystorePa...	keyPassword	alias
client1.p12	123456	123456	client1key
client2.p12	123456	123456	client2key
client3.p12	123456	123456	client3key
client4.p12	123456	123456	client4key
client5.p12	123456	123456	client5key

Return to the HTTP step and select the SSL tab. Use property substitution for the variables. For the password fields, enter `{{keystorePassword}}` and `{{keyPassword}}`.





The test is staged with Stage a Quick Test, and you can see that all of the tests completed successfully. All five certificates were tested against the demo server and client authentication was successful for all of them.



## REST Step

Use the REST step when testing REST applications.

This step is used to send and receive HTTP(S) requests, including GET and POST parameters.

To view parameters, click the Test State button on the right vertical bar. The Test State area slides open. You can dock, pin, and hide the list.

To view the HTTP response, click the Response button.

At the bottom of the Response panel is a selection of filters and an assertion that you can add to the response.

The demo server contains an example of invoking a JSON service to retrieve information from the LISA Bank user database. The URL is `http://localhost:8080/rest-example/`.

An example test case for the REST step is in the examples project. The name of the test case is **rest-example.tst**.

## Web Service Execution (XML) Step


The Web Service Execution (XML) step is designed to execute an operation on a SOAP-based Web Service using an HTTP POST or JMS message.

Access to a WSDL is not required; rather, it is a recommended but optional piece of configuration information. If a WSDL is configured, it helps in the process of building a SOAP message to be sent to the service. This step lets you manipulate the raw SOAP message (XML) directly. This feature provides flexibility and power, but does expose you to the details of how web services work.

In general, the top portion of the editor is dedicated to how and where to send the SOAP message. The bottom portion is dedicated to the contents of the message.

The Web Service Execution (XML) step has a default name using this convention: *Web Service webServiceOperation name*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

After the test step opens, it has two tabs, with each tab having multiple sub tabs.

The PRO  icon switches between basic and advanced options. Some tabs and options are only available when PRO is selected.

## Connection Tab

The Connection tab has fields for the connection. The tab has sub tabs on the top bar and bottom bar.

- The top bar - for viewing the Visual XML, Raw XML, Headers, Attachments.
- The bottom bar - for Request and Response.
  - Basic Configuration
  - Design Time Execution

## Basic Configuration

### Connection

#### WSDL URL


The WSDL URL is an optional but recommended field (denoted by its slightly gray color).

The WSDL URL must be a URL (either file:/, http:/, or https:/). From the More

Options menu  you can:

- Browse the file system for a local WSDL or WSDL Bundle file.
- Search a UDDI Registry (which populates the advanced UDDI access point lookup).
- To migrate from the legacy WS step, select WSDL from hotDeploy.
- Create and use a WSDL bundle. You can also create a WSDL Bundle from the Actions menu, but from the Options menu, DevTest automatically populates the WSDL URL with the resulting WSDL bundle file URL. Or if you already have a WSDL URL populated, it prepopulates the WSDL URL in the WSDL bundle dialog.

When you enter a WSDL URL that is not already a WSDL bundle, DevTest creates a WSDL bundle and stores it locally in the Data/wsdl directory for the project. This action caches the WSDL locally for quicker access. DevTest parses the WSDL and uses its schema to build sample SOAP messages. The Visual XML editor also uses the WSDL to help you manually edit the SOAP message. DevTest first tries to load a cached WSDL bundle whenever processing the WSDL. If the external WSDL has changed and you want to force

the local WSDL cache to update, use the Refresh WSDL Cache  button. You can manually drop a WSDL bundle into the Data/wsdl any time. When the step tries to process the "live" WSDL URL, it uses the cached bundle instead.

#### Service, Port, Operation

If the WSDL URL is populated, the WSDL is processed and the Service, Port, and Operation selections are populated. These optional but recommended fields can be used to build a sample SOAP request message. Selecting a port also updates the Endpoint URL to match the definition in the WSDL. Changing the WSDL URL causes these items to be refreshed. If the Endpoint URL and SOAP Message are unchanged, they update also to correspond to the new WSDL, service, port, and operation that are selected.

If the endpoint was changed and no longer matches the endpoint in the WSDL, a Warning button appears next to the field. A tooltip on the button indicates the differences between the entered value and the WSDL definition. Clicking the button updates the field to match the WSDL definition.

If the SOAP Request Message no longer matches the default, it is not updated automatically. You can force the SOAP Request Message to be updated by

using the Build Message  button next to the Operation field.

### Operation

Any of the following options can be used here:

- Build empty SOAP request message.
- Build full SOAP request message.

### Port

This field indicates the server port on which the service is available.

### On Error

This field indicates what action to be taken when some error occurs during execution.

### Endpoint

The URL to the SAML Query API of the Identity Provider.

### Build Options

When building sample SOAP messages, various build options are used to determine what to do in various situations.

- Use String Pattern for Value: When selected, it populates element values using DevTest string patterns as opposed to using a hard-coded literal value.
- Default Literal Value: When not using string patterns, use this literal value for all string values.
- Build All Choices: By default, only the first element in an XML schema choice is generated. To build all possible choice elements, select this option.

**Note:** The SOAP request is not valid if you include multiple choice elements. However, it provides a sample for each possible choice, which makes it easier to build a message when you do not use the first choice.

- Maximum Elements: Defines the maximum number of elements to include when building the sample message.
- Maximum Type: Defines the maximum number of complex schema types to include when building the sample message.
- Insert Comments: By default, comments that are related to the schema are generated. For example, when an element is optional, alternative choices, nillable elements, are generated. You do not see these comments in the Visual XML Editor, but they are visible in the Raw Editor.

**Tabs available in the Web Service Execution editor are described here:**

[Visual XML Tab](#) (see page 191)

[Raw XML Tab](#) (see page 194)

[Headers Tab](#) (see page 195)

[Attachments Tab](#) (see page 197)

[Addressing Tab](#) (see page 199)

[Security Tab](#) (see page 199)

## Visual XML Tab

The Visual XML Editor (or VXE) is a graphical editor for any XML document. A SOAP message is an XML document that conforms to the SOAP specification (SOAP schema). You can use the editor to build and edit the SOAP message.

The screenshot shows the Visual XML Editor (VXE) interface. At the top, there are tabs for 'Visual XML', 'Raw XML', 'POST SOAP', 'Headers', and 'Attachments'. Below the tabs is a tree view of the XML document structure. The tree shows a 'soapenv:Envelope' containing a 'soapenv:Body' which contains an 'addAddress' element. The 'addAddress' element has a 'username' child and an 'addressObject' child. The 'addressObject' element has several children: 'city', 'id', 'line1', 'line2', 'state', and 'zip'. Each element in the tree has a corresponding row in the table below, showing its type, occurrence, nil status, nillable status, and value.

Node	Type	Occurs	Nil	Nillable	Value
soapenv:Envelope	Envelope	1..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
soapenv:Body	Body	1..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
addAddress	addAddress	1..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
username	string	0..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	{{=:username:}}
addressObject	address	0..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
city	string	0..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	{{=:city:A*}}
id	string	0..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	{{=:id:A*(5-...)}}
line1	string	0..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	{{=:line1:A*}}
line2	string	0..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	{{=:line2:A*}}
state	string	0..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	{{=:state:A*}}
zip	int	1..1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	{{=:zip:1}}

At the bottom of the interface, there are buttons for 'Validation Results', 'View XML Schema Source', 'Error Log', 'Request Editor', 'Request', and 'Response'.

The table shows the XML document, including the SOAP Envelope and Body elements.

### Type

Shows the type for each element.

### Occurs

Indicates how many elements are expected. The first number indicates the minimum number of times the element can occur. Zero means that it is optional and can be removed. The second number indicates the maximum number of times the element can occur. Infinity means that there can be an unlimited number of elements of that name.

### Nil

Lets you set the element value as **nil** or **un-nil**. Selecting the check box removes any element children or values but leaves all attributes alone. Clearing the check box populates all expected children and attributes as defined in the WSDL schema.

### Nillable

Indicates whether the element can be nil. A red icon indicates that the element is non-nillable, but is set to **nil**. A green icon indicates that the element is non-nillable and is not set to **nil**.

### **Value**

You can type the element values directly into the Value column. If the element type is a known type, specialized edit buttons appear.

To select the columns to display or hide, right-click the column heads.

When you right-click the editor, a context-sensitive menu provides options for manipulating the document. This menu contains the following options:

### **Add Schema Attribute/Element**

Lets you select from a list of valid attributes or elements. If a schema is present and an element or attribute is selected in the editor, child elements and attributes can be selected to be added. If more than 20 schema objects are available, a dialog can be used to select the schema object. This dialog contains a search text field, which makes it easier to find a specific schema object when there are dozens of them.

### **Add Element**

Add an element to the document.

### **Add Attribute**

Add an attribute to the document.

### **Add Text**

### **Remove Element/Attribute**

Remove the selected element or attribute.

### **Move Up/Down**

Move the selected node up or down.

### **Convert to Attachment**

A shortcut method for creating a standard referenced attachment. For more information, see [Attachments Tab](#) (see page 197).

### **Convert to XOP Attachment**

A shortcut method for creating a standard referenced XOP Include attachment. For more information, see [Attachments Tab](#) (see page 197).

### **Create XML Data Set**

A shortcut method for generating an XML Data Set. A typical use case is to build a full SOAP message. Then select the section of the XML document that you want as the first record of the data set. Creating an XML Data Set automatically populates the first record with the selected XML element tree. The new data set property is set as the value in the editor.



**Hide Text Nodes**

By default, DevTest hides text nodes that are typically redundant (such as white space). However, it is occasionally useful to view them, including when the XML element is a mixed type element that supports intermixed elements and text.

**Hide Namespace Nodes**

By default, DevTest hides namespace declarations and namespace prefix declarations. You can show them to confirm a prefix value or to change prefixes or namespace scoping.

**Validate**

Validates the XML and displays the results in the Validation Results pane at the bottom of the window.

To switch between displaying and hiding the results of the XML validation, click Validation Results.

You can also use keyboard shortcuts for some tasks:

- To remove the selected element or attribute on Windows, press Ctrl+Backspace.
- To move the selected node up on Windows, press Ctrl+up arrow.
- To move the selected node down on Windows, press Ctrl+down arrow.

Right-clicking an attribute displays a menu with some of the functionality of the general right-click menu.

**Editing the Type Field**

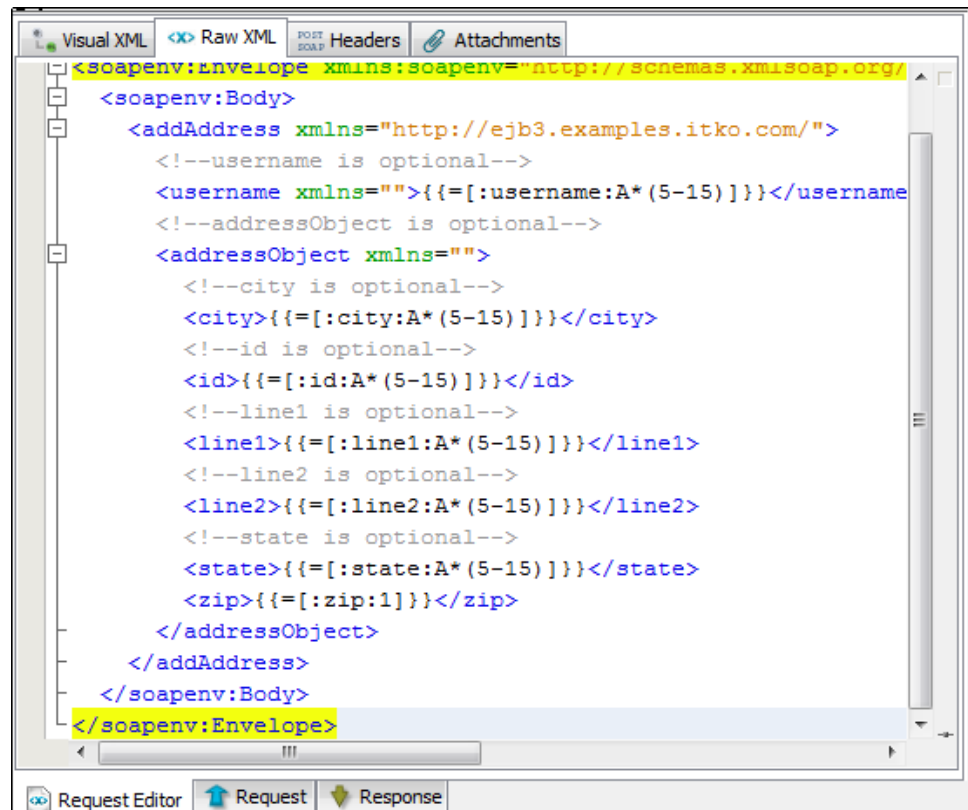
The Type column shows the XML schema type (local name) and has a tool tip to show the fully qualified name (qName) with namespace. This column can be edited for derived XSD types. Only base types with derived types can be edited.

When the column is available for editing, a list of available derived types and the base types are presented in a combo box. You can select a type, and the associated element is now associated with the selected type.

Changing the type removes all child elements and attributes from an element and sets the nil attribute on the element to nil=true.

## Raw XML Tab

The Raw XML Editor is a text-based editor that is XML aware and lets you manually edit the raw XML SOAP message. Any changes that are made are seen when you switch back to the [Visual XML Editor](#) (see page 191) (and conversely).



The Raw XML Editor has a right-click menu for formatting the raw XML.

**Note:** The Remove whitespace option trims whitespace and all ASCII control characters from the beginning and end of each line in the document.

If you edit the document to make it invalid XML, the Visual XML Editor could show an error message.

Fix your changes in the Raw XML Editor, and the Visual XML Editor begins working again.

## Headers Tab

The Headers tab lets you insert headers that are transmitted with the SOAP message (for example, HTTP Headers or JMS properties).

Key	Value
HeaderValue	

Accept  
Accept-Language  
User-Agent  
Connection  
Authorization

+

↑

↓

✖

To add a header row and select a header from the drop-down list, click the plus sign.

### Accept

The Accept request-header field can be used to specify specific media types that are acceptable for the response. Use accept headers to indicate that the request is specifically limited to a small set of types. For example, in the case of a request for an in-line image.

This field contains a semicolon-separated list of representation schemes that are accepted in the response to this request.

```
Accept          = "Accept" ":"
                  #( media-range [ accept-params ] )
```

### Accept - Language

The Accept - Language header field is similar to Accept, but lists the language values that are preferable in the response. A response in an unspecified language is not illegal.

Accept-Language = "Accept-Language" ":"

1#( language-range [ ";" "q" "=" qvalue ] )

language-range = ( ( 1\*8ALPHA \*( "-" 1\*8ALPHA ) ) | "\*" )

#### User - Agent

The User-Agent request-header field contains information about the user agent originating the request.

The headers tab is for statistics, tracing protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid specific user agent limitations. User agents should include this field with requests.

By convention, the product tokens are listed in order of their significance for identifying the application.

User-Agent = "User-Agent" ":" 1\*( product | comment

The Connection general-header field lets the sender specify options that are appropriate for the specific connection. Proxies and *must not* communicate the field over further connections.

Connection = "Connection" ":" 1\*(connection-token)  
connection-token = token

#### Authorization

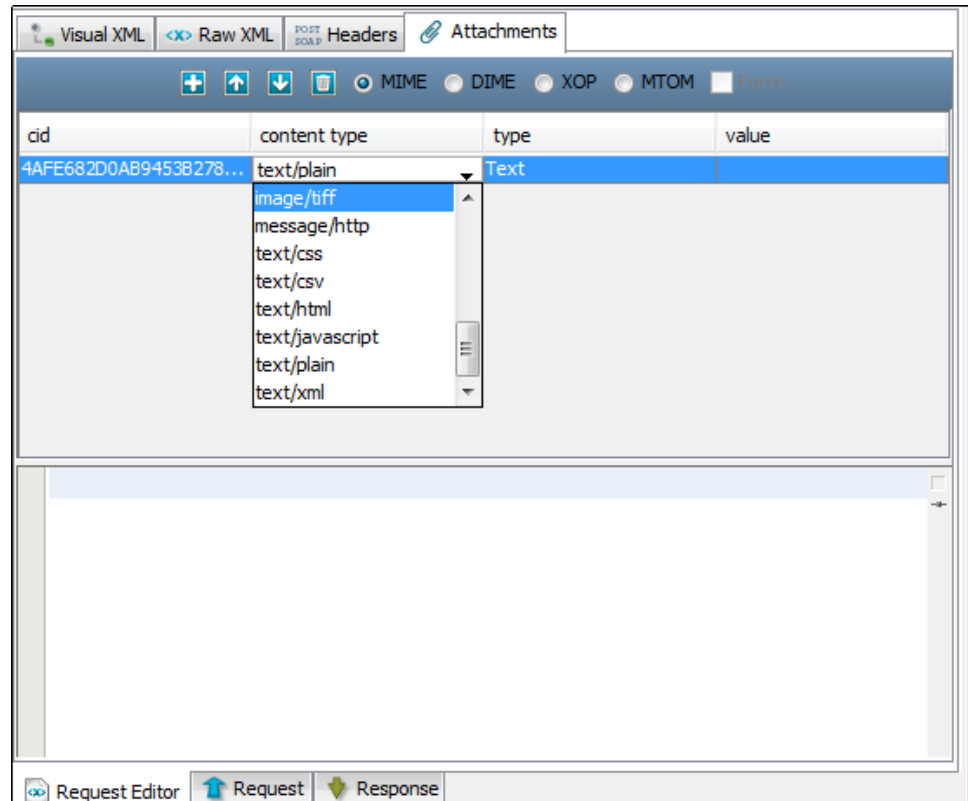
To authenticate itself with a server, a user agent (usually, but not necessarily, after receiving a 401 response) includes an Authorization request-header field with the request. The Authorization field value consists of credentials containing the authentication information of the user agent for the realm of the resource being requested.

Authorization = "Authorization" ":" credential

DevTest does not support using properties in the Headers tab. The UI takes a user name and password during design and calculates the value of the authorization header. To use properties, set the **lisa.http.user** and **lisa.http.pass** properties either in local.properties or in config files and DevTest uses those properties during run time.

## Attachments Tab

The Attachments tab lets you edit attachment data.



### Referenced Attachments

*Referenced attachments* are referenced in the SOAP message. To use referenced attachments, in the VXE, right-click the element that you want to be the referenced attachment and select the appropriate action:

- Convert to Attachment (for MIME and DIME)
- Convert to XOP Attachment (for MTOM/XOP Include style)

This action automatically creates the necessary elements and attributes and configures the content ID that is used to match up the element with the attachment. The action then completes the following actions:

- Switches to the Attachments tab
- Prepopulates a new attachment with the content ID
- Selects a default content type and attachment type
- Populates the value with any existing element data from the VXE

### Unreferenced Attachments

If you plan to use unreferenced (anonymous) attachments, use the Attachments tab to add an attachment manually.

Use the Add, Up, Down, and Delete icons to add, remove, or rearrange the attachments in the table.

#### MIME DIME XOP MTOM

This field controls how the attachments are sent, using MIME, DIME, XOP, or MTOM standards. XOP sends different content headers that are based on the SOAP version.

When MTOM is selected, any base64binary schema types are automatically optimized using the XOP standard. Adding attachments manually is not necessary. If an element is already configured as an attachment, it is left alone. Any extra attachments added manually are also sent.

If you select Force (even if the document contains no base64binary elements) the document is formatted and sent as an attachment (Microsoft MTOM method).

The limitation of automatically optimizing elements is that the VXE must understand the element as a base64binary schema type (or extension/restriction thereof). If you use a data set or property, the expanded property or first data set entry must contain all possible elements to optimize. If the VXE does not display the element that must be optimized, the element is not optimized.

#### cid

The Content ID that can be used in an **href** attribute in the SOAP message to link the element to the attachment data.

#### content type

The mime encoding type that is used to assist the server in how to process the attachment data.

#### type value

The DevTest attachment type determines how to edit and interpret the value data. Each type has its own editor.

### Type Editors

#### XML

An XML-aware text editor to edit the attachment value.

#### Text

A text-based editor to edit the attachment value.

**Base64 Encoded**

A text-based editor to edit the attachment value and a bytes viewer to view the decoded binary data.

**Hex Encoded**

A text-based editor to edit the attachment value and a bytes viewer to view the decoded binary data.

**URL/Text**

A URL field to edit the attachment value and a text data viewer to view the results of loading the data from the URL.

**URL/XML**

A URL field to edit the attachment value and a XML-aware text data viewer to view the results of loading the data from the URL.

**URL/Binary**

A URL field to edit the attachment value and a binary data viewer to view the results of loading the data from the URL.

**Property**

A property field to edit the attachment value. If the resulting property is a string, it sends the attachment as text; otherwise it sends it as binary data.

**Property/URL**

A property field to edit the attachment value. The property value is assumed to be a URL. The URL content is loaded and sent as the attachment data.

## Addressing Tab


For information about the Addressing Tab, see [Advanced Settings](#) (see page 209).

## Security Tab

For more details on the Security Tab, see [Advanced Settings](#) (see page 209).

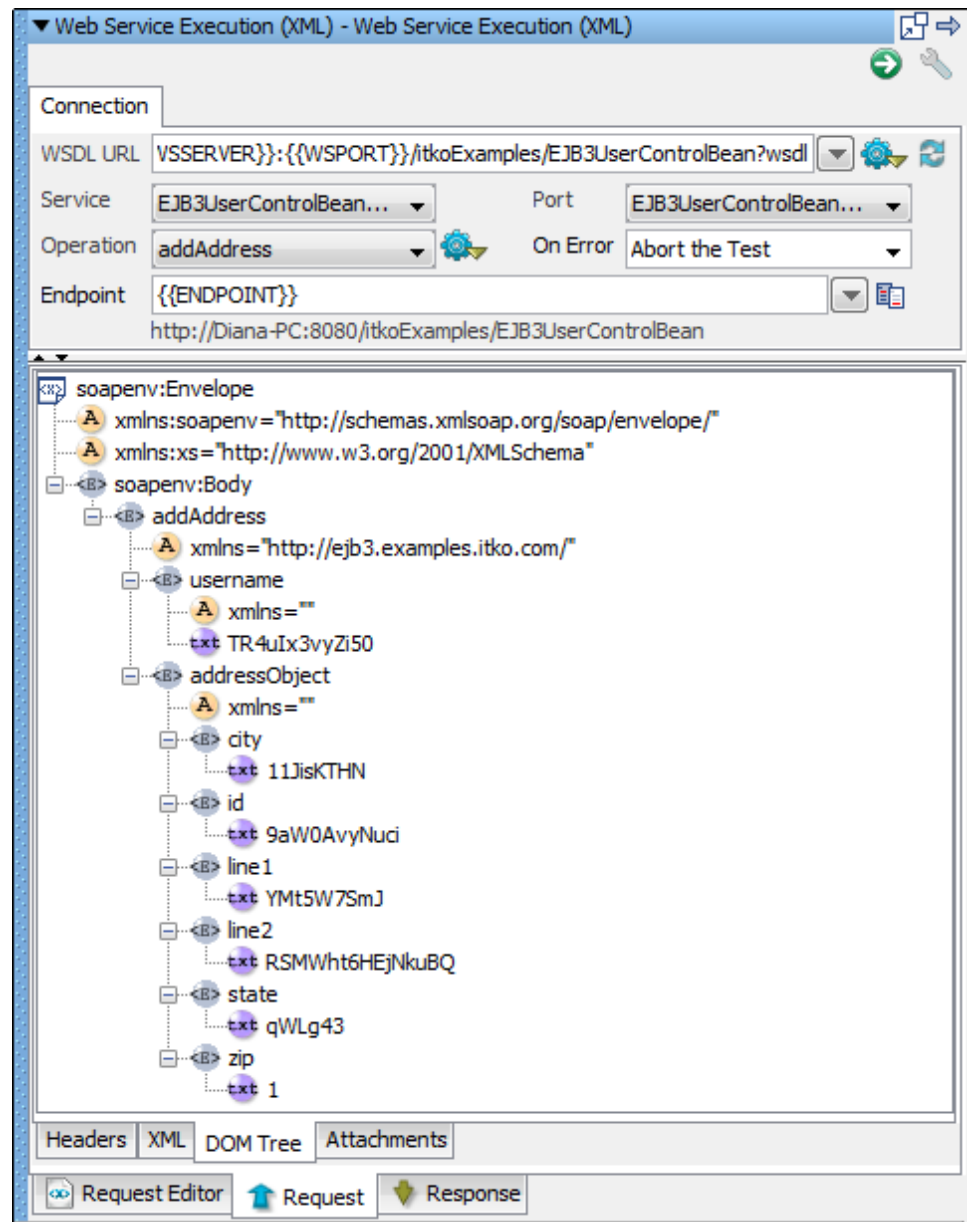
## Design Time Execution

After you finish configuring the connection information and building the SOAP Request Message, you can run the step at design time to test it.

Execute the Web Service operation by clicking Execute  in the upper right corner. After it has executed, the Request and Response tabs will be populated and it will switch to the Response tab automatically.

## Request Tab

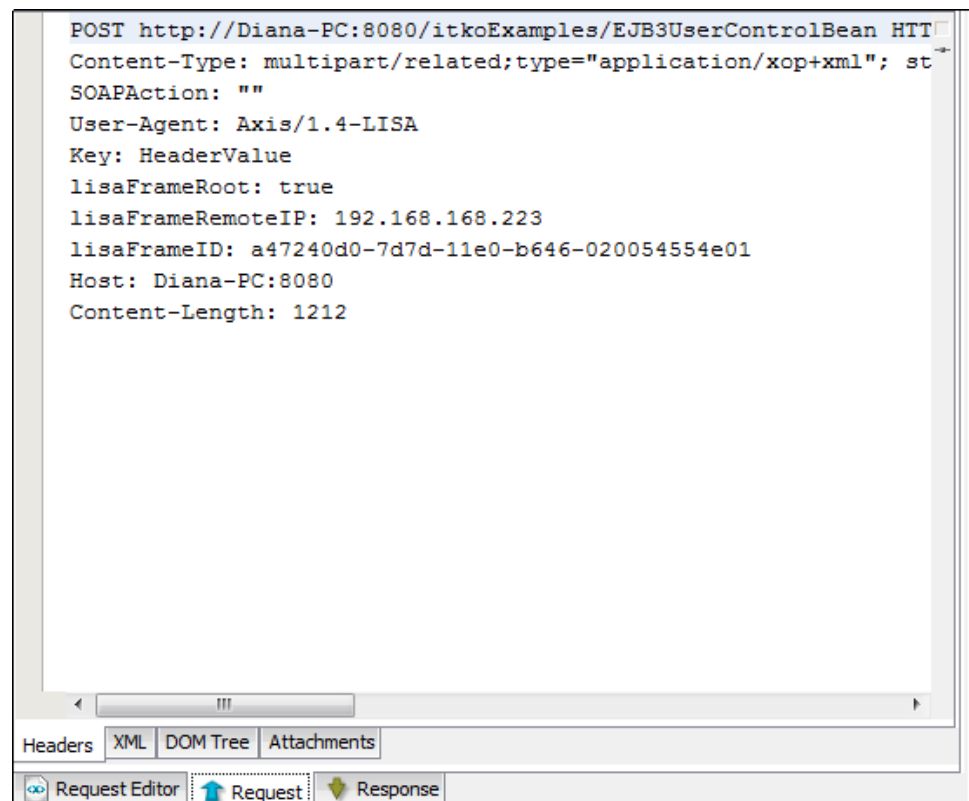
The Request tab shows the resulting request data that was sent after any post processing (for example, substituting DevTest properties). If the message contained any attachment, it does not show the raw MIME or DIME encoded message, but rather the processed message and attachments. To see the raw message, use a tool like TCPMon.



## Header Tab

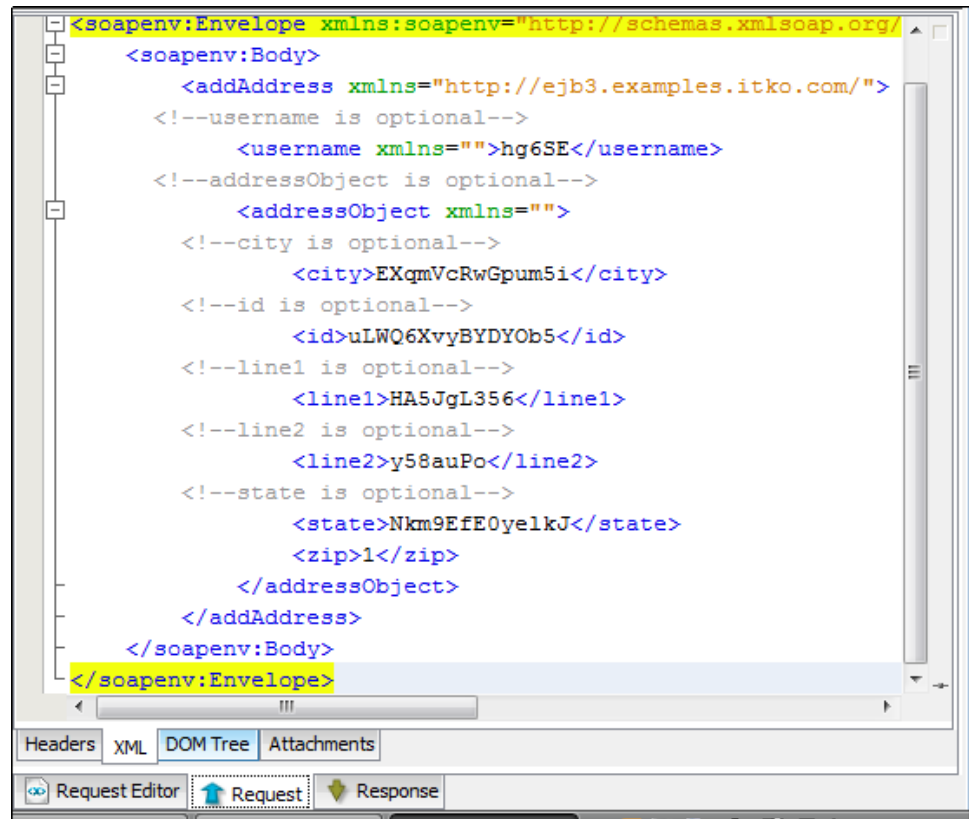
The Header tab shows the Transport Headers that were sent for the request.





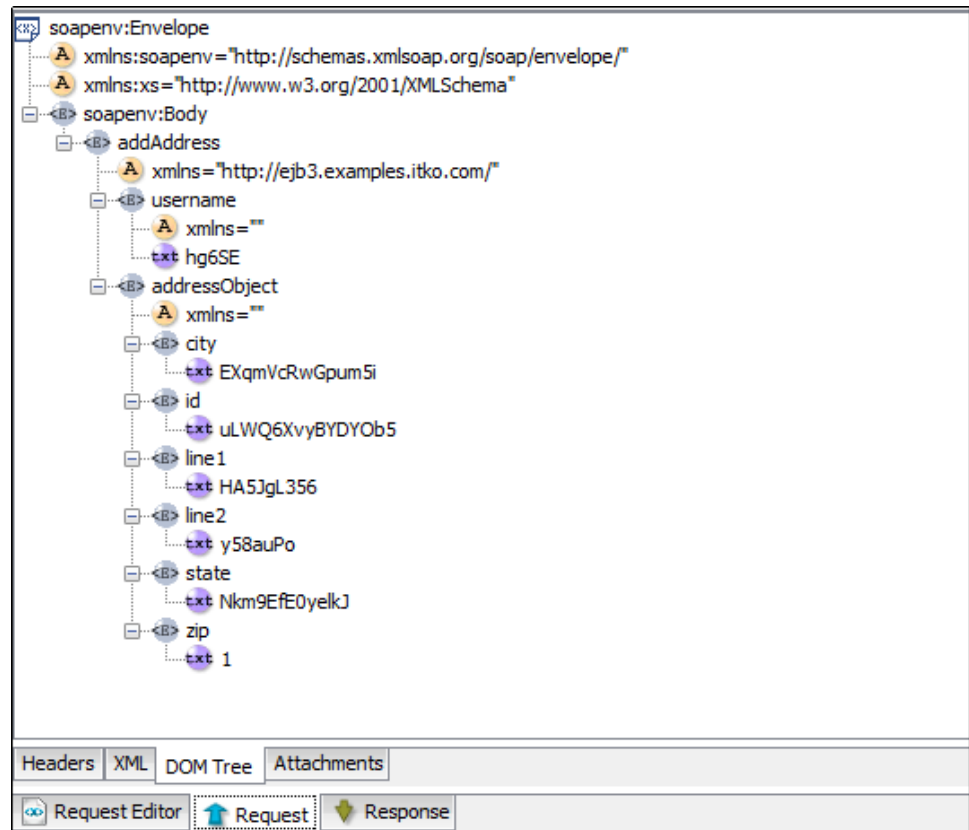
### XML Tab

The XML tab shows the raw SOAP message that was sent after any advanced processing.



### DOM Tree Tab

The DOM Tree tab shows a DOM tree for the SOAP message.

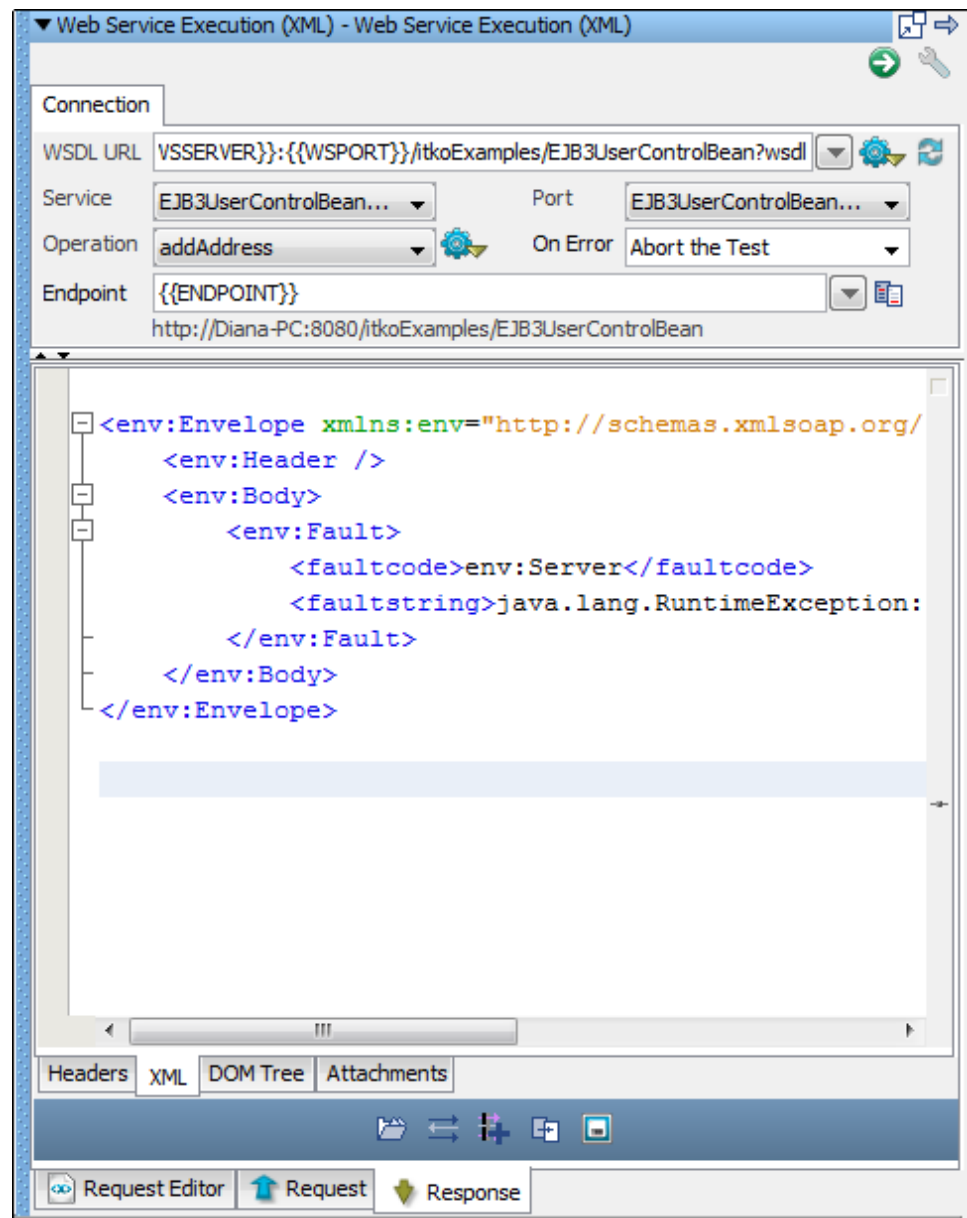


#### Attachments **Tab**

The Attachments tab shows any attachments that were sent.

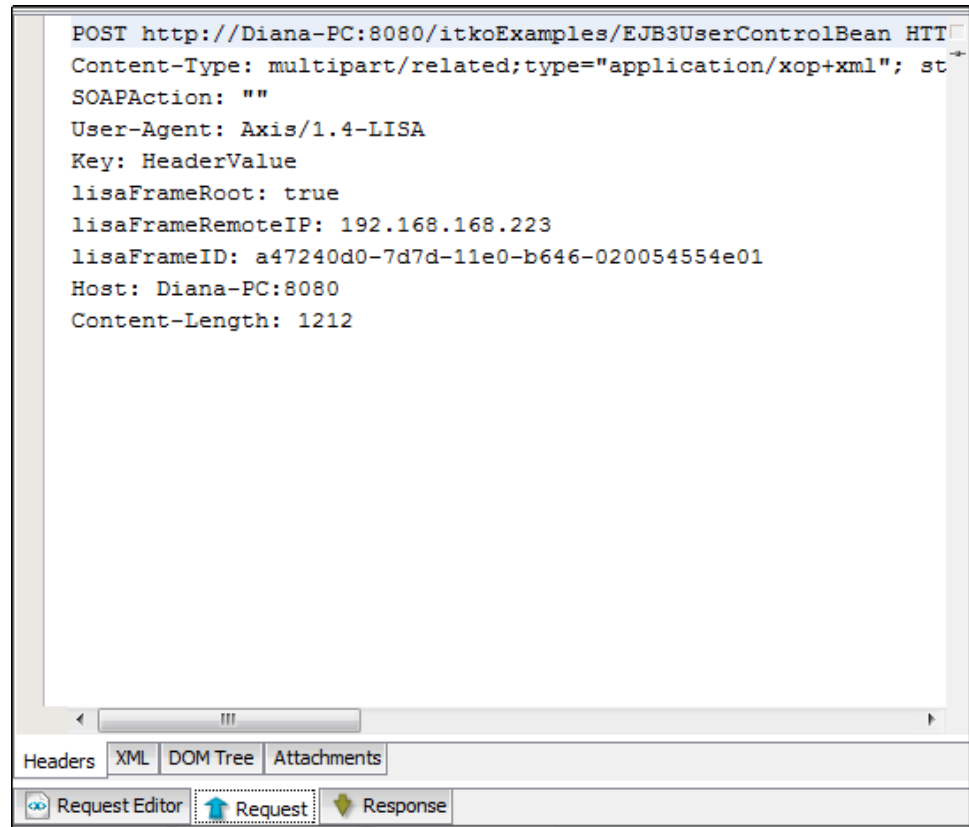
## Response Tab

The Response tab shows the resulting response data that was received. If the message contained any attachment, it does not show the raw MIME or DIME encoded message, but rather the processed message and attachments. To see the raw message, use a tool like TCPMon. If any advanced post-processing options are set (see the following window), the SOAP response message is shown post-processed.



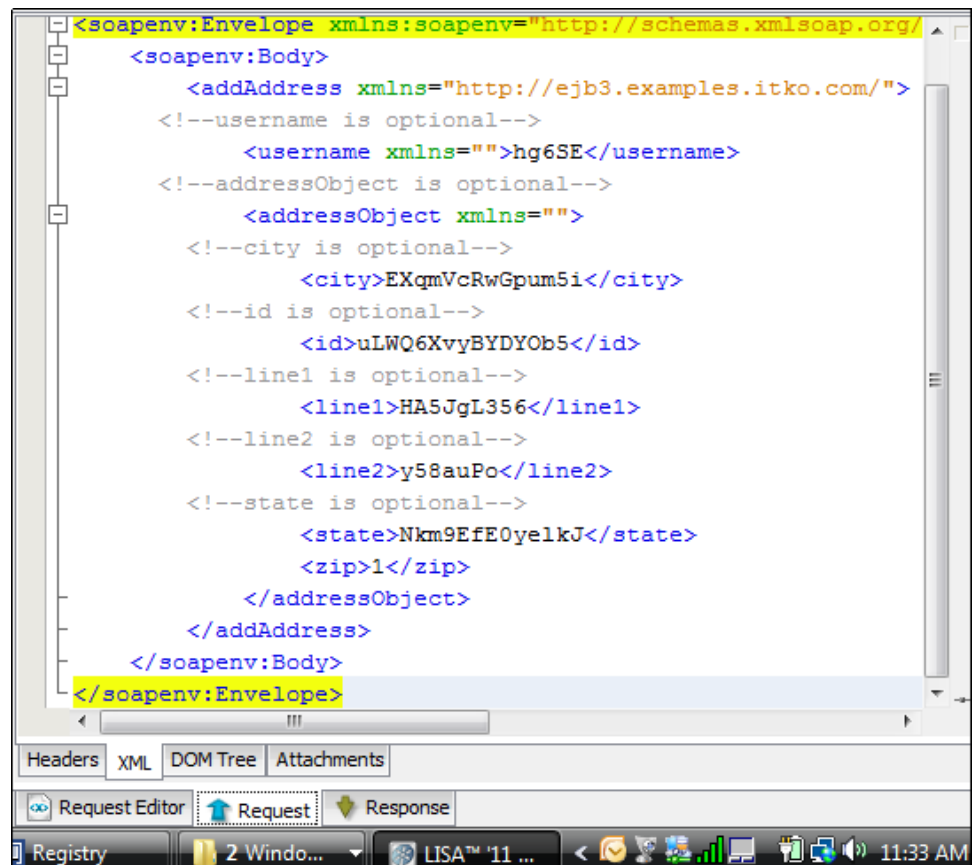
## Header Tab

The Header tab shows the Transport Headers that were received from the response.



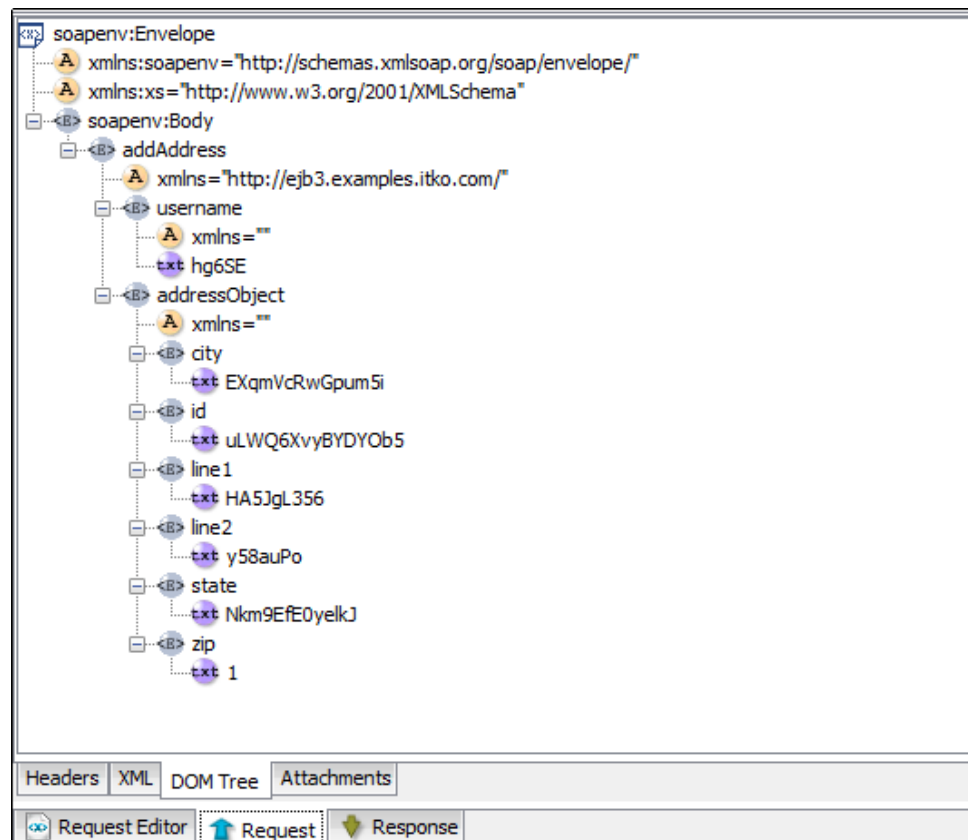
### XML Tab

The XML tab shows the raw SOAP message that was received after any advanced processing.



### DOM Tree Tab

The DOM Tree tab shows a DOM tree for the SOAP message. From this tab, you can quickly add filters and assertions on the resulting SOAP message.



### Attachments Tab

The Attachments tab shows any attachments that were received. You can access the received attachments using automatically generated DevTest properties (for use in later steps, filters, or assertions). For each attachment, the following properties are set.

**lisa.<step name>.rsp.attachment.<cid>**

Attachment value, byte, or String

**lisa.<step name>.rsp.attachment.contenttype.<cid>**

Content type (mime type, for example, text/plain)

**lisa.<step name>.rsp.attachment.<index>**

Attachment value, byte, or String

**lisa.<step name>.rsp.attachment.contenttype.<index>**

Content type (mime type, for example, text/plain)

**lisa.<step name>.rsp.attachment.contentid.<index>**

Content Id (cid)

The parameter **step name** is the DevTest step name that is being executed.

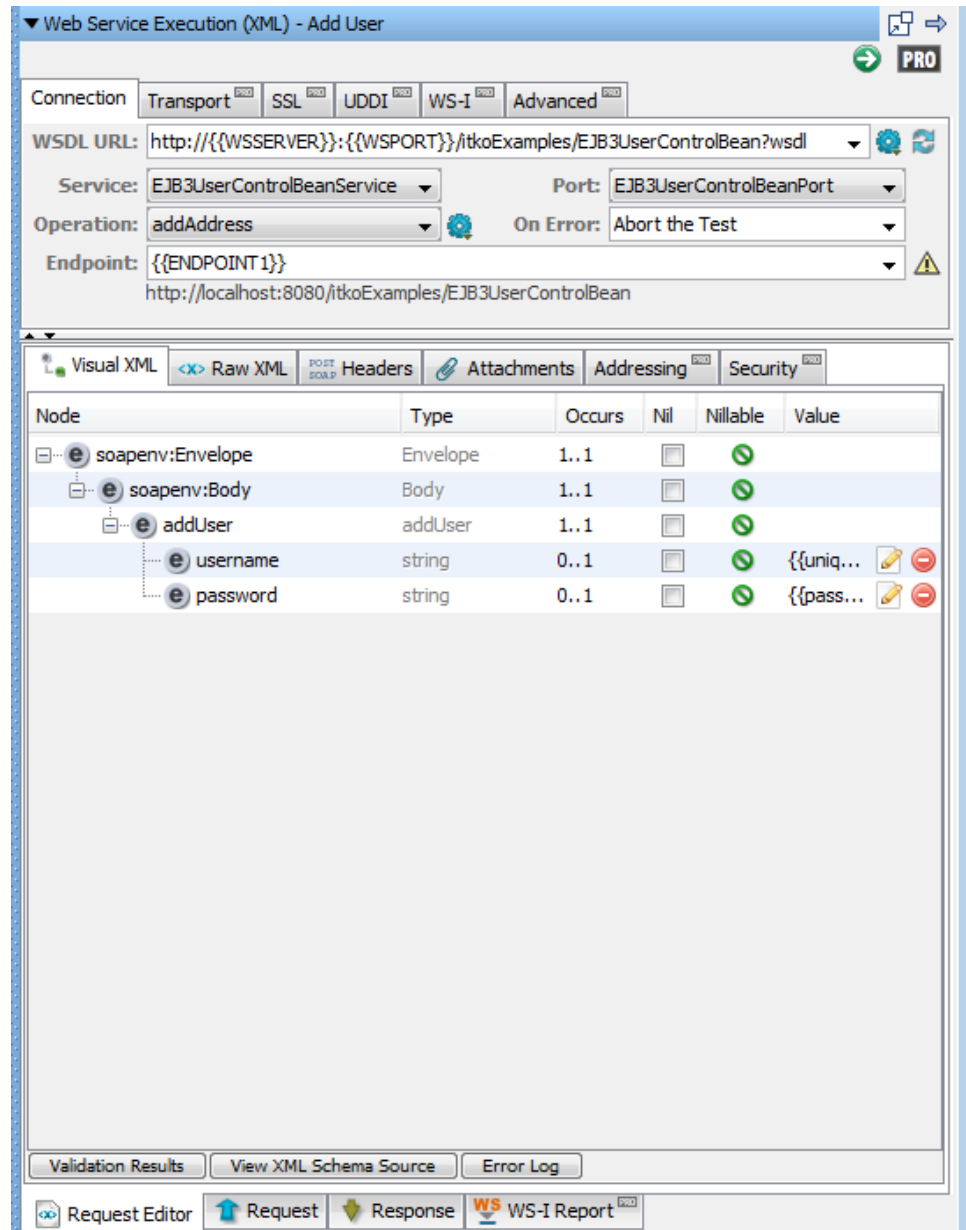
The parameter **cid** is the Content ID that is usually referenced in the SOAP message.

The parameter **index** starts at 0 and is increased for each attachment in the response attachments list.

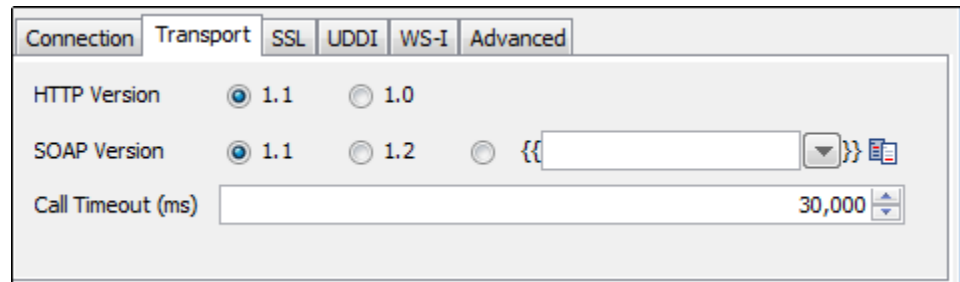


## Advanced Settings

To open the Advanced settings tabs, click PRO **PRO**. Five new tabs open in the top of the panel, and two new tabs open at the bottom level.



## Transport Tab



The screenshot shows a configuration window with five tabs: Connection, Transport, SSL, UDDI, and WS-I. The Transport tab is selected. It contains three settings: HTTP Version with radio buttons for 1.1 (selected) and 1.0; SOAP Version with radio buttons for 1.1 (selected), 1.2, and a custom option with a text box and a dropdown arrow; and Call Timeout (ms) with a text box containing 30,000 and a spin button.

### HTTP Version

This parameter controls which HTTP protocol is used when sending the operation request. The default is 1.1.

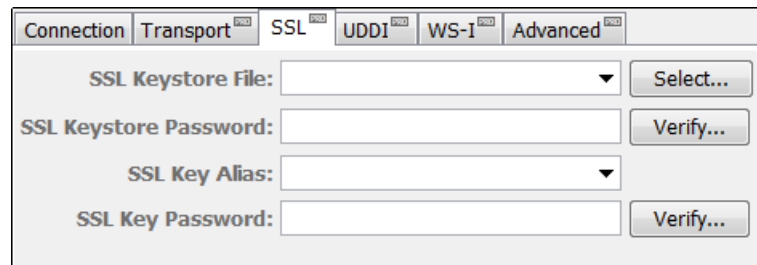
### SOAP Version

The SOAP version is auto-populated based on the WSDL definition. SOAP Version controls the generation of a number of transport headers (for example, SOAPAction and contentType).

### Call Timeout (ms)

This parameter defines how to wait while trying to execute the operation. After the timeout is hit, an exception will be thrown and the On Error handling will occur.

## SSL Tab

**SSL Keystore File**

The name of the keystore file where the client identity certificate is stored. The file can be in JKS or PKCS format.

**SSL Keystore password**

Password for the keystore file.

**SSL Key alias**

The keystore attribute that defines the alias that is used to store and retrieve the private key for the server.

**SSL Key password**

An optional password for the key entry if using a JKS keystore, and key has a different password from keystore.

For an example of using multiple SSL certificates, see [HTTP HTML Request Step](#) (see page 176).

Specify global certificates properties for SSL in your **local.properties** file.

For global certificates (web server, raw SOAP, and web service steps):

**ssl.client.cert.path**

A full path to the keystore.

**ssl.client.cert.pass**

Password for the keystore (this password is automatically encrypted when DevTest runs).

**ssl.client.key.pass**

An optional password for the key entry if you are using the JKS keystore and the key has different password from the keystore. This password is automatically encrypted using AES (Advanced Encryption Standard) when DevTest runs.

**Note:** This option is not available for the WS Test step. To set this option, use the **local.properties** file.

For web service steps only certificates (not raw SOAP steps):

**ws.ssl.client.cert.path**

A full path to the keystore.

**ws.ssl.client.cert.pass**

Password for the keystore. This password is automatically encrypted when DevTest runs.

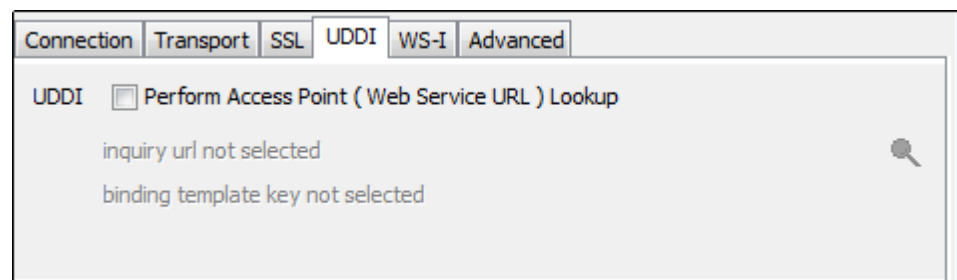
**ws.ssl.client.key.pass**

An optional password for the key entry if you are using the JKS keystore and the key has different password from the keystore. This password is automatically encrypted using AES (Advanced Encryption Standard) when DevTest runs.

**Note:** This option is not available for the WS Test step. To set this option, use the **local.properties** file.


**Note:** If you have duplicate values in **local.properties** and in the general tab, the values in the general tab are used.

## UDDI Tab



**Perform Access Point (Web Service URL) Lookup**

Select the Inquiry URL and Binding Template. To perform the lookup, use the

Search UDDI Server Find  button to navigate to the correct binding.

**Note:** When creating the step, if you used the UDDI Search function when specifying the web service WSDL URL, these values are automatically populated. If the Inquiry URL is specified but the Binding Template is not, you could have performed a Model search. To locate the Binding Template, perform a search at a higher level of the hierarchy. Then drill down to the TModel through a specific Binding Template.

## WS-I Tab

The screenshot shows a configuration window for the WS-I Basic Profile 1.1. It has a tabbed interface with tabs for Connection, Transport, SSL, UDDI, WS-I (selected), and Advanced. The WS-I tab contains the following settings:

- WS-I Basic Profile 1.1**: A dropdown menu set to "Display Only Failed Assertions".
- Validate**: Two checkboxes, "WSDL" and "SOAP Message", both of which are unchecked.
- On Failure Go To**: A dropdown menu set to "Generate Error".

**WS-I Basic Profile 1.1**

You can select four different validation levels in the pull-down menu.

- Display All Assertions
- Display All But Info Assertions
- Display Only Failed Assertions
- Display Only Not Passed Assertions

**Validate**

Select to validate the WSDL, the SOAP message, or both.

**On Failure Go To**

Select the step to redirect to on error.

**Note:** Validation failures are common, but usually do not affect the outcome of the test. A good practice is to set the next step to continue so that you can complete the test.

## Advanced Tab

Connection Transport SSL UDDI WS-I Advanced

SOAP Action

Style ☒ Document ☐ RPC

Use ☒ Literal ☐ Encoded

☒ SOAP Fault is Error ☐ Do not send request ☒ Maintain Session ☒ Clear Session

**SOAP Action**

This field is auto-populated based on the WSDL operation definition. The field is used as the value of the SOAPAction transport header for SOAP 1.1 request message. Change this field manually only in rare cases.

**Style**

This field is auto-populated based on the WSDL operation definition. The field is used to determine how a sample SOAP message is generated. Change this field manually only in rare cases.

**Use**

This field is auto-populated based on the WSDL operation definition. The field is used to determine how a sample SOAP message is generated. If Encoded is selected, you can also edit the Encoded URI in the field next to the choice. Change these fields manually only in rare cases.

**SOAP Fault is Error**

If a SOAP fault is returned, perform On Error handling.

**Do Not Send Request**

When selected, the step execution performs all of the normal SOAP message processing, but it does not send the generated SOAP message. Instead it sets the response to be the request message that would have been sent.

This method is recommended to use transports other than HTTP/S to send SOAP requests. For example, when using a JMS client, you can configure the WS step to only create the request and not send it. Then you can append a Generic JMS step to actually send the request and receive the response.

**Maintain Session**

Select to maintain cookies across invocations.

**Clear Session**

Select to clear cookies across invocations. Clearing the session cookies acts like a new session was created. Old session cookies are not used and any new cookies in the response are not set on the session. However, because the session is not cleared, future steps can still use it.

## Request Editor Tabs

### Addressing Tab

You can send a WS-Addressing header with your request. The WSDL does not specify if WS-Addressing information is required so you must configure it.

	Default	Override
To	<input checked="" type="checkbox"/>	<input type="checkbox"/>
From	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Action	<input checked="" type="checkbox"/>	<input type="checkbox"/>
MessageId	<input checked="" type="checkbox"/>	<input type="checkbox"/>
ReplyTo	<input type="checkbox"/>	<input type="checkbox"/>
FaultTo	<input type="checkbox"/>	<input type="checkbox"/>

Click the Addressing tab, and then specify:

#### Use WS-Addressing

Click to use WS-Addressing.

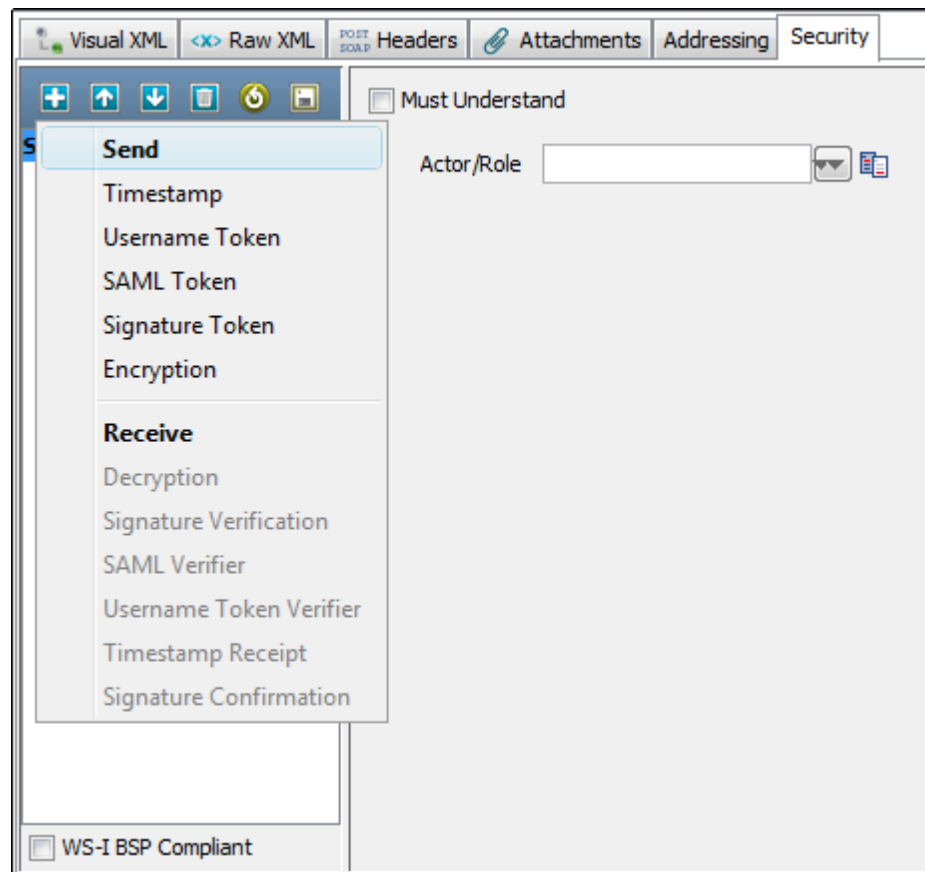
#### Version

Select appropriate version. Several versions of the WS-Addressing specification are listed as options because some web services platforms (for example, .NET) still use the older Draft specifications. Determine which your web service platform is using.

DevTest populates as many values as possible. Then you can select to use the default value or override it. You can select not to send some of the default elements by clearing the Default check box for that element.

To assure that the web service can understand the WSAddressing header, select the Must Understand check box.

### Security Tab




Click the Security tab and click Send.

#### Must Understand

Select to ensure that the server processes the WS-Security header.


#### Actor/Role

Enter the name if needed: most web services do not use multiple Actors/Roles.

Click Add  and select the security action type to add. You are presented with the configuration panel for that security action type.

Adding the security verification to the Response is similar:

Click the Security tab. Click Add , and select Receive.



- Enter the Actor/Role name (if needed)
- Click Add  and select the security action type. The configuration panel for that security action type opens.

**Note:** You can add as many security types as are necessary to execute your web service.



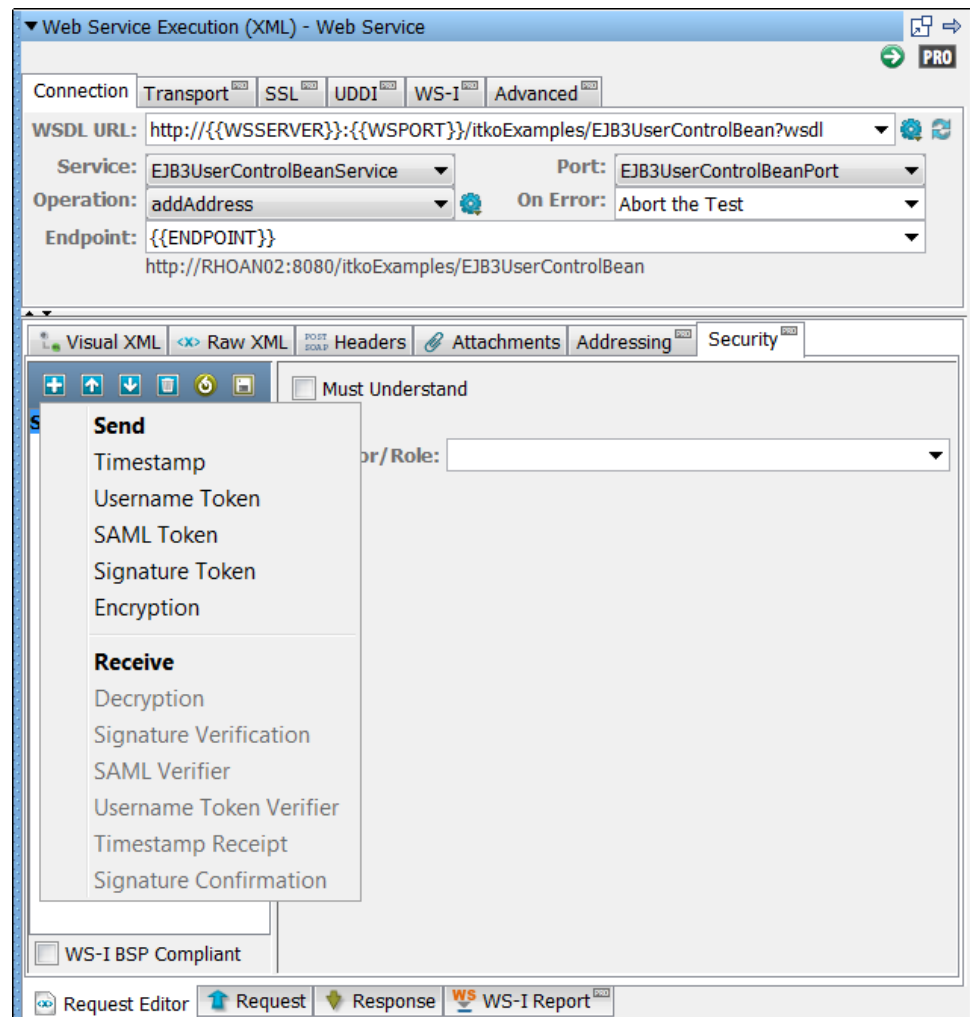
When you use a keystore in a security action type configuration, you can verify in the keystore settings that you are using the correct format, password, alias, and alias password. Clicking the Verify button on the editors for Signature, Encryption/Decryption, and SAML Assertion Token invokes the Keystore Verifier, which produces a verification report.

If you do not know the expected alias name for a WS-Security setting, use the Keystore Verifier to list all of the aliases in the keystore. Leave the Keystore Alias and Alias Password boxes empty and click Verify. The Keystore Verifier is described at the end of this section on WS-Security.

**Note:** You can load and save the security configuration information from and to a .wss file, using the Load  and Save  icons. This capability allows for quick and easy creation of new steps connecting to the same service.

## Security Example

This section describes the parameters that are necessary to run the WS-security example.



## XML Encryption-Decryption

### Encryption

Select the Use Encryption check box.

#### Keystore File

The location of the keystore file.

#### Keystore Password

Enter the password for the keystore.

#### Keystore Alias

Enter an alias for a public key.

#### Alias Password

Leave empty or make the same as Keystore Password for PKCS #12 files.

#### Key ID Type

Select the appropriate key ID type from the pull-down menu.

#### Algorithm

Select Triple DES, AES 128, AED 192, or AES 256.

#### Transport

Select PKCS#1: RSA Encryption Standard v1.5 or Optimal Asymmetric Encryption Padding with RSA Encryption.

The default behavior is to encrypt only the SOAP Body contents.

#### Encrypt Only Parts

To specify different parts to encrypt, click the Select button to identify the parts to be encrypted.

#### Type

##### Values:

- Element: Encrypt the element and the content.
- Content: Encrypt only the content.

#### Namespace URL

Enter the value for the element.

#### Element

Enter the name of the element.

Click Add to repeat the process.

To include the Body element, manually add it. To include the Binary Security Token as a part, use the Element name **Token**.

### **Decryption**

#### **Keystore File**

The location of the keystore file.

#### **Keystore Password**

Enter the password for the keystore.

#### **Keystore Alias**

Enter an alias for a public key.

#### **Alias Password**

Leave empty or make the same as Keystore Password for PKCS #12 files.

## XML Signature Token-Signature Verification

### Signature Token

Select the Add Signature check box.

#### Keystore File

The location of the keystore file.

#### Keystore Password

Enter the password for the keystore.

#### Keystore Alias

Enter an alias for a private key.

#### Alias Password

Leave empty or make the same as Keystore Password for PKCS #12 files.

#### Key ID Type

Select the appropriate key ID type from the pull-down menu.

#### Algorithm

Select DSA with SHA-1.

#### Digest Algorithm

**Values:** SHA-1, SHA-256, SHA-384, SHA-512, RIPEMD-160, or MD5 (not recommended).

The default behavior is to sign only the SOAP Body contents.

#### Sign Only Parts

To specify different parts to sign, click the Select button to identify the parts to be signed.

#### Type

##### Values:

- Element: Encrypt the element and the content.
- Content: Encrypt only the content.

#### Namespace URL

Enter the value for the element.

#### Element

Enter the name of the element.

Click Add to repeat this process.

To include the Body element, add it manually. To include the Binary Security Token as a part, use the Element name **Token**.

### **Signature Verification**

The parameters that are required to configure the signature verification are a subset of the parameters that are required for signing.

#### **Keystore File**

The location of the keystore file.

#### **Keystore Password**

Enter the password for the keystore.

#### **Keystore Alias**

Enter an alias for a public key.

#### **Alias Password**

Leave empty or make the same as Keystore Password for PKCS #12 files.

## Timestamp-Timestamp Receipt

### **Timestamp**

Select the Add Timestamp check box.

#### **Time-To-Live (sec)**

Enter the lifetime of the message in seconds. Enter 0 to exclude an Expires element.

**Note:** Some Web services, particularly .NET 1.x/2.0 with WSE 2.0, are not compliant with standard timestamp date formatting, and do not allow milliseconds. For these web services, clear the Use Millisecond Precision in Timestamp check box.

### Timestamp Receipt

The parameters that are required for Timestamp Receipt are a super set of those parameters that are required for Timestamp. The additional parameter is:

#### **Don't allow expired**

Select this parameter if you do not want to allow expired timestamps.

## Username Token-Username Token Verifier

### Username Token

Select the Add Username Token check box.

#### User Name

Enter the user name.

#### Password

Enter the appropriate password.

#### Password Type

Select the password type from the drop-down list (Text, Digest, None). None is typically used with the Add Signature option.

#### Add Nonce

Specifies whether a nonce is required to protect against replay attacks.

#### Add Created

Select if a timestamp is required.

#### Use Millisecond Precision in Timestamp

To use millisecond precision, select the check box. Some Web services, particularly .NET 1.x/2.0 with WSE 2.0, are not compliant with standard timestamp formatting and do not allow the use of milliseconds.

#### Add Signature

Select to add a signature that is built using a combination of the username and password as the key.

#### Sign Only Parts

To specify different parts to sign, click the Select button to identify the parts to be signed.

#### Type

##### Values:

- Element: Encrypt the element and the content.

Content: Encrypt only the content.

#### Namespace URL

Enter the value for the element.

#### Element

Enter the name of the element.

Click Add to repeat for as many elements as you want.

To include the Body element, manually add it. To include the Binary Security Token as a part, use the Element name **Token**.

#### **UserName Token Verifier**

Select the Verify Username Token check box.

##### **User Name**

Enter the user name.

##### **Password**

Enter the appropriate password.

##### **Use Millisecond Precision in Timestamp**

To use millisecond precision, select the check box. Some Web services, particularly .NET 1.x/2.0 with WSE 2.0, are not compliant with standard timestamp formatting and do not allow the use of milliseconds.

##### **Verify Signature**

Select the check box if signature verification is required.



## SAML Assertion Token-SAML Assertion Receipt

### **SAML Assertion Token**

Select the Add SAML Token check box.

Perform one of the following options:

- Select the From Step Results check box. Select the step whose result is an XML SAML Assertion (like a SAML Query Step or a Parse Text Step with XML manually entered)
- Select the From Property check box and enter the property that contains the XML SAML Assertion.

Click Verify to have DevTest parse the SAML Assertion XML and build the SAML Assertion object as it would when sending the SOAP request. Verifying is useful to confirm that the SAML Assertion that could have been created manually is a valid SAML Assertion. Verifying also attempts to verify any signatures that are associated with the assertion. However, it is likely that DevTest cannot verify the assertion without configuring a public certificate with which to verify.

Select the Signed Sender Vouches check box if the sender must sign the assertion (the sender instead of the bearer/creator of the assertion vouches for its authenticity). When selected, the following information is required:

#### **Keystore File**

The location of the keystore file.

#### **Keystore Password**

Enter the password for the keystore.

#### **Keystore Alias**

Enter an alias for a private key.

#### **Alias Password**

Leave empty or make the same as Keystore Password for PKCS #12 files.

#### **Key ID Type**

Select the appropriate key ID type from the pull-down menu.

#### **Algorithm**

Select DSA with SHA-1.

#### **Digest Algorithm**

**Values:** SHA-1, SHA-256, SHA-384, SHA-512, RIPEMD-160, or MD5 (not recommended).

The default behavior is to sign only the SOAP Body contents.

### **Sign Only Parts**

To specify different parts to sign, click the Select button to identify the parts to be signed.

#### **Type**

##### **Values:**

- Element: Encrypt the element and the content.
- Content: Encrypt only the content.

#### **Namespace URL**

Enter the value for the element.

#### **Element**

Enter the name of the element.

To repeat this process, click Add. To include the Body element, add it manually.

To include the Binary Security Token as a part, use the Element name **Token**.

### **SAML Assertion Receipt**

To scan the response for a SAML Assertion Receipt header in the response, select the Process SAML Assertion check box. If you select this option and there is no SAML Assertion Receipt header, an exception occurs.

### **Signature Confirmation**

To scan the response for a signature confirmation header in the response, select the Signature confirmation check box. If you select this option and there is no confirmation header, an exception occurs.

### **Using the Keystore Verifier**

You can verify your keystore settings to ensure that you are using the correct format, password, alias, and alias password. To produce verification reports, click the Verify button on the editors for SSL, Signature, Encryption/Decryption, and SAML settings.

SSL verification validates the Keystore password only. The SSL verification also confirms at least one of the keys in the keystore can be loaded using the keystore password.

WS-Security verification validates the Keystore password, the alias, and the alias password. A correct validation is indicated with a green entry. Any validation errors that are found are shown in red. Warnings are shown in orange.

**Note:** This verification only verifies the keystore parameters. There could still be issues with the web service, such as a mismatch in certificate sets or an incorrect choice of algorithm. These issues must be validated independently.

### Alias Search

If you do not know the expected alias name for a WS-Security setting, use the keystore verifier. The keystore verifier lists all of the aliases in the keystore. Leave the Keystore Alias and Alias Password boxes empty and click the Verify button.

Aliases have a blue background.

Verification fails because the Keystore Alias and Alias Password boxes were left blank.

### WS-I Report

When you click the Execute button on the Object Editor window, the validation runs. A report is generated and saved (in the reports directory in the DevTest install directory). You can view the report by pressing the WS-I Report tab at the bottom of the window.

**Note:** The format of this report is standard, and the Web-Services-Interoperability Organization (WS-I) dictates the format.

## WSDL Validation

The WSDL Validation step lets you load a WSDL and add one or more assertions to validate the WSDL. This step is a little different in that it lets you load the static WSDL file and perform assertions on it. In most steps, the assertions are made on the response.

The following list describes the most useful assertions:

- **XML Diff Assertion:** Checks that the WSDL has not been changed by comparing it to a control copy of the original WSDL.
- **XML Validator:** Checks for valid XML using Schema or DTD.
- **WS-I Basic Profile 1.1 Assertion:** Checks for compliance with the WS-I Basic Profile.

These assertions are described in [Assertion Descriptions](#) (see page 9).

**Prerequisites:** Familiarity with the three assertions that were named previously.

**Parameter Requirements:** Location of the WSDL to validate.

To configure the WSDL Validation step, enter a WSDL in the WSDL URL field and click Load.

The WSDL appears in the editor. You can view it in XML or DOM View.

You are now ready to add the assertions.

## Web-Raw SOAP Request

The Raw SOAP Request step lets you test a web service by sending a raw SOAP request (raw XML). You can use this step to test legacy SOAP calls or web services that do not have a WSDL. This step also lets you test the reaction of a web service to data of an incorrect type. For example, sending a string when it expects a number, which is not allowed in the Web Service Execution step. Another use for the Raw SOAP request is to reduce overhead during intense load tests. The regular web service step has some additional overhead because it marshals an object into XML to make the request. This step then unmarshals the SOAP XML response back into an object. The Raw SOAP step avoids this overhead and only deals with the raw SOAP XML. Because it has less work to do, it executes faster.

You can type or paste the SOAP request into the editor. The request can also be read from a file, and then parameterized using DevTest properties.

Dynamic WS-Addressing or WS-Security headers are not supported. To have these types of headers, enter them statically as part of the SOAP request in the input area. If your request contains items like a WS-Security signature token, the signed elements cannot be parameterized or the signature is no longer valid.

**Note:** This step is not limited to SOAP calls. You can also do XML or text POSTs.

### To create a Raw SOAP Request:

Complete the following fields:

#### **SOAP Server URL**

Enter the URL of the web service endpoint. The URL is converted into a single property instead of simply substituting the WSSERVER and PORT properties.

#### **SOAP Action**

Complete the SOAP action as indicated in <soap: operation> tag in the WSDL for the method being called. This action is required for SOAP 1.1 and is typically required to be left blank for SOAP 1.2.

#### **Content Type**

Select the Content Type. Use text/HTML for SOAP 1.1, application/SOAP+XML for SOAP 1.2.

#### **Advanced button**

Click to add any custom HTTP headers.

#### **Discard response**

Check to discard the response, replacing it with a small valid but static SOAP text. This feature is intended for load testing where processing a large response limits the scalability of the load generator computers.

Type or paste the SOAP Request into the editor, or click Read Request From File and browse to the file containing the SOAP Request.

Now you can parameterize the request with properties.

To execute the call, click Test.

To examine the response, click the Results tab.

You are now ready to add filters and assertions.

## Base64 Encoder

The Base64 Encoder step is used to encode a file using the Base-64 encoding algorithm. The result can be stored into a property for use elsewhere in the test case.

The Base64 Encoder step accepts a file as input and encodes the file using Base64 encoding. You can store the encoded file in a property. To encode a file, click Load. The Base64 encoded text that is displayed in the editor is read-only.

Complete the following fields:

**File**

Enter the full path and path name, or browse to the file to be encoded.

**Property Key [opt]**

The name of the property in which to store the encoded file.

**Load**


Click to load and test the encoding of the file. Optionally, store it in the specified property.

**If environment error**

Action to take if an environment error occurs.

After the file is encoded, you can add filters and assertions. The valid options are:

- Random Selection Filter
- Parse Value Filter
- XML Xpath Filter
- Create HTML Table ResultSet Filter
- Make Assert on Selection

When you have added filters and assertions, click Load to load the file or Save  to save the editor contents in a new file.

## Multipart MIME Step

The Multipart MIME (Multipurpose Internet Mail Extensions) step lets DevTest load data from a file, encode it, and store it in a property for use as a post parameter on an HTTP request. The encoded document is stored in the property that has been defined previously in an HTTP/HTML Request step.

When a multipart MIME form submit request is recorded, the contents of the file that was uploaded are recorded. Subsequent playback results in the same content being submitted with "file upload" portion of the form again. The multipart MIME step can be used to change what file is uploaded when the test case is played back.

**Prerequisite:** The HTTP/HTML Request step containing the HTTP parameter must exist, and it must be before the Multipart MIME step.

Complete the following fields:

**Step**

Select the name of the HTTP/HTML Request step, or select from the pull-down menu, the step that receives the property containing the encoded document.

**Parameter**

Select the name of the property listed in the step named in the Step field from the pull-down menu.

**File**

Enter the pathname or browse to the document to be encoded.

**MIME Type**

Enter the MIME type that the server expects.

Click Load to encode the file.

▼ Multipart MIME Step - Step2


Step: Account Activity

Parameter: userid

File: {LISA\_PROJ\_ROOT}}/Data/DataSet1.lds Browse

MIME type: application/octet-stream Load

AAAAAGgAAAAACWZpcnN0TmFtZQAibGFzdE5hbWUABnVzZX  
AAAAAScBAAAAAAAAASsAAAAAAAAABlwAAAAAAAAAEzAQAAAA  
AAAAAABQwAAAAAAAAAFHAAAAAAAAAU8BAAAAAAAAAU8AAA  
AAAAAABV8AAAAAAAAABYwEAAAAAAAAABZwAAAAAAAAAFrAA  
dwAAAAAAAAAF7AQAAAAAAAAAF/AAAAAAAAAYMAAAAAAAAAABhw  
AZP////////////////////////////////////  
////////////////////////////////////  
////////////////////////////////////  
////////////////////////////////////8=

To save the contents of the editor to a file, click Save .

In the example in the previous graphic, the Account Activity step has a post-parameter user ID that contains the encoded version of the file **DataSet1.lds**.



## SAML Assertion Query

The SAML Assertion Query step lets you obtain a SAML assertion from an identity provider for use in a Web Service Execution step that uses a WS-Security SAML 1.x Assertion Token.

**Prerequisites:** A cursory understanding of what type of SAML Assertion Query that you must perform. Obtain this information from either the developer of the system that uses SAML Assertions as the form of identity security or from the identity provider administrator.

**Parameter Requirements:** At a minimum, know:

- The URL to the SAML Query interface (endpoint) for the identify provider.
- The Subject information (who/what you want to obtain a SAML Assertion for).
- The type of query you want to perform.
- Some extra information depending on the type of query.

**SAML Assertion Query - SAML Assertion Query**

**Connection**

Endpoint:

SSL Keystore:

SSL Keystore Password:

SAML Version: ☐ 1.0 ☒ 1.1

**Subject**

Name:

Name Qualifier:

Format:

Confirmation Methods: ☐ Holder of Key ☒ Sender Vouches ☐ Bearer ☐ Artifact

**Response (deprecated)**

Respond With	Local Part	Namespace

**Query**

Type: ☒ Attribute ☐ Authorization ☐ Authorization Decision

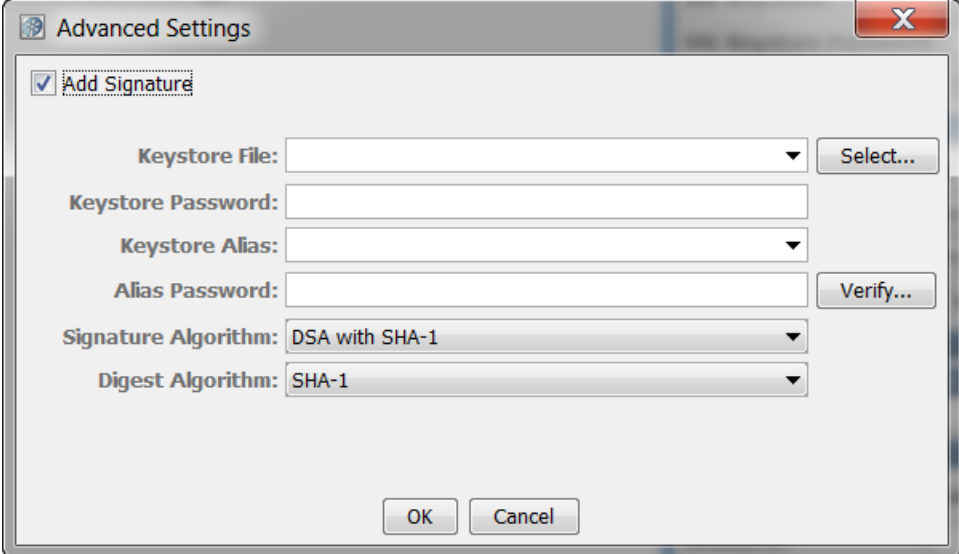
Resource:

Attribute Designators	Name	Namespace

Editor Last Request Raw Query Result Last Response

The SAML Assertion Query Editor has four tabs. The Editor tab lets you configure the query information. After you configure the query, you can test the query using the Test button in the Query section of the editor. After you test the query, you can view the raw request that was sent in the Last Request tab. You can view the raw SOAP response in the Raw Query Result tab. For example, if the query returned multiple assertions, you can see the assertions in the Raw Query Result tab. You can view the step response in the Last Response tab. The Last Response tab shows, for example, what is used in the Web Service WS-Security token.

The Advanced button lets you enter more information.

The image shows a dialog box titled "Advanced Settings" with a close button (X) in the top right corner. Inside the dialog, there is a checked checkbox labeled "Add Signature". Below this, there are several fields and buttons: "Keystore File:" with a text box and a "Select..." button; "Keystore Password:" with a text box; "Keystore Alias:" with a text box and a dropdown arrow; "Alias Password:" with a text box and a "Verify..." button; "Signature Algorithm:" with a dropdown menu showing "DSA with SHA-1"; and "Digest Algorithm:" with a dropdown menu showing "SHA-1". At the bottom of the dialog are "OK" and "Cancel" buttons.

Select the Add Signature check box.

**Keystore File**

The location of the keystore file.

**Keystore Password**

Enter the password for the keystore.

**Keystore Alias**

Enter an alias for a private key.

**Alias Password**

Leave empty or make the same as Keystore Password for PKCS #12 files.

**Signature Algorithm**

Select DSA with SHA-1.

**Digest Algorithm**

**Values:** SHA-1, SHA-256, SHA-384, SHA-512, RIPEMD-160, or MD5 (not recommended).

## Connection

This information describes where the SAML Query API server is and how to connect to it.

**Endpoint**

The URL to the SAML Query API of the identity provider.

**SSL Keystore**

To use client-side identification certificates to connect to the endpoint, select the keystore file using the Select button. Or, select a previously entered item from the pull-down list or enter one manually.

**SSL Keystore Password**

The password for the SSL Keystore if used.

**SAML Version**

The SAML version to use to query the identity provider.

## Subject

This information describes the recipient of a SAML assertion. The subject could be a user, user group, or other entity for which you want to provide an assertion about the current authorization/privileges.

**Name**

The name of the entity (for example, username).

**Name Qualifier**

A group or categorization that is used to qualify the Name (for example, domain).

**Format**

This field describes what format the name is being sent (for example, Full Name as opposed to Username).

**Confirmation Methods**

Select which confirmation method types you want to include in the query. The query only returns assertions that contain at least one of the specified types. If you leave all types cleared, the query returns any assertion regardless of the confirmation method.

## Response (deprecated)


This information describes which assertion statements you want to be returned as part of the SAML Assertion. Response is deprecated as of SAML 1.1.


### Local Part

The element name (for example, AuthenticationStatement, AuthorizationDecisionStatement, and AttributeStatement).

### Namespace

The element namespace (for example, urn:oasis:names:tc:SAML:1.0:assertion).

Click Add  to add more XML elements to the set to be returned.

Click Delete  to remove any elements you have already added.

## Query

A description of which type of query to perform. The query types are:

- Attribute
- Authorization
- Authorization Decision

### Attribute

An attribute query responds with a set of attribute statements. For example, it could tell you which groups a subject is a member of.

### Resource

To limit your query to a specific resource (for example, a specific web service, domain, file) specify the resource name.

### Attribute Designators

A name and namespace (like XML elements) identifies each attribute. You can filter the set of attribute statements that are returned by specifying each attribute type to be returned.

Example:

Name =  
urn:mace:dir:attribute-def:eduPersonScopedAffiliation,  
Namespace = urn:mace:shibboleth:1.0:attributeNamespace:uri)

### Authorization

Used to request authentication statements that are related to a specific Subject SAML Assertions.

### Authorization Method

This parameter limits your query to returning Authorization Statements that are for a specific authorization method.

Example:

urn:oasis:names:tc:SAML:1.0:am:X509-PKI,  
urn:oasis:names:tc:SAML:1.0:am:PGP,  
urn:oasis:names:tc:SAML:1.0:am:password

A set of predefined authorization methods is available from the pull-down list.

### Authorization Decision

Used to request SAML Assertions for specific actions that a subject wants to perform, given the evidence.

### Resource

To limit your query to a specific resource (for example, a specific web service, domain, or file), specify the resource name.

### Actions

Specify at least one action for which to request authorization to perform (for example: login, view, edit) specified with a name (Data) and Namespace (like an XML element).

### Evidence (Assertions)

(Optionally) Specify one or more SAML Assertions to include with the Authorization Decision Query as advice to the Identity Provider. Specify the property that holds the SAML Assertion XML. To use the response from a previous step (for example, another SAML Assertion Query or Parse Text step) use `lisa.<stepname>.rsp`.

### Evidence (Reference IDs)

Optionally specify assertion reference IDs.

## Java-J2EE Steps

**The following steps are available:**

[Dynamic Java Execution](#) (see page 239)

[RMI Server Execution](#) (see page 243)

[Enterprise JavaBean Execution](#) (see page 247)

## Dynamic Java Execution

The Dynamic Java Execution step lets you instantiate and manipulate a Java object. All Java classes on the DevTest classpath are available, including the classes in the JRE classpath. Any user classes can be placed on the classpath by copying them into the hotDeploy directory. The class under test is loaded into the Complex Object Editor where it can be manipulated without having to write any Java code.

This example uses a Java date instance of class **java.util.Date**.

1. Enter the following parameters in the Dynamic Java Execution editor:

### Use JVM

Select Local.

**Note:** You can use In-Container Testing (ICT) to execute a Java object remotely by clicking the Remote option button. However, this mode requires some extra setup before it can be used. For more information, see the *Using the SDK*.

### Local JVM Settings

Select one of the following options:

- **Make New Object of Class:** Select this option button and enter, select, or browse to the Java class to instantiate. This value must be the fully qualified class name including the package of the Java class; for example, **com.example.MyClass**.
- **Load from Property:** Click the option button and enter the name of the property that has the serialized object as its value.

### If environment error

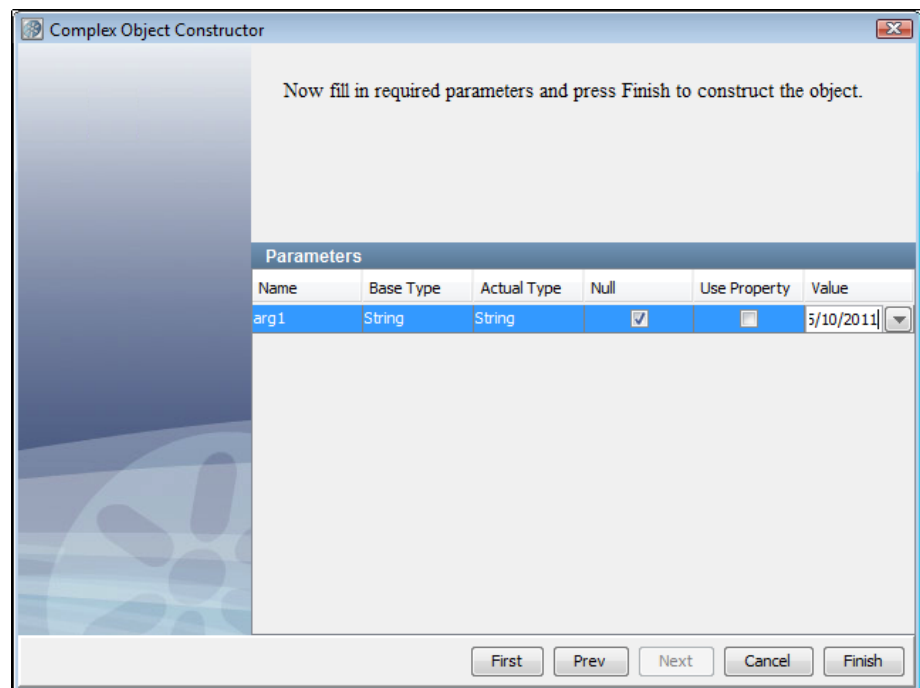
Select the step to redirect to if an environment error occurs while trying to create an object.

**Note:** If you require that the Java object be loaded by its own classloader, add the **Class Loader Sandbox Companion**.

2. Click Construct/Load Object.

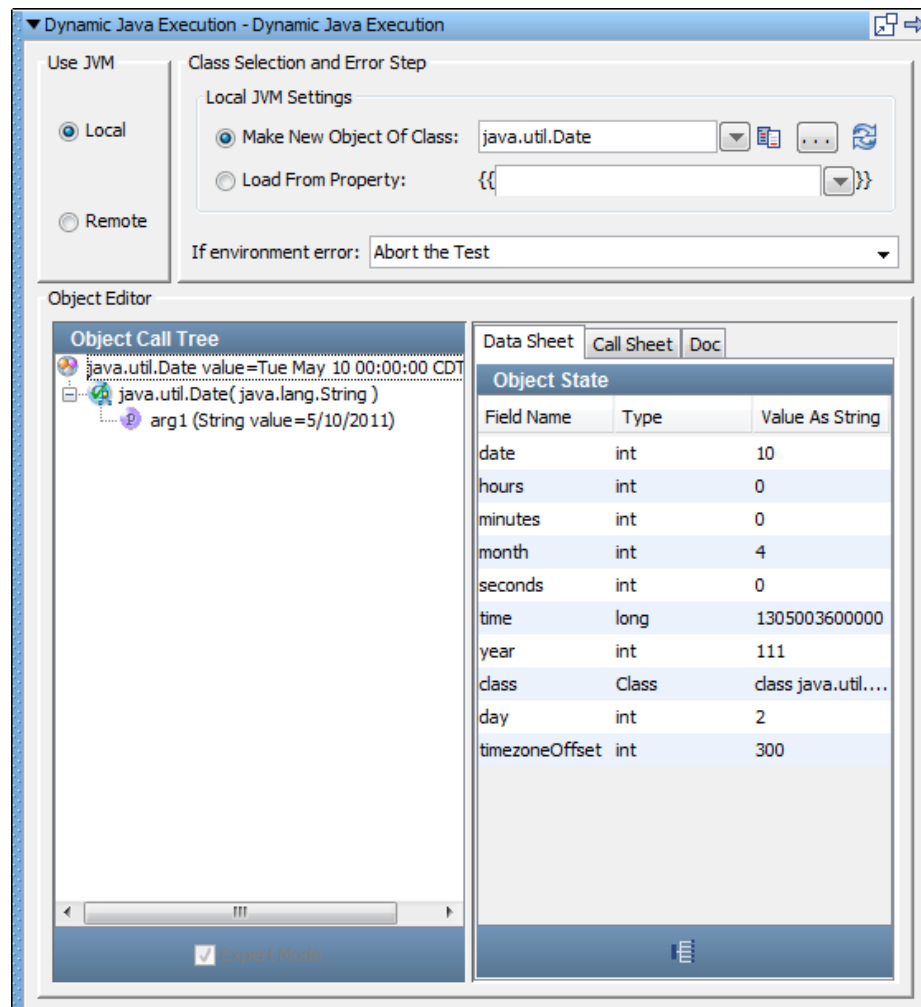
The Complex Object Constructor window opens and lists the available constructors for your object.

3. Select a constructor and click Next.
4. Enter any input parameters that the constructor needs.



5. In this example, enter a string representation of a day (5/10/2011). You can enter a value, a property, or null. DevTest constructs the object and loads it into the Complex Object Editor.



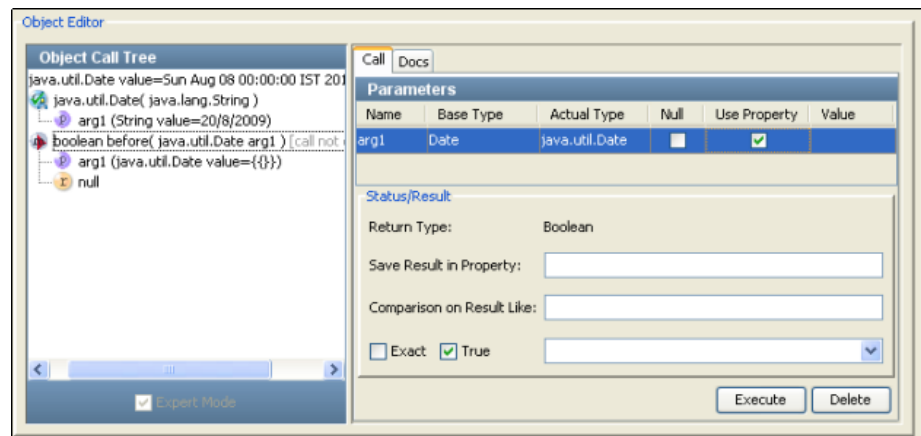


6. You can now manipulate the object, and execute methods, using the Complex Object Editor.

For more information about how to use the Complex Object Editor, see Complex Object Editor (COE) in *Using CA Application Test*.

7. You can use either of the following methods to add filters and assertions:
- Use the inline filter/assertion form (part of the Complex Object Editor)
  - Manually, by selecting filter under your test step in the test case tree

For example, the following graphic is a window before we execute the before method on the Date class.



In the Status/Result section, you can add an inline filter in the Save Results in Property text box. You can also add an inline assertion in the Comparison on Result Like text box.

The Dynamic Java Execution step has a default name using this convention: Dynamic Java Execution. You can change step names at any time.

## RMI Server Execution

The RMI Server Execution step lets you complete the following actions:

- Acquire a reference to a remote Java object through RMI (Remote Method Invocation)
- Call the Java object

**Prerequisites:** Knowledge of the Complex Object Editor is assumed. You must also copy the interface and stub classes for the remote object into the hot deploy directory. These actions are required to contact and interact with the remote object. Get these classes from the remote object developer.

**Parameter Requirement:** You must know how to connect to the RMI Server (usually a host name and port) and you must know the RMI name of the object you want to invoke.

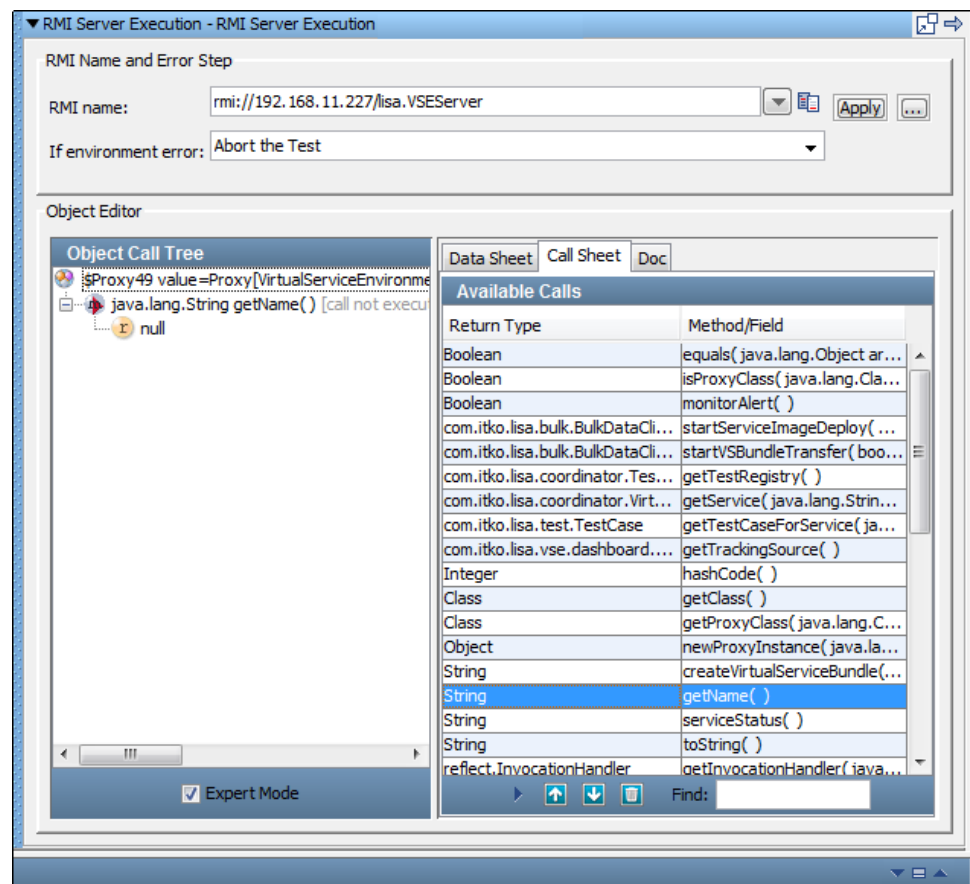
The RMI Server step editor lets you enter the following parameters:

### RMI Name

Complete one of the following actions:

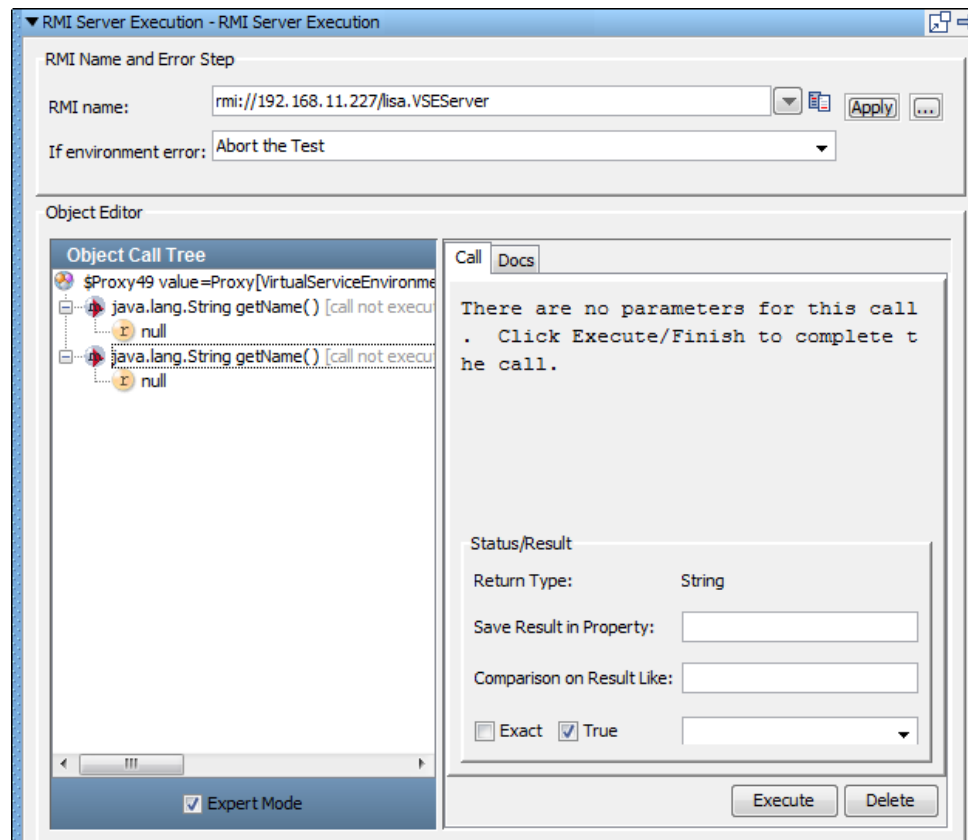
- Enter or select the full RMI name of the object (as shown previously)
- Enter the RMI Server name and click Apply to open a list of available objects.

DevTest constructs the object and loads it into the Complex Object Editor.

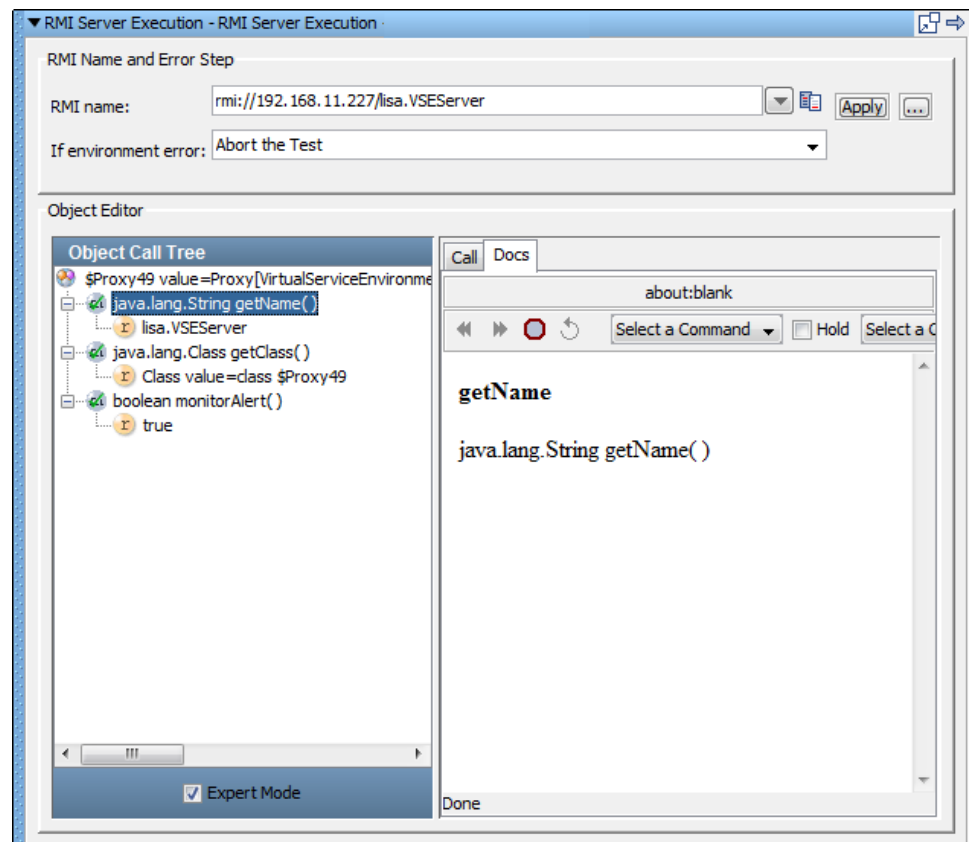


You can now manipulate the object, and execute methods, using the Complex Object Editor.

In the previous graphic, the `getName` method was selected. If you double-click it, it gets added in the Object Call Tree. You can then click Execute in the dialog, to execute it with any required method arguments and information about how to process the result.



The previous graphic shows null as the return value because the method is not yet executed. After you click Execute, it gets executed and the correct return type is shown. The following graphic shows how the Object Call Tree tracks the results of execution of several methods.



You can also use either of the following methods to add filters and assertions:

- Use the inline filter/assertion form
- Manually by selecting filter under your test step in the test case tree

You can see the Status/Result section where you can add an inline filter in the Save Results in Property text box. You can see an inline assertion in the Comparison on Result Like text box.

**Note:** If you have multiple network cards, using localhost in the RMI name can cause errors. You may need to use the IP address, or the host name that corresponds to the IP address.

The RMI Server Execution step has a default name using this convention: *RMI Server Execution - operation name*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

## Enterprise JavaBean Execution

The Enterprise JavaBean Execution step lets you acquire a reference to and make calls on an Enterprise JavaBean (EJB) running in a J2EE application server.

Testing an EJB is similar to testing a Java object. DevTest dynamically connects to the EJB using the Home EJB interface, and then from it creates an instance of an EJB object. This process is a little different for EJBs, as they do not require a home interface. The EJB under test is loaded into the Complex Object Editor, where it can be manipulated without having to write any Java code.

**Prerequisites:** Knowledge of the Complex Object Editor is assumed. The application client JAR and client EJB JAR must be in the DevTest classpath. Both of these JAR files are copied into the hotDeploy directory. The hotDeploy directory contains the jboss-all-client.jar file for the JBoss application server and the examples JAR file so you can run the EJB examples immediately.

**Parameter Requirements:**

- Server connection information (JNDI connection) and user ID and password (if necessary).
- The global JNDI lookup name of your EJB home interface.

The EJB deployer should provide this information.

## Connecting to WebSphere with DevTest Solutions using SIBC

IBM has an EJB and JMS client that you can download and use with the Sun JVM. The client is available [here](#).

### Follow these steps:

1. Download this file; then run their installer.
2. Issue the following command:  

```
java -jar sIBC_install-00902.06.jar jms_jndi_sun <output_directory>
```
3. Get the following files from your <output\_directory>:
  - lib\sIBC.jms.jar
  - lib\sIBC.jndi.jar
  - lib\sIBC.orb.jar
4. To reference the previous three JAR files, Create a LISA\_PRE\_CLASSPATH environment variable.  
  
For example: LISA\_PRE\_CLASSPATH=C:\sIBC.jms.jar;C:\sIBC.jndi.jar;C:\sIBC.orb.jar;
5. Edit **local.properties** and add the following line:  
  
`com.ibm.CORBA.ORBInit=com.ibm.ws.sib.client.ORB`
6. On the JMS step, use the following settings:
  - **JNDI factory class:** com.ibm.websphere.naming.WsnInitialContextFactory
  - **JNDI URL:** iiop://SERVER:PORT



## Example

This example uses the ITKO example server, a JBoss server. To use your local demo server, use localhost as the host name.

1. Enter the following parameters:

### **Choose App Server**

Select your application server from the list. If your application server is not on the list, select the Other/You Specify option.

The lower section of the editor changes, depending on your selection. The previous graphic shows the configuration panel for JBoss.

2. For the JBoss panel, enter the following parameters:

### **Host Name or IP Address**

Enter the hostname or IP address of your application server.

### **Port Number**

Enter the port number.

### **User**

Enter if a user ID is required for the application server.

### **Password**

Enter if a password is required for the application server.

3. Click Next.

### **For the Other/You Specify Window**

1. Enter the following parameters:

### **JNDI Factory**

Enter or select the fully qualified JNDI factory class name for your application server.

### **JNDI Server URL**

Enter or select the JNDI server name.

### **User**

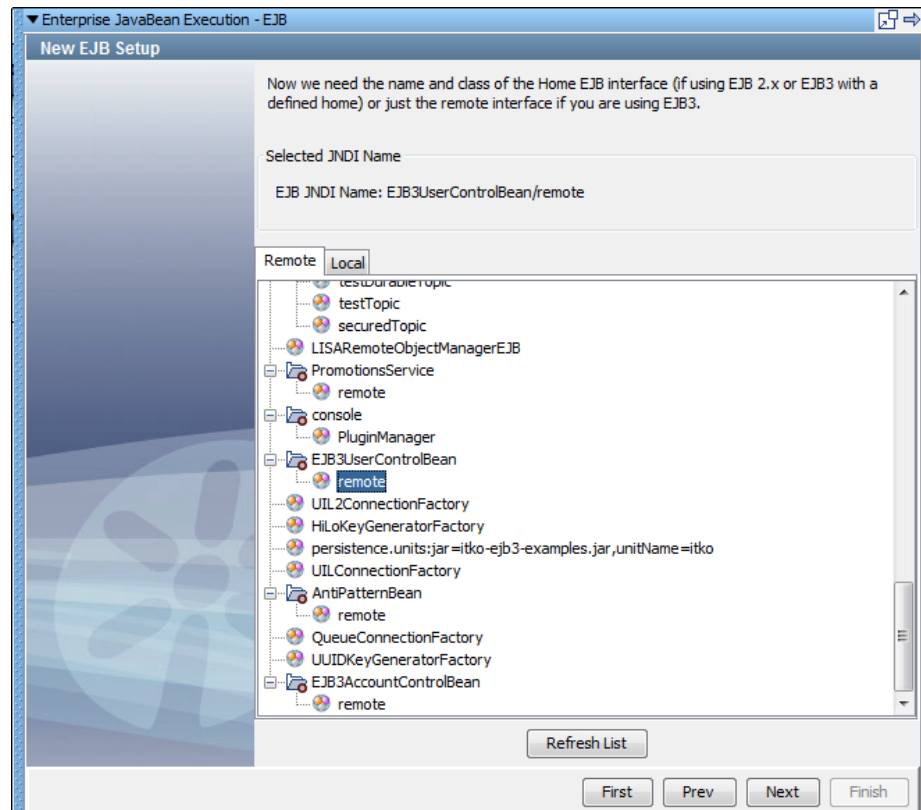
Enter if a user ID is required for the application server.

### **Password**

Enter if a password is required for the application server.

2. Click Next.

The New EJB Setup window opens and lists all the JNDI names that are registered with the application server.

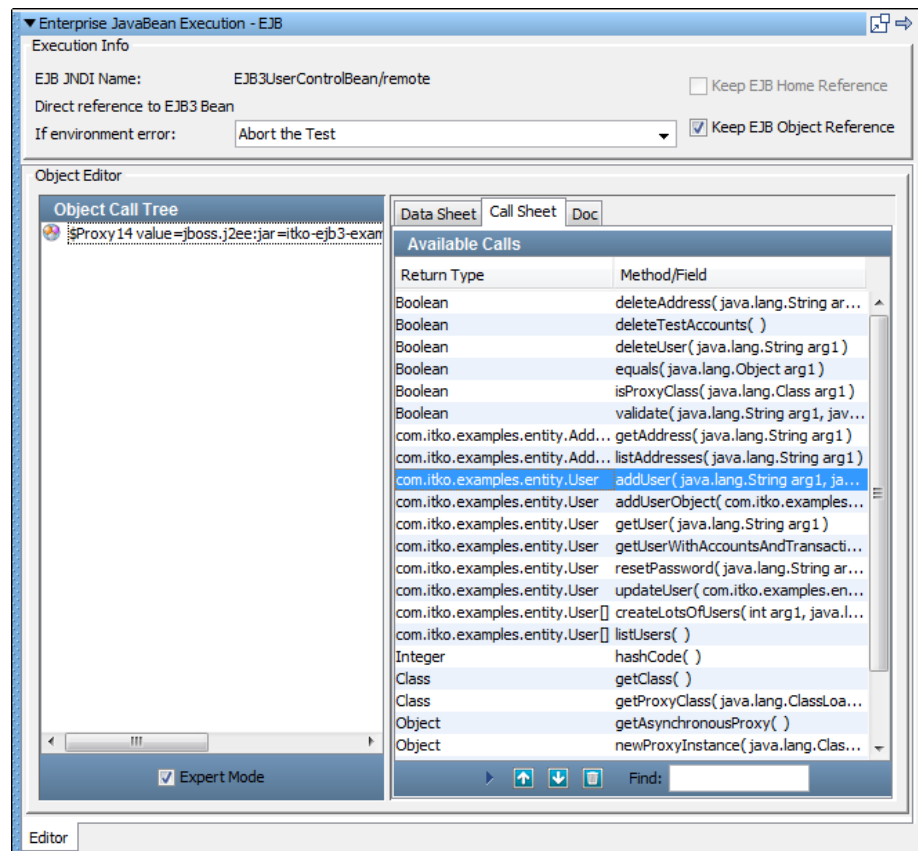


3. Select the name of the appropriate EJB home interface.

In this example, the JNDI name is **com.itko.examples.ejb.UserControlBean**. The EJB3 specification enables stateful and stateless beans to bind to the JNDI tree directly and not require a home interface. If so, the bean can be selected directly and DevTest does not need to create an instance.

4. Click Next.

The object is constructed and loaded into the Complex Object Editor.



In the Execution Info area, the current EJB information is displayed. If you plan to reuse this EJB, you can keep the references to the EJB object and the EJB Home by selecting the following check boxes:

- Keep EJB Home Reference
- Keep EJB Object Reference

If the bean is an EJB 3 bean without a home interface, the Keep EJB Home Reference check box is disabled. Set the If Exception to the step to redirect to if an exception occurs.

The EJB step has a default name using this convention: *EJB javaMethod dynamic java execution*. Before the step is saved, is entered, as shown previously, the default step name is *EJB*. If another step also uses the default step name, a number is appended to it. You can change step names at any time.

5. You can now manipulate the object, and execute methods, using the Complex Object Editor.

The usage is the same as in the [RMI Server Execution](#) (see page 243) step.

## Other Transaction Steps

**The following steps are available:**

[SQL Database Execution \(JDBC\)](#) (see page 253)

[SQL Database Execution \(JDBC with Asset\)](#) (see page 257)

[CORBA Execution](#) (see page 260)

## SQL Database Execution (JDBC)

The SQL Database Execution step lets you connect to a database using JDBC (Java Database Connectivity) and make SQL queries on the database.

Full SQL syntax is supported, but your SQL is not validated. The SQL is passed through to the database where it is validated. If you get an SQL error, it is captured in the response. You can assert on the error. Verify that the SQL is valid for the database manager you are using.

**Prerequisites:** The JDBC driver appropriate for your database must be on the DevTest classpath. You can place the driver JAR file in the hot deploy directory. The DevTest classpath includes the Derby client driver, so you do not need to re-add it.

**Parameter Requirements:** Have the name of the JDBC driver class, the JDBC URL for your database, and a user ID and password for the database. You also must know the schemas for the tables in the database to construct your SQL queries.

SQL Database Execution (JDBC) - JDBC Select Users

**Connection Info**

JDBC Driver: a.derby.jdbc.ClientDriver

Connect String: 1529/lisa-demo-server.db

Max Rows to Fetch: -1

**Execution Info**

User ID: sa

Password: ●●

☐ Keep Connection Open

☐ Use Connection Pool

☒ Returns Result Set

If SQL error: Abort the Test

**SQL Statement**

Select Login, Fname FROM Users

Parameter (?)	Type	Mode	Value
---------------	------	------	-------

Find:

Test Connection Test/Execute SQL

Base Result Set

1. Enter the following parameters in the SQL Database step editor:

Connection Info:

**JDBC Driver**

Enter or select the full package name of the appropriate driver class. Standard driver classes are available in the drop-down list. You can also use the Browse button to browse the DevTest class path for the driver class.

**Connect String**

The connect string is the standard JDBC URL for your database. Enter or select the URL. The JDBC URL templates for common database managers are available in the drop-down list.

**Max Rows to Fetch**

Enter the maximum number of rows you want returned in the result set. This field is required. Enter -1 for unlimited rows.

Execution Info:

**User ID**

Enter a user ID (if the database requires it).

**Password**

Enter a password (if the database requires it).

**Keep Connection Open**

If this option is selected, the database connection that is opened the first time that the step executes is cached. That database connection is then closed when garbage collection happens for the step. If Keep Connection Open is not selected, the connection is closed each time that the step executes.

**Returns Result Set**

Select this check box if your query results in a Result Set being returned; that is, a SELECT type query. Leave cleared for an UPDATE, INSERT, or DELETE. If this check box is set incorrectly, your query causes an error.

**If SQL error**

Select the step to redirect to if an error occurs.

**To communicate with the local demo server, use:**

**JDBC Driver**

org.apache.derby.jdbc.ClientDriver

**Connect String**

jdbc:derby://localhost:1527/reports/lisa-reports.db

**User ID**

sa

**Password**


sa

2. After you have entered the database connection information, including the user ID and password (if necessary), click Test Connection to test your connection.

If the information is correct, a success message in a window appears. Otherwise an error message appears.

3. You are now ready to enter your SQL statement in the lower window.

Properties can be used in your SQL. DevTest substitutes the parameter before passing the SQL string to the database.

The JDBC step supports stored procedure calls. Basic data types (strings, numbers, dates, Boolean) are supported as arguments that a stored procedure uses as input and returns. Click Add  to add a parameter. You cannot edit the numbers in the Parameter column. As you add, delete, and move rows, the numbers in the Parameter column are automatically renumbered.

The JDBC step can also use JDBC prepared statements. You can use question marks in a SQL statement and can add named {{properties}}. This ability allows you to avoid being concerned about the type of the argument or escaping single quotation marks in parameter values. A statement "insert into MYTABLE(COL1,COL2) values (?, ?)" with a reference to {{col1}} and {{col2}} is easier to understand. The type and escape characters are automatically converted.

To include a null in your statement, use this syntax: {{<NULL>}}.

4. After you have created the SQL query, click Test/Execute SQL to execute the query.

A message indicates the result status.

5. Click OK.

Your results display in the Result Set tab.

6. You are now ready to create filters and assertions on the result set.

The icons on the bottom of the Result Set tab provide easy access to the following filters and assertions:

**Get value for another value in a Result Set Row**

You select a search field cell, a value field filter, and enter a property name. If the cell value in the search field is found, the value in the Value field in that row is set as the value of the property that is entered.

**Parse Result for Value filter**

The value in the selected cell is set as the value of the property that is entered.

**Result Set Contents Assertion**

The value in the chosen field (column) is compared to the regular expression that is entered.

For more information about these and other filters and assertions appropriate for result sets, see [Filter Descriptions](#) (see page 122) and [Assertion Descriptions](#) (see page 9).

The SQL Database Execution step has a default name using this convention: *JDBC SQLfunction tablename*. The supported functions in step name defaults are *select*, *insert*, *delete*, *update*, and *perform*. You can change step names at any time.



## SQL Database Execution (JDBC with Asset)

The SQL Database Execution step lets you connect to a database using JDBC (Java Database Connectivity) and make SQL queries on the database. Use this step when you have a [JDBC connection asset](#) (see page 58) defined.

**Note:** If you have a legacy SQL Database Execution (JDBC) step, you can export the connection information from the existing step instead of creating it manually. For more information see Create Assets from Test Steps.

Full SQL syntax is supported, but your SQL is not validated. The SQL is passed through to the database where it is validated. If you get an SQL error, it is captured in the response. You can assert on the error. Verify that the SQL is valid for the database manager you are using.

1. Enter the following parameters in the SQL Database step editor:

### Connection Info:

#### Connection Asset

Select a [connection asset](#) (see page 58) that contains the connection parameters for the JDBC connection.

#### Max Rows to Fetch

Enter the maximum number of rows you want returned in the result set. This field is required. Enter -1 for unlimited rows.

**Execution Info:**

**Returns Result Set**

Select this check box if your query results in a Result Set being returned; that is, a SELECT type query. Leave cleared for an UPDATE, INSERT, or DELETE. If this check box is set incorrectly, your query causes an error.

**If SQL error**


Select the step to redirect to if an error occurs.

2. After you have entered the database connection information, including the user ID and password (if necessary), click Test Connection to test your connection.

If the information is correct, a success message in a window appears. Otherwise an error message appears.

3. You are now ready to enter your SQL statement in the lower window.

Properties can be used in your SQL. DevTest substitutes the parameter before passing the SQL string to the database.

The JDBC step supports stored procedure calls. Basic data types (strings, numbers, dates, Boolean) are supported as arguments that a stored procedure uses as input and returns. Click Add  to add a parameter. You cannot edit the numbers in the Parameter column. As you add, delete, and move rows, the numbers in the Parameter column are automatically renumbered.

The JDBC step can also use JDBC prepared statements. You can use question marks in a SQL statement and can add named {{properties}}. This ability allows you to avoid being concerned about the type of the argument or escaping single quotation marks in parameter values. A statement "insert into MYTABLE(COL1,COL2) values (?, ?)" with a reference to {{col1}} and {{col2}} is easier to understand. The type and escape characters are automatically converted.

To include a null in your statement, use this syntax: {{<NULL>}}.

4. After you have created the SQL query, click Test/Execute SQL to execute the query.

A message indicates the result status.

5. Click OK.

Your results display in the Result Set tab.

6. You are now ready to create filters and assertions on the result set.

The three icons on the bottom of the Result Set tab provide easy access to the following filters and assertions:

**Get value for another value in a Result Set Row**

You select a search field cell, a value field filter, and enter a property name. If the cell value in the search field is found, the value in the Value field in that row is set as the value of the property that is entered.

**Parse Result for Value filter**

The value in the selected cell is set as the value of the property that is entered.

**Result Set Contents Assertion**

The value in the selected field (column) is compared to the regular expression that is entered.

For more information about these and other filters and assertions appropriate for result sets, see [Filter Descriptions](#) (see page 122) and [Assertion Descriptions](#) (see page 9).

The SQL Database Execution step has a default name using this convention: *JDBC with Connection Asset SQLfunction tablename*. The supported functions in step name defaults are *select*, *insert*, *delete*, *update*, and *perform*. You can change step names at any time.

## CORBA Execution

The CORBA Execution step is used to make CORBA calls using the Java RMI-IIOP library. You are required to provide appropriate skeleton classes.

**Follow these steps:**

1. Copy the **corbaserver.jar** file to the DevTest lib directory before starting execution. This JAR is available in the CORBA server lib directory.
2. To open the step editor, select the CORBA step.
3. Complete the fields.

**Object IOR**

This field contains the raw IOR string for the object or the name service. This string can be taken from the output (generated IOR) of running **nameserver.sh** batch file.

**Note:** Copying and pasting from nameserver printed output to the Object IOR field can introduce spaces. The string must not contain spaces.

**Class Name**

IOR references this object class.

You can also use the IOR construction dialog. When open, it parses any IOR string that is entered and fills in the individual parts. Ignore the strange looking key. IORs store the key in a byte format so not every byte can be displayed properly. To use the parsed version of a raw IOR, do not edit the field.

4. Click Construct/Load Object.  
The dynamic object editor shows the call sheet for the object.
5. Select the method to call and execute it.

The default CORBA Execution step uses the following convention: *CORBA classname*. You can change step names at any time.

## Utilities Steps

The following steps are available:

[Save Property as Last Response](#) (see page 261)

[Output Log Message](#) (see page 261)

[Write to Delimited File](#) (see page 262)

[Read Properties from a File](#) (see page 264)

[Do-Nothing Step](#) (see page 265)

[Parse Text as Response](#) (see page 265)

[Audit Step](#) (see page 266)

[Base64 Encoder Step](#) (see page 267)

[Checksum Step](#) (see page 268)

[Convert XML to Element Object](#) (see page 269)

[Compare Strings for Response Lookup](#) (see page 271)

[Compare Strings for Next Step Lookup](#) (see page 273)

[Send Email](#) (see page 275)

## Save Property as Last Response

The Save Property as Last Response step lets you save the value of a property as the last response.

Enter the property name of an existing property or select it from the pull-down menu. The value of the property is loaded as the last response (and the step response). The value can then be accessed immediately as the last response, or later in the test case using the property **lisa.thisStepName.rsp**, where **thisStepName** is the name of the current step.

Each step in a test case has a response that is associated with it. When that step runs, the response is automatically saved in two properties: **LASTRESPONSE** and **lisa.thisStepName.rsp**. Use this step so you can have filters and assertions apply to a property value instead of the real response of the step.

## Output Log Message

The Output Log Message step lets you write a text message to the log. This step is useful for tracking and logging test cases as they execute. Your log message is usually a combination of text and properties.

Enter your log message in the editor. This log message appears when this step executes in the Interactive Test Run (ITR), and the message is logged during a test run.

## Write to Delimited File

The Write to Delimited File step lets you save the current value of some properties in a CSV file. This step is commonly used when properties are shared among multiple test cases, or for debugging purposes when a test has failed. The properties can be existing properties or new properties. You can save existing properties under a different property name.

Complete the following fields:

### **File Name**

Enter the path name of the file, or use the Browse button to browse to the file.

### **Encoding**

Accept the default encoding of UTF-8 or select an alternate encoding from the drop-down list.

### **Include BOM**

If the encoding you select includes a Byte Order Mark, you can select this check box to include that BOM at the beginning of the file.


### **Delimiter**

Enter the delimiter character to use for your file.

### **Line end**

Select from the drop-down list the correct line end character for your file.

### **Properties to write**

Click Add  to add a row. Then enter the properties (Header) and corresponding values (Value) to save. Your list of properties can contain existing or new properties. When specifying existing properties (Header), you can override their current value by specifying a new value, or you can use their existing value.

In the previous example:

- The first two properties are being stored without change
- The third property was stored under a new name (Header), Date
- The fourth property is using Reg Expression as its value

The properties are stored in a CSV file where the first line in the file is the set of property names being saved. The second line contains the values corresponding to each of the properties.

**Note:** This step is used to pass data between test cases. For example, assume Test1 adds customers to the bank and then returns a list of new account numbers. Those accounts could be written out to a file. A second test could get the accounts and could make deposits. The second test would use a data set to read in the file that was created in the first test.

Be aware of the following warning when writing to a delimited file. The Data directory can be used as the location of the CSV file only if both tests support rerun. The first test must be run again to create the CSV in the **lads** folder so the second test can run. The **lads** directory stores files temporarily while a test case or suite is running.

To let the second test run without the first, you must put the data set in a common location outside the project or MAR.

The Write to Delimited File step writes out one (and only one) data row for each execution. The only exception is on the first execution when (presumably) the target file does not exist. If the file does not exist, if indicated, a byte order mark for the selected encoding is written, followed by a row containing the keys.

The default name for the Write to Delimited File step is *Write Properties to File <file>*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

## Read Properties from a File

The Read Properties from a File step is used to read the properties from an external file. You can read the properties in two ways:

- In name/value pairs
- In XML tags

Complete the following fields:

### File Name

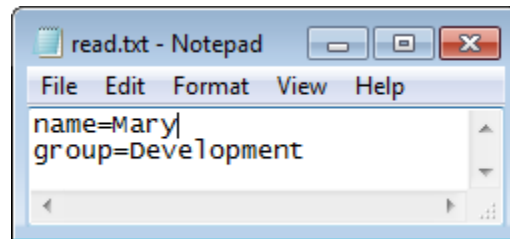
Enter the path name of the file, or use the Browse button to browse to the file.

### File Encoding

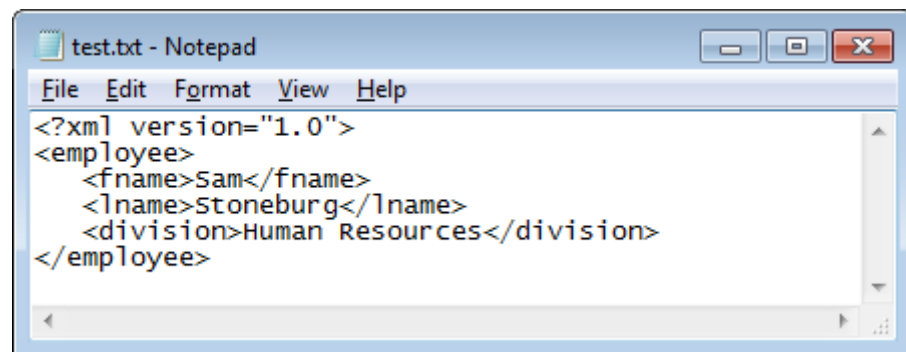
Accept the default encoding of UTF-8 or select an alternate encoding from the drop-down list. You can also select Auto-detect and click Detect to have DevTest select an encoding type for you.

### Type of File

Select the type of a file; either Name-Value-Pairs or XML Tags.



The previous graphic shows the Name/Value pair type of properties file, where *name* is a property and *Mary* is a value of the name property.



The previous graphic shows the XML Tags type of properties file, where *fname* is a property and *Sam* is a value of the fname property.



The Read Properties from a File step has a default name using this convention: *Properties Reader from <filename>* where *filename* is the leaf name of the entered or selected file. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

## Do-Nothing Step

The Do-Nothing step does not take any parameters, nor does it have any functionality by itself. However, this step is useful in specific situations, as you can add assertions to the step.

For example, you can use the scripted assertion to add a quick custom assertion, to compare numbers or dates, or some numerical comparison test.

## Parse Text as Response

The Parse Text as Response step lets you enter textual content from a file that can be saved as the last response. The content can be stored in a property. You can type or paste the text into the editor, or you can load it from a file.

Complete the following fields:

**Property Key**

The name of the property to store the content (optional).

**Load From File**

Click to browse to the file. Otherwise, type or paste the text into the editor.

The content is now available for you to parameterize, filter, and add assertions.

Click Test to show the resulting text.

## Audit Step

The Audit step lets you apply an audit document to a current test step, remote test, or virtual service.

This step allows a model to be verified in terms of the events it produces during execution.

To open its editor, click the step.

### Mode

This step has two modes:

- Start Monitoring
- Apply Audit Document.

### Start Monitoring **Mode**

If you select the Start Monitoring Mode:

#### Audit Document

Enter or browse for the audit document.

#### Target

Select the target for the audit document:

- This Model: Applies to the current model.
- Test Run: Applies to the test run.
- Virtual Model: Applies to the virtual model.

#### If Environment Error

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

### Apply Audit Document **Mode**

If you select the Apply Audit Document mode:

#### If Audit Fails

Select the step to run if the audit fails.

#### If Environment Error

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

## Base64 Encoder Step

This step is used to encode a file using Base 64 encoding algorithm.

The result can be stored in a property file to be used anywhere in the test run.

To open its editor, click the step.

Complete the following fields:

### **File**

Enter the name of file to be encoded or browse to the target location.

### **Property Key**

The name of the property to store the encoded file. This field is optional.

### **If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

Click Load to load the file in the editor.

Here you can apply filters or assertions, or both, if necessary, from the Command menu at the bottom.

## Checksum Step

The Checksum step calculates the checksum of a file and can save that value in a property.

Complete the following fields:

**File**

Enter the pathname or browse to the file on which you want the checksum to be calculated.

**Property key**

Enter the name of the property that stores the checksum value.

Click Load.

The checksum is displayed as the response, and if a property name was entered the value is in that property.

The checksum value is now available for you to filter and add assertions.

## Convert XML to Element Object

The Convert XML to Element Object step converts a raw XML into an object of one of the following types:

- Message Element Array
- Message Element
- DOM Element

This step is useful when you have a web service API that takes any type using strict processing. This type of WSDL element requires a Message Element Array as an input parameter. You can capture the raw XML from a previous step (such as Read from File or Parse Text as Response) and store it in a property. That property becomes an input parameter for this step.

**Prerequisites:** The XML must be already stored in a property.

Complete the following fields:

**Load XML from Property**

Enter the property that contains the XML. This property can be a user-defined property or a built-in DevTest property.

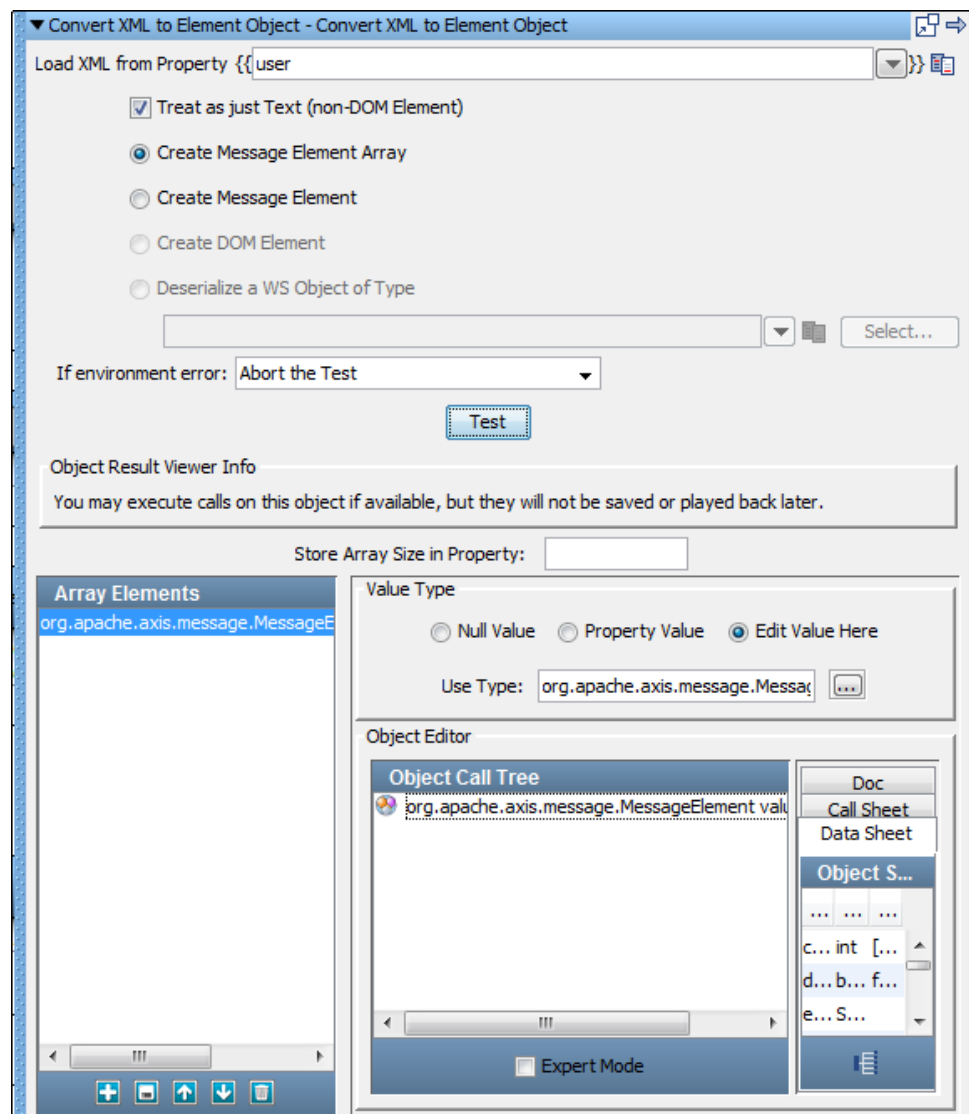
**Treat as just Text**

Select this check box if you must use plain text as your input instead of XML. This selection results in a message element that contains plain text.

Select the type of object you want from the available types by clicking the respective option button.

Click Test to do the conversion.

If you select the Treat as Just Text check box, you see the results as seen in the following graphic. The Create Message Element, Create DOM Element, and Deserialize a WS Object of Type options are dimmed.



Use the response from this step when this object is required as a parameter in another step. To save the response in a filter, use the Save Step Response as a Property filter. Or, you can refer to it as **`lisa.<stepname>.rsp`**.

## Compare Strings for Response Lookup

The Compare Strings for Response Lookup step is used to inspect an incoming request to a virtual service and determine the appropriate response. You can match the incoming requests using partial text match, regular expression, and others.

This step is automatically filled out and added to a virtual service when using the Virtual Web Service HTTP Recorder.

To open a step editor, click the step.

Complete the following fields:

**Text to match**

Enter the text against which criteria is matched. This value is typically a property reference, such as LASTRESPONSE.

**Range to match**

Enter the Start and End of the range.

**If no match found**

Select the step to be executed, if no match is found.

**If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

**Store responses in a compressed form in the test case file**

Selected by default. This option compresses the responses in the test case file.

**Case Response Entries**

In this table you can add, move, and delete entries by clicking Add, Move, and Delete. The columns in the table are as follows:

**Enabled**

Selected by default when you add an entry. Clear to ignore an entry.

**Name**

Enter a unique name for the case response entry.

**Delay Spec**

Enter the delay specification range. The default is 1000-10000, which indicates to use a randomly selected delay time from 1000 through 10000 milliseconds. (The syntax is the same format as Think Time specifications.)

**Criteria**

This area provides the response of this step if the entry matches the Text to match field. To edit the criteria, in the Criteria Column area select the appropriate row, and enter a different setting in the criteria list. Click Enter to update it in the subsequent row.

**Compare Type**

Select an option for the Compare type from the list:

- Find in string (default)
- Regular expression
- Starts with
- Ends with
- Exactly equals

**Response**

Allows for the update of the step response for an entry.

**Criteria**

Updates the criteria string for an entry.

**Response**

This area provides the response of this step if the entry matches the Text to match field. To edit the response, in the Case Response Entries area select the appropriate row, and enter a different setting in the Response list. To get it updated in the previous row, click Enter.



## Compare Strings for Next Step Lookup

This step is used to inspect an incoming request and determine the appropriate next step.

You can match incoming requests using partial text match, regular expression, among others. Each matching criterion specifies the name of the step to which to transfer if the match succeeds.

This step is automatically filled out and matches to a Virtual Service when using the JDBC Database Traffic Recorder.

To open a step editor, click the step.

Enter the following parameters:

### **Text to match**

Enter the text against which criteria is matched. This value is typically a property reference, such as LASTRESPONSE.

### **Range to match**

Enter the Start and End of the range.

### **If no match found**

Select the step to be executed, if no match is found.

### **If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

### **Next Step Entries**

Click Add to add an entry. Use Move and Delete to move or delete an entry. Enter text in the Find field to find an entry.

The columns in the Next Step Entry are:

#### **Enabled**

Selected by default when you add an entry. Clear to ignore an entry.

#### **Name**

Enter a unique name for the next step entry.

#### **Delay Spec**

Enter the delay specification range. The default is 1000-10000, which indicates to use a randomly selected delay time from 1000 through 10000 milliseconds. The syntax is the same format as Think Time specifications.

#### **Criteria**

This area defines the criteria to compare to the Text to match field.

**Compare Type**

Select from five options:

- Find in string (default)
- Regular expression
- Starts with
- Ends with
- Exactly equals

**Next Step**

The step name.

**Criteria**

Updates the criteria string for an entry.

## Send Email

To send an email notification from a test case, use the Send Email step.

Complete the following fields:

**Connection**

Specifies the [asset](#) (see page 67) that contains the connection parameters to the SMTP mail server. Select a connection from the drop-down list. To add an asset, click Add New Asset.

**From**

Identifies the email sender. Enter an email address or a property.

**To**

Identifies the email recipient. To send to multiple recipients, use a comma between email addresses. You can also use a property or list of properties.

**Example:**

`{{Ops_email}},{{QA_email}},InfoTechnology@company.com`

**Cc**

Identifies the secondary email recipient. To send to multiple recipients, use a comma between email addresses. You can also use a property or list of properties.

**Bcc**

Identifies the tertiary email recipient. To send to multiple recipients, use a comma between email addresses. The tertiary recipients are not identified to other recipients. You can also use a property or list of properties.


**Subject**

Specifies the subject of the email. You can use a property or multiple properties in the Subject field.

**Message**

Specifies the body of the e-mail with HTML support. If you use a buffered image as a property in the email body, the image is embedded in the message. You can use a property or multiple properties in the Message field. For the Selenium Integration step, you can use the **selenium.last.screenshot** property to embed a screenshot from the Selenium Integration step in the email message.

**Attachments**

Lists the attachments to the email. Click , then Browse to locate the path and file name of the file to be attached to the email.

**If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

To test the email connection, click Send Test Email.

The Send Email step has a default name using this convention: *Send Email Step*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

## External-Subprocess Steps

**The following steps are available:**

[Execute External Command](#) (see page 277)

[File System Snapshot](#) (see page 280)

[Execute Subprocess](#) (see page 281)

[JUnit Test Case-Suite](#) (see page 282)

[Read a File \(Disk URL or Classpath\)](#) (see page 283)

[External - FTP Step](#) (see page 284)

## Execute External Command

Use the Execute External Command step to run an external program (such as an operating system script, an operating system command, or an executable) and capture its contents for filtering or making assertions.

The external program syntax depends on your operating system.

Enter the following parameters in the Execute External Command editor:

### **Execute from directory**

The directory that is considered current when the external command is run. DevTest creates the directory (subject to file system permissions) if it does not exist on the system that is running the test. If the directory does not exist and cannot be created, the step fails.

### **Time Out (Seconds)**

How long to wait before transferring to the step defined by On Time Out Execute.

### **If timeout**

The step to run if the external command does not complete execution before the timeout value.

### **If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

### **Output Encoding**

Accept the default encoding of UTF-8 or select an alternate encoding from the drop-down list. You can also select Auto-detect to have an encoding type be selected for you.

### **Allow Properties**

This check box determines whether properties are allowed for the next four parameters. This option changes the look of the command editor interface.

With the Allow Properties check box cleared, five check boxes are available. Here the only choice is to select the parameter or not.

### **Wait for Completion**

If you select this check box, the step waits until the execution finishes to filter or assert on the results. If this check box is cleared, filters and assertions execute; however execution does not wait for the result of the command being run.

### **Kill at Test End**

If the Wait for Completion check box is cleared, select this check box to kill the process after the test case finishes. This parameter lets a process run while a test case runs, then shuts down the process. A property contains the process ID of the started command.

### Spawn Process

Creates a process in the operating system in which to run the command. This parameter is helpful in the following cases:

- You want a long-running background process
- You must ensure that a new set of environment variables is set and your environment contains nothing that DevTest set

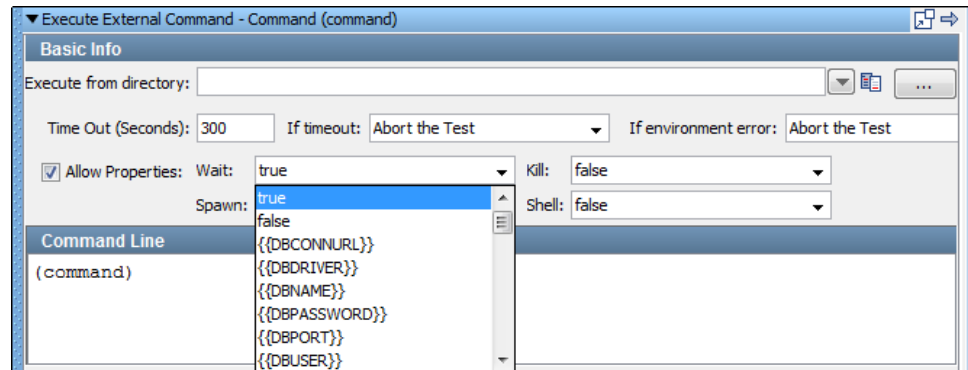
### Exec Shell

Lets you run the contents of the command line in a system shell. This option is required if you must use shell process features such as redirecting (pipe) output streams to files or other commands. Depending upon your system, this option may be required for you to run system commands such as dir or ls. This check box must be selected for a Windows operating system.

### Append to Environment

When the step defines environment variables, DevTest appends them to the existing environment, instead of creating an environment that defines only these variables.

With the Allow properties check box selected, there are pull-down menus that have the same functionality as shown previously, but each parameter can now be a property.



### Command Line

The external command is typically only one command that is written as a shell script or batch file. You can run multiple commands when the Exec Shell option is also selected. The command string must be valid for the operating system that you are running.

### Environment Variables

Allows existing environment variables to be overridden with new environment variables. If nothing is entered, the existing environment variables are used for the command. If you define an environment variable, the new variable sets are used instead of the environment variables that were used to start DevTest.

**Exit Codes**

Lets you base the outcome of the test on the exit code of the completed process. Enter a comma-delimited string of exit codes with a corresponding step to run when the process exits with this code.

To test your command, click Execute.

The content is now available for you to filter and add assertions.

The Execute External Command step has a default name using this convention: *Command commandfirstword*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

## File System Snapshot

The File System Snapshot step lets you list the files in a directory in a format that is operating system independent.

You can list a single file, all the files in a directory, or all the files in a directory tree.

To open a step editor, click the step.

Enter the following parameters:

**Execute from directory**

Enter the path name or browse to the file or directory.

**Recurse Subdirectories**

Select if you want the complete directory tree including sub directories.

**Include File Size**

Select if you want the file sizes to be listed.

**Include Date/Time**

Select if you want the last modified date to be listed.

**If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

Click Execute Now to run and start scanning the file system.

The content is now available for you to filter and add assertions.



## Execute Subprocess

The Execute Subprocess step lets you call a subprocess test case as a single step.

This step is used to call a subprocess and receive the outputs. This step is commonly used when a specific function is performed in numerous test cases. For example, if a specific validation always works the same way, you can create a validation subprocess and can add it to different test cases.

For more information, see [Building Subprocesses](#).

Complete the following fields:

**Subprocess**

Select the subprocess from the pull-down menu.

**Fully Expand Props**

When the parameters have nested properties, fully expand the properties before sending to the subprocess.

**Send HTTP Cookies**

Select to forward cookies to the subprocess.

**Get HTTP Cookies**

Get the HTTP cookies from the subprocess.

**If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

The documentation that was entered into the Documentation area of the subprocess is displayed.

The Parameters to Subprocess panel contains a list of the parameters that the subprocess requires. These keys and values must be present in your current test case. Edit the Value column as necessary to supply the correct values.

The Result Properties panel lists all the properties that are produced in the subprocess. Select the properties to be returned from the subprocess. These properties are used in your test case. You are not limited to a single return value.

When this step runs, it appears to run as a single step. The Interactive Test Run (ITR) utility shows the events that fired while the step ran. The short name of the events combines the name of the current step and the name of the subprocess step where the event was fired..

If an assertion is triggered in the subprocess, then the appropriate event appears in the Test Events tab of the Interactive Test Run (ITR) utility.

The default Execute Subprocess step uses the following convention: *Subprocess subprocessname*. You can change step names at any time.

### JUnit Test Case-Suite

The Execute JUnit Test Case/Suite step lets you run a JUnit test case or a JUnit test suite in a DevTest step. If the JUnit test passes, then so does the test step. After a failure, you can redirect to another test step.

**Prerequisite:** Your JUnit test must be on the classpath. Drop it into the hot deploy directory.

Complete the following fields:

#### Test Class

Enter the package name of the JUnit test class or test suite class. You can browse the classpath using the Browse button. This technique also confirms that your class is on your classpath.

#### If environment error

Select the step to redirect to if the JUnit test fails.

Click Load to load the class files. The class tree is displayed in the left panel.

Click Execute to run the JUnit test. The standard JUnit results are displayed in the right panel.

The JUnit Test Case/Suite step has a default name using this convention: *JUnit Test Case Suite testclassname*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

## Read a File (Disk URL or Classpath)

The Read a File step reads a file from your file system, a URL, or the classpath.

Files are used as a source of data for testing. This step can be paired with the Load a set of File Names data set to provide source data for testing.

You can read a text file or a binary file. The contents of the file can optionally be stored in a property.

Complete the following fields:

### **File**

Enter the path name, a URL, or a classpath, or browse to the file using the Browse button.

### **File Encoding**

Accept the default encoding of UTF-8 or select an alternate encoding from the drop-down list. You can also select Auto-detect and click the Detect button to have <IdtF> select an encoding type for you.

### **Property Key**

Enter the name of the property in which to store the file contents (optional).

### **Load as Byte []**

To load contents as a byte array, select this check box. This capability is useful when loading a file to be used as a binaryData type in a web service execution parameter.

### **If environment error**

Select the step to redirect to if the Read a File test fails.

### **Display as characters**

The contents are displayed as hexadecimal encoded bytes unless you select this check box. This check box is only visible if the Load as Byte [] box is selected.

Click Load to load and display the file. The content is now ready for you to filter and add assertions.

**Note:** If a binary file is loaded but you do not select to load as byte, DevTest converts the data to characters (many of which are unreadable) and the step response is a string.

The Read a File step has a default name using this convention: *Read file [set the File Name variable]*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

## External - FTP Step

The FTP step lets you send or receive a file using FTP protocol. After entering the FTP information, user name, and user password you can either upload a file or download a file.

If necessary, you can drag the FTP step editor window to the left to display the Execute Now button.

Complete the following fields:

**Host**

Enter the host name of the FTP server (without the protocol).

**Port**

Enter the port for FTP server access. A port is optional; the default port is 21.

**User**

Enter the user ID for FTP server access.

**Password**

Enter the password for FTP server access.

**Direction**

Indicate if the data flow is an upload or a download.

**Mode**

Specify passive or active FTP, or enter a property that indicates passive or active.

**Transfer Type**

Select the file transfer type; Binary or ASCII, or enter a property that indicates binary or ASCII.

**Host Path**

Enter the path to the source file (either on the FTP server or local computer).

**Local Path**

Enter the path for the destination file (either on the FTP server or local computer).

**Note:** Host Path is the path of the file on the remote computer. Local Path is always the path to the file on the local computer that is running DevTest. When you upload a file, you are uploading the file from the Local Path to the Host Path. When you download a file, you are downloading the file from the Host Path to the Local Path.

**If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

Click Execute Now to initiate the send/receive action.

The response from a download is the file itself. The response from a successful upload is the string success.

**Note:** If a file by the same name exists, it is overwritten without a warning message.

The FTP step has a default name using this convention: *FTP action (put or get) hostname*; for example, **FTP get ftp.download.com**. You can change step names at any time.

## JMS Messaging Steps

**The following steps are available:**

[JMS Messaging \(JNDI\)](#) (see page 286)

[JMS Messaging - Message Consumer](#) (see page 295)

[JMS Send Receive Step](#) (see page 299)

## JMS Messaging (JNDI)

The JMS Messaging (JNDI) step lets you send messages to, and receive messages from, topics and queues. You can also receive, change, and forward an existing message. The list of possible queues and topics can be browsed using JNDI. Provide client libraries where DevTest can read them.

All the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended) are supported.

The JMS Messaging (JNDI) step is configured using a single editor regardless of the messaging requirements. The input options vary on the messaging requirements. The editor only allows valid configurations, so when you enable some features, others could become inactive.

The JMS Messaging (JNDI) step has a default name using this convention: *JMS publish-queueName publish*. If there is not a publish queue name, the default step name is *JMS subscribe-queueName subscribe*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements in *Administering*.

**Parameter Requirements:** This step requires the connection parameters and the queue or topic names that are used in the application under test. Other parameters could be required, depending on your environment. Get these parameters from the application developers. In most cases, you can browse for server resources to get some of these required parameters.

The **jms.tst** test case in the examples project shows the step that this section describes.

The **jms.tst** test case uses a publish/subscribe JMS step to send a message and listen on a temporary queue. A Message Driven Bean (MDB) on the server handles the message and drops the message onto the temporary queue. The message type is text. The message is an XML payload that is created by inserting properties dynamically into the XML elements. The properties are read from the **order\_data** data set. After the response message is received, the XML from the JMS message is put into a property. The next step does an assertion validating the order ID. After this check asserts true, the existing message object is changed, and the message is sent to another JMS destination.

The **jms.tst** test case shows how to listen for and intercept messages as they move through a multipoint messaging service backbone. You can run this test case against the demo server on your computer. The application backend is available there.

The editor for the JMS Messaging (JNDI) step contains the following tabs:

- The Base tab is where you define the connection and messaging parameters.

- The Selector Query tab lets you specify a selector query to be run when listening for a message on a queue.
- The Send Message Data tab is where you create the message content.
- The Response Message tab is where the response messages are posted.

## Base Tab

The Base tab is where you define the connection and messaging parameters.

The following graphic shows the Base tab. The tab is divided into the following sections:

- Server Connection Info
- Subscriber Info
- ReplyTo Info
- Publisher Info
- Error Handling and Test

The screenshot shows the 'JMS Messaging (JNDI) - JMS {{JNDI\_QUEUE}} subscribe' dialog box, Base tab. The dialog is divided into several sections:

- Server Connection Info:** Contains fields for JNDI Factory Class ({{JNDI\_FACTORY}}), JNDI Server URL ({{JNDI\_URL}}, JMS ConnectionFactory ({{JNDI\_CONNECTION\_FACTORY}}, User ({{JNDI\_USERNAME}}, and Password (masked). There are checkboxes for 'Share Sessions' and 'Share Publishers', and buttons for 'Stop All' and 'Advanced'.
- Subscriber Info:** Contains an 'enable' checkbox, Name ({{JNDI\_QUEUE}}, Type (Queue - ASYNC), Timeout (secs) (30), Async Key (ASYN), Durable Session Key, Session Mode (Auto Acknowledge), and checkboxes for 'use temporary queue' and 'make payload last received'.
- ReplyTo Info:** Contains an 'enable' checkbox, Name, and Type (Queue).
- Publisher Info:** Contains an 'enable' checkbox, 'use transaction' checkbox, Name, Type (Topic), and Message (Empty). There is an 'Advanced' button.
- Error Handling and Test:** Contains a dropdown for 'If environment error' (Abort the Test) and a 'Test...' button.

At the bottom, there is a tab bar with 'Base', 'Selector Query', 'Send Message Data', and 'Response Message'.

To enable and disable the Subscriber Info, Publisher Info, and ReplyTo Info, use the enable check box in the top left corner of each section. This option allows you to configure the step to be a publish step, a subscribe step, or both. You can also select to include a JMS reply to component in the step.

When you finish configuring the test step, click Test in the Error Handling and Test section to test the configuration settings.



### Server Connection Info

Enter the JNDI information in the Server Connection Info section of the Base tab.

To simplify changing the application under test, parameterize these values with properties that are in your configuration. The previous graphic shows an example of this approach.

The following parameters are available to you for the system under test:


#### **JNDI Factory Class**

The fully qualified class name of the context factory for the JNDI provider.

#### **JNDI Server URL**

The URL for connecting to the JNDI server. The format of the URL depends on the specific JNDI provider being used.

#### **JMS Connection Factory**

Use Search  to browse available resources on the server. Select or enter a connection factory to use for this step execution according to the JMS specification.

The pull-down menus contain common examples or templates for these values.

The user and password could be optional.

#### **User**

The user name for connecting to the JNDI provider and getting a handle to the connection factory.

#### **Password**

The password for connecting to the JNDI provider and getting a handle to the connection factory.

#### **Share Sessions and Share Publishers**

To share JMS sessions and publishers throughout the test case, use these check boxes. This approach can lower overhead, but does not always provide a realistic simulation because typically JMS clients want to release resources. If you select the Share Publishers check box, the Share Sessions check box is also selected. You cannot share publishers without sharing sessions. For more detailed information about these parameters, see the *Deliberate Delays in VSE* (<https://support.ca.com/iri/portal/kbtech?docid=603964&searchID=TEC603964&fromKBResultsScreen=T>) knowledge base article.

#### **Stop All**

Lets you stop any listeners at design time now. Some listeners can get orphaned, but still consume messages. When they do, it is difficult to create test cases.

#### **Advanced**

Displays a panel where you can add custom properties that are sent with the connection information and you can configure the second-level authentication.

**Note:** The Server Connection Info user and password fields are for connecting to the JNDI provider and getting a handle to the connection factory. The user and password fields in the Second Level Authentication tab are for getting a handle to the actual JMS connection.


#### **Publisher Info**

To set up the ability to send messages, select the enable check box.

To execute a commit when the message is sent, select the use transaction check box.


Enter the following parameters:

##### **Name**

The name of the topic or queue. Use Search  to browse the JNDI server for the topic or queue name.

##### **Type**

Select whether you are using a topic or queue. To see what messages are

waiting to be consumed from a queue (only), use Browse  to the right of this field.

##### **Message**

Select the type of message you are sending. The supported types are Empty, Text, Object, Bytes, Message, and Mapped (Extended).

##### **Advanced**


Displays a panel where you can edit the message headers and can add message properties.

#### **Subscriber Info**

To set up the ability to receive messages, select the enable check box.

Enter the following parameters:

**Name**

The name of the topic or queue. Use Search  to browse the JNDI server for the topic or queue name.

**Type**

Select whether you are using a topic or queue, and whether to listen in synchronous or asynchronous mode. For asynchronous mode, you also must have an entry in the Async Key field. To see what messages are waiting to be

consumed from a queue, use the Browse icon  to the right of this field.

**Timeout (secs)**

The period to wait before there is an interrupt waiting for a message. Use 0 for no timeout.

**Async Key**

The value that is necessary for identifying asynchronous messages. This field is necessary only in asynchronous mode. The field is used in a subsequent Message Consumer step to retrieve asynchronous messages.

**Durable Session Key**

By entering a name here, you are requesting a durable session. You are also providing a key for that session. A durable session lets you receive all of your messages from a topic even if you log out and then you log in again.

**Session Mode**

The options available are:

- Auto Acknowledge: The JMS client libraries acknowledge the JMS Messages as soon as they are received.
- Client Acknowledge: The JMS client must explicitly acknowledge the JMS Messages.
- Use Transaction: The JMS Session operates under a transaction. The acknowledge mode is ignored.
- Auto (Duplicates Okay): The JMS client library automatically acknowledges at unknown intervals. As a result, duplicate messages could be received if the automatic acknowledgment does not arrive before the JMS Provider retries the delivery.

Auto Acknowledge and Client Acknowledge and Auto (Duplicates Only) have no practical differences. With Client Acknowledge, each received message is acknowledged immediately upon receipt. The only difference is that the acknowledge call is made explicitly instead of letting the JMS client library do it. With Auto (Duplicates Only), the behavior is the same as Auto Acknowledge except under high loads.

The Use Transaction option is not strictly an acknowledgment mode setting. The option is included in the list for two reasons:

- If the JMS Session is operating under a transaction, then the acknowledgment modes are ignored. Messages are acknowledged by committing the session transaction.
- The Use Transaction mode is still a way of controlling guaranteed delivery of messages from JMS. If a message is received and the session transaction is not committed, the message is resent, as if it was not acknowledged.

#### Use temporary queue/topic

If you want the JMS provider to set up a temporary queue/topic on your behalf, select this check box. When a temporary queue/topic is used, DevTest automatically sets the **JMS ReplyTo** parameter of the message you send to the temporary queue/topic. The temporary queue/topic feature must always be used with a publisher so that a reply can be sent. If you use a temporary queue/topic, the ReplyTo section is disabled.

#### Make payload last response

To make the payload response the last response, select this check box.


#### ReplyTo Info

To set up a destination queue/topic, select the enable check box.


If your application requires a destination, it is set up in this section.

Enter the following parameters:

##### Name

The name of the topic or queue. Use Search  to browse the JNDI server for the topic or queue name.

##### Type

Select whether you are using a topic or queue. To see what messages are waiting to be consumed from a queue (only), use Browse  to the right of this field.

#### Error Handling and Test

If an error occurs, the Error Handling and Test section lets you redirect to a step.

##### If Environment Error

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

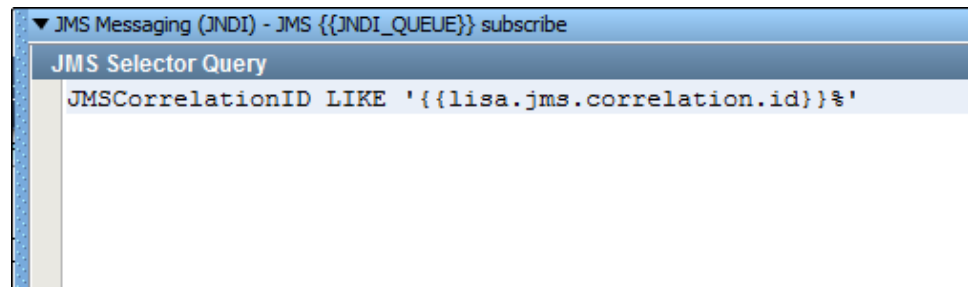
Click Test to test your step configuration settings.

## Selector Query Tab

You can enter a JMS selector query in the Selector Query tab. The syntax closely follows SQL. The query is a subset of SQL92.

A JMS selector query can be specified when listening for a message on a queue that is a response to a published message.

The following graphic shows a query looking for a **JMSCorrelationID** that matches the **lisa.jms.correlation.id** property as sent with the original message.

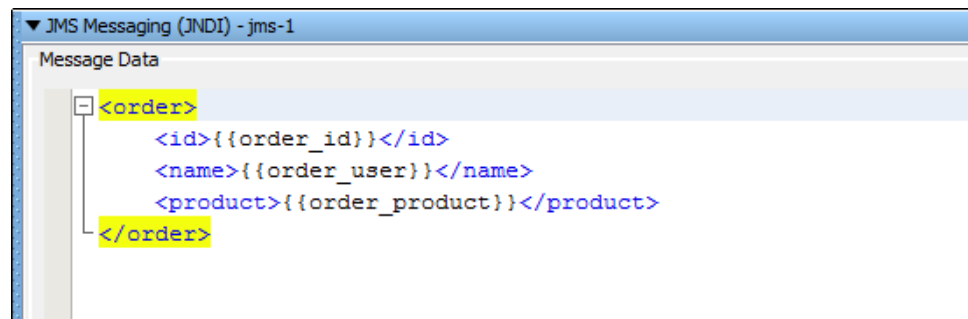


## Send Message Data Tab

If your step is configured to publish, you compose the message in the Send Message Data tab.

You can type the text, or you can click Read Message from File to read from a file. You can also store text in a property, in which case you would place the property in the editor, for example, **{{property\_name}}**.

The following graphic shows an XML fragment with properties. Using the properties allows the message to be created dynamically during the test run.



## Response Message Tab

If your step is configured to subscribe, the response appears in the Response Message tab after you click Test in the Base tab.

The tab shows the Complex Object Editor for the returned object. The returned object varies with the type of application server. You have access to all the JMS parameters returned in addition to the message itself. The object is loaded into the Complex Object Editor, where it can be manipulated like any other Java object.

The following graphic shows a text response from a JBoss object.

The screenshot shows a window titled "JMS Messaging (JNDI) - JMS {{JNDI\_QUEUE}} subscribe". Inside, the "Object Editor" is active, displaying the "Object Call Tree" on the left and the "Object State" table on the right. The "Object Call Tree" shows a single node: "org.jboss.mq.SpyMessage value=org.jboss.mq.SpyMessage". The "Object State" table lists various JMS parameters and their values.

Field Name	Type	Value As String
JMSCorrelationID	String	(null)
JMSDeliveryMode	int	-1
JMSExpiration	long	0
JMSMessageID	String	(null)
JMSPriority	int	-1
JMSRedelivered	boolean	false
JMSTimestamp	long	0
JMSType	String	(null)
JMSCorrelationIDAsB...	[B	[Access Not Allowed]
JMSDestination	Destination	(null)
JMSReplyTo	Destination	(null)
class	Class	class org.jboss.mq.S...
outdated	boolean	false
propertyNames	Enumeration	java.util.Collections\$...

At the bottom of the window, there are tabs: "Base", "Selector Query", "Send Message Data", and "Response Message". The "Response Message" tab is currently selected.

## JMS Messaging - Message Consumer

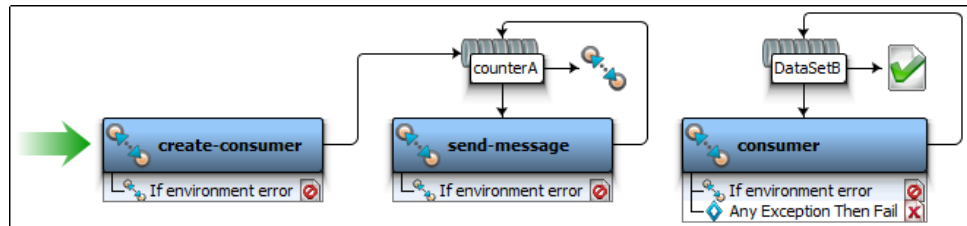
The Message Consumer step lets you consume asynchronous messages in a test case. This step can attach to a known queue or topic and can get messages posted for this subscriber. You identify yourself with a unique key. You must have already subscribed to the queue or topic and the messages were pushed to that destination.

**Prerequisite:** You must have the demo server running before you execute the example test case.

**Parameter Requirements:** Knowledge of the queue or topic being used in the application under test.

The Message Consumer step has a default name using this convention: *Listen on subscribe-queueName*. Before the queueName is entered, the default step name is Async JMS. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

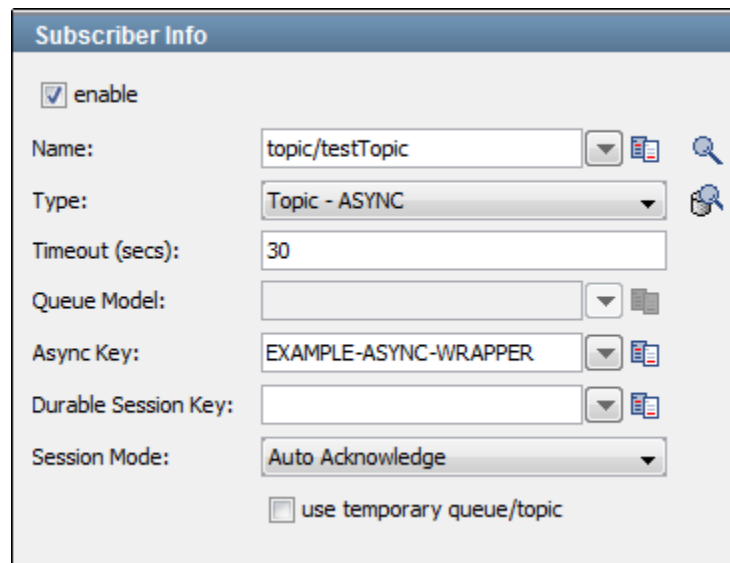
Review the **async-consumer-jms.tst** test case in the examples project to see what is being described in this section. The following graphic shows the **async-consumer-jms.tst** test case.



The **create-consumer** step subscribes to asynchronous message (topic/testTopic) using the Async Key (EXAMPLE-ASYNC-WRAPPER). The **send-message** step publishes message to a queue (queue/C). The number of messages to be published is controlled using a data set (counterA). The message consumer step has an Async queue (EXAMPLE-ASYNC-WRAPPER) using which message subscribed by the create-consumer step is consumed. The number of messages to be consumed is controlled using the data set (DataSetB).



**Note:** The Async Key specified in create-consumer and consumer steps must match.



The following graphic shows an example of the subscriber section of a step.





**Subscriber Info**



☒ enable



Name:   


Type:   

Timeout (secs):

Queue Model:   

Async Key:   

Durable Session Key:   

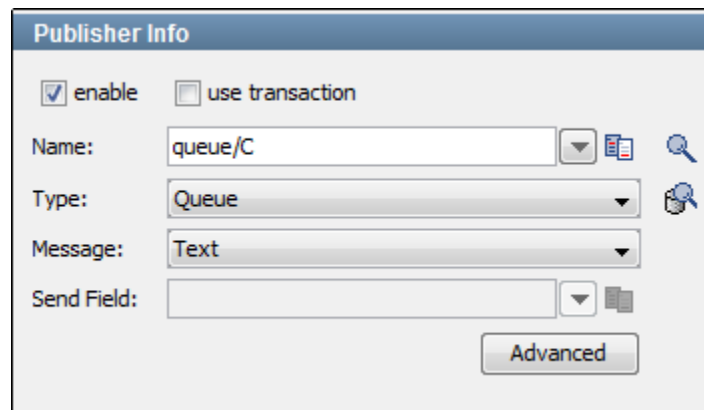
Session Mode:  

☐ use temporary queue/topic

To set up and enable the ability to listen to (subscribe to) messages, select the enable check box.



Notice that an asynchronous topic is specified in the Type field, and an Async Key parameter is defined. This key is necessary as an input in the current step.



The following graphic shows an example of the publisher section of a step.






**Publisher Info**

☒ enable ☐ use transaction

Name:   

Type:   


Message:  

Send Field:   

To set up the ability to send (publish) messages, select the enable check box. To execute a commit when the message is sent, select the use transaction check box.

Enter the following parameters:


**Name**

The name of the topic or queue. Use Search  to browse the JNDI server for the topic or queue name.

**Type**



Select whether you are using a topic or queue. To see what messages are

waiting to be consumed from a queue (only), use Browse  to the right of this field.

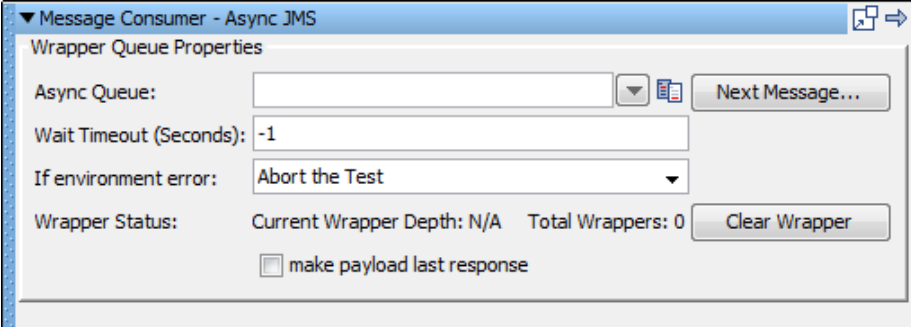
### Message

Select the type of message you are sending. The supported types are Empty, Text, Object, Bytes, Message, and Mapped (Extended).

### Advanced


Displays a panel where you can edit the message headers and can add message properties.

*Equation 1: JMS Messaging - Message Consumer step*



▼ Message Consumer - Async JMS

Wrapper Queue Properties

Async Queue:   Next Message...

Wait Timeout (Seconds):

If environment error:

Wrapper Status: Current Wrapper Depth: N/A Total Wrappers: 0 Clear Wrapper

☐ make payload last response

Enter the following parameters:

**Async Queue**

Enter or select the Async Key parameter that was named in a preceding subscriber step (EXAMPLE-ASYNC-WRAPPER). These names must match.

**Wait Timeout (Seconds)**

Enter the interval, in seconds, to wait for the next message.

**If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

**Wrapper Status**

The wrapper status contains two output status values:

- **Current Wrapper Depth:** Number of messages that are left to be read in the current wrapper.
- **Total Wrappers:** Number of wrappers (destinations).

**Make payload last response**

- To make the payload as the last response in the step, select the option.

If some messages are waiting, they can be read by clicking **Next Message**. The **Complex Object Editor** displays the message.

You are now ready to manipulate this object.

A wrapper is a FIFO list for holding responses from asynchronous topics and queues. A wrapper provides a place for the application to put responses for consumption later. Messages wait in this list for subsequent processing (in this **Message Consumer** step).

## JMS Send Receive Step

The JMS Send Receive step lets you connect to any JMS-compatible message server and do the following actions:

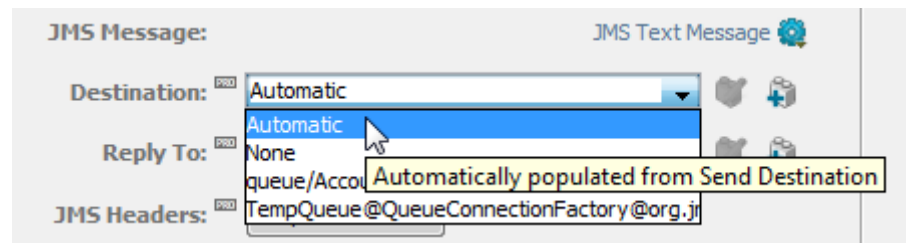
- Send a request message
- Receive a response message

The step editor has basic and advanced parameters. To display the advanced parameters, click PRO at the top of the editor.

Each parameter has a tooltip that describes the purpose of the parameter.

Some parameters include the value Automatic as an option. This value indicates that the actual setting is taken from another parameter. If you click the drop-down arrow and you place the mouse pointer over the value Automatic, a tooltip displays the name of the other parameter.

In the following graphic, the tooltip indicates that the value of the Destination parameter is automatically populated from the Send Destination parameter.



Some parameters let you change the editor so that you can enter a property as the value.









Some parameters that provide a discrete set of values let you change the editor so that you can enter the value directly. For example, the JMS Delivery Mode parameter has two values: Persistent and Non-persistent. In the JMS API, these values map to the numbers 2 and 1, respectively. To enter the value directly, switch to the direct editor. The direct editor also lets you specify a value that is not in the official enumeration of values.

## Send and Receive

The JMS Send Receive step has separate areas for configuring the JMS send operation and the JMS receive operation.

Specify two [destinations](#) (see page 60) in the step editor: one for the send operation, and one for the receive operation. The destinations can be the same or different. The lists are populated with the JMS destination assets from the active configuration.

The following graphic shows an example in which the send and receive operations have different destinations.

<b>JMS Send:</b> <input checked="" type="checkbox"/> Enabled	<b>JMS Receive:</b> <input checked="" type="checkbox"/> Enabled
Send Destination: <input type="text" value="queue/A"/>   	Receive Destination: <input type="text" value="queue/B"/>   
JMS Message: <input type="text" value="JMS Text Message"/> 	Correlation Scheme: <input type="checkbox"/> Enabled JMS Message ID to ... 

To send a message without receiving a response, disable the JMS receive operation. To wait for a message without having to send one first, disable the JMS send operation.

If you disable both operations, the step does nothing.

By default, the receive operation uses a synchronous [consumer](#) (see page 60). The advanced parameters include a check box for specifying an asynchronous consumer.

Each asset has a runtime scope. The JMS Send Receive step lets you specify a minimum runtime scope for the send and receive operations. The scope parameters are advanced parameters.

## JMS Message

The JMS Send Receive step lets you configure the message that is sent or received.

The following message types are supported:

- Text
- Bytes
- Stream
- Map
- Empty

You configure the payload of the message in the Content area.

The available editors in the Content area depend on the message type.

Text messages have the following editors:

- Plain text
- JSON
- EDI
- XML

Bytes messages have the following editors:

- Binary
- Graphic image

Stream messages have an editor that lets you add objects.

Map messages have an editor that lets you add key/value pairs.

Empty messages do not have an editor.

If you change the message type, the step checks whether the existing payload can be converted. If the conversion is not possible, the step discards the existing payload.

In JMS, a message includes a set of headers and an optional set of custom properties. You can configure these headers and properties in the step editor.

## JMS Correlation Schemes

Consider a messaging service with two clients that are running simultaneously. Each client can send a request message to the same request queue. The service can process the requests in any order and can send the response messages to a response queue. How does each client receive its own response and not receive the response that is meant for the other client?

The most common solution is to incorporate some type of identifier into the request, and copy the identifier into the response. This identifier is known as a *correlation ID*. Each client uses a unique correlation ID. Some mechanism is used to ensure that each client can only receive a response containing the correlation ID for that client.

You can enable correlation in the JMS Send Receive step. The following correlation schemes are available:

- JMS Correlation ID
- JMS Message ID to Correlation ID
- JMS Payload

Correlation schemes differ based on:

- How they route responses to the correct client by correlation ID
- Where the correlation ID is located in the request and response messages

## JMS Correlation ID

Most messaging platforms have a correlation ID field. In this scheme, the client generates a unique ID and sends a request message containing that ID in the correlation ID field. Before the service sends the response message, the service copies the correlation ID from the request message to the response message. The client listens for a response message that contains the same correlation ID as the original request.

By default, the correlation ID is automatically generated. To specify the value manually, use the Manual Value field.

The Reuse ID list indicates whether to generate a new correlation ID for each new transaction or use the same correlation ID throughout the test.

## JMS Message ID to Correlation ID

This scheme is similar to the JMS Correlation ID scheme.

Most messaging platforms also have a message ID field, which contains a unique identifier for the message. The message ID is automatically generated.

Before the service sends the response message, the service copies the message ID of the request message to the correlation ID of the response message. The client listens for a response message that contains a correlation ID that matches the message ID of the original request.

You cannot specify the message ID manually. You cannot reuse message IDs.

## JMS Payload

This scheme is based on a correlation ID that is embedded in the payload of the request and response messages. The scheme has an associated *payload scheme* that controls how the correlation ID is embedded. The default payload scheme lets you define XPath expressions for obtaining the correlation ID from the messages.

The following graphic shows an example of the JMS Payload correlation scheme with the XPath payload scheme.

The screenshot displays the configuration for the JMS Payload correlation scheme. It is divided into two main sections: 'Correlation Scheme' and 'Payload Scheme'. In the 'Correlation Scheme' section, the 'Enabled' checkbox is checked, and the 'JMS Payload' icon is highlighted with a red box. Below this, there are three dropdown menus: 'Manual Value' (empty), 'Reuse ID' (set to 'No'), and 'Auto Generate' (set to 'Yes'). The 'Payload Scheme' section shows the 'XPath' icon highlighted with a red box. Below this, there are two text input fields: 'Request XPath' containing 'request/id/getText()' and 'Response XPath' containing 'response/id/getText()'.

The Reuse ID list indicates whether to generate a new correlation ID for each new transaction or use the same correlation ID for the duration of the test.

You cannot mix the JMS Payload correlation scheme with the other two correlation schemes on the same queue. If one listener is listening on a queue with the JMS Payload correlation scheme, then all listeners on that queue must be using JMS Payload. However, you can use JMS Correlation ID and JMS Message ID to Correlation ID as payload schemes for the JMS Payload correlation scheme.

## Test the JMS Send Receive Step

You can verify the functionality of the JMS Send Receive step in the editor.

The execution log provides a high-level view of the activity that happens behind the scenes. For example, the following lines in the execution log show the creation of various [JMS client assets](#) (see page 60):

```
Creating JMS Connection
Starting JMS Connection
Creating JMS Session
Performing JNDI lookup with name: queue/B
Creating JMS Consumer on Queue B
Performing JNDI lookup with name: queue/A
Creating JMS Producer
```

**Follow these steps:**

1. Click the green Execute button in the step editor.
2. To cancel the step while it is executing, click Cancel.

The Response tab shows the response that was received when the operation finishes.

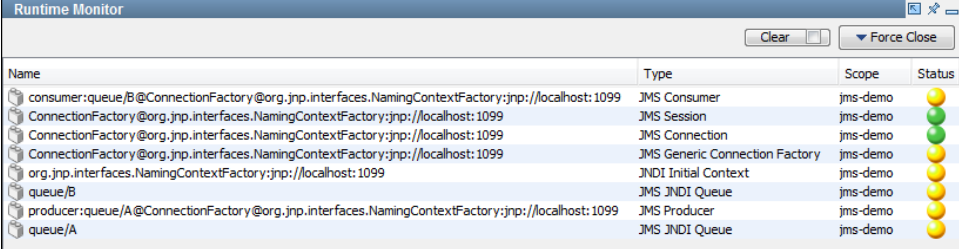
3. To view the step activity, click Execution Log.
4. To [monitor the asset instances](#) (see page 305), click Runtime Monitor.
5. To view the request that was sent, click the Request tab.



## Monitor and Close Cached Asset Instances

When you [test the JMS Send Receive step](#) (see page 304), a runtime monitor in the Response tab lets you monitor asset instances. You can also manually close the assets.

The following graphic shows an example of the runtime monitor.



Name	Type	Scope	Status
consumer:queue/B@ConnectionFactory@org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099	JMS Consumer	jms-demo	Yellow
ConnectionFactory@org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099	JMS Session	jms-demo	Green
ConnectionFactory@org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099	JMS Connection	jms-demo	Yellow
ConnectionFactory@org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099	JMS Generic Connection Factory	jms-demo	Yellow
org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099	JNDI Initial Context	jms-demo	Yellow
queue/B	JMS JNDI Queue	jms-demo	Yellow
producer:queue/A@ConnectionFactory@org.jnp.interfaces.NamingContextFactory:jnp://localhost:1099	JMS Producer	jms-demo	Yellow
queue/A	JMS JNDI Queue	jms-demo	Yellow

By default, the runtime monitor automatically removes assets that are closed. In the following procedure, you disable this behavior.

### Follow these steps:

1. In the Response tab, click Runtime Monitor.
2. Select the check box that appears inside the Clear button.
3. Execute the step again.

The runtime monitor displays the assets that were created during the execution of the step.

4. View the following information:
  - a. Name: The name of the asset.
  - b. Type: The type of asset.
  - c. Scope: The name of the test step, test case instance, test case, or DevTest component that corresponds to the runtime scope of the asset. The value has a tooltip that shows the scope.
  - d. Status: The color green indicates that the asset is active. The color yellow indicates that the asset is idle. The color gray indicates that the asset is closed.
5. To display more information about an asset, click the status icon.
6. To close an asset immediately, click the status icon and select Close or Force Close.
7. To close a set of assets, click Force Close.

## Tutorial - Send and Receive a JMS Message

This tutorial provides an introduction to using the JMS Send Receive test step.

### Prerequisites

- The registry is running.
- The demo server is running.

## Step 1 - Create JMS and JNDI Assets

In this procedure, you create JMS and JNDI assets from test steps in an example test case.

### Follow these steps:

1. Go to DevTest Workstation and open the examples project.
2. In the Tests folder, open the **jms.tst** test case.
3. Select the first JMS Messaging (JNDI) step. This step has the name **jms-1**.
4. Click the Generate assets from the selected steps button in the model toolbar.
5. Select the second JMS Messaging (JNDI) step. This step has the name **send-msg-post-update**.
6. Click the Generate assets from the selected steps button in the model toolbar.
7. In the Configs folder, open the project configuration.
8. Confirm that the JMS and JNDI assets were generated. In the next procedure, you select two of the JMS destination assets.

## Step 2 - Configure the JMS Send Receive Step

In this procedure, you perform the following actions:

- Add the JMS Send Receive step to a new test case.
- Configure the step to send a JMS text message to a queue and receive the response from a different queue.

The following graphic shows a portion of the step editor. The fields that you change are highlighted.

The screenshot shows the configuration interface for a JMS Send Receive step. It is split into two panels. The left panel, titled 'JMS Send: [x] Enabled', contains a 'Send Destination' dropdown menu set to 'queue/A' and a 'JMS Message' field set to 'JMS Text Message'. The right panel, titled 'JMS Receive: [x] Enabled', contains a 'Receive Destination' dropdown menu set to 'queue/B' and a 'Correlation Scheme' field set to 'Enabled'. Below these panels is a 'Content' area with a text input field containing 'Hello, world.'.

The Send Destination field specifies the queue to which the message is sent.

The Receive Destination field specifies the queue from which the response is received.

The Content area specifies the text of the message.

### Follow these steps:

1. Create a test case in the examples project.
2. Add a JMS Send Receive step.  
The step editor appears.
3. In the Send Destination list, select **queue/A**.
4. In the Receive Destination list, select **queue/B**.
5. In the Content area, type a sentence.
6. Save the test case.

## Step 3 - Examine JMS and JNDI Assets

In this procedure, you examine one of the JMS destination assets that the JMS Send Receive step uses. You also examine the JNDI context asset that the JMS destination asset uses.

**Follow these steps:**

1. Click the Edit Selected Asset button that appears to the right of the Send Destination list.  
The editor for the JMS destination asset appears.
2. Review the parameters, but do not change them. You can place the mouse pointer over the parameter names to display tooltips.
3. Click the Edit Selected Asset button that appears to the right of the JNDI Context list.  
The editor for the JNDI context asset appears.
4. Review the parameters, but do not change them.
5. Click Cancel to return to the editor for the JMS destination asset.
6. Click Cancel to return to the step editor.

## Step 4 - Test the JMS Send Receive Step

In this procedure, you verify that the JMS Send Receive step is configured correctly.

**Follow these steps:**

1. Click the green Execute button in the step editor.
2. Confirm that the Request tab and the Response tab contain the same sentence.
3. Close the step editor.

## BEA Steps

**The following steps are available:**

[WebLogic JMS \(JNDI\)](#) (see page 309)

[Message Consumer](#) (see page 315)

[Read a File](#) (see page 316)

[Web Service Execution \(XML\)](#) (see page 316)

[Raw SOAP Request](#) (see page 316)

[FTP Step](#) (see page 316)

## WebLogic JMS (JNDI)

The WebLogic JMS (JNDI) step lets you send messages to, and receive messages from, topics and queues. You can also receive, change, and forward an existing message.

WebLogic JMS (JNDI) supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

The WebLogic JMS (JNDI) step is configured using a single editor, regardless of the messaging requirements. The input options vary on the messaging requirements. The editor only allows valid configurations. When you enable some features, others can become inactive.

**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see *Third-Party File Requirements in Administering*.

**Parameter Requirements:** This step requires the connection parameters and the subject names that are used in the application under test. The following sections describe the necessary parameters. There can be other parameters that are required, depending on your environment. Get these parameters from the developers of the application.

The editor for the WebLogic JMS (JNDI) step contains the following tabs:

The Base tab is where you define the connection and messaging parameters.

The Selector Query tab lets you specify a selector query to be run when listening for a message on a queue.

The Send Message Data tab is where you create the message content.

The Response Message tab is where the response messages are posted.

## Base Tab

The Base tab is divided into the following sections:

- Server Connection Info
- Subscriber Info
- Publisher Info
- ReplyTo Info
- Error Handling and Test

To enable and disable the Subscriber Info, Publisher Info, and ReplyTo Info sections, use the enable check box in each section. Using these check boxes, you can configure the step to be a publish step, a subscribe step, or both. You can also include a JMS reply to component in the step.

When you finish configuring the test step, click Test in the Error Handling and Test section to test the configuration settings.

### Server Connection Info

Enter the JNDI information.

These values are parameterized with properties from your configuration. These properties simplify changing the application under test. By default, the WLS\_SERVER property in the JNDI Server URL is used. This property must be added to your configuration if you plan to use it.

The five parameters are available to you for the system under test. The pull-down menus contain common examples or templates for these values.


#### **JNDI Factory Class**

The fully qualified class name of the context factory for the JNDI provider.

#### **JNDI Server URL**

The URL for connecting to the JNDI server. The format of the URL depends on the specific JNDI provider being used.

#### **JMS Connection Factory**

Use Search  to browse available resources on the server. Select or enter a connection factory to use for this step execution according to the JMS specification.

#### **User**

The user name for connecting to the JNDI provider and getting a handle to the connection factory.

### Password

The password for connecting to the JNDI provider and getting a handle to the connection factory.

### Share Sessions and Share Publishers

To share JMS sessions and publishers throughout the test case, use these check boxes. This approach can lower overhead, but does not always provide a realistic simulation because typically JMS clients want to release resources. If you select the Share Publishers check box, the Share Sessions check box is also selected. You cannot share publishers without sharing sessions. For more detailed information about these parameters, see the *Deliberate Delays in VSE* (<https://support.ca.com/irj/portal/kbtech?docid=603964&searchID=TEC603964&fromKBResultsScreen=T>) knowledge base article.

### Stop All

Lets you stop any listeners at design time now. Some listeners can get orphaned, but still consume messages. When they do, it is difficult to create test cases.

### Advanced


Displays a panel where you can add custom properties that are sent with the connection information and you can configure the second-level authentication.

### Publisher Info


To set up the ability to send (publish) messages, select the enable check box. To execute a commit when the message is sent, select the use transaction check box.

Enter the following parameters:

#### Name

The name of the topic or queue. Use Search  to browse the JNDI server for the topic or queue name.

#### Type

Select whether you are using a topic or queue. To see what messages are waiting to be consumed from a queue (only), use Browse  to the right of this field.

#### Message

Select the type of message you are sending. The supported types are Empty, Text, Object, Bytes, Message, and Mapped (Extended).

#### Advanced




Displays a panel where you can edit the message headers and can add message properties.

### Subscriber Info

Select the enable check box to set up to enable the ability to receive (subscribe to) messages.


Enter the following parameters:

#### **Name**

The name of the topic or queue. Use Search  to browse the JNDI server for the topic or queue name.

#### **Type**

Select whether you are using a topic or queue, and whether to listen in synchronous or asynchronous mode. For the asynchronous mode, you must

also have an entry in the Async Key field. Use Browse  to the right of this field to see what messages are waiting to be consumed from a queue (only).

#### **Timeout (secs)**

The period to wait before the application interrupts waiting for a message (this field can be left blank for no timeout).

#### **Async Key**

The value that identifies asynchronous messages. This value is only necessary in asynchronous mode. This value is used in a subsequent Message Consumer step to retrieve asynchronous messages.

#### **Durable Session Key**

By entering a name here you are requesting a durable session. You are also providing a key for that session. A durable session lets you receive all of your messages from a topic, even if you log out and then you log back in.

#### **use transaction**

To execute a Commit when a message is received, select the use transaction check box.

#### **use temporary queue/topic**

To have the JMS.provider set up a temporary queue/topic on your behalf, select the use temporary queue/topic check box. When a temporary queue/topic is used, the JMS **ReplyTo** parameter of the message you send to the temporary queue/topic is automatically set. The temporary queue/topic feature must always be used with a publisher so that a reply can be sent. If you use a temporary queue/topic, the ReplyTo section is disabled.

### **make payload last response**

To make the payload the response of this step, select the make payload last response check box.


### **ReplyTo Info**

To set up a destination queue or topic, select the enable check box.


If your application requires a destination, it is set up in this section.

Enter the following parameters:

#### **Name**

The name of the topic or queue. Use Search  to browse the JNDI server for the topic or queue name.

#### **Type**

Select whether you are using a topic or queue. To see what messages are waiting to be consumed from a queue (only), use Browse  to the right of this field.

### **Error Handling and Test**

If an exception occurs, the Error Handling and Test section lets you redirect to a step.

#### **If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

Click Test to test your step configuration settings.

## Selector Query Tab

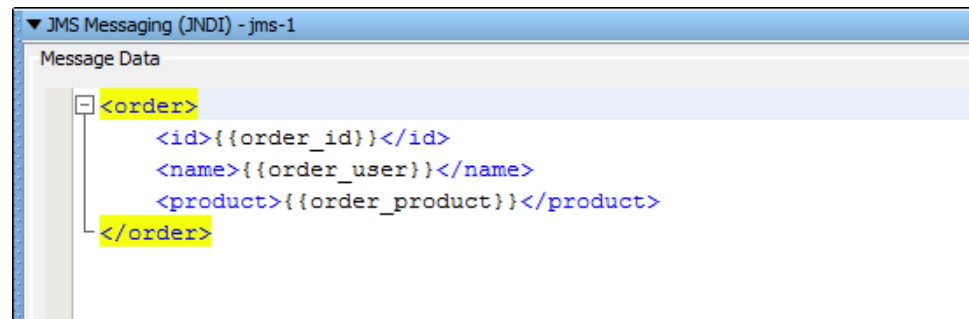
You can enter a JMS selector query in this editor. The syntax closely follows SQL and is a subset of SQL92. A JMS selector query can be specified when listening for a message on a queue that is a response to a published message. The previous graphic shows a specific query for a **JMSCorrelationID** that matches one set in a property as sent with the original message.

A built-in mechanism allows a test creator to set the **JMSCorrelationID** for a message before sending it. Before the message is sent, you can set the correlation ID by setting the **lisa.jms.correlation.id** property.

A non-zero value is detected, and the message **JMSCorrelationID** property is set before the message is sent.

## Send Message Data Tab

If your step is configured to publish, this tab lets you compose your message. The Send Message Data tab view in the following example shows a text message.



This example shows an XML fragment with properties being used. You can type the text, click Read Message from File to read from a file, or you can store the fragment in a property. If you store the text in a property, simply place the property in the editor: for example, **LISA\_PROP**.

Notice that properties are used in the message XML allowing the message to be created dynamically during the test run.

## Response Message Tab

If your step is configured to subscribe, your response is shown here. For more information, see [JMS Messaging \(JNDI\)](#) (see page 286).

## Message Consumer

For detailed information about this step, see [JMS Messaging - Message Consumer](#) (see page 295).

## Read a File

For detailed information about this step, see [Read a File \(Disk, URL or Class Path\)](#) (see page 283).

## Web Service Execution (XML)

For detailed information about this step, see [Web Service Execution \(XML\) Step](#) (see page 186).

## Raw SOAP Request

For detailed information about this step, see [Web Raw SOAP Request](#) (see page 229).

## FTP Step

For detailed information about this step, see [External - FTP Step](#) (see page 284).

## Sun JCAPS Steps

**The following steps are included.**

[JCAPS Messaging \(Native\)](#) (see page 317)

[JCAPS Messaging \(JNDI\)](#) (see page 320)

**More information:**

[Web Service Execution \(XML\) Step](#) (see page 186)

[SQL Database Execution \(JDBC\)](#) (see page 253)

[Read a File \(Disk URL or Classpath\)](#) (see page 283)

[External - FTP Step](#) (see page 284)

[Message Consumer](#) (see page 315)

[Raw SOAP Request](#) (see page 316)

## JCAPS Messaging (Native)

The JCAPS Messaging (Native) step lets you send and receive messages from topics and queues. You can also receive, change, and forward an existing message.

JCAPS Messaging (Native) supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

The JCAPS Messaging (Native) step is configured using a single editor regardless of the messaging requirements. The input options vary on the messaging requirements. The editor only allows valid configurations, so when you enable some features others can become inactive.

The default JCAPS Messaging (Native) step has a name uses the following convention: *JCAPS queueName publish*. If there is not a publish queue name, the default step name is *JCAPS queueName subscribe*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements in *Administering*.

**Parameter Requirements:** This step requires the connection parameters and the subject names that are used in the application under test. The following sections describe the parameters that you require. There can be other required parameters, depending on your environment. Get these parameters from the developers of the application.

The messaging step editor for JCAPS Messaging (Native) is used to configure this step.

**JCAPS Messaging (Native) - JCAPS**

**Server Connection Info**

Host: localhost  
Port: 18007  
Advanced

**Subscriber Info**

☐ enable  
Name:   
Type: Queue  
Timeout (secs): 30  
Async Key:   
Durable Session Key:   
Session Mode: Auto Acknowledge  
☐ use temporary queue/topic  
☒ make payload last response

**ReplyTo Info**

☐ enable  
Name:   
Type: Queue

**Publisher Info**

☐ enable ☐ use transaction  
Name:   
Type: Queue  
Message: Empty  
Advanced

**Error Handling and Test**

If environment error: Abort the Test Test...

Base Selector Query Send Message Data Response Message

The editor for the JCAPS Messaging (Native) step contains the following tabs.

- The Base tab is where you define the connection and messaging parameters.
- The Selector Query tab lets you specify a selector query to be run when listening for a message on a queue.
- The Send Message Data tab is where you create the message content.
- The Response Message tab is where the response messages are posted.

## Base Tab

The Base tab is divided into the following sections:

- Server Connection Info
- Subscriber Info
- ReplyTo Info
- Publisher Info
- Error Handling and Test

To enable and disable the Subscriber Info, Publisher Info, and ReplyTo Info sections, use the enable check box in the top left corner of each section. Using these check boxes, you can configure the step to be a publish step, a subscribe step, or both. You can also select to include a JMS reply to component in the step.

When you finish configuring the test step, click Test in the Error Handling and Test section to test the configuration settings.

### Server Connection Info

The Server Connection Info section displays two parameters available to you for the system under test.

#### **Host**

The name of the JMS server.

#### **Port**

The port number the JMS server is running on.

The Advanced button displays a panel where you can add custom properties that are sent with the connection information.

All other tabs are defined in detail in [JMS Messaging \(JNDI\)](#) (see page 286).

## JCAPS Messaging (JNDI)

The JCAPS Messaging (JNDI) step lets you send and receive messages from topics and queues. You can also receive, change, and forward an existing message.

JCAPS Messaging (JNDI) supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

The JCAPS Messaging (JNDI) step is configured using a single DevTest editor regardless of the messaging requirements. The input options vary on the messaging requirements. The editor only allows valid configurations, so when you enable some features others can become inactive.

**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see *Third-Party File Requirements in Administering*.

**Parameter Requirements:** This step requires the connection parameters and the subject names that are used in the application under test. Other parameters may be required, depending on your environment. Get these parameters from the developers of the application.

Detailed information about parameters and fields for JCAPS Messaging can be found in [JMS Messaging \(JNDI\)](#) (see page 286).

**Default Step Names:** The default JCAPS Messaging (JNDI) step name uses the following convention: *JCAPS queueName publish*. If there is not a publish queue name, the default step name is *JCAPS queueName subscribe*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

## Oracle Steps

**The following steps are available:**

[Oracle OC4J \(JNDI\)](#) (see page 321)

[Oracle AQ Steps](#) (see page 322)



**More information:**

[Web Service Execution \(XML\) Step](#) (see page 186)

[Web-Raw SOAP Request](#) (see page 229)

[SQL Database Execution \(JDBC\)](#) (see page 253)

[Read a File \(Disk URL or Classpath\)](#) (see page 283)

[External - FTP Step](#) (see page 284)

[Message Consumer](#) (see page 315)

## Oracle OC4J (JNDI)

The Oracle OC4J (JNDI) step lets you send messages to, and receive messages from, topics and queues. You can also receive, modify, and forward an existing message. Oracle OC4J (JNDI) supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

The Oracle OC4J (JNDI) step is configured using a single editor regardless of the messaging requirements. The input options on the messaging requirements vary. The editor only allows valid configurations, so when you enable some features others could become inactive.

**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see *Third-Party File Requirements in Administering*.

**Parameter Requirements:** You need the connection parameters and the subject names that are used in the application under test. There may be other parameters that are required, depending on your environment. Get these parameters from the developers of the application.

DevTest, by default uses the OC4J\_SERVER property in the JNDI Server URL. This property must be added to your configuration if you plan to use it.

Detailed information about the parameters and fields for this step can be found in [JMS Messaging \(JNDI\)](#) (see page 286).

## Oracle AQ Steps

Oracle AQ is a messaging provider, like IBM WebSphere MQ, webMethods Broker, TIBCO EMS, and others. Oracle AQ fits into DevTest like any of those other messaging providers.

The Oracle AQ step is added to a test case to allow sending messages, receiving messages, receiving messages asynchronously, or some combination of the three.

For AQ, there are two separate step types (JMS and JPUB). They have the same functionality but represent two different methods for communicating with the Oracle AQ messaging provider.

Both Oracle AQ steps have the standard configuration sections for messaging steps.

- A Connection section for configuring the connection information.
- An optional Subscriber section for configuring the location from which the step receives its message and the type of message it receives.
- An optional Publisher section for configuring the location to which the step sends its message and the type of message it sends. The contents of the message to send are configured in a separate tab.
- The step sends one message, receives one message, or both. The step can run multiple times in a loop in the same test case to send or receive multiple messages.
- When the step is used as an asynchronous subscriber, then it is only run once in a test case. An extra Consumer step is run to consume each message received from the provider.

Two distinct ways to use AQ are:

- [JMS](#) (see page 323)
- [JPUB](#) (see page 329)

## Oracle AQ (JMS)

Oracle Advanced Queuing (AQ) is a messaging provider that is built into the Oracle database itself. AQ is used as the default JMS provider for many Oracle products, such as Oracle Enterprise Service Bus.

One of the two ways to use AQ is JMS.

The screenshot shows the 'Oracle AQ (JMS) - Step 3' configuration window. It is divided into four main sections: **Server Connection Info**, **Subscriber Info**, **ReplyTo Info**, and **Publisher Info**.   
**Server Connection Info** includes fields for JDBC URL (jdbc:oracle:thin:@localhost:1521:ord), JDBC Driver (oracle.jdbc.driver.OracleDriver), User, and Password. There are checkboxes for 'Share Sessions' and 'Share Publishers', and a 'Stop All' button.   
**Subscriber Info** includes fields for Schema, Name, Type (Queue), Timeout (secs) (30), Async Key, Durable Session Key, and Session Mode (Auto Acknowledge). There are checkboxes for 'enable', 'use temporary queue/topic', and 'make payload last response'.   
**ReplyTo Info** includes fields for Schema, Name, and Type (Queue).   
**Publisher Info** includes fields for Schema, Name, Type (Queue), and Message (Empty). There are checkboxes for 'enable' and 'use transaction', and an 'Advanced' button.   
**Error Handling and Test** section at the bottom has a dropdown for 'If environment error' (set to 'Abort the Test') and a 'Test...' button.   
The bottom of the window shows four tabs: 'Base', 'Selector Query', 'Send Message Data', and 'Response Message'.

Using the JMS library works like any other normal JMS provider, with a few notable differences:

- The JMS connection is not made through JNDI. The connection is made using a JDBC connection and involves entering a JDBC URL, driver class name, username, and password.
- Queues and topics are tied to schemas in the database. To send to a queue or receive from a queue, give both the queue name and queue schema.
- Each queue or topic is restricted to a specific type of JMS Message. For example, if a queue typically transports JMS Text Messages, then you cannot use that same queue to transport JMS Object Messages, or JMS Byte Messages.
- Setting up JMS queues and topics in the Oracle database involves running stored procedures.

Four tabs are available at the bottom of the editor.

- The Base tab is where you define your connection and messaging parameters.

- The Selector Query tab lets you specify a selector query to be run when listening for a message on a queue.
- The Send Message Data tab is where you create your message content.
- The Response Message tab is where your response messages are posted.

## Base Information Tab

The Base tab view is shown in the previous example contains five major sections:

- Server Connection Info
- Subscriber Info
- Publisher Info
- ReplyTo Info
- Error Handling and Test

The Server Connection Info and Error Handling and Test sections are always active. You can use the enable check box in the top left corner of each section to enable or disable Subscriber Info, Publisher Info, and ReplyTo Info. Using these check boxes, you can configure the step to publish a step, subscribe a step, or both. You can also select to include a replyto component in the step.

When you have configured your test step, click Test in the Error Handling and Test section to test your configuration settings.

### Server Connection Info

Here you enter the JDBC-related information.

Parameterize these values with properties that are in your configuration, making it easy to change the application under test.

DevTest, by default, uses the **oracle.jdbc.driver.OracleDriver** in the JDBC Driver location.

The following parameters are available to you for the system under test. The pull-down menus contain common examples or templates for these values.

#### JDBC URL

This field is prepopulated with default values.

#### JDBC Server

This field is prepopulated with default values.

#### User

Enter the user name.

#### Password

Enter the password.

#### Share Sessions and Share Publishers

To share JMS sessions and publishers throughout the test case, use these check boxes. This approach can lower overhead, but does not always provide a realistic simulation because typically JMS clients want to release resources. If you select the Share Publishers check box, the Share Sessions check box is also selected. You cannot share publishers without sharing sessions. For more detailed information about these parameters, see the *Deliberate Delays in VSE* (<https://support.ca.com/irj/portal/kbtech?docid=603964&searchID=TEC603964&fromKBResultsScreen=T>) knowledge base article.

### **Stop All**

Lets you stop any listeners at design time now. Some listeners can get orphaned, but still consume messages. When they do, it is difficult to create test cases.

### **Publisher Info**

To set up the ability to send (publish) messages, select the enable check box.

To execute a commit when the message is sent, select the use transaction check box.

Enter the following parameters:

#### **Schema**

Enter the name of the schema to use.

#### **Name**

Enter the name of the topic or queue to use.

#### **Type**

Select whether you are using a topic or queue.

#### **Message**

Select the type of message you are sending. The supported types are Empty, Text, Object, Bytes, Message, and Mapped (Extended).

#### **Advanced**

Displays a panel where you can edit the message headers and can add message properties.

### **Subscriber Info**

To set up to enable the ability to receive (subscribe to) messages, select the enable check box.

Enter the following parameters:

**Schema**

Enter the name of the schema to use.

**Name**

Enter the name of the topic or queue to use.

**Type**

Select whether you are using a topic or queue, and whether to listen in synchronous or asynchronous mode. For asynchronous mode, you also must have an entry in the Async key field. To see what messages are waiting to be consumed from a queue (only), click Browse, to the right of this field.

**Timeout (secs)**

Indicates the number of seconds before DevTest interrupts waiting for a message. For no timeout, leave this field blank.

**Async Key**

Enter the value that is necessary to identify asynchronous messages. This value is only required in asynchronous mode. This value is used in a subsequent Message Consumer step to retrieve asynchronous messages.

**Durable Session Key**

By entering a name here you are requesting a durable session. You are also providing a key for that session. A durable session lets you receive all of your messages from a topic even if you log out, and then you log in again.

**Session Mode**

Select the appropriate mode from the available options by clicking the drop-down list. Options are: Auto Acknowledge, Client Acknowledge, Use Transaction, Auto (Duplicates Okay).

**ReplyTo Info**

If your application requires a destination, it is set up in this section.

To set up a destination queue/topic, select the enable check box.

Enter the following parameters:

**Schema**

Enter the name of the schema to use.

**Name**

Enter the name of the topic or queue to use.

**Type**

Select whether you are using a topic or queue.

### **Error Handling and Test**

If an error occurs, the Error Handling and Test section lets you redirect to a step.

#### **If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

Click Test to test your step configuration settings.



## Oracle AQ (JPUB)

Two distinct ways to use AQ are:

- JMS
- J PUB

The Oracle AQ JMS API is a layer that is built on top of a lower-level AQ API. This lower-level API is much more difficult to deal with, and it acts almost nothing like JMS.

The principal distinction is the message format. The low-level AQ messages contain a payload, which can be any type that is defined in the database. The type could be a varchar or a clob, but usually it is a user-defined structured database type. Similar to AQ JMS Queues, each AQ low-level queue can only handle one payload type.

Oracle provides a utility with the name **JPUB** that can generate the Java objects that can deal with these user-defined structured types. J PUB works in the same way that Axis generates the Java objects that use web services. The low-level AQ step, **Oracle AQ J PUB**, can automatically use this utility to generate the client classes that are based on the queue information. The user then fills in their payload object using a standard COE.

A distinction between queues or topics does not exist. A client can:

- remove the next message from the AQ queue, making it essentially a queue, or
- read the next message from the AQ queue without removing it, making it essentially a topic.

Setting up the low-level AQ queues is again done through stored procedures. It may be necessary to create a specific user-defined structured type in the database before you create an AQ queue around it. Technically, you can interact with AQ JMS queues using the low-level API. The JMS queues simply have a specific payload type that is structured like a standard JMS message. However, you cannot use the AQ JMS API to interact with low-level AQ queues; that is, queues that do not use a JMS payload type.

To add an Oracle AQ (JPUB) step to a test case, click the step to open its editor.

▼ Oracle AQ (JPub) - Step2

Server Connection Info	Subscriber Info
JDBC URL: jdbc:oracle:thin:@localhost:	<input type="checkbox"/> enable
JDBC Driver: oracle.jdbc.driver.OracleDriver	Schema: [ ]
User: [ ]	Name: [ ]
Password: [ ]	Type: Queue
<input type="checkbox"/> Share Sessions	Timeout (secs): 30
<button>Stop All</button>	Async Key: [ ]
	<button>Generate JPub Classes</button>
	Payload Class Name: [ ]
	<button>Advanced</button>
	<input checked="" type="checkbox"/> make payload last response
Error Handling and Test	Publisher Info
If environment error: Abort the Test	<input type="checkbox"/> enable
<button>Test...</button>	Schema: [ ]
	Name: [ ]
	<button>Generate JPub Classes</button>
	Payload Class Name: [ ]
	<button>Advanced</button>

Base Condition Send Message Data Response Message

Four tabs are available at the bottom of the editor.

- The Base tab is where you define your connection and messaging parameters.
- The Condition tab lets you specify a condition to be run.
- The Send Message Data tab is where you create your message content.
- The Response Message tab is where your response messages are posted.

## Base Information Tab

The Base tab view that the previous graphic shows is the default view and has the following sections:

- Server Connection Info
- Subscriber Info
- Publisher Info
- Error Handling and Test

The Server Connection Info and Error Handling and Test sections are always active. The Subscriber Info and Publisher Info sections can be enabled or disabled using the enable check box in the top left corner of each section. Using these check boxes, you can configure the step to publish a step, subscribe a step, or both.

When you have configured your test step, click Test in the Error Handling and Test section to test your configuration settings.

### Server Connection Info

To simplify changing the application under test, parameterize the values with properties from your configuration. By default, the **oracle.jdbc.driver.OracleDriver** in the JDBC Driver location is used.

#### JDBC URL

This field is prepopulated with default values.

#### JDBC Server

This field is prepopulated with default values.

#### User

Enter the user name.

#### Password

Enter the password.

### Share Sessions and Share Publishers

To share JMS sessions and publishers throughout the test case, use these check boxes. This approach can lower overhead, but does not always provide a realistic simulation because typically JMS clients want to release resources. If you select the Share Publishers check box, the Share Sessions check box is also selected. You cannot share publishers without sharing sessions. For more detailed information about these parameters, see the *Deliberate Delays in VSE* (<https://support.ca.com/irj/portal/kbtech?docid=603964&searchID=TEC603964&fromKBResultsScreen=T>) knowledge base article.

### **Stop All**

Lets you stop any listeners at design time now. Some listeners can get orphaned, but still consume messages. When they do, it is difficult to create test cases.

### **Publisher Info**

To set up the ability to send (publish) messages, select the enable check box.

Enter the following parameters:

#### **Schema**

Enter the name of the schema to use.

#### **Name**

Enter the name of the topic or queue to use.

#### **Generate JPub classes**

Click to generate the JPub classes.

#### **Payload Class Name**

Enter the payload class name.

#### **Advanced button**

To open the Publisher Advanced dialog, click Advanced, then enter or select the Correlation and click OK.

### **Subscriber Info**

To set up to enable the ability to receive (subscribe to) messages, select the enable check box.

Enter the following parameters:

#### **Schema**

Enter the name of the schema to use.

#### **Name**

Enter the name of the topic or queue to use.

#### **Type**

Select whether you are using a topic or queue, and whether to listen in synchronous or asynchronous mode. For asynchronous mode, you also must have an entry in the Async key field. To see what messages are waiting to be consumed from a queue (only), click Browse, to the right of this field.

**Timeout (secs)**

Indicates the number of seconds before DevTest interrupts waiting for a message. For no timeout, leave this field blank.

**Async Key**

Enter the value that is necessary to identify asynchronous messages. This value is only required in asynchronous mode. This value is used in a subsequent Message Consumer step to retrieve asynchronous messages.

**Generate JPub classes**

Click to generate the JPub classes.

**Payload Class Name**

Enter the payload class name.

**Advanced**

Click to open the Subscriber Advanced dialog.

In the Advanced dialog, you can enter the Consumer Name, Correlation, and Message ID.

**Error Handling and Test**

If an error occurs, the Error Handling and Test section lets you redirect to a step.

**If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

Click Test to test your step configuration settings.

## TIBCO Steps

**The following steps are available:**

[TIBCO Rendezvous Messaging](#) (see page 335)

[TIBCO EMS Messaging](#) (see page 341)

[TIBCO Direct JMS](#) (see page 342)

**More information:**

[Web Service Execution \(XML\) Step](#) (see page 186)

[Web-Raw SOAP Request](#) (see page 229)

[SQL Database Execution \(JDBC\)](#) (see page 253)

[Read a File \(Disk URL or Classpath\)](#) (see page 283)

[External - FTP Step](#) (see page 284)

[Message Consumer](#) (see page 315)

## TIBCO Rendezvous Messaging

The TIBCO Rendezvous Messaging step lets you send and receive messages from Rendezvous "Subjects" using Native Rendezvous protocol. You can also receive, change, and forward an existing message.

The TIBCO Rendezvous Messaging step is configured using a single editor regardless of the messaging requirements. The input options vary on the messaging requirements. The editor only allows valid configurations, so when you enable some features others could become inactive.

The default TIBCO Rendezvous Messaging step uses the following convention: *RV queueName publish*. If there is not a publish queue name, the default step name is *RV queueName subscribe*. If another step also uses the default step name, a number is appended to the step name. You can change step names at any time.

**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements in *Administering*.

**Parameter Requirements:** This step requires the connection parameters and the subject names that are used in the application under test. The following sections describe the parameters that you require. There could be other required parameters, depending on your environment. Get these parameters from the developers of the application.

The screenshot shows the 'TIBCO Rendezvous Messaging - RV queue.sample subscribe' configuration window. It is divided into several sections:

- Server Connection Info:** Includes fields for Service (7474), Network, Daemon (192.168.10.221:7474), and Client Mode (Native Client). A 'Stop All' button is at the bottom.
- Subscriber Info:** Includes a checked 'enable' checkbox, Subject (queue.sample), Timeout (secs) (30), and Async Key (ASync).
- Certified Transport Info:** Includes an unchecked 'enable' checkbox, Sender Name, Advisory Subject (\_RV.INFO.RVCM.DELIVERY.CONFIRM. >), and Time Limit.
- Publisher Info:** Includes an unchecked 'enable' checkbox, an unchecked 'Enable Inbox Type' checkbox, Subject, Message (Empty), and Send Field.
- ReplyTo Info:** Includes an unchecked 'enable' checkbox and Subject.
- Error Handling and Test:** Includes a dropdown for 'If environment error' (Abort the Test) and a 'Test...' button.

At the bottom, there are three tabs: 'Base', 'Send Message Data', and 'Response Message'.

The editor for the TIBCO Rendezvous Messaging step contains the following tabs:

- The Base tab is where you define the connection and messaging parameters.
- The Send Message Data tab is where you create the message content.
- The Response Message tab is where the response messages are posted.



## Base Tab

The Base tab is divided into the following sections:

- Server Connection Info
- Subscriber Info
- Certified Transport Info
- Publisher Info
- ReplyTo Info
- Error Handling and Test

To enable and disable the Subscriber Info, Publisher Info, and ReplyTo Info sections, use the **enable** check box in the top left corner of each section. To configure the step to be a publish step, a subscribe step, or both, use these check boxes. You can also select to include a JMS reply to component in the step.

When you finish configuring the test step, click Test in the Error Handling and Test section to test the configuration settings.

### Server Connection Info

Enter the connection information specific to Rendezvous information in the Server Connection Info area.

The following parameters are available for the system under test:

#### **Service, Network, and Daemon**

These parameters enable connection to the RV network that you want to communicate on.

#### **Client Mode**

Select either the Rendezvous Native client or Java Client mode. Usually you use the more versatile client mode.

To simplify changing the system under test, parameterize these values with properties from your configuration.

### Publisher Info

To set up the ability to send messages, select the enable check box.

Complete the following fields:

### Subject

The name of the subject to use. You can define your own subjects. A valid subject name is: **queue.sample**. An invalid subject name is: **queue.....My\_Samples** (null element) or **.My.Queue..** (three null elements).

### Message

Select the type of message you are sending. The supported types are Empty, Text, Object, Bytes, Message, and Mapped (Extended).

### Send Field

RV messages are actually maps of fields and values. This field is used to enable quick single field messages. When you enter a value here, the Send Message data is put into the value of a field with this name. This value is overridden with Mapped (Extended) type messages as they let you do multiple fields and values in a single message.

### Enable Inbox Type

To enable the Inbox Timeout and Enable sendReply fields, select this check box.

### Enable sendReply

To specify an Inbox Timeout or to enable sendReply functionality for the publisher, select the Enable Inbox Type.

## Certified Transport Info

To provide transport information, select the enable check box.

### Sender Name

The name that is the correspondent name of the CM transport.

### Advisory Subject

Rendezvous software constructs the subject names of system advisory messages using this template: **\_RV.class.SYSTEM.name**. Rendezvous certified message delivery software constructs the subject names of advisory messages using these templates: **\_RV.class.RVCM.category.condition.subject** and **\_RV.class.RVCM.category.condition.name**. Distributed queue software constructs the subject names of advisory messages using this template: **\_RV.class.RVCM.category.role.condition.name**. Rendezvous fault tolerance software constructs the subject names of advisory messages using this template: **\_RV.class.RVFT.name.group**.

### Time Limit

The time limit in which a message exists.

**Subscriber Info**

Selecting the **enable** box turns on the subscriber function and lets you set up the ability to receive messages.

Complete the following fields:

**Subject**

The name of the subject to use. You can define your own subjects.

**Timeout (secs)**

Indicates the number of seconds before DevTest interrupts waiting for a message. For no timeout, leave this field blank.

**Async Key**

Enter the value that is necessary to identify asynchronous messages. This value is only required in asynchronous mode. This value is used in a subsequent Message Consumer step to retrieve asynchronous messages.

**ReplyTo Info**

To set up a destination subject, select the **enable** check box.

If your application requires a destination, it is set up in this section.

Enter the following parameter:

**Subject**

The name of the subject to use.

**Error Handling and Test**

If an error occurs, the Error Handling and Test section lets you redirect to a step.

**If Environment Error**

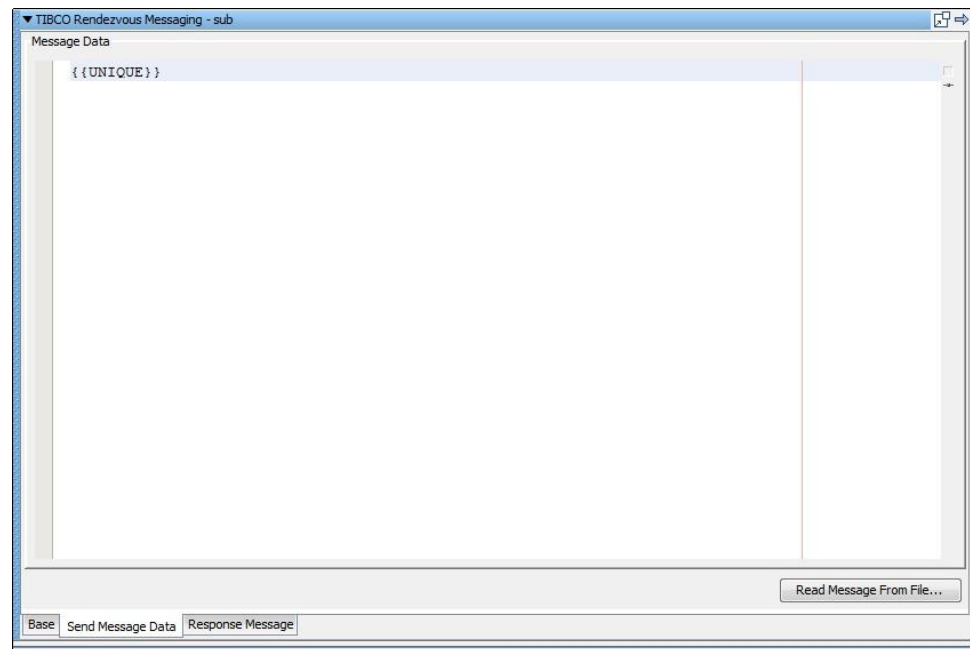
Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

Click Test to test your step configuration settings.

## Send Message Data

If your step is configured to publish, compose the message on this tab. The Send Message Data tab view in the following example shows a text message.



This example shows an XML fragment with properties being used. You can type the text, or you can click Read Message From File to read from a file. The text can also be stored in a property, in which case you would place the property in the editor: for example, LISA\_PROP.

Notice that properties are used in the message XML allowing the message to be created dynamically during the test run.

## Response Message Tab

If your step is configured to subscribe, the response is shown. For more information, see [JMS Messaging \(JNDI\)](#) (see page 286).

## TIBCO EMS Messaging

The TIBCO EMS Messaging step lets you send and receive messages from topics and queues. You can also receive, change, and forward an existing message.

All the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended) are supported.

The default TIBCO EMS Messaging step names uses the following convention: *EMS queueName publish*. If there is not a publish queue name, the default step name is *EMS queueName subscribe*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see *Third-Party File Requirements in Administering*.

**Parameter Requirements:** This step requires the connection parameters and the subject names that are used in the application under test. The default is the `TIBCO_SERVER` property in the JNDI Server URL. This property must be added to your configuration if you plan to use it. Your environment may require other parameters. Get these from the application developers.

For more detailed information about parameters and fields, see [JMS Messaging \(JNDI\)](#) (see page 286).

## TIBCO Direct JMS

The TIBCO Direct JMS step lets you send messages and receive messages from topics and queues without using the JNDI libraries. You can also receive, change, and forward an existing message.

All the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended) are supported.

The TIBCO Direct JMS step is configured using a single editor regardless of the messaging requirements. Input options vary on the messaging requirements. The editor only allows valid configurations. Enabling some features can make other unavailable.

The default TIBCO Direct JMS step name uses the following convention: *EMS queueName publish*. If there is not a publish queue name, the default step name is *EMS queueName subscribe*. If another step also uses the default step name, DevTest appends a number to the step name. You can change step names at any time.

**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements in *Administering*.

**Parameter Requirements:** This step requires the connection parameters and the subject names that are used in the application under test. DevTest, by default, uses the TIBCO\_SERVER property in the JNDI Server URL. This property must be added to your configuration if you plan to use it. Your environment may require other parameters. Get these from the application developers.

For more detailed information about parameters and fields, see [JMS Messaging \(JNDI\)](#) (see page 286).

## Sonic Steps

**The following steps are available:**

[SonicMQ Messaging \(Native\)](#) (see page 344)

[SonicMQ Messaging \(JNDI\)](#) (see page 345)

**More information:**

[Web Service Execution \(XML\) Step](#) (see page 186)

[Web-Raw SOAP Request](#) (see page 229)

[SQL Database Execution \(JDBC\)](#) (see page 253)

[Read a File \(Disk URL or Classpath\)](#) (see page 283)

[External - FTP Step](#) (see page 284)

[Message Consumer](#) (see page 315)

## SonicMQ Messaging (Native)

The SonicMQ Messaging (Native) step lets you send and receive messages from topics and queues using native Sonic protocol. You can also receive, change, and forward an existing message.

SonicMQ Messaging (Native) supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

The SonicMQ Messaging (Native) step is configured using a single editor regardless of the messaging requirements. The input options vary on the messaging requirements. The editor only allows valid configurations, so when you enable some features others could become inactive.

**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see *Third-Party File Requirements in Administering*.

**Parameter Requirements:** This step requires the connection parameters and the subject names that are used in the application under test.

The following parameters are required for the system under test.

- Broker Host
- Broker Port
- User
- Password

There could be other parameters that are required, depending on your environment. Get these parameters from the developers of the application.

**Default Step Names:** The default SonicMQ Messaging (Native) step name uses the following convention: *Sonic queueName publish*. If there is not a publish queue name, the default step name is *Sonic queueName subscribe*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

For more detailed information about parameters and fields, see [JMS Messaging \(JNDI\)](#) (see page 286).



## SonicMQ Messaging (JNDI)

SonicMQ Messaging (JNDI) supports all the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended).

The SonicMQ Messaging (JNDI) step lets you send and receive messages from topics and queues. You can also receive, modify, and forward an existing message.

The SonicMQ Messaging (JNDI) step is configured using a single DevTest editor regardless of the messaging requirements. The input options vary on the messaging requirements. The editor only allows valid configurations, so when you enable some features others could become inactive.

**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see *Third-Party File Requirements in Administering*.

**Parameter Requirements:** You need the connection parameters and the subject names that are used in the application under test. DevTest, by default, uses the SONICMQ\_SERVER property in the JNDI Server URL. This property must be added to your configuration if you plan to use it. There could be other parameters that are required, depending on your environment. Get these parameters from the developers of the application.

**Default Step Names:** The SonicMQ Messaging (JNDI) step has a default name using the following convention: *Sonic queueName publish*. If there is not a publish queue name, the default step name is *Sonic queueName subscribe*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

For more detailed information about parameters and fields, see [JMS Messaging \(JNDI\)](#) (see page 286).

## webMethods Steps

**The following steps are available:**

[webMethods Broker](#) (see page 347)

[webMethods Integration Server Services](#) (see page 352)

**More information:**

[HTTP-HTML Request Step](#) (see page 176)

[REST Step](#) (see page 186)

[Web Service Execution \(XML\) Step](#) (see page 186)

[Web-Raw SOAP Request](#) (see page 229)

[SQL Database Execution \(JDBC\)](#) (see page 253)

[Read a File \(Disk URL or Classpath\)](#) (see page 283)

[External - FTP Step](#) (see page 284)

[Message Consumer](#) (see page 315)

## webMethods Broker

webMethods Broker supports Mapped (Extended) messages that create Broker Events.

The webMethods Broker step lets you send and receive messages from the Broker. You can also receive, change, and forward existing Broker Events/ Messages.

The webMethods Broker step is configured using a single editor, regardless of the messaging requirements. The input options vary on the messaging requirements. The step editor only allows valid configurations, so when you enable some features others can become inactive.

**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements in *Administering*.

**Parameter Requirements:** This step requires the connection parameters and the subject names that are used in the application under test. The following sections describe the parameters that you require. Other parameters could be required, depending on your environment. Get these parameters from the developers of the application.

The screenshot shows the 'webMethods Broker - webM' configuration window. It is divided into several sections:

- Server Connection Info:** Fields for Broker Host (localhost), Host Port (6849), Broker Name, Client ID, Client Group, and App Name (LISA).
- Subscriber Info:** Includes an 'enable' checkbox, docType, Timeout (secs) (30), Async Key, and an 'Auto convert to' section with radio buttons for 'string' (selected) and 'xml'.
- ReplyTo Info:** Includes an 'enable' checkbox and a Name field.
- Publisher Info:** Includes an 'enable' checkbox, docType, Message (webMethods Broker), Force Document Pre-fill, Deliver Enabled, Deliver Client ID, and Envelope Tag.
- Error Handling and Test:** A section with 'If environment error:' set to 'Abort the Test' and a 'Test...' button.

At the bottom, there is a tabbed interface with 'Base' selected, and other tabs for 'Send Message Data' and 'Response Message'.

The messaging step editor for webMethods Broker includes three tabs at the bottom of the page:

- The Base tab is where you define your connection and messaging parameters.
- The Send Message Data tab is where you create your message content.
- The Response Message tab is where your response messages are posted.

## Base Tab

The Base tab view that the previous graphic shows is divided into the following sections:

- Server Connection Info
- Subscriber Info
- Publisher Info
- ReplyTo Info
- Error Handling and Test

The Server Connection Info and Error Handling and Test sections are always active. To enable or disable the Subscriber Info, Publisher Info, and ReplyTo Info sections, use the enable check box in the top left corner of each section. Using these check boxes, you can configure the step to be a publish step, a subscribe step, or both. You can also include a **replyto** component in the step.

When you have configured your test step, click Test in the Error Handling and Test section to test your configuration settings.

### Server Connection Info

In the Server Connection Info section, enter the connection information specific to webMethods Broker.

The four parameters must be available to you for the system under test.

#### **Broker Host**

#### **Host Port**

#### **Broker Name**

#### **Client ID**

#### **Client Group**

This value is the Client group that is able to see the Broker destinations to use.

#### **App Name**

Specify the application using the Broker here. This parameter is optional and mostly used in server logs for debugging. The default is "DevTest". Using this parameter is a good practice, but if you must use something else for application logic you can.

To simplify changing the system under test, parameterize these values with properties from your configuration.

### Publisher Info

To set up the ability to send (publish) messages, select the enable check box.

Enter the following parameters:

#### **docType**

Enter the name of the docType to use.

#### **Message**

Select the type of message you are sending from the pull-down menu. The supported messages are: webMethods Broker, Object, Message, and Mapped (Extended).

#### **Force Document Pre-fill**

The selected docType is inspected and the message with the required fields is preloaded. This check box lets you change fields and decide whether to re-add any missing fields. DevTest does not write over existing fields with the same name. This property is only a design time effect and does nothing at test run time.

#### **Deliver Enabled**

Select to enable the Deliver Client ID field.

#### **Deliver Client ID**

The broker Client Identification for the connection. If the value is null, the broker generates an identifier automatically. An error can be returned if the value is already in use by another connection.

#### **Envelope Tag**

This parameter lets you set the env.tag property on a broker event message.

### Subscriber Info

To set up to enable the ability to receive (subscribe to) messages, select the enable check box.

Enter the following parameters:

#### **docType**

Enter the name of the docType to use.

#### **Timeout (secs)**

Indicates the number of seconds before DevTest interrupts waiting for a message. For no timeout, leave this field blank.

#### **Async Key**

Enter the value that is necessary to identify asynchronous messages. This value is only required in asynchronous mode. This value is used in a subsequent Message Consumer step to retrieve asynchronous messages.

**Auto convert to**

To return a string representation of the payload, enter a string that calls the toString() function on the payload object; xml returns the payload in XML format.


**ReplyTo Info**

To set up a destination queue/topic, select the enable check box.

If your application requires a destination, it is set up in this section.

Enter the following parameters:

**Name**

The name of the topic or queue. Use Search  to browse the JNDI server for the topic or queue name.

**Error Handling and Test**

If an error occurs, the Error Handling and Test option lets you redirect to a step.

**If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

Click Test to test your step configuration settings.

## Send Message Data Tab

This tab is where you compose your message, if your step is configured to publish.

## Response Message Tab

If your step is configured to subscribe, your response is shown here. For more information, see [JMS Messaging \(JNDI\)](#) (see page 286).

### Default Step Names

The default webMethods Broker step name uses the following convention: *webM queueName publish*. If there is not a publish queue name, the default step name is *webM queueName subscribe*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

### webMethods Integration Server Services

The webMethods Integration Server Services step lets you execute Integration Server services through the native Java APIs. This is done using IData objects so it works with services not exposed through HTTP transports.

**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see *Third-Party File Requirements in Administering*.

**Parameter Requirements:** This step requires the connection parameters and the subject names that are used in the application under test. The following sections describe the parameters that you need. Other parameters could be required, depending on your environment. Get these parameters from the application developers.



## Base Tab: Server Connection Info

Enter the following parameters:

**Host**

The host name.

**User**

The userid.

**Password**

The password.

**Package**

The package in which the service is located.

**Service**

The name of the actual service that you want to call.

**Input Type**

Select the input type from Property, IData Object, or Force IData Pre-fill.

**Output Type**

Select the output type from XML or IData Object.

**If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

Click Execute to connect. You see an object response. To pull the payload or other properties from the response, which is itself an IData object, export it to a Java Execution step. To complete this task, create a Java Step in DevTest and load from a property, specifying the step name pattern for a last response. Use the `lisa.<stepName>.rsp` property.

▼ webMethods Integration Server Services - IntegrationServerInvoker~1

Host: 192.168.11.158

User: Administrator

Password: ●●●●●●

Package: iTKOTraining.flows

Service: addUser

Input Type: ☐ Property ☒ IData Object ☐ Force IData Pre-fill

Output Type: ☐ XML ☒ IData Object

If environment error: Abort the Test

Execute

Base Pipeline Input Pipeline Output

## Pipeline Input Tab

webMethods Integration Server Services - IntegrationServerInvoker~1

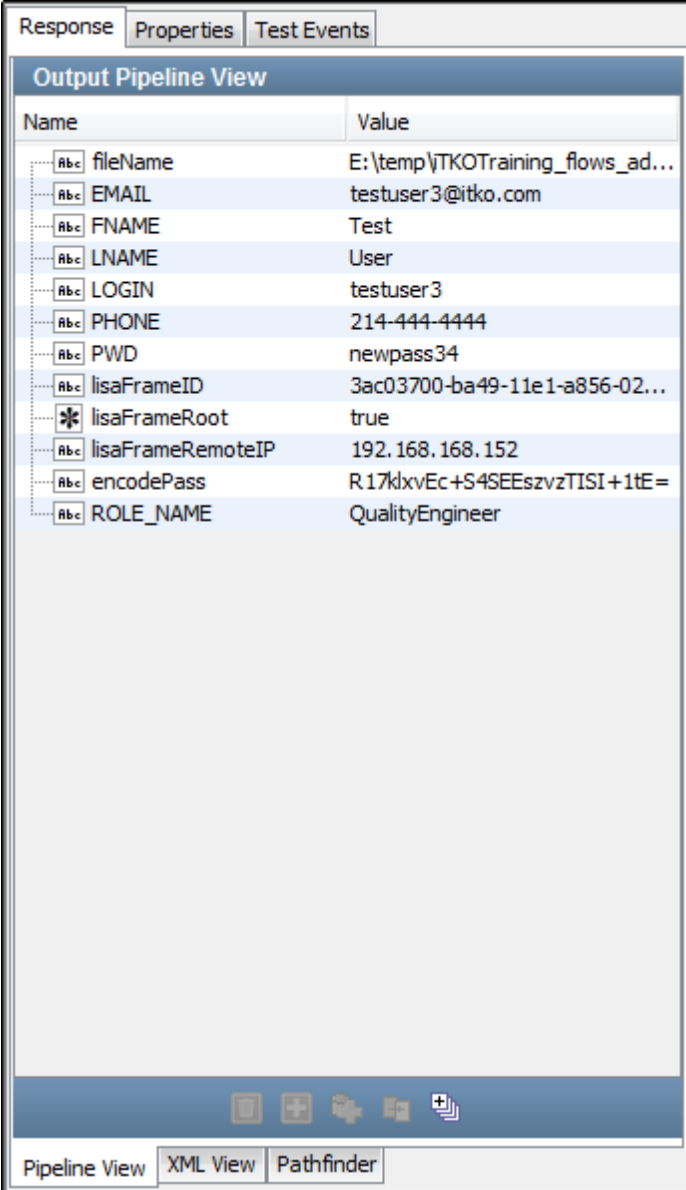
### Input Pipeline View

Name	Value
EMAIL	testuser3@itko.com
FNAME	Test
LNAME	User
LOGIN	testuser3
PHONE	214-444-4444
PWD	pass
ROLE_NAME	QualityEngineer

Base Pipeline Input Pipeline Output

Load From File

## Pipeline Output Tab



Name	Value
fileName	E:\temp\TKOTraining_flows_ad...
EMAIL	testuser3@itko.com
FNAME	Test
LNAME	User
LOGIN	testuser3
PHONE	214-444-4444
PWD	newpass34
lisaFrameID	3ac03700-ba49-11e1-a856-02...
* lisaFrameRoot	true
lisaFrameRemoteIP	192.168.168.152
encodePass	R.17klxvEc+S4SEeszvzTISI+1tE=
ROLE_NAME	QualityEngineer

## Default Step Names

The default webMethods Integration Server Services step name uses the following convention: *IntegrationServerInvoker ServiceName@HostName*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

## IBM Steps

**The following steps are available:**

[IBM WebSphere MQ Step](#) (see page 357)

**More information:**

[Message Consumer](#) (see page 315)

## IBM WebSphere MQ Step

The IBM WebSphere MQ step lets you send messages to, and receive messages from, topics and queues. You can also receive, modify, and forward an existing message.

All the common message types including Empty, Text, Object, Bytes, Message, and Mapped (Extended) are supported.

The IBM WebSphere MQ step is configured using a single editor regardless of the messaging requirements. The input options vary on the messaging requirements. The editor only allows valid configurations, so when you enable some features, others could become inactive.

The default IBM WebSphere MQ step name uses the following convention: *MQ queueename publish*. If there is not a publish queue name, the default step name is *MQ queueename subscribe*. If another step uses the default step name, DevTest appends a number to this step name to keep it unique. You can change step names at any time.

**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see *Third-Party File Requirements in Administering*.

**Parameter Requirements:** You must have the connection parameters for your system under test. The following sections describe the required parameters.

The editor for the IBM WebSphere MQ step contains the following tabs:

- The Base tab is where you define the connection and messaging parameters.
- The Selector Query tab lets you specify a selector query to be run when listening for a message on a queue.
- The Send Message Data tab is where you create the message content.
- The Response Message tab is where the response messages are posted.

**Note:** This topic describes the Base tab. For information about the other tabs, see [JMS Messaging \(JNDI\)](#) (see page 286).

## IBM WebSphere MQ: Base Tab

The following graphic shows the Base tab. The tab is divided into the following sections:

- Server Connection Info
- Subscriber Info
- ReplyTo Info
- Publisher Info
- Error Handling and Test

The screenshot shows the 'IBM Websphere MQ - MQ' configuration window. It is divided into four main sections: 'Server Connection Info', 'Subscriber Info', 'ReplyTo Info', and 'Publisher Info'. The 'Server Connection Info' section includes fields for Host Name, TCP/IP Port, Channel, Queue Manager, CCID, User, Password, and Client Mode (set to JMS). The 'Subscriber Info' section includes an 'enable' checkbox, Name, Type (set to Queue), Timeout (secs) (set to 30), Queue Model, Async Key, Durable Session Key, Session Mode (set to Auto Acknowledge), and checkboxes for 'use temporary queue/topic', 'make payload last response' (checked), and 'use correlation ID for subscri'. The 'ReplyTo Info' section includes an 'enable' checkbox, Name, Type (set to Queue), and Queue Manager. The 'Publisher Info' section includes an 'enable' checkbox, Name, Type (set to Queue), Message (set to Empty), and Alt QManager. At the bottom, there are buttons for 'Advanced', 'Stop All', 'Share Sessions', 'Subscribe Properties', and 'Message Properties'. A tab bar at the very bottom shows 'Base', 'Selector Query', 'Send Message Data', and 'Response Message'.

To enable and disable the Subscriber Info, Publisher Info, and ReplyTo Info sections, use the enable check box in the top left corner of each section. Using these check boxes, you can configure the step to be a publish step, a subscribe step, or both. You can also include a reply to component in the step.

When you finish configuring the test step, click Test in the Error Handling and Test section to test the configuration settings.

### Server Connection Info

To connect to WebSphere MQ, enter the following information:

**Host Name**

The name of the host.

**TCP/IP Port**

The port for a client connection.

**Channel**

A connection property that is used for routing and management in the message bus.

**Queue Manager**

A connection property that is used for routing and management.

**CCID**

Optional for connections and only applies if character transformation must occur between the client (DevTest) and server.

**User Name**

The login user name, if applicable.

**Password**

The login password, if applicable.

**Client Mode**

Lets you select how you want to interact with the WebSphere MQ server.

- JMS: A pure Java implementation that is based on the JMS specification. We recommend that you use the JMS Transport Protocol instead of MQ for this implementation.
- Native Client: A pure Java implementation using IBM-specific APIs.
- Bindings: Requires access to the native libraries from a WebSphere MQ client installation. Verify that these libraries are accessible by the DevTest application run time. In most cases, having these libraries available in the PATH environment works.

**Share Sessions**

Select to specify sharing everything in MQ Native Mode, including the connection.


### Publisher Info

To set up the ability to send (publish) messages, select the enable check box.


To execute a commit when the message is sent, select the use transaction check box.

Enter the following parameters:

**Name**

The name of the topic or queue. Use Search  to browse the JNDI server for the topic or queue name.

**Type**

Select whether you are using a topic or queue. To see what messages are waiting to be consumed from a queue (only), use Browse  to the right of this field.

**Message**

Select the type of message you are sending. The supported types are Empty, Text, Object, Bytes, Message, and Mapped (Extended).

**Alt QManager**

The queue manager that hosts the publish queue, if it is different from the queue manager to which you are connecting.

**Message Properties**


Lets you set publish properties on the message. The list of properties changes depending on the client mode. For more information, see the WebSphere MQ documentation and the *JMS and MQ Message Properties* knowledge base article. If the client mode is JMS, then you can add custom message properties.

**Subscriber Info**

To set up to enable the ability to receive (subscribe to) messages, select the enable check box.

Enter the following parameters:

**Name**

The name of the topic or queue. Use Search  to browse the JNDI server for the topic or queue name.

**Type**

Select whether you are using a topic or queue, and whether to listen in synchronous or asynchronous mode. For asynchronous mode, you also must have an entry in the Async key field. To see what messages are waiting to be consumed from a queue (only), click Browse, to the right of this field.

**Timeout (secs)**



Indicates the number of seconds before DevTest interrupts waiting for a message. For no timeout, leave this field blank.

**Queue Model**

MQ requires this value to create temporary destinations. The queue model is configured on the MQ server, and it is only active when use temporary queue/topic is checked. In this case, the ReplyTo Info section is disabled.

**Async Key**

Enter the value that is necessary to identify asynchronous messages. This value is only required in asynchronous mode. This value is used in a subsequent Message Consumer step to retrieve asynchronous messages.

**Durable Session Key**

By entering a name here you are requesting a durable session. You are also providing a key for that session. A durable session lets you receive all of your messages from a topic even if you log out, and then you log in again.

**Session Mode**

Select the appropriate mode from the available options by clicking the drop-down list. Options are: Auto Acknowledge, Client Acknowledge, Use Transaction, Auto (Duplicates Okay).

- Auto Acknowledge: The session automatically acknowledges the receipt of a message by a client.
- Client Acknowledge: This option instructs the client to acknowledge messages programmatically.
- Use Transaction: To execute a commit when a message is received.
- Auto (Duplicates Okay): This option instructs the session to acknowledge the delivery of messages.

**Use temporary queue/topic**

If you want JMS.provider to set up a temporary queue/topic on your behalf, select the use temporary queue/topic check box. When a temporary queue/topic is used, the JMS ReplyTo parameter of the message you send to the temporary queue/topic is automatically set. The temporary queue/topic feature must always be used with a publisher so that a reply can be sent. If you use a temporary queue/topic, the ReplyTo section is disabled.

**Make payload last response**

To make the payload as the last response, select this option.

**Use correlation ID for subscribe**

Enables filtering of incoming messages using the values in Subscribe Properties.

**Subscribe Properties**

Lets you set subscribe properties on the message. For more information, see the WebSphere MQ documentation and the JMS and MQ Message Properties [knowledge base article](#).


### ReplyTo Info

To set up a destination queue/topic, select the enable check box.

If your application requires a destination, it is set up in this section.


Enter the following parameters:

#### Name

The name of the topic or queue. Use Search  to browse the JNDI server for the topic or queue name.

#### Type

Select whether you are using a topic or queue. To see what messages are

waiting to be consumed from a queue (only), use Browse  to the right of this field.

#### Queue Manager

Allows the ReplyTo to be on a different Queue Manager than the Publisher (in this step).

### Error Handling and Test

If an error occurs, the Error Handling and Test section lets you redirect to a step.

#### If Environment Error

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

Click Test to test your step configuration settings.

## SAP Steps

**The following steps are available:**

[SAP RFC Execution](#) (see page 364)

[SAP IDoc Sender](#) (see page 367)

[SAP IDoc Status Retriever](#) (see page 370)

## SAP RFC Execution

The SAP RFC Execution step lets you connect to an SAP system to execute an RFC (Remote Function Call).

**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see *Third-Party File Requirements in Administering*.

**To create an SAP RFC Execution step:**

1. Select a Destination [asset](#) (see page 64) that includes the connection information for either an application server or a message server.
2. If you are using a message server:

In Windows, add the following line to the end of the  
C:\Windows\System32\drivers\etc\services file: *sapmsCR2 3600/tcp*.

In Linux, add the following line to the end of /etc/services: *sapmsCR2 3600/tcp*.

▼ SAP RFC Execution - RFC\_READ\_TABLE

Destination: SAP Application Server

RFC name filter:

RFC name: RFC\_READ\_TABLE

Import parameters:

Name	Description	Value
DELIMITER	Sign for indicating fiel...	
NO_DATA	If <> SPACE, only FI...	
QUERY_TABLE	Table read	EDIDS
ROWCOUNT		
ROWSKIPS		

Table parameters:

Name	Description	Value
DATA	Data read (out)	
WA	Character field lengt...	
FIELDS	Names (in) and struc...	
FIELDNAME	Field Name	DOCNUM
OFFSET	Offset of a field	
LENGTH	Length (No. of Chara...	
TYPE	ABAP data type (C,D...	
FIELDTEXT	Short Description of ...	
OPTIONS	Selection entries, "W...	
TEXT	Text line of a message	TID = '8DCA4723009...

- Enter the following parameters in the SAP RFC Execution step editor. You can use properties for RFC input parameters.

#### RFC name filter

To filter the RFC function names, enter a filter value, which can include a wildcard character of \*. When you click the RFC name field drop-down arrow, DevTest requests the SAP system to retrieve the RFC names that the filter qualified, then populates the field.

After you select or enter an RFC function name, DevTest sends a request to the SAP system to retrieve the RFC function input parameters. Both Key and Description of each parameter are returned from the SAP system. Although the description is available for most parameters, not all of them have it. You can enter the parameter values before executing the RFC function.

**RFC name**

Field Description

**Import Parameters**

**Name**

The name of a parameter

**Description**

The description of a parameter

**Value**

The value of a parameter

Each table cell in the value column is a drop-down list of the available properties.

**Table Parameters**

**Name**

The name of a parameter

**Description**

The description of a parameter

**Value**

The value of a parameter

Each table cell in the value column is a drop-down with the available properties to select from.

**If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

4. Click Test to populate the Response tab, which contains subtabs for the output for XML and DOM Tree.

## SAP IDoc Sender

The SAP IDoc Sender step lets you connect to an SAP system to send SAP IDocs.

**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see *Third-Party File Requirements in Administering*.

**To create an SAP IDoc Sender step:**

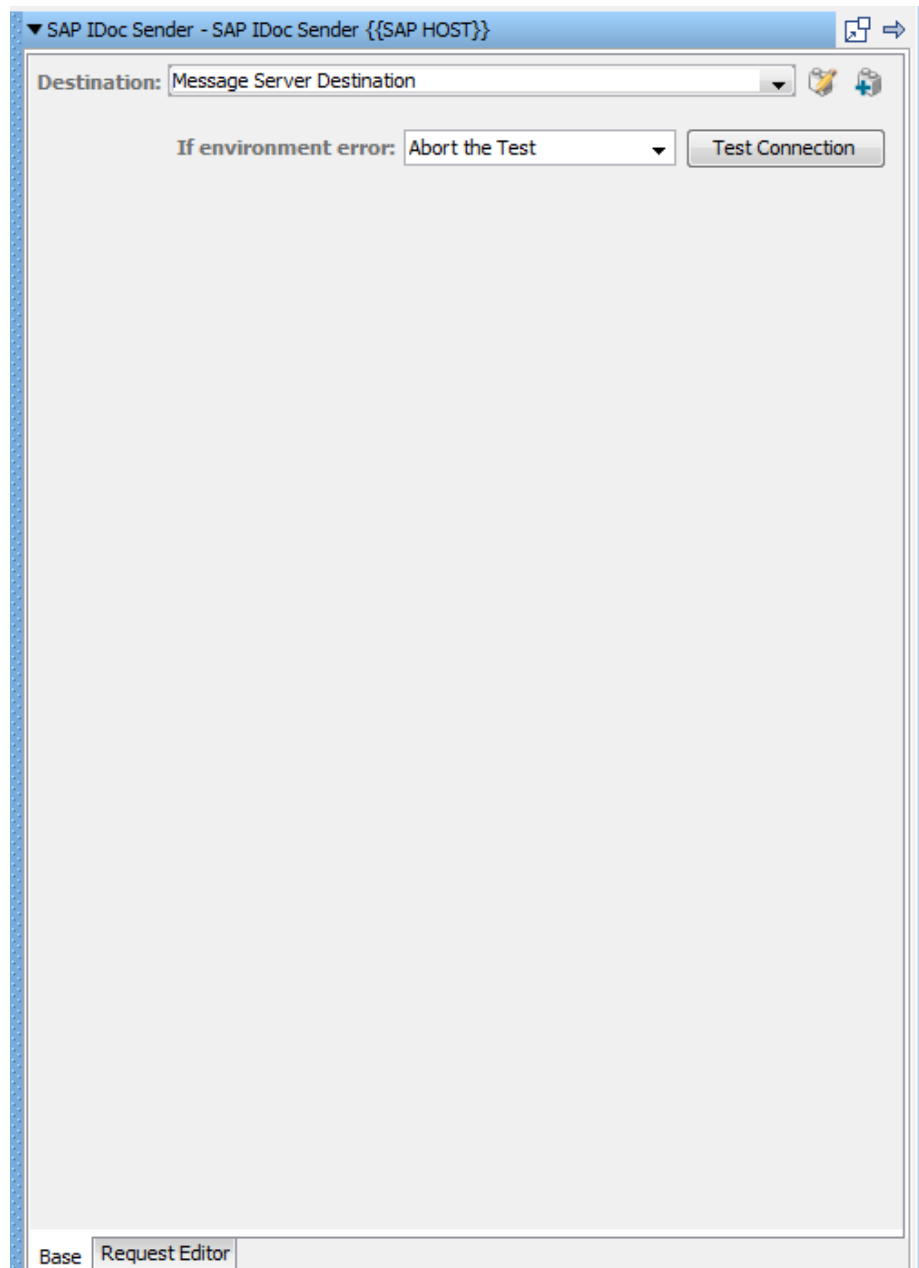
1. Select a Destination [asset](#) (see page 64) that includes the connection information for either an application server or a message server.

2. If you are using a message server:

In Windows, add the following line to the end of the

C:\Windows\System32\drivers\etc\services file: *sapmsCR2 3600/tcp*.

In Linux, add the following line to the end of */etc/services*: *sapmsCR2 3600/tcp*.





3. Enter the following parameter in the SAP IDoc Sender step editor.

**If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

4. To test the connection to the SAP server, click Test Connection.
5. To display the file locator, click Read IDoc From File. Navigate to the location of your IDoc and select it.

SAP IDocs are supported in XML and raw text formats.

## SAP IDoc Status Retriever

The SAP IDoc Status Retriever step lets you connect to an SAP system and poll SAP IDocs status periodically until it completes or a specified interval ends.

**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see *Third-Party File Requirements in Administering*.

### To create an SAP IDoc Status Retriever step:

1. Select a Destination [asset](#) (see page 64) that includes the connection information for either an application server or a message server.
2. If you are using a message server:

In Windows, add the following line to the end of the  
C:\Windows\System32\drivers\etc\services file: *sapmsCR2 3600/tcp*.

In Linux, add the following line to the end of */etc/services*: *sapmsCR2 3600/tcp*.

The screenshot shows the 'SAP IDoc Status Retriever' step editor. The title bar reads '▼ SAP IDoc Status Retriever - SAP IDoc Status Retriever {{SAP HOST}}'. The interface includes several configuration fields: 'Destination' is a dropdown menu set to 'Application Server Destination'; 'Timeout (secs)' is a dropdown menu set to '600'; 'Polling Interval (secs)' is a dropdown menu set to '30'; 'SAP Transaction ID' is a dropdown menu set to '{{Isa.SAP\_TRANSACTION\_ID}}'. Below these fields are two more dropdown menus: 'If environment error:' set to 'Abort the Test' and 'If timeout:' set to 'Abort the Test'. A 'Test Connection' button is located to the right of the 'If environment error:' dropdown. At the bottom left of the editor, there is a 'Base' tab.

3. Enter the following parameters in the SAP IDoc Status Retriever step editor. You can use properties for SAP input parameters.

**Timeout (secs)**

The duration of polling operation for SAP IDoc status

**Polling Interval (secs)**

The frequency of polling operation for SAP IDoc status

For example, with the default values, the SAP IDoc Status Retriever step polls the IDoc status every 30 seconds for a total duration of 10 minutes. If the IDoc is completed any time in the duration, the SAP IDoc Status Retriever step stops immediately, without waiting for the entire 10 minutes.

**SAP Transaction ID**

An alphanumeric transaction ID or a DevTest property. If an SAP IDoc Sender step is used before this IDoc Status Retriever step, use the default property {{lisa.SAP\_TRANSACTION\_ID}}.

### **If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

### **If timeout**

Select the step to be executed or action to take if there is a timeout.

4. To validate the connection to the SAP server, click Test Connection.

## Selenium Integration Steps

The Selenium steps let you import test scripts for web-based user interfaces from Selenium Builder into CA Application Test. You can use the exported JSON test script to build test steps for testing a user interface.

To create a Selenium step, complete the following tasks:

- [Create and Export a Selenium Builder Recording](#) (see page 374)
- [Import the Selenium Builder JSON into CA Application Test](#) (see page 375)

**Note:** Browser support for recording Selenium Integration tests is limited to Firefox. After you import these steps to CA Application Test, you can run the test cases on Google Chrome, Mozilla Firefox 24 or later, or Internet Explorer 8.0 or later.

You can also export a CA Application Test Selenium test case to a JSON script. For more information, see:

- [Export a Selenium Test Case to a JSON Script](#) (see page 378)

## Set Advanced Options on the Selenium Web Driver

You can set advanced options on the Selenium web driver to meet your requirements.

### Follow these steps:

1. Locate the **selenium-capabilities.conf** in the Data directory of the examples project in the LISA\_HOME directory.

This file contains sample parameters to configure the Selenium web driver. Refer to the URLs contained in the file for the most up-to-date information from Selenium about these parameters.

The file is a plain text file. Use any text editor, such as Notepad on Windows or vi on UNIX and Linux, to view and edit the file.

2. Enter the values for any parameters that require an update.
3. To update parameters for the Selenium steps, specify the location of the **selenium-capabilities.conf** in local.properties or in the active project config file, using the **selenium.WebDriver.DesiredCapabilities.filePath** property.

To use the default location of the file, copy it to the Data directory of your project.

## Create and Export a Selenium Builder Recording

Selenium Builder is a Firefox add-on that lets you record actions in a web-based user interface and create Selenium tests. The first step in creating a Selenium test step in CA Application Test is to create a Selenium Builder test script.


If Selenium Builder is not installed, download and install it from the following URL:  
<http://sebuilder.github.io/se-builder/>.

**Note:** We recommend that you turn off the automatic upgrade option:

- a. Click Firefox, Add Ons on the main Firefox menu.  
The Add-ons Manager opens.
- b. Click Extensions and double-click Selenium Builder.
- c. Select the Off option for Automatic Updates.

**Follow these steps:**

1. Open your Firefox browser.
2. Click Firefox, Web Developer, Launch Selenium Builder on the main Firefox menu.  
The Selenium Builder page opens.

**Note:** If you have the Add-on Bar enabled, you can click  in the bottom right corner of the Firefox browser window.

3. Enter the URL of the web application you want to test in the **Start recording at** field.
4. Click Selenium 2.

**Note:** Selenium 1 does not support the required ability to export to JSON.

5. Navigate to the web application and perform the actions you want to test.
6. When you are done, return to the Selenium Builder page and click Stop Recording.

**Note:** For more information about creating Selenium Builder tests, see the Selenium Builder documentation at <http://sebuilder.github.io/se-builder/>.

7. Click File, Export.  
The Choose Export Format dialog opens.
8. Click Save as JSON.
9. Browse to the directory where you want to save the JSON file.
10. Enter a name for the JSON file and click Save.
11. To verify that the script is valid, run the script in Selenium Builder.
12. Click Run, Run test locally.



The results and any script errors appear.

## Import a Selenium Builder JSON into CA Application Test

Use the Selenium Integration test steps to import a [JSON test script that you created](#) (see page 374) in Selenium Builder to create a test step or script that is based on that JSON file.

### Follow these steps:

1. Open an existing or create a new test case.
2. Perform one of the following actions:

- Click Selenium .
- Click Add Step , Selenium, Selenium Import/Export.

The Import tab of the Selenium Integration page opens.

3. Click Browse next to the Input JSON Script field and browse to the location of the JSON test script you want to import.
4. Select one of the following output options:

#### Selenium Script

Imports the complete JSON test script in one test step. This option also supports Selenium Builder suites.

#### Selenium Step

Divides the JSON script to create a separate test step for each action in the script. This option does not support Selenium Builder suites.

5. Click Build.

A new step (Selenium Step) or new steps (Selenium Script) appear in your test case.

The Selenium Integration Import dialog opens. This page notifies you of any warnings or errors that occurred during the import. The page also shows you the number of steps that the import process generated.

6. Click Close.
7. Double-click each step to view the JSON script in the Selenium Script or Selenium Step tab of the Elements tree in the right panel.
8. Complete the following fields for each step:

#### Alert Behavior (Optional)

The Selenium step can provide "inline" alert handling. The fields in the Alert Behavior area work in parallel with the value of the **unexpectedAlertBehaviour** parameter, which is defined in the file that is named by the **selenium.WebDriver.DesiredCapabilities.filePath** property.

These fields let you specify how the test step should react to a modal alert dialog displayed in the web app.

### Alert Action (Optional)

Defines the action to take in response to a modal alert dialog.

**Values:** Accept, Dismiss, Answer

### Input Text (Optional)

Defines text to input when the Alert Action is Answer. The step inputs the text that you put into this text box, then clicks OK.

### On Step Fail

Defines the action to take if a specific step in the test case fails. Select the step to execute (Go to:) or the action to take if the step fails. For more information, see [Configure Next Step and Generate Warnings and Errors](#).

**Note:** The JSON script supports variable substitution by property for any value that you input. You can also use properties with encrypted values, such as `{{password_enc}}`, to avoid exposing sensitive data.

The following Selenium Builder steps are mapped to your test case as described:

#### Store

The name/value pair becomes a standard property in your test case. The name is prefixed with "selenium" to distinguish it from other properties. For example, the following JSON definition becomes a new property with the name **selenium.window\_title**. The value is populated after the step runs.

```
{
  "type": "storeTitle",
  "variable": "window_title"
},
```

#### Verify

The verify step in Selenium Builder is used to validate user interface elements. If the validation fails, the state of the step is set to ERROR, but the test case execution flow continues to the next step. An associated DevTest error event (in red) is created for the failure.

#### Assertion

The assertion step in Selenium Builder validates user interface elements. If the validation fails, the state of the step is set to FAIL, and the test case execution flow stops. An associated DevTest error event (in red) is created for the failure.

#### saveScreenshot



If a `saveScreenshot` step in your script does not include a full path for saving the screen shot, DevTest tries to create the file under a `$LISA_HOME\tmp\selenium` directory. You can also use variables for the supplied file name. For example:

```
c:\testcase1\snapshot1-{{LISA_TEST_RUN_ID}}.png
```

or

```
c:\{{testCase}}\{{LISA_TEST_RUN_ID}}\snapshot1.png
```

If any part of the parent directory for the target file does not exist, that portion of the directory is automatically created. If the target file already exists, it is deleted before writing new data.

## Export a Test Case with Selenium Test Steps to a JSON Script

You can export a test case with Selenium steps to a JSON script that can be stored on the file system.

Before you export a Selenium test case, you must install the DevTest plugin in the Firefox Selenium Builder plugins directory.

After you export a test case, you can use Selenium Builder to rerecord web applications. You can then reimport the newly generated JSON from Selenium Builder to DevTest Workstation and merge the changes into an existing test case.

Load exported JSON scripts into Selenium Builder using the **Open a script exported by CA Application Test** menu item. This menu item is only available after you install the DevTest plugin.

After you load the JSON script into Selenium Builder, you can make updates using regular Selenium Builder functionalities, such as:

- Edit step
- Add step
- Delete step
- Change the execution order of steps

### To install the DevTest plugin:

1. Copy the entire **lisa** directory, and its contents, from [LISA\_HOME]\addons\sebuilder-plugin on the DevTest Workstation to the [Firefox profile]\SeBuilder\plugins directory. To find your Firefox profile:
  - a. For Firefox 3.6 and later, from Firefox, select Help, Troubleshooting information.
  - b. Under Application Basics:
    - On Windows and Linux, depending on the Firefox version, click Show Folder (Windows), Open Directory (Linux), or Open Containing Folder.
    - On OS x, click Show in Finder.


**Note:** The Firefox menu bar contains the File, Edit, View, History, Bookmarks, Tools, and Help menu items. On Windows, the menu bar may be hidden. To show a hidden menu bar temporarily, click the Alt key.
2. Copy the **lisa** directory to the SeBuilder/plugins directory.

### To export a Selenium test case:

1. Open an existing test case.

2. Perform one of the following actions:

- Click Selenium  .

- Click Add Step  , Selenium, Selenium Import/Export.

The Selenium Integration page opens.

3. Click the Export tab.
4. Click Browse next to the Output JSON Script field and browse to the location of the JSON test script you want to export, or enter a path and file name.
5. Click Save.

## LISA Virtual Service Environment Steps

The CA Service Virtualization test steps are described in Editing a VSM in *Using CA Service Virtualization*.

## CAI Steps

**The following step is included:**

[Execute Transaction Frame](#) (see page 380)

## Execute Transaction Frame

The Execute Transaction Frame step lets you run a transaction frame on a DevTest Java Agent.

This step is useful when you want to verify the functionality in a system under test, but you do not have a public access point. In addition, this step can be automatically included in baselines.

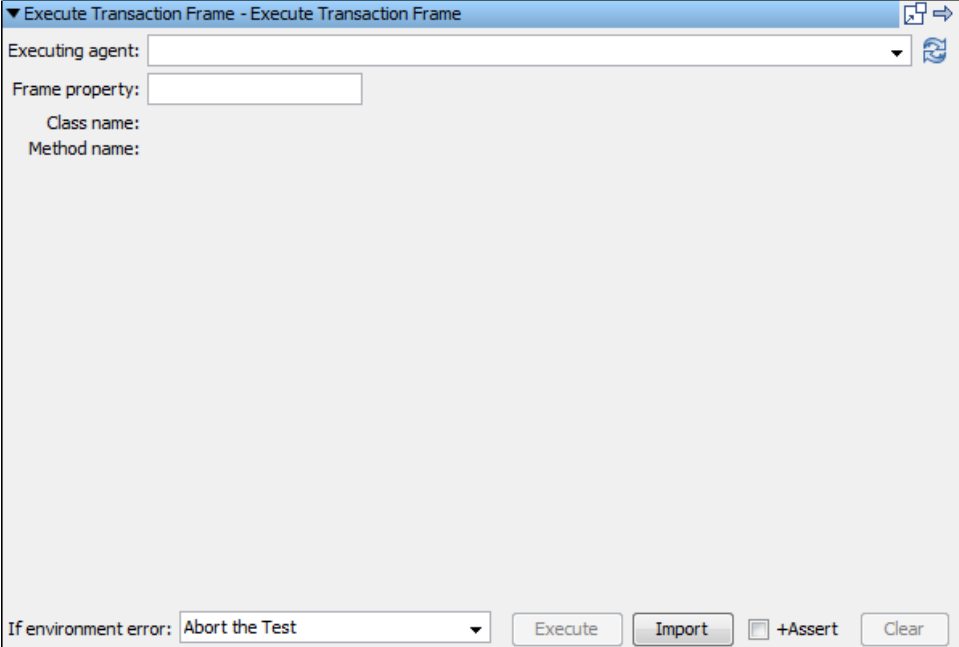
This step can be used on any type of transaction frame that the agent can capture.

If you manually add the step to a test case, do either of the following:

- Import a transaction
- Specify the property in which to find the frame

If you do both, DevTest examines the property first and uses the imported transaction only if the property does not refer to a frame, either as an actual frame or as an XML representation of it. The frame can be imported from an XML file or from a zip file. You can obtain frames by exporting from the CAI Console.

The following graphic shows the initial view of the step. To specify the frame, click Import. To have the import process create an assertion on the response for the imported frame and add it to the step automatically, select the Assert check box before you click Import.



The screenshot shows a dialog box titled "Execute Transaction Frame - Execute Transaction Frame". It contains the following fields and controls:

- Executing agent:** A dropdown menu with a refresh icon.
- Frame property:** A text input field.
- Class name:** A text input field.
- Method name:** A text input field.
- If environment error:** A dropdown menu currently set to "Abort the Test".
- Buttons:** "Execute", "Import", "+Assert" (with a checkbox), and "Clear".

After a successful import, three tabs are added:

- State

- Request
- Response

You can now configure and invoke the frame.

The following graphic shows the step, after a frame was imported.

▼ Execute Transaction Frame - Execute Transaction Frame

Executing agent: JBoss\_LISABank (1088413135)

Frame property:

Class name: com.itko.examples.ejb3.EJB3UserControlBean  
Method name: getUser

State Request Response

XML Document

Node	Type	Occurs	Nil	Nilable	Value
EJB			<input type="checkbox"/>		
java.naming.factory.initial			<input type="checkbox"/>		{{JNDIFACTORY}}
java.naming.provider.url			<input type="checkbox"/>		{{JNDIURL}}
jndi.name			<input type="checkbox"/>		{{JNDINAME}}
isEjb3			<input type="checkbox"/>		true
serializedFully			<input type="checkbox"/>		false

Validation Results View XML Schema Source Error Log

If environment error: Abort the Test

Execute Import +Assert Clear

The upper portion contains the following items:

#### Executing agent

The agent that runs the frame. The color of the agent in the drop-down list indicates whether it is active. Execution from this editor works only if the agent is active. To refresh the contents, click the Refresh button to the right of the drop-down list. A property can also be specified.

#### Frame property

The property from which the frame to execute is obtained when the step is run. This field is automatically filled in for consolidated baseline tests that the CAI Console creates.

#### Class name

The name of the class of the method.

#### Method name

The name of the method that the agent intercepted.

The State tab contains metadata that the underlying protocol requires. For example, the state for an EJB frame includes the JNDI lookup information. You can change the state before execution.

The Request tab contains the input to the method for the frame. You can change the request before execution.

In the State and Request tabs, the heading bar indicates the type of payload that is being shown. You can change the payload type by clicking the icon in the right portion of the heading bar and selecting the appropriate type.

The Response tab compares the expected response from invoking the frame with the actual response. The format is the same as the Graphical XML Side-by-Side Comparison assertion.

The lower portion contains the following items:

**If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

**Default:** Abort the test.

**Execute**

Invoke the frame on the currently selected agent. You can use this button to test the step "in place" in the editor.

**Import**

Import a frame.

**Assert**

Add an assertion to the step that verifies the response. The assertion is when you import a frame.

**Clear**

Remove the imported frame.

## Mobile Steps

Mobile test steps are automatically created when you record a mobile test case. The test steps represent the mobile actions or gestures that are captured during the recording. Each test step represents the actions that you performed on a specific screen in the application.

When the recorded test case is complete, you can modify the test case in a variety of ways by manually reordering the actions, inserting additional actions, or adding assertions.

**This section contains the following topics:**

[Modify Mobile Test Steps](#) (see page 384)

## Modify Mobile Test Steps

### Follow these steps:

1. Open the mobile test case that you want to update.
2. To view the details of each step:
  - a. Click the test step that you want to review.
  - b. Click Mobile testing step in the element tree on the right to expand the details for the step.

The Mobile Testing Step tab opens and shows a screenshot of the test application. The Actions section at the top of the tab shows the individual actions that are performed in the test step.
  - c. To view the screenshot associated with a specific action, click the action in the Actions section.
3. To manually add a new action, click Add (+) in the Actions section and enter the Key/Value pair for that action.
4. To rearrange the order of the actions, click the action you want to move and use the up and down arrows to move the action.
5. To delete an action, click the action and click Delete.
6. You can also add the following element or screen actions by right-clicking the screenshot for a recorded action:

#### Back

Inserts a Back command to return to the previous screen in the application. Android only.

#### Get element

Performs a Get on the selected element. The name of the selected element appears next to the Get element option.

#### Tap element

Performs a Tap action on the selected element. The name of the selected element appears next to the Tap element option.

#### Send keys

Brings up the keyboard and simulates pressing the specified keys to set the value of the text element.

**Note:** In some instances, the application does not display the typed values. For example, a password field might not display the entered values.

#### Set value

Sets the value for the selected element. The name of the selected element appears next to the Set value option.



**Note:** You can also use this action to set the value for a slider. For example, setting a value of 0.8 moves the slider 80% to the right. Use increments of 0.1 when setting the value for a slider. Setting a value of 0.25 is unrecognized and does not move the slider.

**Long Tap screen**

Performs a long tap on the screen, as opposed to a specific element.

**Long Tap element**

Performs a long tap on the selected element. The name of the selected element appears next to the Long Tap element option.

**Wait**

Inserts a pause in the test. You can express the wait like ThinkTime at the step level. For example, **1s-10s** inserts a random pause between 1 and 10 seconds. **100z** inserts a pause of 100 milliseconds.

**Comment**

Lets you insert a comment for a specific action. The comment performs no action, it is for documentation only.

**Script**

Lets you insert an arbitrary beanshell script. Typically, this script does not interact with the device or simulator. This action provides the means of inserting a custom assertion.

**Change Orientation**

Changes the orientation of the device.

**Shake**

Performs a shake gesture on the device.

**Go to background**

Performs the action of pressing the home button. This action forces the application to the background for 10 seconds and then brings it back. This action is useful for testing because it typically forces the application to release memory and stop any CPU-intensive work. iOS only.

**Assertion**

For more information, see [Add an Assertion to a Mobile Test Step](#).

7. Click Save.

## Custom Extension Steps

**The following steps are available:**

[Custom Test Step Execution](#) (see page 386)

[Java Script Step \(deprecated\)](#) (see page 387)

[Execute Script \(JSR-223\) Step](#) (see page 389)

## Custom Test Step Execution

The Custom Test Step runs a test step that is written by your team using the Software Developer Kit (SDK). This step is documented in *Using the SDK*.

## Java Script Step (deprecated)

**Note:** The Java Script step has been deprecated. Use the [Execute Script \(JSR-223\)](#) (see page 389) step instead.

The Java Script step gives you the flexibility of writing and executing a Java script to perform some function or procedure. Your script is executed using the BeanShell interpreter. You have access to all the properties in the test case, including built-in objects.

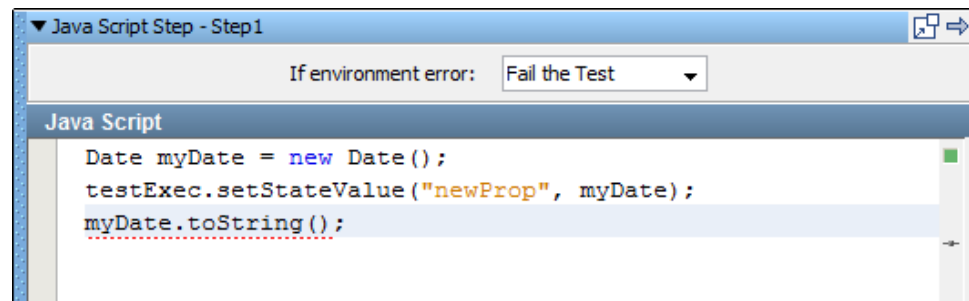
**Prerequisites:** Some knowledge of BeanShell. For more information about BeanShell, see <http://www.beanshell.org/>.

The step includes a script editor. Double-clicking an item in the Available Objects list pastes that variable name into the editor.

The last value that is exposed in the script is saved as the response of this step.

The following graphic shows the script editor. The script contains the following statements:

- A new Date object is created, initialized to the current date and time.
- This object is stored in a new property, **newProp**, using one of the DevTest exposed objects, testExec. For more information, see *Using the SDK*.
- The **toString()** value of the Date object is set as the response of the step.



The Test button lets you test the script. You see the result from executing the script, or an error message describing the error that occurred.

DevTest property name syntax is flexible and can include spaces. The property names that are not valid Java identifiers are converted for use in this step. An underscore (\_) replaces any invalid characters.

If you use properties **{{exampleprop}}** in a script, DevTest substitutes the properties for the actual property values at run time before it runs the script.

Properties with "." in their names are imported into the script environment with "\_" replacing ".". So **{{foo.bar}}** in a script is the same as **foo\_bar**.

You can produce a log event inside a script step or assertion. A **testExec** object is useful. To produce a log event, code the following instead of using the log4j logger. The **testExec.log()** object causes an actual event to be raised and you can see it in the ITR.

```
testExec.log("Got here");
```

A test case run has only one scripting instance. If there are multiple Java scripting steps, whether in the same test case or in subprocesses, DevTest uses the same instance to run the entire test case.

By default, variables are global to the instance. This behavior extends to subprocesses.

If you want the scope of a variable to be local, place the code inside an opening curly brace and a closing curly brace. For example:

```
{
    String var= "local";
    return var;
}
```

Parameter names for subprocesses are treated as global variables.

## Execute Script (JSR-223) Step

**Note:** The Java Script step has been deprecated. Use the Execute Script (JSR-223) step instead.

The Execute Script (JSR-223) step lets you write and run a script to perform some function or procedure. You can select from:

- Applescript (for OS X)
- Beanshell
- Freemarker
- Groovy
- JavaScript
- Velocity

To use additional scripting languages, see "Enabling Additional Scripting Languages."

You have access to all the properties in the test case, including built-in objects.

Complete the following fields:

**If Environment Error**

Specifies the action to take or the step to go to if the test fails because of an environment error.

Default: **Abort the test.**

**Script Language**

Designates the scripting language to use.

**Values:**

- Applescript (for OS X)
- Beanshell
- Freemarker
- Groovy
- JavaScript
- Velocity

**Default:** Beanshell

**Copy properties into scope**

Allows you to specify which properties to download for use in the step.

**Values:**

- Test state and system properties: all properties for the test case and system
- Test state properties: properties that provide information about the test case
- TestExec and logger only: only the TestExec and logger properties

**Default:** Test state properties

Click Test to open a window with the result of the script execution or a description of the errors that occurred.

Anywhere you use {{expressions}}, you can specify a scripting language by using this syntax:

`{{=%\language%}}`

Examples are in the **scripting** test case in the Examples project.

Set the default scripting language by using the **lisa.scripting.default.language** property.

# Chapter 2: Test Document Reference

---

This section contains the following topics:

[Events](#) (see page 391)

[Metrics](#) (see page 397)

## Events

The following table describes the standard events. The table is sorted by event name.

Event Name	Description	Short Info	Long Info
Abort	This event ends a test in a "cannot finish" state. It is a failing type of end event.	Event Aborted	
Assert evaluated	A non-embedded assertion will generate this event if it does not fire. These events are the assertions that did not get to execute their consequence.	The name of the assertion	
Assertion fired	A non-embedded assertion will generate this event if it does fire. Firing means it is true and its consequence will be followed.	The name of the assertion	The log message of the assertion, or a DevTest-generated message if there is no log message set
Call made	Steps that perform object calls like web services or EJBs use this event to report each call that is made on the object.	The name of the step	The call as a string, for example, <b>void setName( java.lang.String name[Basic Checking] )</b>
Call result	Steps that perform object calls like web services or EJBs use this event to report the response they get from calls.	The name of the step	The call response as a string. If the response is an object, then an XML view of the object is shown.

Configure for load test		Configure for load test	The test run has been configured as a load test. Selecting individual events has been disabled.
Coordinator ended	A coordinator has been removed.	The name of the coordinator server	
Coordinator server ended	The coordinator server was ended.	The name of the coordinator server	
Coordinator server started	The coordinator server was created.	The name of the coordinator server	
Coordinator started	A new coordinator was created.	The name of the coordinator server	
Cycle ended normally	The test execution completed in a successful state.	The name of the test case	
Cycle ending	The instances have been instructed to stop testing.		
Cycle failed	The test execution failed. That there are no exceptions in the test case, but either logic errors in the test case or in the system under test caused the test case not to complete as expected.	The name of the test case	
Cycle history	Every model that runs will generate one of these events. It is the final trace of all the details of its run.	Cycle History	
Cycle initialized	The test has been initialized (loaded).	The name of the test case	
Cycle runtime error	An abnormal DevTest error occurred. For example, the coordinator lost its connection to a simulator while running a test.	Varies but it is usually the name of the test element (step, data set, or filter) that has the error	Usually a message that explains the error
Cycle started	This test instance and cycle have just started.	The name of the test case	



Data set read	Data sets generate an event that makes it clear what values are about to be used.	Data set read	
HTTP performance	This event is generated for every HTTP transaction that DevTest executes to capture the performance statistics.	HTTP performance statistics	
Info message	Basic logging data. For example, the HTTP/HTML request step sends this message with the step name in the short field and the URL being sent to the server in the long field.	Usually the name of the step during which this message was generated	Usually a message that DevTest generated
Instance ended	Sent when a simulator instance has finished.	The name of the simulator	
Instance started	Sent when a simulator creates an instance.	The name of the simulator	
Log message	Basic logging data. Can be turned off to minimize overhead when filtering events.	The message sent to the log	
Metric alert	A metric has been collected and is reporting its value.	The short name of the metric, for example, <b>DevTest: Avg Response Time</b>	The value of the metric collected
Metric started	Metrics that are collected generate real-time events of their values.	Metric Started	
Metric value	Metrics that are collected generate real-time events of their values.	Metric value	
Model definition error	A test case error was discovered during execution of the test. For example, the name is constructed from a property that does not exist.	Varies but it is usually the name of the test element (step, data set, or filter) that has the error	Usually a message that explains the error

Pathfinder	The system being tested has DevTest integration enabled. This event contains the DevTest integration XML data that was captured.	The name of the step	The XML representation of the DevTest Integration data captured
Property removed	A property was removed.	The property key	The word "<removed>"
Property set	A property was set.	The property key	The property value
Simulator ended	Sent when a simulator has ended.	The name of the simulator	
Simulator started	Sent when a simulator has started.	The name of the simulator	
Step bandwidth consumed	Approximate amount of data sent and received from the system under test for the step execution.	The name of the step	Actual number of bytes read/received
Step error	An error has occurred in the system under test. For example, there was no response from a web server. This event is for a step. The EVENT_TESTFAILED refers to the complete test case.	The name of the step	If available, a message to help determine the cause of the failure
Step history	Every step has a history event that fires of type <b>com.itko.lisa.test.NodeExecutionContextHistory</b> in its long info.	Step history	
Step request	Steps that support this event use it to report the actual request made to the system under test.	The name of the step	The request data as a string
Step request bandwidth			
Step response	A step was completed against the system under test.	The name of the step	The response data as a string

Step response bandwidth			
Step response time	The amount of time a step took to execute against the system under test.	The name of the step	The number of milliseconds to execute the step
Step started	A step is being executed.	The name of the step	
Step target	Every step has a target, such as the URL for a web request or the JNDI name of an EJB.		
Step warning	A warning was recorded. For example, a filter took the default value because it could not find the current value.	Step warning	
Subprocess finished	A subprocess has finished executing.		
Subprocess ran	A subprocess has started.		
Suite aborted	When a setup test (defined in a suite document) has failed, then the suite will not run the tests defined in the suite. This event indicates that this happened.	The name of the suite	
Suite ended	All the tests running as part of a suite have finished.	The name of the suite	
Suite history	Every suite that runs will generate one of these events. It is the final trace of all the details of its run.	Suite History	
Suite setup/teardown	The suite has a setup or teardown test defined and that test has just started to run.	The name of the suite	The name of the test, path to the test, and other information
Suite started	A suite is starting to run.	The name of the suite	

Suite test failed	A test running as part of a suite ended in failure.	The name of the suite	
Suite test passed	A test running as part of a suite ended successfully.	The name of the suite	
Suite test staged	A test is staged to run as part of a suite.	The name of the suite	The name of the test, path to the test, and other information
Test ended	Sent when the coordinator stops the test.	The name of the test case	
Test not active	A test in a suite is marked as inactive.	Event Test Not Active	
Test started	Sent when the coordinator starts the test.	The name of the test case	
VS log message	VSE internal logging	VS log message	
VS no transaction match	A virtual service did not match a transaction request to at least one recorded response.	VS transaction no match	
VS service ended	A virtual service was stopped.	VS service stopped	
VS service started	A virtual service was started.	VS service started	
VS transaction finished	A virtual service finished processing a transaction request.		
VS transaction match	A virtual service matched a transaction request to at least one recorded response.	VS transaction match	
VSE server reset	A service reset request was made of a server.	VSE server resets	
VSE server shutdown	A virtual service environment was asked to shut down.	VSE server shutdown	
VSE server stop	A service stop request was made of a server.	VSE server stops	

# Metrics

**This section describes the types of metrics available:**

[DevTest Whole Test Metrics](#) (see page 397)

[DevTest Test Event Metrics](#) (see page 398)

[SNMP Metrics](#) (see page 400)

[JMX Metrics](#) (see page 402)

[TIBCO Hawk Metrics](#) (see page 405)

[Windows Perfmon Metrics](#) (see page 407)

[UNIX Metrics Via SSH](#) (see page 409)

## DevTest Whole Test Metrics

Whole Test Metrics, as the name suggests, collect all the basic information about the test case and provide six sub metrics.

The following sub metrics are collected. The response time metrics are reported in milliseconds.

- **Instances** (the number of virtual users)
- **Avg Resp Time**
- **Max Resp Time**
- **Min Resp Time**
- **Last Resp Time**
- **Steps Per second**

**Note:** The number of virtual users (Instances), average response times (Avg Resp Time), and Steps Per Second sub metrics are added by default to the metric list in a staging document.

## DevTest Test Event Metrics

Test Event Metrics provide metrics in regard to DevTest events. These metrics include both counters and gauges.

Event metrics let you filter events of a specific type by including a regular expression to match the short description field of the event.

The Test Event Metrics category has eight submetrics:

- **Coordinator server started**
- **Coordinator server ended**
- **Coordinator started**
- **Coordinator ended**
- **Test started**
- **Test ended**
- **Instance started**
- **Instance ended**

### To add test event metrics:

In addition to selecting the metric, you can specify:

- **Key Expression Match:** Enter an expression to say to sample the chosen event only if it has this expression in its short description field. If you leave this field blank, or enter \*, then every event of this type is reported.
- **Metric is a Counter:** If this box is selected, the value counts over time are recorded. If the box is cleared, the absolute value is recorded (metric is functioning as a gauge).

Test Event metrics are meant to track the presence of specific events in the workflow engine. When the Counter check box is selected, the metric collector tracks the number of times that the chosen event was observed. The count increases as the test runs; however, the graphs in DevTest Workstation and the reporting console shows the value that was collected during the sampling period. This is the default option for Test Event Metrics.

If the Counter check box is not selected, the metric collector attempts to parse the long description of the event as a number. During the sampling period the collector keeps a running total of these parsed values. When the metric is reported it is a simple average of the parsed numbers. The running total is reset for each sampling period. If there is not a number value that is contained in the long description of the event, the metric value is 0.

The default sampling period is 1 second and can be modified in the staging document.

- Click OK to add this metric to the list of metrics.

## SNMP Metrics

The SNMP metrics use the Simple Network Management Protocol (SNMP) to monitor the system performance.

### Setting up SNMP Support

Configure SNMP before you can collect the SNMP metrics. For more details about setting up SNMP on UNIX and Windows, see Install and Configuring SNMP in *Installing*.

To add SNMP Metrics:

1. Select SNMP Metric from the dialog and click OK.

The Add SNMP Metric dialog opens.

The MIB Groups tree displays all the SNMP metrics that come standard with DevTest Workstation.

A MIB is a predefined database of a set of metrics on a specific domain.

#### Host

The Host field provides system information about a server hosting a coordinator server. Some host metrics include **hrProcessorLoad** for CPU utilization and **hrSystemUptime** for the amount of time after this host was last initialized.

#### Server O/S

Provides information that is related to the system, like up time, date, and number of users.

#### BEA WebLogic

Provides JDBC, JMS, JVM, socket, servlet, and web application information about a Server running BEA WebLogic. Some BEA WebLogic metrics include **jvmRuntime-HeapFreeCurrent** for the current amount of free memory in the JVM heap in bytes, and **webAppComponentRuntimeOpenSessionsCurrentCount** for the current total number of open sessions in this component.

#### RDBMS

Provides information about a server running a generic relational database management system. Some RDBMS metrics include **rdbmsSrvInfoPageReads** for the number of physical page reads that were completed after the RDBMS was last restarted, and **rdbmsSrvInfoPageWrites** for the number of single page writes that were completed after the RDBMS was last restarted.

#### Oracle



Provides information about a server running Oracle. Some Oracle metrics include **oraDbSysUserCommits** for the number of user commits, and **oraDbSysUserRollbacks** for the number of times data has rolled back.

#### Microsoft SQL Server

Provides information about a server running Microsoft SQL Server. Some Microsoft SQL Server metrics include **mssqlSrvInfoCacheHitRatio** for the percentage of time that a requested data page was found in the data cache (instead of being read from disk), and **mssqlSrvInfoUserConnections** for the number of open user connections.

2. To select metrics, browse through these choices.
3. Click the target metric in the left panel.

The required parameters for this target metric are completed in the MIB Object form in the right panel. A description of the metric is displayed in the text box. The other information can be ignored or accepted.

4. Enter the domain name or IP address of the host computer (Host) where you are collecting the metrics. To add this metric, click OK.
5. Repeat until you have added all the target SNMP metrics.

To add SNMP metrics that are not in the MIB Group tree, manually enter the data (OID) into the **MIB Object** form. An OID is the unique identifier of a specific metric, using a tree-structured naming scheme. The domain root OID for ITKO is ".1.3.6.1.4.1.12841.1.1", so all SNMP metrics start from there.

The Load From MIB button lets you browse the file system for MIBs.

The SNMP Walk button lets you browse an SNMP tree.

**Note:** All SNMP MIBs are supported. The set of MIBs displayed in the Add SNMP Metric dialog are only a sample of the MIBs that are supported. The Add SNMP Metric dialog makes it easier to understand the Object IDs (OIDs) in those MIBs. However, any valid OID works, not only those OIDs that are displayed in the Add SNMP Metric dialog. A set of all the standard MIBs from IETF and IANA are provided. Those MIBs are stored in the **LISA\_HOME\snmp\ietf** and **LISA\_HOME\snmp\iana** directories.

## JMX Metrics

The JMX metrics use the Java Management Extension (JMX) API to provide metrics.

These JMX connectors are provided for an easy setup:

- Any JSR 160 RMI connection
- JBoss 3.2-4.0
- JSE 5 Connector
- Oracle AS (OCJ4)
- Tomcat 5.0.28
- WebLogic 6.1-8.1
- WebLogic 9.x
- WebSphere 5.x
- ITKO JMX Agents

Each of these applications requires slightly different connection parameters. The values that you require for your server are available from your server administrator. Each provider provides slightly different metrics. To use other JMX features, you can invoke them as RMI steps.

Only numerical attributes are supported.

The following example uses JBoss.

**Follow these steps:**

1. Select JMX Attribute Reader from the dialog.
2. Click OK.

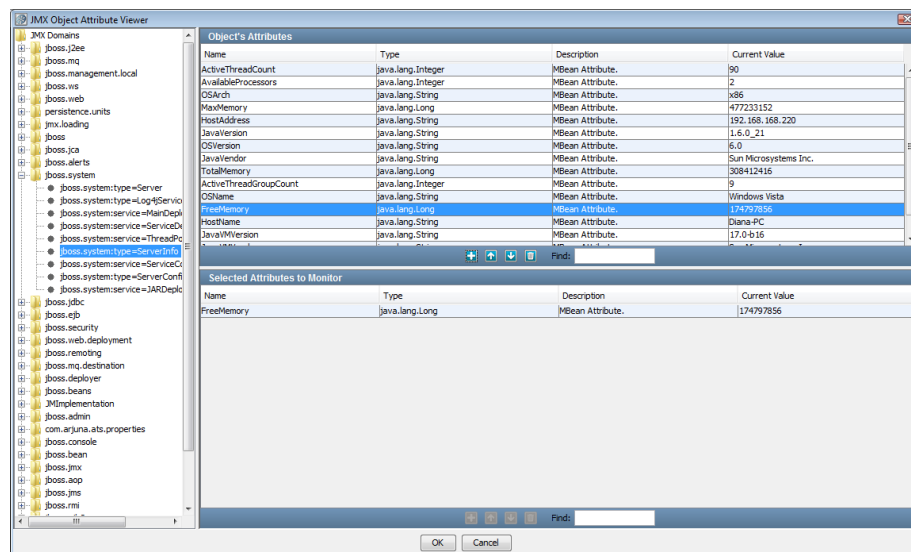
The Select and Configure JMX Agent dialog opens.

3. Select **JBoss** from the list of JMX Connectors.
4. Enter the Server Naming URL and Agent RMI name for your installation.
5. The values that you enter in the remaining fields depend on whether your JBoss server requires authentication to access JMX data.
  - If your JBoss server does not require authentication for JMX access, use the default context factory "org.jnp.interfaces.NamingContextFactory".
  - If your JBoss server does require authentication:
    - a. Change the context factory to "org.jboss.security.jndi.JndiLoginInitialContextFactory".
    - b. Enter the username and password (principal and credentials) needed for the authentication.

c. Enter the JAAS login module that will authenticate these credentials.

6. Click OK.

The JMX Object Attribute Viewer opens.




On the left is the JMX Domain hierarchy for JBoss. JMX metrics use an object-attribute model. In this model, domain objects are published by the specific application server (for example, "system"), and attributes are name/value pairs inside the object.

When you select an object from this tree, then the base attributes of that object are also displayed in the tree. After you select a base attribute, the rest of the attribute name appears in the list in the top right Object Attribute panel.

In the previous example, we selected the domain object to be `jboss.system`, the base attribute to be `ServerInfo`, and the rest of the attribute name to be `FreeMemory`. The attributes for `ServerInfo` are displayed in the Object Attribute panel.

7. To select one of these attributes (metrics), select the metric, and click Add .

This metric is added to the list of selected metrics in the Selected Attributes to Monitor panel at the bottom of the window.

To remove an attribute from this panel, click Delete .

8. Repeat this process until all of your chosen metrics appear in your list (bottom panel).

9. Click OK to return to the main metrics panel.

Notice that the JMX metrics are now on the list of metrics.


Depending on the application server, vendor-specific JARs could be required to enable the JMX communication with that application server.

10. Similarly, you can also select the **JSE 5 Connector** from the JMX Connector list.

11. Click OK to connect to the JMX Agent.

## Enable JMX Metrics for Tomcat

### Follow these steps:

1. Modify the **catalina.bat** file and add **CATALINA\_OPTS**.  
You can use the embedded Jakarta-Tomcat that is packaged with DevTest Solutions.
2. Connect to Tomcat through JConsole.
3. Start DevTest, open a staging document, and go to the Metrics tab.
4. Click Add .
5. Select JMX Attribute Reader and click OK.
6. Select Tomcat 5.0.28.  
All parameters other than user name and password are prepopulated.
7. Click OK and connect to the JMX console.
8. Select a few objects specific to Catalina and add them in the list.
9. Click OK.  
These attributes appear in the metrics list.


## TIBCO Hawk Metrics

TIBCO Hawk is a tool for monitoring and managing distributed applications and operating systems. Unlike other monitoring solutions, TIBCO Hawk software uses TIBCO Messaging software for communicating and inherits many of its benefits. These benefits include a flexible architecture, enterprise-wide scalability, and location-transparent product components that are simple to configure.

DevTest has out-of-the-box integration with TIBCO Hawk for monitoring distributed applications and operating systems metrics in the context of testing. TIBCO Hawk provides in-container metrics for TIBCO BusinessWorks process archives. By using TIBCO Hawk, DevTest can monitor metrics of all activities in any TIBCO BusinessWorks process that is deployed. This integration facilitates peering inside of a process to understand where bottlenecks are occurring.

**Prerequisites:** Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see *Third-Party File Requirements in Administering*.

### Follow these steps:

1. Create a staging document.
2. On the Metrics tab of the staging document, click Add .
3. Select TIBCO Hawk from the drop-down list.

The following parameters are available:

- **Transport:** Select **Rendezvous** or **EMS**.
- **Hawk Domain:** Enter the TIBCO Hawk domain.
- **Rendezvous Service**
- **Rendezvous Network**
- **Rendezvous Daemon**
- **EMS Url**
- **EMS User**
- **EMS Password**

4. Click OK.

The Hawk Object Attribute Viewer opens.

5. Select the Process Archive and expand beneath it.
6. The **GetProcessDefinitions** method retrieves the process definitions that are defined in the Process Archive.
7. Use the Process Definition Name as parameter for the **GetActivities** method.

TIBCO Hawk provides a number of metrics for Process Activities.

8. Provide a filter with the Process Definition Name and the Activity Name.
9. Select Method Return Values and click OK.

Activity Metrics are now monitored for test cases staged with this staging document.

TIBCO Hawk can also be invoked in a test case, through TIBCO Hawk APIs.

**Note:** It takes a few minutes for TIBCO Hawk to initialize itself and achieve a ready state where it can report metrics. TIBCO notifies DevTest when this ready state is achieved. It is important to set **`lisa.net.timeout.ms`** to a fairly high value (at least 2-3 minutes) to avoid timeouts when using staging documents that collect the TIBCO Hawk metrics.

## Windows Perfmon Metrics

The Perfmon Metrics use Microsoft Windows Perfmon to provide metrics to monitor the system performance on a Windows operating system. These metrics are similar to the SNMP metrics.

### Setting up Perfmon

Before you can collect the Perfmon metrics, configure Perfmon on your Windows computer.

For information about setting up Perfmon, see *Install Performance Monitor (Perfmon) in Installing*.

**Note:** If you are collecting Perfmon metrics for remote computers, ensure the Remote Registry service is running. This service does not run by default on Windows 7 or Windows 8. You must manually start the Remote Registry service from the Windows Services Manager.

#### Follow these steps:

1. Select Windows Perfmon Metrics from the Metrics dialog and click OK.

The Windows Perfmon Machine Name dialog opens.

2. Complete the following fields:

#### Machine Name

Enter the Machine Name of your Windows installation. You can find your machine name by right-clicking My Computer, selecting Properties, and going to the Computer Name tab.

#### User Name

Enter the user name of the remote computer.

#### Password

Enter the password of the remote computer.

#### Domain

Enter the domain name. A domain is optional.

3. Click OK to proceed.

After you enter valid login credentials, a window appears showing all the available metric types. The Windows Perfmon Metrics window opens.

The Perfmon Metric window has three panels.

- The left panel displays the Selected Performance Category.

- The top right panel displays the description of the selected category in the Counter Description section.
  - The bottom right panel lists the metric that is selected in the left panel.
4. Select a metric from the Performance Category list.

This list shows all available categories that can be monitored:

- .NET CLR Remoting/ LocksandThreads
- .NET CLR Data/Networking
- Job Objects/Job Object Details
- Performance/RSVP Service
- Memory/Print Queue
- ICMP/Process
- Outlook/Logical disk
- IP/Server/Cache

After you select the category, it is added to the left panel.

5. Double-click the target metric in the left panel to add it to the Selected Counters to Monitor table on the right panel.
6. Repeat until you have added all the target Perfmon metrics.
7. Click OK.



## UNIX Metrics Via SSH

This metric is used to collect command-line metrics. This metric gathers inputs like authentication details, host name, and the selection of metrics to be collected.

The metric data is stored in an XML file that is located in the **LISA\_HOME/umetrics** directory by default. However, you can define a new location by updating the key **stats.unix.xml.folder** in **local.properties**.

Each file in that directory has a unique file name that is based on the operating system:

- Linux.xml
- osx.xml
- Unix.xml
- windows.xml

The XML file is used to feed the information about the command and metrics to be collected on a specific operating system. For example, to make the command and metrics known on a Linux operating system, a linux.xml file must be in the **LISA\_HOME/umetrics** folder.

The following graphic is an example of an OSX command parser for **iostat** that collects CPU and disk0 metrics.

```
<UnixMetrics>
  <Platform>OS X</Platform>
  <MetricCollectorSet>
    <Name>iostat</Name>
    <Description>Statistics on the IO for our Unix box</Description>
    <!-- command that will be invoked. Must not require interactive prompts! -->
    <WindowsCommand>cmd ssh john@myserver.itko.com iostat -w 1</WindowsCommand>
    <UnixCommand>ssh john@myserver.itko.com iostat -w 1</UnixCommand>

    <!-- Most commands have a header that we want to know to skip -->
    <IgnoreLineCount>3</IgnoreLineCount>

    <!-- Sometimes headers are repeated, or random break lines appear, so put tokens that warn our
         parser not to consider that line here -->
    <IgnoreTokenList>load,MB</IgnoreTokenList>

    <!-- And here is the metric you want, the assumption is that every "real" line has a value for it -->
    <Metric name="UserCPU" start="38" end="41" decimalPlaces="0" gauge="true">
      Summary of % time spent in user code of across all CPU cores
    </Metric>
    <Metric name="SystemCPU" start="41" end="44" decimalPlaces="0" gauge="true">
      Summary of % time spent in kernel code of across all CPU cores
    </Metric>
    <Metric name="Disk0MBs" start="13" end="19" decimalPlaces="2" gauge="true">
      The first disk MB per second
    </Metric>
    <Metric name="Idle CPU" start="44" end="47" decimalPlaces="0" gauge="true">
      Percent of time CPUs are idle
    </Metric>
  </MetricCollectorSet>
</MetricCollectorSet>
```

In the Specify Remote Machine details window, enter the operating system: Linux, OS X, or Solaris.

Enter the remote computer details:

- Host
- User

Select authentication type and enter information:

- Password
- Private Key
- Encrypted Private Key

When you use a private key or an encrypted private key, the private key file must be in PEM format and in the Data folder under the project. Use Browse to select the file.

Each operating system has a slightly different set of metrics from which to select. To select the metrics you would like to collect, use the check box in the left column.

# 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 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 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 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 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 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) 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 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 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 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 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 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 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 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) 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) 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 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 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 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 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 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 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 in *Using CA Application Test*.

**subprocess**

A *subprocess* is a test case that another test case calls. For more information, see Building Subprocesses 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 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 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 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) and Staging Document Editor - 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*.