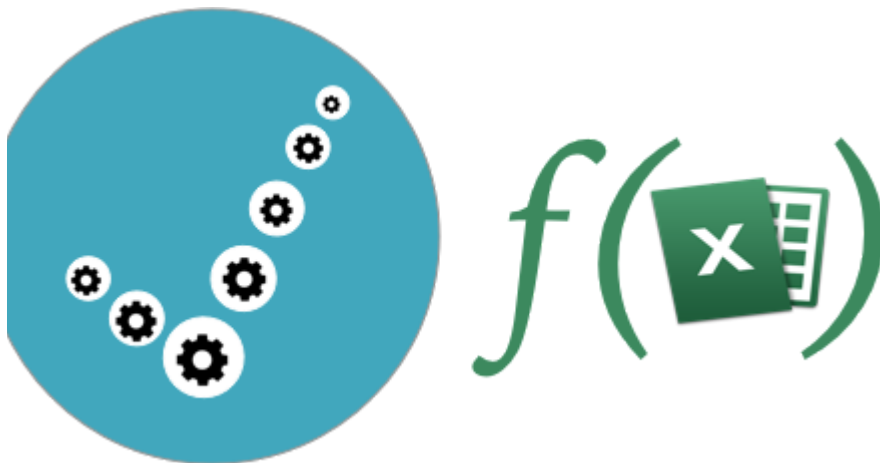


# Path To QA Experience

RIT



MAY 12, 2016JUNE 1, 2016 / HARINIVAS GANAPATHY

## RIT Reporting Using Excel Custom Functions

*This post is a tutorial providing a comprehensive method to create and update test results to the excel workbook using the Custom Functions available in RIT-OpenFramework. This post can also be used as a tutorial for Custom Functions available in GitHub-@harinivas-ganapathy. (<https://github.com/harinivas-ganapathy/RIT-OpenFramework/>). Here we would see how to use WriteToExcel function update Test data sheet with test status and also we would use the examples, project setup to re-execute only failed tests and update their results. We do this in automation fashion.*

For demonstrating the concept and implementation we'll use the ConvertTemp web service available from <http://www.websvcex.net> (<http://www.websvcex.net>).

Pre-Requisites –

(1) Rational Integration Tester (Henceforth will be referenced as RIT)

(2) Internet access to open ConvertTemp WSDL available from

<http://www.websvcex.net/ConvertTemperature.asmx>

(<http://www.websvcex.net/ConvertTemperature.asmx>)

(3) Download ExcelFunction\_<version\_number>.jar from the GitHub repository –

<https://github.com/harinivas-ganapathy/RIT-OpenFramework.git> (<https://github.com/harinivas-ganapathy/RIT-OpenFramework.git>)

(4) Sample Excel File with Test cases for the ConvertTemp operation. Is also available from the GitHub repository –

[https://github.com/harinivas-ganapathy/RIT-](https://github.com/harinivas-ganapathy/RIT-OpenFramework/blob/master/Examples/ConvertTemp_TestCases_V1.00.xlsx)

[OpenFramework/blob/master/Examples/ConvertTemp\\_TestCases\\_V1.00.xlsx](https://github.com/harinivas-ganapathy/RIT-OpenFramework/blob/master/Examples/ConvertTemp_TestCases_V1.00.xlsx) ([http://](http://https://github.com/harinivas-ganapathy/RIT-OpenFramework/blob/master/Examples/ConvertTemp_TestCases_V1.00.xlsx)

[https://github.com/harinivas-ganapathy/RIT-](https://github.com/harinivas-ganapathy/RIT-OpenFramework/blob/master/Examples/ConvertTemp_TestCases_V1.00.xlsx)

[OpenFramework/blob/master/Examples/ConvertTemp\\_TestCases\\_V1.00.xlsx](https://github.com/harinivas-ganapathy/RIT-OpenFramework/blob/master/Examples/ConvertTemp_TestCases_V1.00.xlsx))

## Getting Started –

### RIT Installation

First we would need RIT installation to proceed further. RIT provides a free Starter Edition for all learning purpose and acustom with the application. It can be download from IBM's Continuous Testing Page –

#### Mac Version –

<https://developer.ibm.com/testing/resources/downloading-rational-integration-tester-starter-edition-for-mac-64-bit/> ([http:// https://developer.ibm.com/testing/resources/downloading-rational-integration-tester-starter-edition-for-mac-64-bit/](http://https://developer.ibm.com/testing/resources/downloading-rational-integration-tester-starter-edition-for-mac-64-bit/))

#### Windows Version –

<https://developer.ibm.com/testing/resources/rational-integration-tester-starter-edition-download/> ([http:// https://developer.ibm.com/testing/resources/rational-integration-tester-starter-edition-download/](http://https://developer.ibm.com/testing/resources/rational-integration-tester-starter-edition-download/))

There is not much of installation procedure for the RIT Starter Edition (*RIT SE*). Just extract the downloaded zip or gzip archive. And double clicking the IntegrationTester.app in Mac brings up the RIT – Starter Edition Splash screen –



## Rational Integration Tester

### Starter Edition

Licensed Materials - Property of IBM Corp. © IBM Corporation and other(s) 2002, 2016. IBM, the IBM logo and Rational are trademarks of IBM Corp. in the United States, other countries and regions or both. Built on Eclipse logo is a trademark of Eclipse Foundation, Inc. Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates. Other company, product or service names may be trademarks or service marks of others.

Version 9.0.0.0



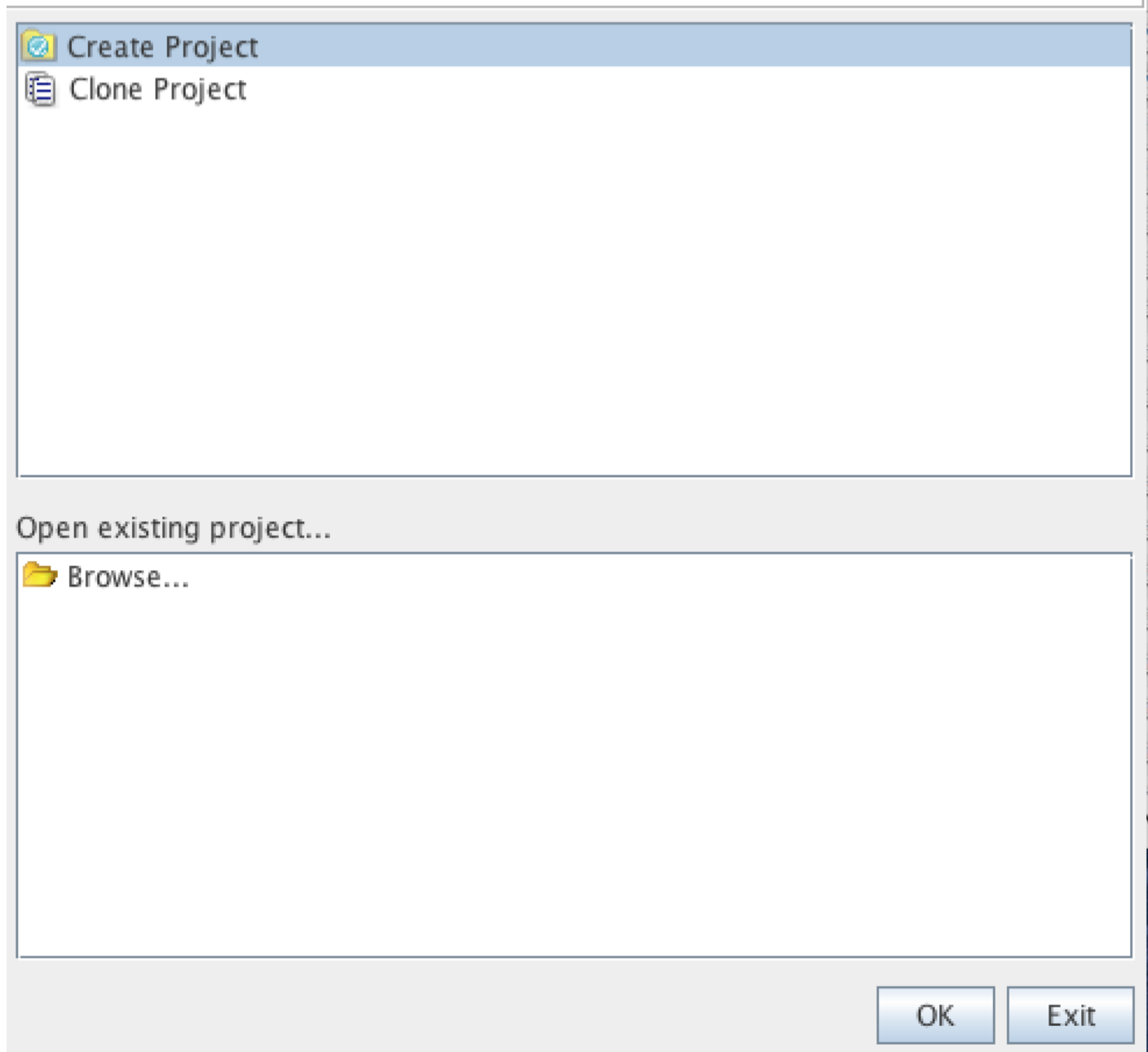
[https://harinivasganapathy.files.wordpress.com/2016/05/rit-se-splash-screen\\_full.png](https://harinivasganapathy.files.wordpress.com/2016/05/rit-se-splash-screen_full.png)

Later it would show Welcome window where we will create a new RIT Project

**Step 1:** Select Create Project from the Welcome Screen

## Welcome to IBM Rational Integration Tester Starter Edition 9.0.0.0a

Select an item from the lists below



([https://harinivasganapathy.files.wordpress.com/2016/05/1462796618\\_full.png](https://harinivasganapathy.files.wordpress.com/2016/05/1462796618_full.png))

On the displayed Create New Project screen

- Give a name for the project
- Specify a location for the project
- Click Finish

## New IBM Rational Integration Tester project details

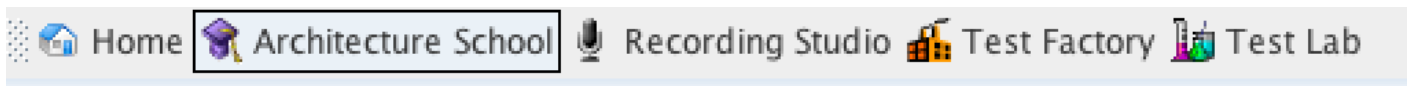
Enter your details to create a new IBM Rational Integration Tester project on your local machine



Project Name	<input type="text" value="ConvertTemp"/>
Owner	<input type="text" value="harinivas"/>
Comments	<input type="text" value="Project created on May 9, 2016"/>
Project Directory	<input type="text" value="/Users/harinivas/ConvertTemp"/> <input type="button" value="Browse..."/>
<input type="button" value="Cancel"/> <input type="button" value="Back"/> <input type="button" value="Next"/> <input type="button" value="Finish"/>	

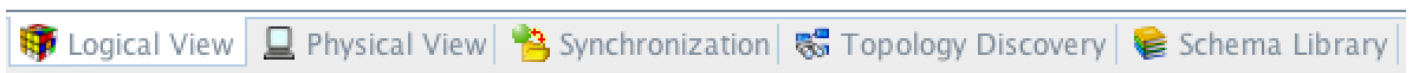
[https://harinivasganapathy.files.wordpress.com/2016/05/1462796700\\_full.png](https://harinivasganapathy.files.wordpress.com/2016/05/1462796700_full.png)

**Step 2:** Switch to the Architecture School selecting it from the Tool bar on the top



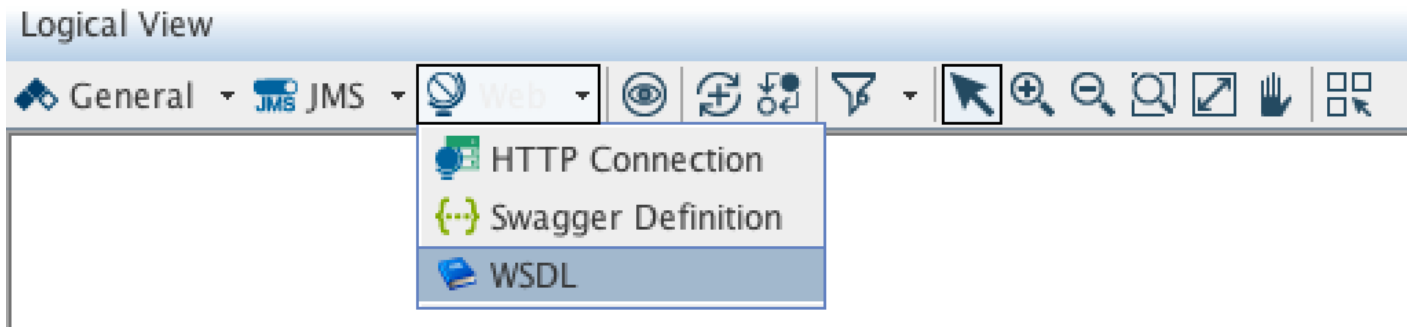
[https://harinivasganapathy.files.wordpress.com/2016/05/1462796903\\_full.png](https://harinivasganapathy.files.wordpress.com/2016/05/1462796903_full.png)

Ensure you're on the Logical View Tab of Architecture School



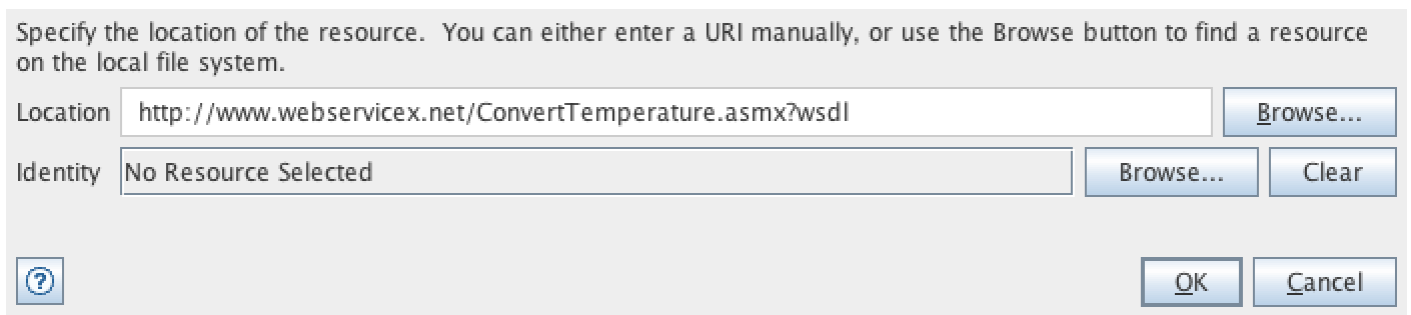
[https://harinivasganapathy.files.wordpress.com/2016/05/1462796935\\_full.png](https://harinivasganapathy.files.wordpress.com/2016/05/1462796935_full.png)

**Step 3:** Add the ConvertTemp WSDL by selecting wsdl from the Web drop down menu select



[https://harinivasganapathy.files.wordpress.com/2016/05/1462797168\\_full.png](https://harinivasganapathy.files.wordpress.com/2016/05/1462797168_full.png)

**Step 4:** From the “Create a New Synchronization Resource” – click “New...” to add ConverTemp Wsdl to the project



[https://harinivasganapathy.files.wordpress.com/2016/05/1462797980\\_full.png](https://harinivasganapathy.files.wordpress.com/2016/05/1462797980_full.png)

**Synchronization Source**  
Configure the connection details to your Synchronization Source

Type

Configuration

Configuration

Current Location

Environment

Choose existing or enter unique name

Show advanced options

[https://harinivasganapathy.files.wordpress.com/2016/05/1462797798\\_full.png](https://harinivasganapathy.files.wordpress.com/2016/05/1462797798_full.png)

**Step 5:** Click 'OK'

**Step 6:** Give a name for environment – like – SIT, UAT, ProdFIX.


**Step 7:** Click Next on the "Create a New Synchronization Resource"

**Step 8:** On the Generate Assets for Operations page – Click Next

**Step 9:** Click finish on the Summary Page

## Generate Assets for Operations

Choose which operations you wish to generate tests and stubs for. You can use the default asset name or specify your own name.

	<input checked="" type="checkbox"/>	Create Tests	<input type="checkbox"/>	Create Stubs
☝  ConvertTemperature				
▶ ConvertTemp [ConvertTemperatureSoap]	<input checked="" type="checkbox"/>	ConvertTemp [Co...	<input type="checkbox"/>	ConvertTemp [Co...
▶ ConvertTemp [ConvertTemperatureHttp0]	<input checked="" type="checkbox"/>	ConvertTemp [Co...	<input type="checkbox"/>	ConvertTemp [Co...
▶ ConvertTemp [ConvertTemperatureHttp1]	<input checked="" type="checkbox"/>	ConvertTemp [Co...	<input type="checkbox"/>	ConvertTemp [Co...

Cancel      << Back      **Next >>**      Finish

[https://harinivasganapathy.files.wordpress.com/2016/05/1462797992\\_full.png](https://harinivasganapathy.files.wordpress.com/2016/05/1462797992_full.png)

Service Component and operation would be created in the Logical View and synchronized in the Synchronization view



# ConvertTemperature

ConvertTemp  
[ConvertTemperatureHttpGet]



ConvertTemperatureHttpPost

ConvertTemp  
[ConvertTemperatureHttpPost]



ConvertTemperatureSoap

ConvertTemp  
[ConvertTemperatureSoap]



ConvertTemperatureSoap12



ConvertTemperatureHttpGet

([https://harinivasganapathy.files.wordpress.com/2016/05/1462798133\\_full.png](https://harinivasganapathy.files.wordpress.com/2016/05/1462798133_full.png))

This completes project setup.

## Loading the Custom Functions

From the the GitHub page download the ExcelFunctions<version>.jar and place it inside the Functions folder of the Project Home Directory

Switch to RIT Window and select from the Menu bar – **Tools -> Functions -> Reload Custom Functions/Behavior**

To Verify that that Custom Function is loaded select from the Menu bar – **Tools -> Functions->View All Functions**

### View All Functions

This is a list of all functions loaded in IBM Rational Integration Tester. There are 43 functions in total, including custom functions.

Name	Description	Syntax
insert	Data Model/insert	insert(tag_name)
join	Join	join( Delimiter, expr, expr, ... )
le	Assertion/Less than or equal	le(expr,expr)
lookupExcelCol	LookupExcelCol	lookupExcelCol(filepath,sheet...
lookupTD	Lookup Test Data	lookupTD(dataSetPath,[keyCol...
lt	Assertion/Less than	lt(expr,expr)
mod	Maths/Modulo	mod(expr,expr)
multiply	Maths/Multiply	multiply(expr,expr)
ne	Assertion/Not equal	ne(expr,expr)
not	Assertion/Not	not( expression )
null	Null	null()
or	Assertion/Or	or( expression, expression, ....)
readFromExcel	ReadFromExcel	readFromExcel(FilePath,Sheet...
regex	String/Regular expression	regex(string, expr, [instance], ...
regexEscape	String/Regular expression esc...	regexEscape(literal)
replaceTags	Replace tag(s)	replaceTags(value)
resetTags	Reset tag(s)	resetTags([tagNamePattern])
round	Maths/Round	round(expr[, decimal places])
setTag	Store a tag	setTag(tagName, value)
settlementDate	Settlement Date	settlementDate ( "startDate", ...
subtract	Maths/Subtract	subtract(expr,expr)
validateXSD	Validate against XSD	validateXSD(xml, [schemaURL...
writeToExcel	WriteToExcel	writeToExcel(filepath,sheetna...
XMLdbQuery	XML database query	XMLdbQuery(connectionId, qu...
xpath	XPath query	xpath(xml, xpath)

([https://harinivasganapathy.files.wordpress.com/2016/05/1462798558\\_full.png](https://harinivasganapathy.files.wordpress.com/2016/05/1462798558_full.png))

# Preparing the Test Cases and Test Data Source

After the RIT Project is setup, let's ensure the test cases are documented. For demonstration I have sample test cases available in a specific format.

Create a new folder 'Test cases' under Project Home.

Download the sample workbook from my GitHub repository in the Test cases folder.

Test_ID	Name	Test Details				Test Data			Test Baseline		Result	Comments
		Scenario	Allow_Execution?	Condition	Steps	Temperature	FromUnit	ToUnit	ConvertTempResult	Actual Value		
1	ConvertTemp_V_001	Conversion from Positive Integer Fahrenheit to Positive Integer Celsius	Y	1) Temperature should be positive Integer 2) FromUnit should be "degreeFahrenheit" 3) ToUnit should be "degreeCelsius"	Send request with positive value greater than 37.5 for Temperature and FromUnit as "degreeFahrenheit" and ToUnit as "degreeCelsius".	37	degreeFahrenheit	degreeCelsius	37			
2	ConvertTemp_V_002	Conversion from Positive Integer Fahrenheit to negative Integer Celsius	Y	1) Temperature should be positive Integer 2) FromUnit should be "degreeFahrenheit" 3) ToUnit should be "degreeCelsius"	Send request with positive value greater than 0 for Temperature and FromUnit as "degreeFahrenheit" and ToUnit as "degreeCelsius".	37	degreeFahrenheit	degreeCelsius	37			
3	ConvertTemp_V_003	Conversion from negative Integer Fahrenheit to negative Integer Celsius	Y	1) Temperature should be negative Integer 2) FromUnit should be "degreeFahrenheit" 3) ToUnit should be "degreeCelsius"	Send request with positive value less than or equal to 0 for Temperature and FromUnit as "degreeFahrenheit" and ToUnit as "degreeCelsius".	-37	degreeFahrenheit	degreeCelsius	-37			
4	ConvertTemp_V_004	Conversion from Positive Double Fahrenheit to Positive Double Celsius	Y	1) Temperature should be positive Double 2) FromUnit should be "degreeFahrenheit" 3) ToUnit should be "degreeCelsius"	Send request with positive value greater than 37.5 for Temperature and FromUnit as "degreeFahrenheit" and ToUnit as "degreeCelsius".	70.87	degreeFahrenheit	degreeCelsius	70.87			
5	ConvertTemp_V_005	Conversion from Positive Double Fahrenheit to negative Double Celsius	Y	1) Temperature should be positive Double 2) FromUnit should be "degreeFahrenheit" 3) ToUnit should be "degreeCelsius"	Send request with positive value greater than 0 for Temperature and FromUnit as "degreeFahrenheit" and ToUnit as "degreeCelsius".	70.87	degreeFahrenheit	degreeCelsius	70.87			

[https://harinivasganapathy.files.wordpress.com/2016/05/1463054935\\_full.png](https://harinivasganapathy.files.wordpress.com/2016/05/1463054935_full.png)

The test case workbook is organized in a way that –

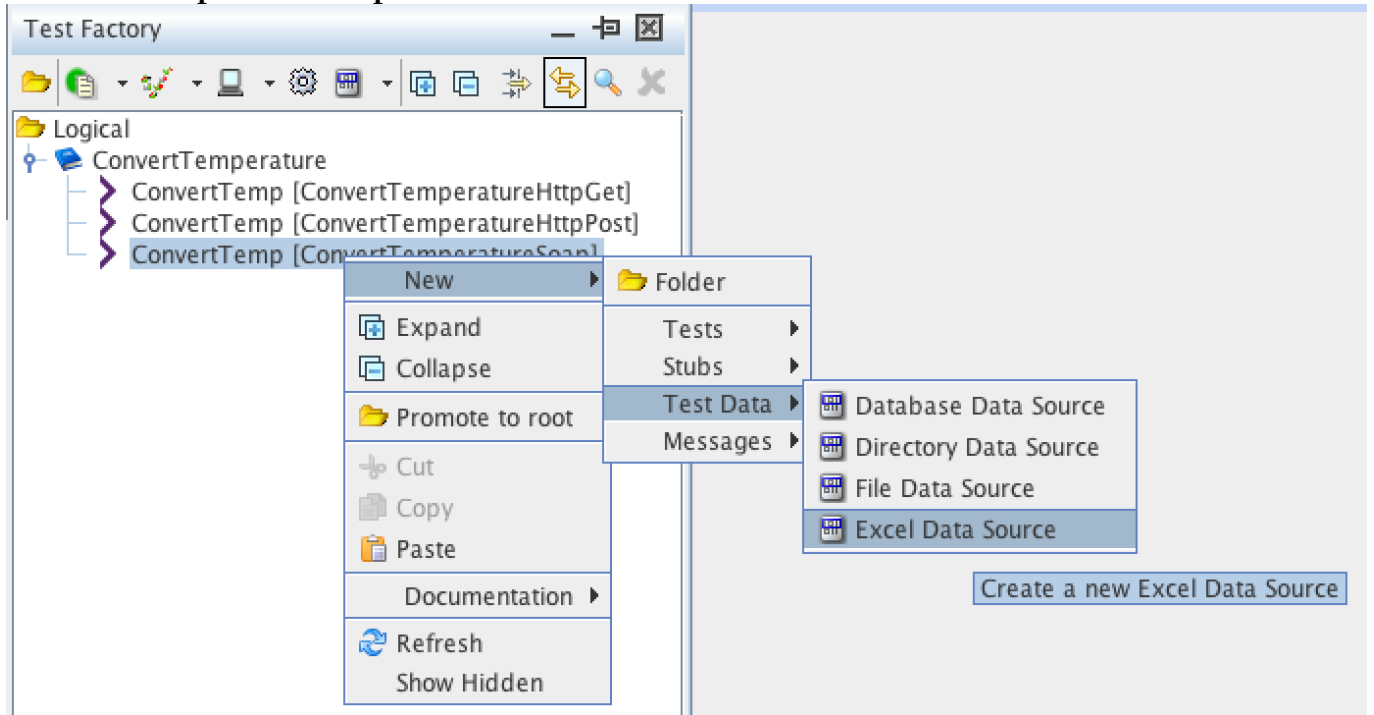
1. It has Unique test ID
2. It has unique test name that can mapped to ALM.
3. It has test execution control flag "Allow\_Execution". The idea of the flag to determine which tests to execute in RIT from Excel instead of in RIT. This gives a flexibility to control test execution even without opening RIT when run in Command line mode.
4. It has standard test condition and test steps to describe the purpose of the test.
5. Additionally it has the test Data section. The purpose of having test data part of test case is to isolate automated test from manual test by feeding test data directly into RIT tests using parameterization avoids manual test data setup in RIT tests.
6. It has baseline section that is referenced with in RIT to compare the values returned during execution with the baseline to determine if the test is Passed.
7. It has a result section to write back test result back to this Excel after execution of each step. For this purpose we will be using the ExcelFunctions available in GitHub RITOpenFramework.

### Note:

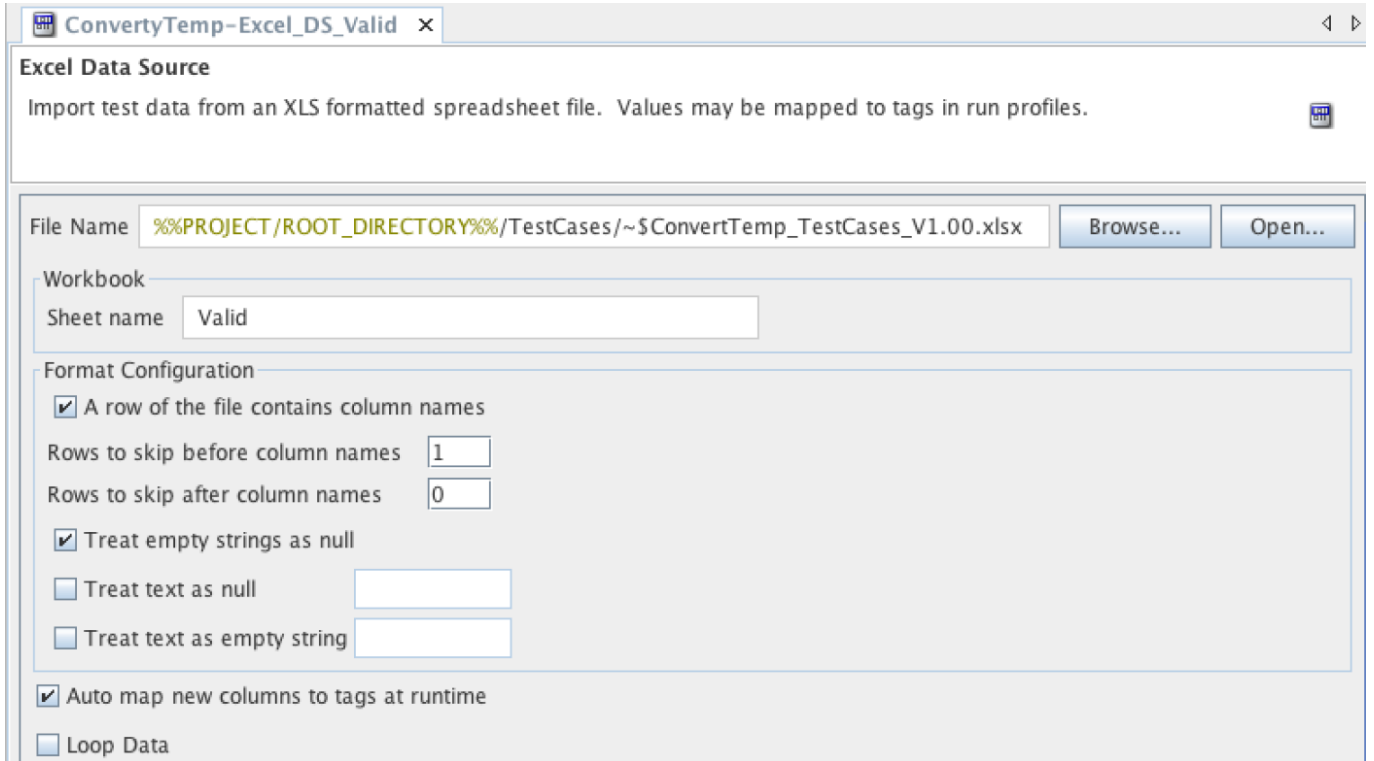
I have intentionally modified the value of ConvertTempResult for Tests 2, 4 & 6. So that we could have both results (Passed & Failed) updated in the test sheet and will use it for demonstrating other two functionality – ReadFromExcel & LookupExcelCol custom functions.

## Build Data source for parameterization

1. Create a new Excel data source by selecting and right click on the operation – **ConvertTemp [ConvertTemperatureSoap]** -> **New -> Tests Data -> Excel Data Source**



2. Give it name like Convert-Temp\_Excel\_DS
3. Double-Click the newly created Data Source to configure with the Excel workbook downloaded from GitHub page.
4. Click Browse on the Data source Editor and navigate to the Test case workbook under the Test case folder.
5. Specify the test sheet name as – “Valid”
6. Ensure “A row of the file contains column names” is checked.
7. Mention 1 in the “Rows to skip before column names”.



8. Click refresh to preview the data from the work book

Preview (max 25 rows)

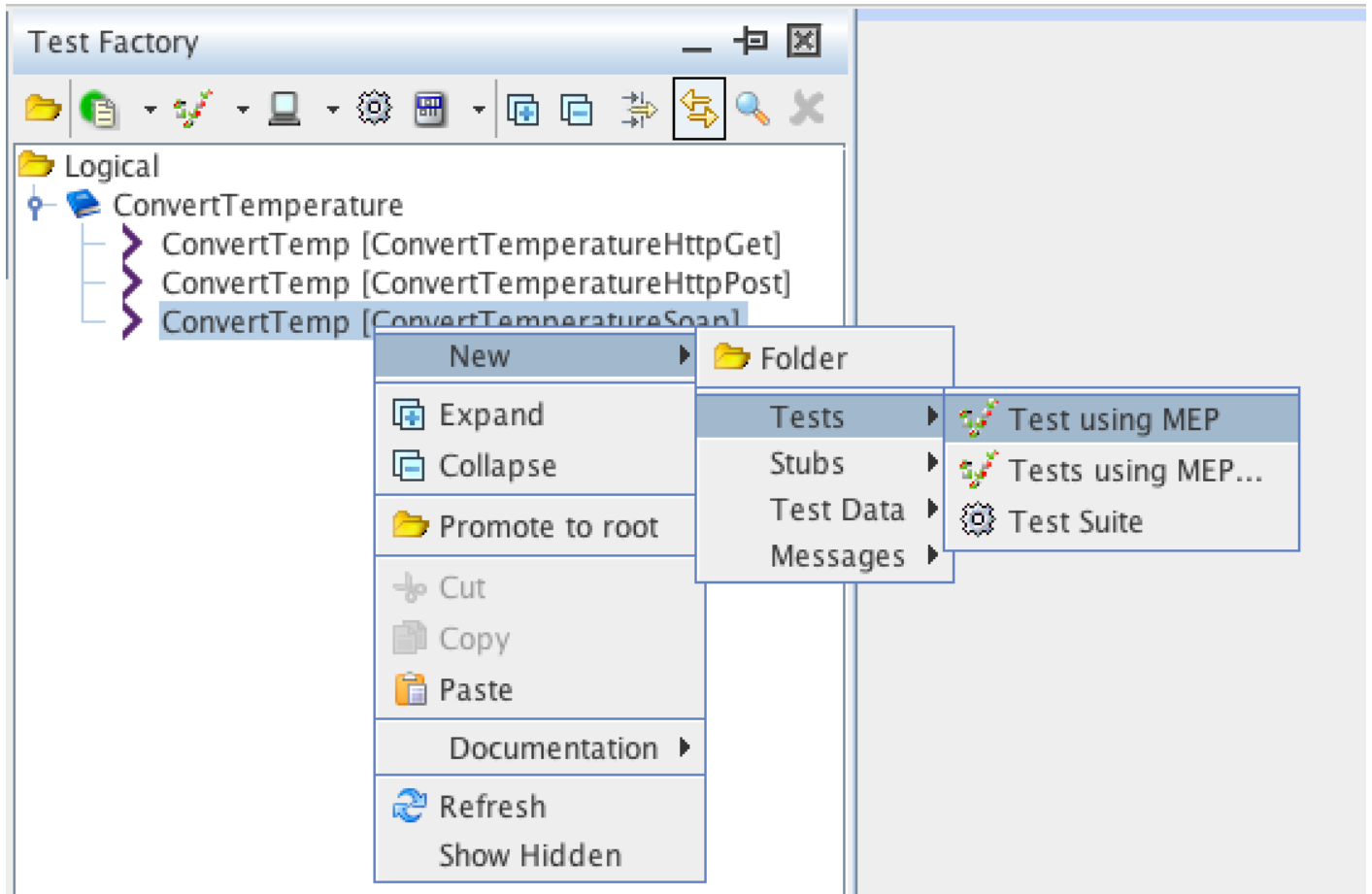
Test_ID	Name	Scenario	Allow_Execution?	Condition	Steps	Temperature	FromUnit	ToUnit	ConvertTempResult	Actual Value	Result	Comments
2	Convert...	Conversion f...	Y	1) Temperature s...	Send request wit...	37	degreeFahrenheit	degreeCelsius	2.7777777777777715			
3	Convert...	Conversion f...	Y	1) Temperature s...	Send request wit...	-37	degreeFahrenheit	degreeCelsius	-38.333333333333343			
4	Convert...	Conversion f...	Y	1) Temperature s...	Send request wit...	70.87	degreeFahrenheit	degreeCelsius	21.594444444444434			
5	Convert...	Conversion f...	Y	1) Temperature s...	Send request wit...	70.87	degreeFahrenheit	degreeCelsius	21.594444444444434			
6	Convert...	Conversion f...	Y	1) Temperature s...	Send request wit...	-14.76	degreeFahrenheit	degreeCelsius	-25.97777777777776			
7	Convert...	Conversion f...	Y	1) Temperature s...	Send request wit...	37	degreeCelsius	degreeFahrenheit	2.222222222222217			
8	Convert...	Conversion f...	Y	1) Temperature s...	Send request wit...	37	degreeCelsius	degreeFahrenheit	2.222222222222217			

Refresh Copy column names to clipboard

## Test Building in Test Factory

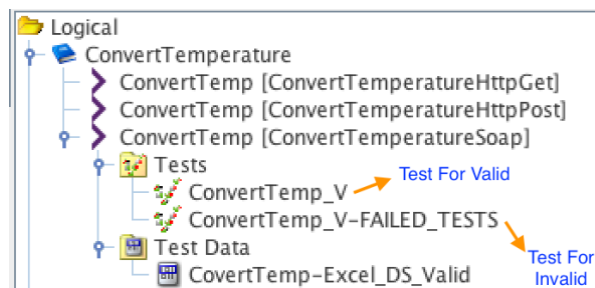
### Create Test Cases in RIT

- Switch to Test Factory view in RIT
- Select and right click *ConvertTemp [ConvertTemperatureSoap]* operation from the Test Factory explorer pane.



([https://harinivasganapathy.files.wordpress.com/2016/05/1463056850\\_full.png](https://harinivasganapathy.files.wordpress.com/2016/05/1463056850_full.png))

- o Name the test something like – *ConvertTemp\_Valid* to denote this test handles all positive conditions of the operation. Using this test we will demonstrate WriteToExcel Custom Function behavior.



### Edit to Construct Test using Actions and Parameterization

- o From the Excel Data Source Tab – after refreshing to preview the Data in the workbook sheet. Click on 'Copy Column names to clipboard' button.
- o Open the *ConvertTemp\_Valid* by double clicking in under the Tests folder in Test Factory Pane.

- Open the Tag Editor and click on the Paste Icon to paste the Excel Column headers as Tags to

Name	Description	Default Value
Actual_Value		
Allow_Execution?		
Comments		
Condition		
ConvertTempResult		
FromUnit		
ITR_Result		
Name		
Result		
Scenario		
Steps		
Temp		
Temperature		
Test_ID		
ToUnit		

the test.


- Under the Steps tab, in the test editor, select the Test Steps section. And add Iterate Test Data

The screenshot shows the Test Editor interface with the 'Steps' tab selected. The 'Test Steps' section is expanded, showing an 'Iterate Test Data' action added to the flow. The flow includes an 'Initialize' action and an 'Iterate Test Data' action over 'ConvertTemp-Excel\_DS\_Valid'.


from 'Flow' Menu.

- Double Click Iterate Test Data action to configure it.

- o Click 'Browse' to select the excel data source we create before.

**Iterate Test Data**  
Iterate over a test data set 

**Config** | Filter | Store

Test data set  CovertTemp-Excel\_DS\_Valid

Group data by column

Iterations

**Pacing**

Enable pacing

Pacing mode

Period (seconds)

**Runtime Settings**

Creates new test iteration

Continue on fail

**Iteration Timing**

Limit the length of each iteration

Maximum iteration time in seconds

Limit the length of the entire iterate action

Maximum total time in seconds

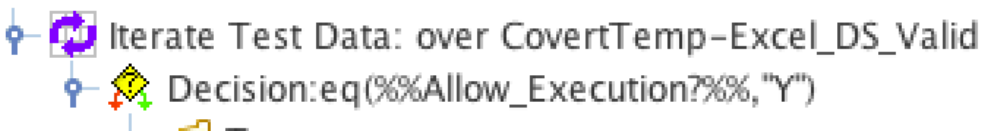
- o Switch to the 'Store' Tab and ensure all Excel Column Headers are mapped to the tags in the Tag Store. There might be tags in Test that shows 'No Mapping'. It is ok if there is no matching



columns in the Excel workbook. They might be added for other actions in the test.

Tag name	Data
Condition	Condition
Result	Result
Comments	Comments
Scenario	Scenario
FromUnit	FromUnit
Temperature	Temperature
Name	Name
ConvertTempResult	ConvertTempResult
Steps	Steps
Test_ID	Test_ID
ToUnit	ToUnit
Allow_Execution?	Allow_Execution?
ITR_Result	No mapping
Actual_Value	No mapping
Temp	No mapping

- Purpose of this action is to iterate every row in the excel work book, pick up each test case per iteration and perform test execution and validation steps.
- Next add a decision action from the 'Flow' menu.



- This action checks if a particular test is to be executed based on how the control flag – Allow\_Execution? is set in test data sheet.

**Decision**

Choose evaluation expressions to determine test execution flow. Use predefined functions, custom plug-ins and tagged data.

---

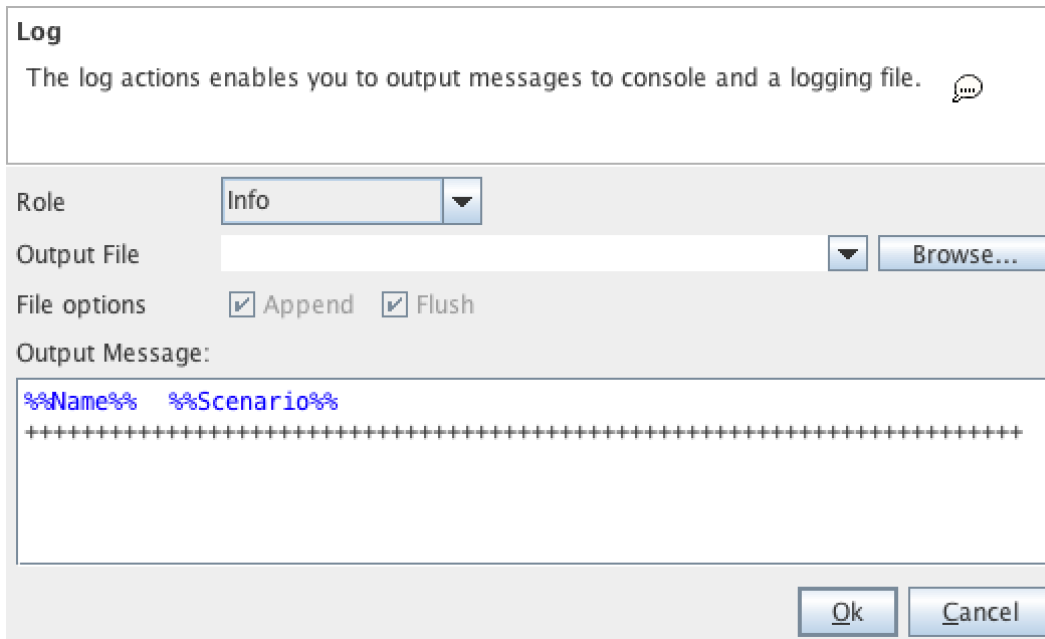
Enter a description of what the script is intended to do

---

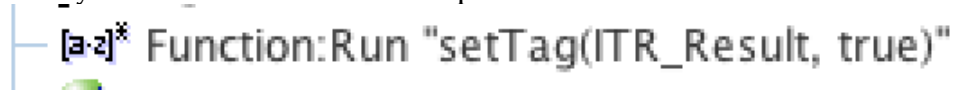
Script Language: Legacy

`eq(%%Allow_Execution?%%, "Y")`

- In the true part we'll add and configure the test action steps required to trigger the test and validate it.
- In the false part we add a Warning Message that this is not marked for execution and so it is skipped.
- We add a Log action to output a message to the RIT console to display the Test name and scenario for readability and identifying the message for each test.



- In RIT we can capture the overall result of a test, but cannot capture a result of iteration. Hence we create a Tag ITR\_Result to set and use for this purpose.
- We begin our test implementation by setting the tag ITR\_Result as 'True'. We do this at the start of every iteration so that result of previous iteration is not carried forward.



- Then we add Send Request and Recieve Reply Step.



- We add the below XML to the Request

```
<?xml version="1.0" encoding="utf-8"?><soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soap:Body>
<ConvertTemp xmlns="http://www.webserviceX.NET/">
<Temperature>%%Temperature%%</Temperature>
<FromUnit>%%FromUnit%%</FromUnit>
<ToUnit>%%ToUnit%%</ToUnit>
</ConvertTemp>
</soap:Body>
</soap:Envelope>
```

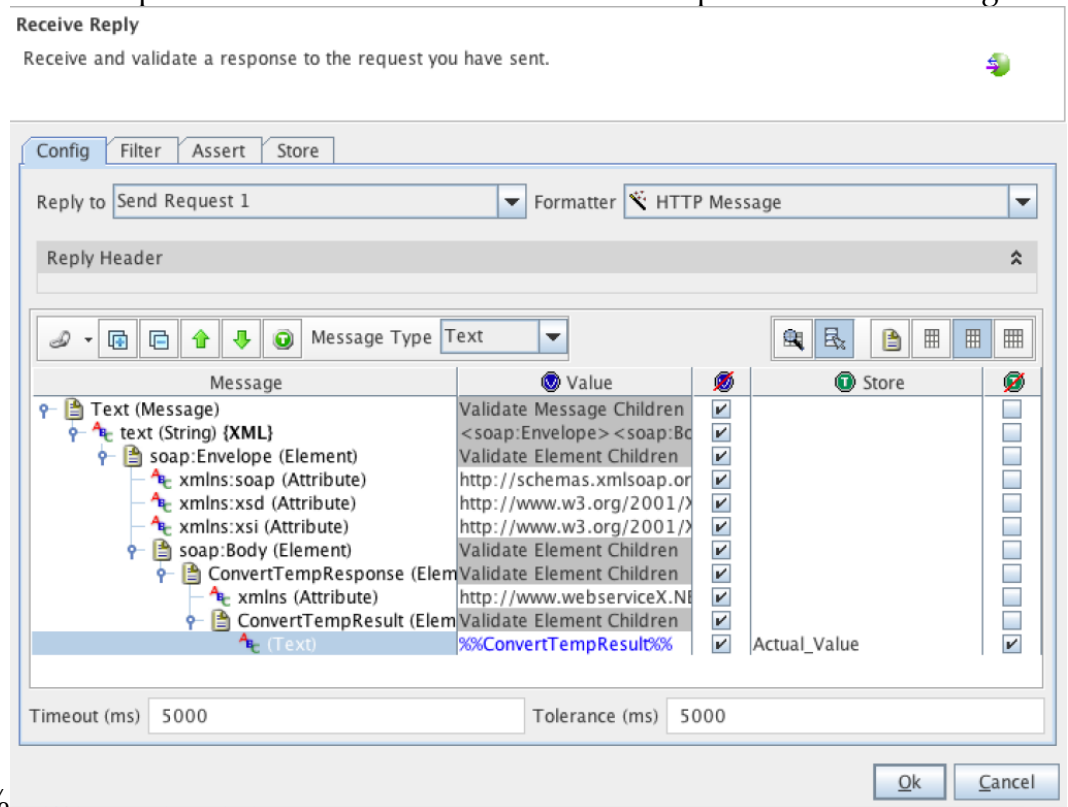
- We add the below XML to the Response

```

<?xml version="1.0" encoding="utf-8"?><soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soap:Body>
<ConvertTempResponse xmlns="http://www.webserviceX.NET/
<ConvertTempResult>%%ConvertTempResult%%</ConvertTempResult
</ConvertTempResponse>
</soap:Body>
</soap:Envelope>

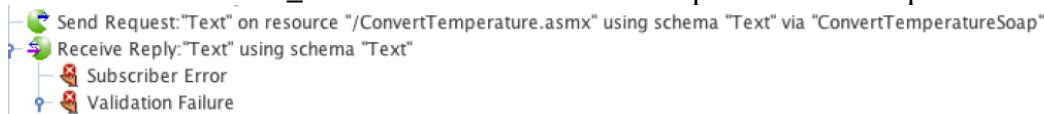
```

- We validate if ConvertTempResult is as per expectation by comparing it with the %%ConvertTempResult%% tag that comes from the Excel Data sheet. This is done by mentioning this Tag in the validate tab of the field editor for 'Equality'.
- Additionally configure the response to store the result of ConvertTempResult field into Tag



%%Actual\_Value%%

- If the Response validation fails the test stops and we won't be able to access or display or validate the %%ITR\_Result%%. So add a failure path for the response.



- Under 'Validation Failure' add a Function action to set the %%ITR\_Result%% as false.
- Add another function action to write that result to Excel worksheet.

```

writeToExcel("%PROJECT/ROOT_DIRECTORY%/TestCases/ConvertTemp_TestCases_V1.00.
xlsx",Valid,regex(add(%TEST/ITERATION/NUMBER%,1),"\\d+",1),L,FAILED)

```

**First parameter** –

%%PROJECT/ROOT\_DIRECTORY%%/TestCases/ConvertTemp\_TestCases\_V1.00.xlsx – points to the Excel workbook file.

**Second parameter** – *Valid* – refers to the sheet in excel workbook

**Third parameter** – *regex(add(%%TEST/ITERATION/NUMBER%%,1),"\\d+",1)* – Retrieves the Iteration number that is used to refer as the row number.

**Fourth parameter** – *L* – refers to the Column 'Result' in the Excel Workbook.

**Fifth parameter** – *FAILED* – refers to the value written to the cell identified by third and fourth parameter.

- Same way we add Function action to include the Custom Function as below –

```
writeToExcel("%%PROJECT/ROOT_DIRECTORY%%/TestCases/ConvertTemp_TestCases_V1.00.xlsx",Valid,regex(add(%%TEST/ITERATION/NUMBER%%,1),"\\d+",1),K,%%Actual_Value%  
)
```

The only difference from earlier function is the fourth and fifth parameter

**Fourth parameter** – *L* – refers to the Column 'Actual\_Value' in the Excel Workbook.

**Fifth parameter** – *%%Actual\_Value%%* – refers to the value written to the cell identified by third and fourth parameter. Note that *%%Actual\_Value%%* comes from the actual response received.

- Add another third function action to write the timestamp into the Comments column in the Test Data sheet.


```
writeToExcel("%%PROJECT/ROOT_DIRECTORY%%/TestCases/ConvertTemp_TestCases_V1.00.xlsx",Valid,regex(add(%%TEST/ITERATION/NUMBER%%,1),"\\d+",1),M,%%SYSTEM/CURRENT_DATE_TIME%%)
```

The only difference from earlier function is the fourth and fifth parameter


**Fourth parameter** – *M* – refers to the Column 'Comments' in the Excel Workbook.

**Fifth parameter** – *%%SYSTEM/CURRENT\_DATE\_TIME%%* – refers to the timestamp value written to the cell identified by third and fourth parameter.

- As a last step inside the Failure path of the Response add a Fail action, without which even if the validation fails and Validation Failure Path is executed overall Test Result will be always be

 Validation Failure

- [a-z]\* Function:Run "setTag(ITR\_Result,false)"
- [a-z]\* Function:Run "writeToExcel("%%PROJECT/ROOT\_DIRECTORY%%/"
- [a-z]\* Function:Run "writeToExcel("%%PROJECT/ROOT\_DIRECTORY%%/"
- [a-z]\* Function:Run "writeToExcel("%%PROJECT/ROOT\_DIRECTORY%%/"

 Fail:-NO MESSAGE DEFINED-

PASSED.

- o What follows next is the subsequent steps to Receive Reply. Which means actions to be performed if the Receive Reply validation is successful.
- o We add a decision to check if the ITR\_Result is still true, and if true we add 3 functions actions in the order
  1. Write the status as Passed to Column K in the Test Data sheet
  2. Write the Actual Value from the response in Column L
  3. Timestamp value in Column M

```
True
[!#] Function:Run "writeToExcel("%%PROJECT/ROOT_DIRECTORY%%/TestCases/ConvertTemp_TestCases_V1.00.xlsx",Valid,regex(Add000TEST/ITERATION/NUMBER%%,1),"\d+",1),L,PASSED)"
[!#] Function:Run "writeToExcel("%%PROJECT/ROOT_DIRECTORY%%/TestCases/ConvertTemp_TestCases_V1.00.xlsx",Valid,regex(Add000TEST/ITERATION/NUMBER%%,1),"\d+",1),K,%Actual_Value%%" (2)
[!#] Function:Run "writeToExcel("%%PROJECT/ROOT_DIRECTORY%%/TestCases/ConvertTemp_TestCases_V1.00.xlsx",Valid,regex(Add000TEST/ITERATION/NUMBER%%,1),"\d+",1),M,%SYSTEM/CURRENT_DATE,
```

- o False part can be left blank as the possible actions that can be included here are already included within Validation Failure path.

## Verifying the Results

Now that test is executed, open the excel workbook from the below location –

%%PROJECT/ROOT\_DIRECTORY%%/TestCases/ConvertTemp\_TestCases\_V1.00.xlsx

Test	Name	Scenario	Allow_Execution?	ConvertTempRes	Actual_Value	Result	Comments
1	ConvertTemp_V_001	Conversion from Positive Integer Fahrenheit to Positive Integer Celsius	Y	2.77777777777715	2.77777777777715	PASSED	14-May-2016 08:52:07
2	ConvertTemp_V_002	Conversion from Positive Integer Fahrenheit to negative Integer Celsius	Y	2.77777777777715	2.77777777777715	FAILED	14-May-2016 08:52:07
3	ConvertTemp_V_003	Conversion from negative Integer Fahrenheit to negative Integer Celsius	Y	-38.33333333333343	-38.33333333333343	PASSED	14-May-2016 08:52:08
4	ConvertTemp_V_004	Conversion from Positive Double Fahrenheit to Positive Double Celsius	Y	21.59444444444434	21.59444444444434	FAILED	14-May-2016 08:52:08
5	ConvertTemp_V_005	Conversion from Positive Double Fahrenheit to negative Double Celsius	Y	21.59444444444434	21.59444444444434	PASSED	14-May-2016 08:52:08
6	ConvertTemp_V_006	Conversion from negative Double Fahrenheit to negative Double Celsius	Y	-25.9777777777776	-25.9777777777776	FAILED	14-May-2016 08:52:08
7	ConvertTemp_V_007	Conversion from Positive Integer Celsius to Positive Integer Fahrenheit	Y	98.60000000000009	98.60000000000009	PASSED	14-May-2016 08:52:08

## Reading value from Excel Workbook

This far we used the WriteToExcel function available from RIT-OpenFramework (available in GitHub) to update status of every test to a excel workbook. We will continue to use other two functions – LookupExcelCol() and ReadFromExcel() to re-execute only failed tests from first run.

Remember that we purposefully had a wrong value in the ConvertTempResult column on the test data sheet so that we see how PASSED & FAILED and written back to the excel worksheet.

we will now demonstrate the use of LookupExcelCol() to check if a specific test is failed by checking the Result column having FAILED text. Now the tests that had intentionally left wrong ConvertTempResult is corrected. If Result column has FAILED text we will attempt to re-executed the test.

				Test Data			Test Baseline			
Test	Name	Scenario	Allow_Execution?	Temperature	FromUnit	ToUnit	ConvertTempRes	Actual_Value	Result	Comments
1	ConvertTemp_V_001	Conversion from Positive Integer Fahrenheit to Positive Integer Celsius	Y	37	degreeFahrenheit	degreeCelsius	2.7777777777777775	2.7777777777777775	PASSED	15-May-2016 12:38:00
2	ConvertTemp_V_002	Conversion from Positive Integer Fahrenheit to negative Integer Celsius	Y	37	degreeFahrenheit	degreeCelsius	2.7777777777777775	2.7777777777777775	FAILED	15-May-2016 12:38:00
3	ConvertTemp_V_003	Conversion from negative Integer Fahrenheit to negative Integer Celsius	Y	-37	degreeFahrenheit	degreeCelsius	-38.33333333333334	-38.33333333333334	PASSED	15-May-2016 12:38:00
4	ConvertTemp_V_004	Conversion from Positive Double Fahrenheit to Positive Double Celsius	Y	70.87	degreeFahrenheit	degreeCelsius	22.594444444444434	21.594444444444434	FAILED	15-May-2016 12:38:00
5	ConvertTemp_V_005	Conversion from Positive Double Fahrenheit to negative Double Celsius	Y	70.87	degreeFahrenheit	degreeCelsius	21.594444444444434	21.594444444444434	PASSED	15-May-2016 12:38:01
6	ConvertTemp_V_006	Conversion from negative Double Fahrenheit to negative Double Celsius	Y	-14.76	degreeFahrenheit	degreeCelsius	-25.977777777777775	-25.977777777777775	FAILED	15-May-2016 12:38:01

## Building the Test

- Switch to the Test Factory
- In the Test Factory Pane -Right click the test – ‘ConvertTemp\_V’ and click ‘Copy’
- Right click the test folder and click ‘Paste’
- Select the newly created test – ‘ConvertTemp\_V (2)’ and right click to select ‘Rename’
- Give it a value – ‘ConvertTemp\_V-FAILED\_TESTS’
- The only change we will make it this test is modify the Design logic which checks if %%Allow\_Execution?%% is “Y”.
- Double click the decision to edit it.
- Add one more test condition by click the ‘Add’ button
- Add the below condition to it –

```
`eq(readFromExcel(%%PROJECT/ROOT_DIRECTORY%%/TestCases/ConvertTemp_TestCases_V1.00.xlsx,Valid,regex(add(%%TEST/ITERATION/NUMBER%%,1)," \d+",1),L),FAILED)`
```

In the above condition we are using the Excel function readFromExcel() to read the cell content marked by row and column.

This function takes 4 parameters –

First Parameter –

`%%PROJECT/ROOT_DIRECTORY%%/TestCases/ConvertTemp_TestCases_V1.00.xlsx`

Identifies the Excel file to read from.

Second Parameter – *Valid* identifies the sheet within the excel workbook.

Third parameter – *regex(add(%%TEST/ITERATION/NUMBER%%,1))* identifies the Row

Fourth parameter – *L* – identifies the Col

This Function call returns a value and we check this returned value against the Text – FAILED. This ensures only test which are allowed for execution and failed are executed.

## Executing the Test

- Save the test.
- Run the test by selecting 'Run' button from the tool bar or press F5.

From the console log we could see that only tests that failed was executed. Not all tests. Tests that were not executed just displayed a message – **[Warning] Test Not Executed**

```
<terminated> ConvertTemperature/ConvertTemp [ConvertTemperatureSoap]/ConvertTemp_V-FAILED_TESTS
[05/16/2016, 6:13:24.051 AM] Initializing...
[05/16/2016, 6:13:24.054 AM] Using environment: SIT
[05/16/2016, 6:13:24.054 AM] - - - - Starting main steps - - - -
[05/16/2016, 6:13:24.055 AM] Test No: 1
[05/16/2016, 6:13:24.076 AM] [Warning] Test Not Executed
[05/16/2016, 6:13:24.077 AM] Test No: 2
[05/16/2016, 6:13:24.103 AM] Send Request:"Text" on resource "/ConvertTemperature.asmx" using schema "Text" via "ConvertTemperatureSoap" - Send message
[05/16/2016, 6:13:24.338 AM] Receive Reply:"Text" using schema "Text" - Message validation passed
[05/16/2016, 6:13:24.338 AM] 2.777777777777715
[05/16/2016, 6:13:24.572 AM] Test No: 3
[05/16/2016, 6:13:24.589 AM] [Warning] Test Not Executed
[05/16/2016, 6:13:24.590 AM] Test No: 4
[05/16/2016, 6:13:24.614 AM] Send Request:"Text" on resource "/ConvertTemperature.asmx" using schema "Text" via "ConvertTemperatureSoap" - Send message
[05/16/2016, 6:13:24.694 AM] Receive Reply:"Text" using schema "Text" - Message validation passed
[05/16/2016, 6:13:24.694 AM] 21.594444444444434
[05/16/2016, 6:13:24.921 AM] Test No: 5
[05/16/2016, 6:13:24.936 AM] [Warning] Test Not Executed
[05/16/2016, 6:13:24.937 AM] Test No: 6
[05/16/2016, 6:13:24.960 AM] Send Request:"Text" on resource "/ConvertTemperature.asmx" using schema "Text" via "ConvertTemperatureSoap" - Send message
[05/16/2016, 6:13:25.039 AM] Receive Reply:"Text" using schema "Text" - Message validation passed
[05/16/2016, 6:13:25.039 AM] -25.9777777777776
[05/16/2016, 6:13:25.275 AM] Test No: 7
[05/16/2016, 6:13:25.293 AM] [Warning] Test Not Executed
[05/16/2016, 6:13:25.294 AM] Test No: 8
[05/16/2016, 6:13:25.317 AM] [Warning] Test Not Executed
[05/16/2016, 6:13:25.318 AM] Test No: 9
[05/16/2016, 6:13:25.342 AM] [Warning] Test Not Executed
[05/16/2016, 6:13:25.344 AM] Test No: 10
[05/16/2016, 6:13:25.370 AM] [Warning] Test Not Executed
[05/16/2016, 6:13:25.371 AM] Test No: 11
[05/16/2016, 6:13:25.395 AM] [Warning] Test Not Executed
[05/16/2016, 6:13:25.396 AM] Test No: 12
[05/16/2016, 6:13:25.420 AM] [Warning] Test Not Executed
[05/16/2016, 6:13:25.422 AM] [Passed] 12 iteration(s) completed successfully
Logging summary: Info (15), Warnings (9), Errors (0)
Overall status:SUCCESSFUL
```

Opening the excel sheet we could see that FAILED tests were updated as PASSED.

Test	Name	Scenario	Allow_Execution?	Test Data			Test Baseline		Actual_Value	Result	Comments
				Temperature	FromUnit	ToUnit	ConvertTempRes				
1	ConvertTemp_V_001	Conversion from Positive Integer Fahrenheit to Positive Integer Celsius	Y	37	degreeFahrenheit	degreeCelsius	2.7777777777777775	2.7777777777777775	PASSED	15-May-2016 12:38:00	
2	ConvertTemp_V_002	Conversion from Positive Integer Fahrenheit to negative Integer Celsius	Y	37	degreeFahrenheit	degreeCelsius	2.7777777777777775	2.7777777777777775	PASSED	16-May-2016 06:13:24	
3	ConvertTemp_V_003	Conversion from negative Integer Fahrenheit to negative Integer Celsius	Y	-37	degreeFahrenheit	degreeCelsius	-38.33333333333343	-38.33333333333343	PASSED	15-May-2016 12:38:00	
4	ConvertTemp_V_004	Conversion from Positive Double Fahrenheit to Positive Double Celsius	Y	70.87	degreeFahrenheit	degreeCelsius	21.594444444444434	21.594444444444434	PASSED	16-May-2016 06:13:24	
5	ConvertTemp_V_005	Conversion from Positive Double Fahrenheit to negative Double Celsius	Y	70.87	degreeFahrenheit	degreeCelsius	21.594444444444434	21.594444444444434	PASSED	15-May-2016 12:38:01	
6	ConvertTemp_V_006	Conversion from negative Double Fahrenheit to negative Double Celsius	Y	-14.76	degreeFahrenheit	degreeCelsius	-25.97777777777776	-25.97777777777776	PASSED	16-May-2016 06:13:25	

This concludes our tutorial on using Excel Function to update test results to Excel and Read back from it. The same concept can be used to write values to a separate sheet should we have to maintain a separate report of execution status. Also remember to log bugs or feature / enhancement request on Excel Custom Functions on GitHub [Issues page \(https://github.com/harinivas-ganapathy/RIT-OpenFramework/issues\)](https://github.com/harinivas-ganapathy/RIT-OpenFramework/issues).



APRIL 24, 2016 MAY 3, 2016 / HARINIVAS GANAPATHY

## Rational integration tester – A Guide to Visual Scripting Experience

*The idea of this is post is help any beginners with little programming experience, who start with RIT a fresh, to have a feel of what RIT offers for Scripting to automate Web Services Testing. This post also portraits RIT usage in a Developers perspective. The post also highlights advanced RIT features, usage with examples for other RIT users who have medium to advanced exposures.*

When you have started with IBM’s Rational Integration Tester product or referred the product documentation several times while working on a project, you would have come across this statement, or told by a tutor or had a feeling that

*“RIT is a script free automation tool”*

This statement is both *true* and *false*. *How?* is the entire idea of the discussion. RIT offers two kinds of experience

- o Basic Beginner Mode



- Core Pro Mode

Beginner mode is for novice users who has just started their journey. For such users RIT offers a Visual Basic kind of experience where there are visual graphical elements for almost every primary fundamental programming constructs. Like Controls in VB, in RIT they are referred as *field actions*. Just like in VB how we drag elements and place them in applications forms, in RIT field actions can be visually selected and added inside tests with clickable configuration changes. They can also be repositioned by drag 'n' drop. In this fashion it's true that you can use RIT to test and automate Web Services application without even writing a single piece of code. However the power of RIT is its abilities and flexibility in pro-scripting mode. In order to provide a script free experience RIT has concealed its true power so that it looks simple to use the features and have a test ready in no time. On first look it would seem so obvious and quite deceiving that RIT is truly scripting free test automation product. But on the contrary it is a full featured true scripting compatible test development environment. More over unlike other Testing products which supports either JavaScript or VBScript, RIT Script engine supports both JS and also any other scripting languages that support *jar* extensions like – Python, Groovy etc. It gives great flexibility in selecting the scripting language to use for automation. In addition to support scripting language RIT also supports building libraries in Java as *Custom Functions* that can be used anywhere within the test script. In the following section we elaborate the different visual elements in RIT that corresponds to basic programming constructs.

## Constants & Variables

Like in any programming language we need named placeholders to store values that can be processed later by the program, RIT offers *Tags* to store values in named placeholders that can be used later in the Tests.

*Note:*

RIT does not distinguishes between Constant and Variables. It leaves it to test designer to decide on how to handle. It is up to the designer to ensure the integrity of tags to treat it as Constant – *Just like in JavaScript.*

## Accessing Constants & Variables

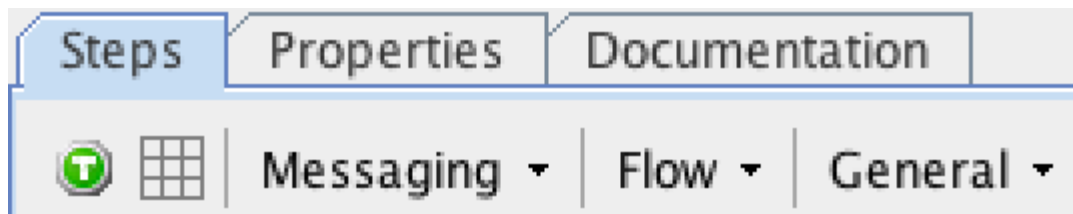
Once Values are stored in Tags they are accessed using a special pattern – name of the tag preceded and succeeded by '%%' like *%%variable%%*. This way RIT will know to look for a tag named variable in the tag table.

*Note:*

To reference Tags inside Function action for legacy – you can use the same pattern as above. However to within ECMAScript use the pattern *tags["variable"]*.

## Statement

Statements are the smallest standalone element of a programming language. Likewise *Action* are the smallest standalone elements within RIT that can be executed. Statements are made of other elements like expression, variable & operators, RIT actions include other actions from different category – Messaging, Flow & General in Starter Edition and Full version additionally has – BPM, GUI, Map & Stub behaviours.



## Operators

RIT is all about functions. Functions are the individual unit to perform computations within RIT tests. Everything in RIT is implemented as Functions. Operators used in programming statements are also designed, implemented and made available as Functions from “*Select Function*” context menu within *action* or *field* editors. Basic operators like addition, subtraction, multiplication and division are made available as Functions – Add, Subtract, Multiply & Division respectively. Below is the table showing different operators in a typical programming language and corresponding functions available in RIT and sample usage example.

Operator	Function	Usage	example	Result Returned
<b>addition</b>	add	add(expr,expr, ...)	add(1,1)	2.0
<b>Subtraction</b>	subtract	subtract(expr,expr)	subtract(1,1)	0.0
<b>Multiply</b>	multiply	multiply(expr,expr)	multiply(5,2)	10.0
<b>Division</b>	divide	divide(expr,expr)	divide(10,5)	2.0

*Note:*

The result of all mathematical functions always returns a decimal value. Most of times we would need Integer value like used to Iterate. In such case there is no direct way in RIT to convert decimal into integers.

## **Data Structures**

In general data structures specify ways a programming language provides to store and manipulate data. Examples – Array, List, HashMap, HashTable. Likewise RIT stores and process data in form of Tags. Yes, it's the same Tags we discussed in Section – Constants & Variables. RIT provides a mechanism to treat a Tag to store single Values as well as multiple values like Array. Tags that store multiple values are called List Tags. Unlike normal Tags, List Tags cannot be created in the Tag Store. Instead they have to be configured in the store tab of field editor and select the checkbox "Append to list of values". This will ensure multiple values are stored in the same Tag like arrays without existing value. There is also another way to create and add values to list using ECMAScript which we will discuss in later post.

Field Editor

Name

Type

Value Validate Store

Field is optional

Ignore rule cache

Store copy of field with parent 'tns:FromCurrency' in tag 'tns:FromCurrency'

Action Type

Description

Tag

Append to list of values

To access individual items in the List Tag, index numbers are used within set brackets before the trailing '%%' in the Tag reference. Example %%ListTag[0]%% gives the first value in the list, %%ListTag[1]%% gives the second value in the list. Index of ListTags can also be referenced using Other Tags. Example -Create a Tag for *index* in the Tag store and assign a value of zero inside a Function action using ECMAScript and remember to use Regex to trim the decimal zero.  

```
index=0;
index = regex(index,"\\d+",1)
```

**Function**  
Execute a function and optionally tag the results. Use predefined functions, custom plug-ins and tagged data. [a-z]\*

[a-z]\* Function  Store

Script Language:

```
index=0; index = regex(index, "\\d+", 1)
```

Values from the list tag can be accessed using the Index tag in *log* or *field* actions using the format – %%TagName[IndexTagName]%%. Example -%%ListTag[index]%%

**Log**  
The log actions enables you to output messages to console and a logging file. [a-z]\*

Role

Output File

File options  Append  Flush

Output Message:

```
First Item in the List : %%ListTag[index]%%
```

**Note:**

1. Index reference of ListTags always starts with zero.
2. Though List Tags seems like and can be treated like Arrays, actually, inside RIT, they are implemented as ListArrays in Java. Knowing this little secret will open doors to manipulate them with in ECMAScripts with function steps.
3. When referencing the ListTag using index, during test design, RIT displays it in Red-Error color to denote as if that is not vailable in the Tag Store. That’s because RIT treats “ListTag[index]” as a single tag during design. But it will work at runtime. So it is safe to ignore the color while using index for ListTag.

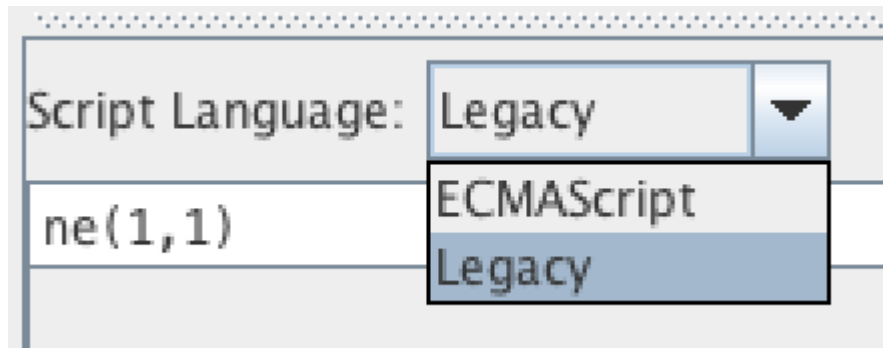
## Control Structures

Traditional programming languages has control structures like if, if-else, switch etc. RIT provides *Decision Flow* action to be used and configured like *if*, *if-else*, *nested if* and *Switch* Statements.. There is no dedicated *switch* statement but similar behavior can be simulated using Cascaded decision action.

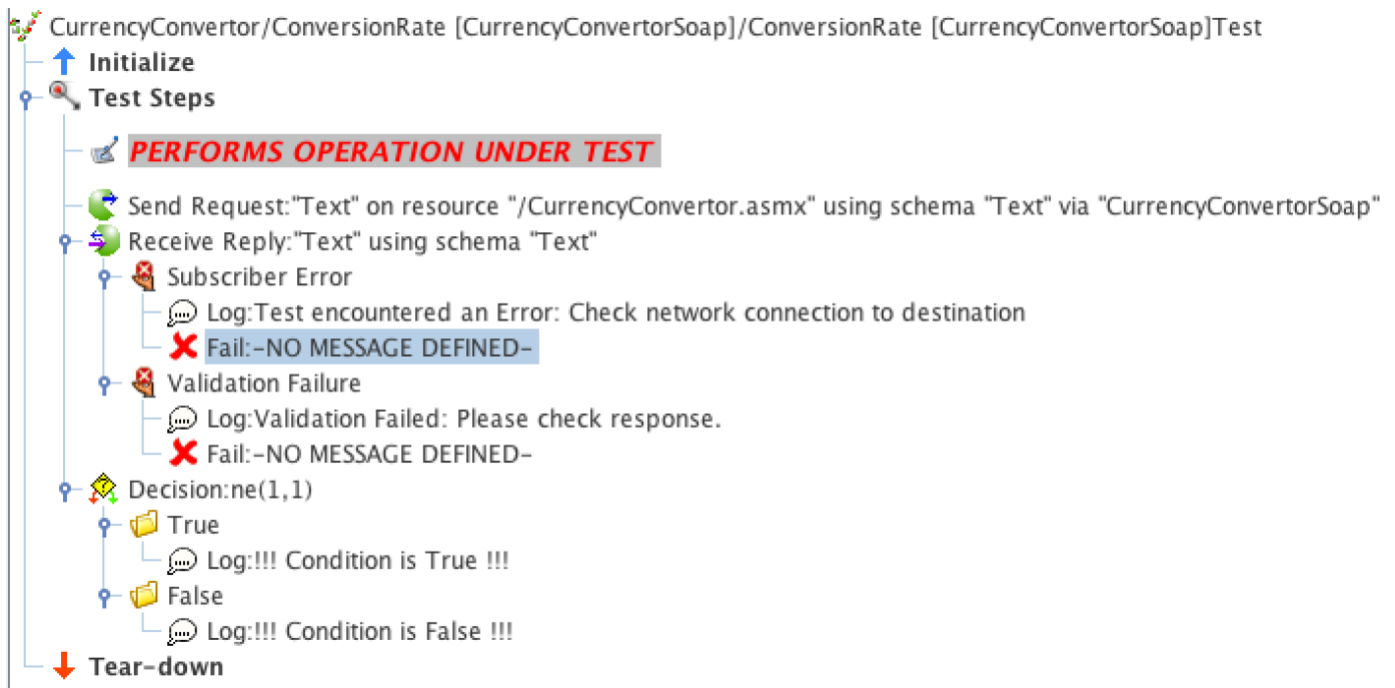
## if ... then ...

To make decision in our tests RIT provides *decision* action. There is 3 parts to this action –

1. Condition – To check for condition based on which either actions inside True or False are executed.



2. True – If the condition evaluates to true, all statements within 'True' part is executed.
3. False – If the condition evaluates to false all statements within 'False' part is executed.



To help with comparing values within conditions, RIT provides below condition functions –

Operator	Function	Usage	Example	Evaluated Result
Equal To	eq	eq(Value1,Value2)	eq(1,1)	TRUE
Not Equal To	ne	ne(Value1,Value2)	ne(1,2)	FALSE
Less than	lt	lt(value1,value2)	lt(1,2)	TRUE
Greater than	gt	gt(Value1,Value2)	gt(2,2)	FALSE
Less than or Equal To	le	le(Value1,Value2)	le(2,2)	TRUE
Greater than or Equal To	ge	ge(Value1,Value2)	gt(2,2)	TRUE

*Note:* There is no if only matching conditional in RIT. *Decision* action always has *true* block and the *false* block. Though to replicate the if only behavior the false block can be left empty.

The always added false block step can put to productive use. Add a log statement within false block to check if decision condition evaluated is true or false. Loops

## Loop

RIT provides 3 actions to support 3 types of looping support by modern programming languages –

1. For Loop
2. While Loop
3. Loop through data collection

## For Loop –

RIT provides Iterate test action that simulates For loop. Number of times the loop to be repeated is mentioned within the Iteration text field, example 1-5.

The image shows a screenshot of the RIT (Rational Integration Tester) interface. On the left, a test suite tree is visible for 'CurrencyConverter/ConversionRate [CurrencyConvertoSoap]/ConversionRate [Cu...'. The 'Test Steps' section includes: 'Initialize', 'Fetch Test Data: from "Test Data Input"', 'Iterate: 1-5', 'Log: This Loops 5 Times', and 'Tear-down'. The 'Iterate: 1-5' step is highlighted with a red box and the text 'IMITATES FOR LOOP'.

On the right, the configuration dialog for the 'Iterate' action is shown. It has two tabs: 'Config' and 'Store'. The 'Config' tab is active, showing the following settings:

- Iterations:** 1-5
- Pacing:**
  - Enable pacing
  - Pacing mode: Minimum iteration time
  - Period (seconds): 0
- Runtime Settings:**
  - Creates new test iteration
  - Continue on fail
- Iteration Timing:**
  - Limit the length of each iteration
    - Maximum iteration time in seconds: 0
  - Limit the length of the entire iterate action
    - Maximum total time in seconds: 0

Type	Task	Progress	Status
	rsionRate [CurrencyConvertorSoap]Test-3	100%	Passed

Console

```

<terminated> CurrencyConvertor/ConversionRate [CurrencyConvertorSoap]/ConversionRate [CurrencyConvertorSoap]Test-3
[04/24/2016, 6:20:07.846 PM] Initializing...
[04/24/2016, 6:20:07.868 PM] Using environment: Harinivass-MacBook-Pro-4
[04/24/2016, 6:20:07.869 PM] - - - Starting main steps - - -
[04/24/2016, 6:20:07.870 PM] Fetch Test Data:from "Test Data Input" Opening data source for initial use when no grouping
applied
[04/24/2016, 6:20:07.873 PM] This Loops 5 Times
[04/24/2016, 6:20:07.873 PM] This Loops 5 Times
[04/24/2016, 6:20:07.873 PM] This Loops 5 Times
[04/24/2016, 6:20:07.873 PM] This Loops 5 Times
[04/24/2016, 6:20:07.874 PM] This Loops 5 Times
[04/24/2016, 6:20:07.874 PM] [Passed] 5 iteration(s) completed successfully
Logging summary: Info (5), Warnings (0), Errors (0)
Overall status:SUCCESSFUL

```

Iteration can also be controlled by using Tags. Example – 1-%%Counter%%

```

CurrencyConvertor/ConversionRate [CurrencyConvertorSoap]/ConversionRate [Cu
↑ Initialize
Test Steps
  ↳ IMITATES FOR LOOP
  ↳ Fetch Test Data:from "Test Data Input"
  ↳ Iterate:1-%%Counter%%
  ↳ Log:Value of Counter: %%Counter%%
↓ Tear-down

```

Config Store

Iterations

Pacing

Enable pacing

Pacing mode

Period (seconds)

Runtime Settings

Creates new test iteration

Continue on fail

Iteration Timing

Limit the length of each iteration

Maximum iteration time in seconds

Limit the length of the entire iterate action

Maximum total time in seconds

```

<terminated> CurrencyConvertor/ConversionRate [CurrencyConvertorSoap]/ConversionRate [CurrencyConvertorSoap]Test-3
[04/24/2016, 6:41:24.776 PM] Initializing...
[04/24/2016, 6:41:24.780 PM] Using environment: Harinivass-MacBook-Pro-4
[04/24/2016, 6:41:24.780 PM] - - - Starting main steps - - -
[04/24/2016, 6:41:24.781 PM] Value of Counter: 5
This Loops 5 Times
[04/24/2016, 6:41:24.781 PM] Value of Counter: 5
This Loops 5 Times
[04/24/2016, 6:41:24.782 PM] Value of Counter: 5
This Loops 5 Times
[04/24/2016, 6:41:24.782 PM] Value of Counter: 5
This Loops 5 Times
[04/24/2016, 6:41:24.782 PM] Value of Counter: 5
This Loops 5 Times
[04/24/2016, 6:41:24.783 PM] [Passed] 5 iteration(s) completed successfully
Logging summary: Info (5), Warnings (0), Errors (0)
Overall status:SUCCESSFUL

```

## While Loop



RIT provides 'Iterate while' action to simulate While Loop behavior in tests. Iterate while takes one or more conditions and executes as long as the condition is true. Remember to avoid infinite loop alter the value of condition variable within the Iterate while action. For Example in below test – Counter variable is initialized to 0 and a condition 'counter values less than or equal to 5' is added to Iterate while action and the counter value is incremented by 1 after every iteration within the iterate while action so that the Log statements within it are executed 5 times.

The screenshot shows the RIT test configuration interface. On the left, a tree view shows the test steps: Initialize, Test Steps (containing 'IMITATES FOR LOOP'), Function:Run "setTag(Counter, 0)", Iterate While:le(Counter, 5), Log:Value of Counter: %%Counter%%, and Function:Run "setTag(Counter, add(Counter, 1))". On the right, the 'Config' panel is visible, showing the condition 'le(Counter, 5)' and various settings like Pacing, Runtime Settings, and Iteration Timing.

Type	Task	Progress	Status
	rsionRate [CurrencyConvertoSoap]Test-3	100%	Passed

The screenshot shows the RIT console output for the test. The log messages are as follows:

```

<terminated> CurrencyConverto/ConversionRate [CurrencyConvertoSoap]/ConversionRate [CurrencyConvertoSoap]Test-3
[04/25/2016, 1:51:21.649 AM] Initializing...
[04/25/2016, 1:51:21.652 AM] Using environment: Harinivass-MacBook-Pro-4
[04/25/2016, 1:51:21.653 AM] - - - - Starting main steps - - - -
[04/25/2016, 1:51:21.654 AM] Value of Counter: 0
    This Loops 5 Times
[04/25/2016, 1:51:21.656 AM] Value of Counter: 1.0
    This Loops 5 Times
[04/25/2016, 1:51:21.657 AM] Value of Counter: 2.0
    This Loops 5 Times
[04/25/2016, 1:51:21.658 AM] Value of Counter: 3.0
    This Loops 5 Times
[04/25/2016, 1:51:21.660 AM] Value of Counter: 4.0
    This Loops 5 Times
[04/25/2016, 1:51:21.661 AM] Value of Counter: 5.0
    This Loops 5 Times
[04/25/2016, 1:51:21.662 AM] [Passed] 1 iteration(s) completed successfully
    Logging summary: Info (6), Warnings (0), Errors (0)
    Overall status:SUCCESSFUL
  
```

## Iterate Data Collections

Like we use to iterate for every value in data collections like in Array, List or Dictionaries in modern programming languages, RIT provides a mechanism to iterate through values in ListTags.Example:

## Approach 1

**STEP 1:** Configure Test Data Set as shown below –

File Name: /Users/harinivas/Documents/CurrencyConvertor\_TestData.csv

Format Configuration

- A row of the file contains column names
- Rows to skip before column names: 0
- Rows to skip after column names: 0

Delimiter Options

- Comma
- Tab
- Space
- Semi-colon
- Other
- Ignore delimiters within quoted strings
- Column count: [ ]
- Calculate

Treat empty strings as null

Treat text as null [ ]

Treat text as empty string [ ]

Auto map new columns to tags at runtime

Loop Data

Preview (max 25 rows)

FromCurrency	ToCurrency
USD	INR
AFA	ALL
DZD	ARS
BSD	XOF
VND	YER

Refresh Copy column names to clipboard

**STEP 2:** Iterate over the test data set created above **STEP 3:** Use *Function* actions to read individual tag values and store them into a tag as *List*. **STEP 4:** Use *Function* to calculate the number of items in the list using ECMAScript and `size()` function available in it. And create a new tag "index" to be used to iterate with in the list. Set the index tag to zero. Remember to trim the the decimal from the result of the Index. **STEP 5:** Use *Iterate While* action to check for the condition `index <= ListCount` (No of Items in the list `lt(%%index%%, %%ListCount%%)`)

## Iterate While

Repeat test steps while a specified condition is true.



**Config**

**Condition**

Enter a description of what the script is intended to do

Script Language: Legacy    'OR' Expressions

`lt(%%index%%,%%ListCount%%)`

**Pacing**

Enable pacing

Pacing mode: Minimum iteration time

Period (seconds): 0

**Runtime Settings**

Creates new test iteration

Continue on fail

**Iteration Timing**

Limit the length of each iteration

Maximum iteration time in seconds: 0

Limit the length of the entire iterate action

Maximum total time in seconds: 0

**STEP 6:** Within *Iterate while* use message action to *Send request* and *Receive Reply*

## Test

Construct a test by using one or more of the available actions. These can be inserted by using the buttons on the actions toolbar, or by using the context menu. Double click an action to edit it.



Steps Properties Documentation

Messaging ▾ Flow ▾ General ▾

CurrencyConvertor/ConversionRate [CurrencyConvertorSoap]/ConversionRate [CurrencyConvertorSoap]Test-Iterate\_List

- Initialize
- Test Steps
  - Iterate Test Data: over Test Data Input
    - Function:Run "%FromCurrency%", append to"xFromCurrency"
    - Function:Run "%ToCurrency%", append to"xToCurrency"
    - Log:New FromCurrency List %xFromCurrency%
    - Function:Run "ListCount=xFromCurrency.size(); index=0; index = regex(index,"\\d+",1)"
    - Function:-NO FUNCTION DEFINED-
    - Log:Size of FromCurrencyList %ListCount%
    - Iterate While:lt(%index%,%ListCount%)
      - Log:%xFromCurrency[index]%
      - Send Request:"Text" on resource "/CurrencyConvertor.asmx" using schema "Text" via "CurrencyConvertorSoap"
      - Receive Reply:"Text" using schema "Text"
      - Function:Run "regex(add(index,1), "\\d+",1)", store into"index"
- Tear-down

**STEP 7:** Inside of Send request List Tag and Index can be used to tag the input fields like in be screenshot.



### Send Request

Publish a message and wait for a response to be received. This can then be validated accordingly.



Config Value Store

Transport CurrencyConvertorSoap

Browse...

Formatter HTTP Message

#### Message Header

HTTP Properties

HTTP Headers

Resource Path /CurrencyConvertor.asmx

HTTP Method POST

Follow Redirects

Message Type Text



Message

Value

- Text (Message)
  - text (String) {Document-Literal SOAP}
    - ConversionRate\_\_INPUT\_\_ConversionRateSoapIn
      - tns:ConversionRate (Element)
        - xmlns:tns (Attribute)
        - tns:FromCurrency (Element)
        - (Text)
        - tns:ToCurrency (Element)
        - (Text)

Process Children	<input checked="" type="checkbox"/>
<ConversionRate__INPUT__ConversionRateSoapIn> <tns:Co	<input checked="" type="checkbox"/>
Process Children	<input checked="" type="checkbox"/>
Process Children	<input checked="" type="checkbox"/>
http://www.webserviceX.NET/	<input checked="" type="checkbox"/>
Process Children	<input checked="" type="checkbox"/>
Process Children	<input checked="" type="checkbox"/>
%%xFromCurrency[index]%%	<input checked="" type="checkbox"/>
Process Children	<input checked="" type="checkbox"/>
%%xToCurrency[index]%%	<input checked="" type="checkbox"/>

Ok

Cancel

```

Console
<terminated> CurrencyConverter/ConversionRate [CurrencyConverterSoap]/ConversionRate [CurrencyConverterSoap]Test-Iterate_List
[05/02/2016, 11:32:52.564 PM] Initializing...
[05/02/2016, 11:32:52.567 PM] Using environment: Harinivass-MacBook-Pro-4
[05/02/2016, 11:32:52.567 PM] - - - - Starting main steps - - - -
[05/02/2016, 11:32:52.569 PM] New FromCurrency List USD
New ToCurrency List INR
[05/02/2016, 11:32:52.571 PM] New FromCurrency List {USD, AFA}
New ToCurrency List {INR, ALL}
[05/02/2016, 11:32:52.572 PM] New FromCurrency List {USD, AFA, DZD}
New ToCurrency List {INR, ALL, ARS}
[05/02/2016, 11:32:52.574 PM] New FromCurrency List {USD, AFA, DZD, BSD}
New ToCurrency List {INR, ALL, ARS, XOF}
[05/02/2016, 11:32:52.576 PM] New FromCurrency List {USD, AFA, DZD, BSD, VND}
New ToCurrency List {INR, ALL, ARS, XOF, YER}
[05/02/2016, 11:32:52.583 PM] Size of FromCurrencyList 5
[05/02/2016, 11:32:52.584 PM] From Currency @ Index 0 USD
To Currency @ Index 0 INR
[05/02/2016, 11:32:52.594 PM] Send Request:"Text" on resource "/CurrencyConverter.asmx" using schema "Text" via "CurrencyConverterSoap" - Send message
[05/02/2016, 11:32:52.686 PM] Receive Reply:"Text" using schema "Text" - Message validation passed
[05/02/2016, 11:32:52.691 PM] From Currency @ Index 1 AFA
To Currency @ Index 1 ALL
[05/02/2016, 11:32:52.704 PM] Send Request:"Text" on resource "/CurrencyConverter.asmx" using schema "Text" via "CurrencyConverterSoap" - Send message
[05/02/2016, 11:32:52.788 PM] Receive Reply:"Text" using schema "Text" - Message validation passed
[05/02/2016, 11:32:52.793 PM] From Currency @ Index 2 DZD
To Currency @ Index 2 ARS
[05/02/2016, 11:32:52.814 PM] Send Request:"Text" on resource "/CurrencyConverter.asmx" using schema "Text" via "CurrencyConverterSoap" - Send message
[05/02/2016, 11:32:52.900 PM] Receive Reply:"Text" using schema "Text" - Message validation passed
[05/02/2016, 11:32:52.906 PM] From Currency @ Index 3 BSD
To Currency @ Index 3 XOF
[05/02/2016, 11:32:52.917 PM] Send Request:"Text" on resource "/CurrencyConverter.asmx" using schema "Text" via "CurrencyConverterSoap" - Send message
[05/02/2016, 11:32:52.998 PM] Receive Reply:"Text" using schema "Text" - Message validation passed
[05/02/2016, 11:32:53.004 PM] From Currency @ Index 4 VND
To Currency @ Index 4 YER
[05/02/2016, 11:32:53.015 PM] Send Request:"Text" on resource "/CurrencyConverter.asmx" using schema "Text" via "CurrencyConverterSoap" - Send message
[05/02/2016, 11:32:53.103 PM] Receive Reply:"Text" using schema "Text" - Message validation passed
[05/02/2016, 11:32:53.110 PM] [Passed] 5 iteration(s) completed successfully
Logging summary: Info (11), Warnings (0), Errors (0)
Overall status:SUCCESSFUL

```

Steps showing building up of List Tag using JavaScript

Values fetched using the index variable

## Approach 2:

Another straight forward approach is to avoid calculating the list tag size and using it within iterate while action. Instead just mention the list tag in the condition field like below – How ever remember that all steps within the Iterate while action will be repeated for every item in the list and we have to still use the index tag/value to access the individual elements in the list tag.

### Iterate While

Repeat test steps while a specified condition is true.



**Config**

Condition

Enter a description of what the script is intended to do

---

Script Language: Legacy 'OR' Expressions  Add Delete Test

%%FromCurrency%%

**BlockBlock** is a set of statements grouped together. in C / Java opening and closing curly braces are used to denote the beginning and end of a block. However RIT provides block as a visual element – Action Group – with a folder icon image to have test steps grouped and displayed like a tree

structure. Nested blocks are also possible.

## Function

Functions in programming languages are series of related steps (instructions) grouped together as a block to perform a task. RIT provides a visual element named 'Functions' to add functions to tests. This is available from the General Menu. Opening up the Function step provides two modes to work with them.

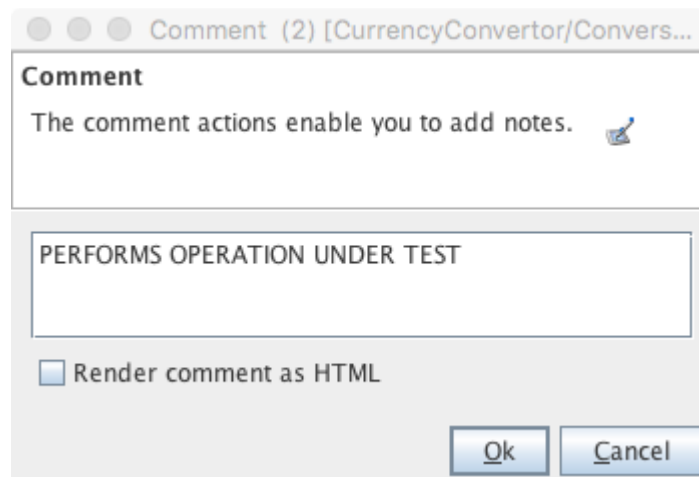
1. *ECMAScript* – allows to add JavaScript snippets to Function step.
2. *Legacy* – allows using built-in Functions from the library.

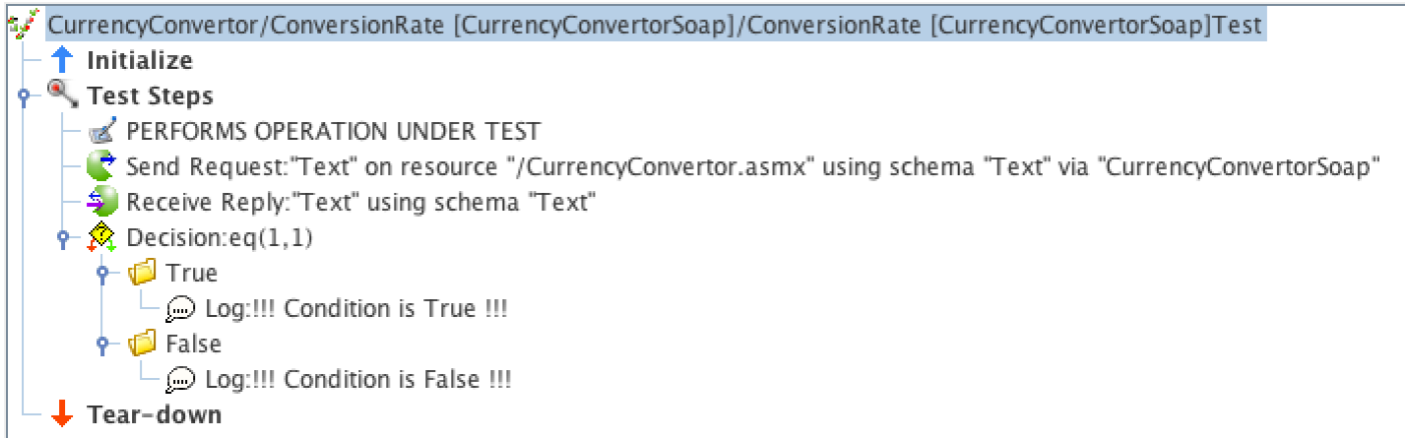
### *Note:*

The difference between using ECMAScript and Legacy is that the way Tags should be referenced. In ECMAScript they can be referenced just with their names or using notation 'tags["name"]'. RIT notation of tags – %%TagName%% will not work in ECMAScript and ECMAScript notation of referencing tag won't work in Legacy. One another difference is that functions supported by script engine (like JavaScript or python) will be available to use within ECMAScript along with built in library but only Functions from built-in library available for use in Legacy mode. We can of course create Custom Functions using eclipse and add them to built-in library.

## Comments

Adding comments to program helps in documentation and helps readability in programs. Just like it RIT supports comments by providing Comment action as a visual element that can be added as steps at point inside the test. Comment actions are not executed when test is run but only added to aid in readability of the test.





### Note

Unlike in programming languages where comments are just text, RIT allows text to be displayed as HTML Content. This allows to add Font Color and Text Background to signify the importance of the following test step and add more readability to tests.

Comment <HTML> <Title></Title> <Body><H3><FONT COLOR=RED bgcolor="silver"><B><I>PERFORMS OPER...

**Comment**

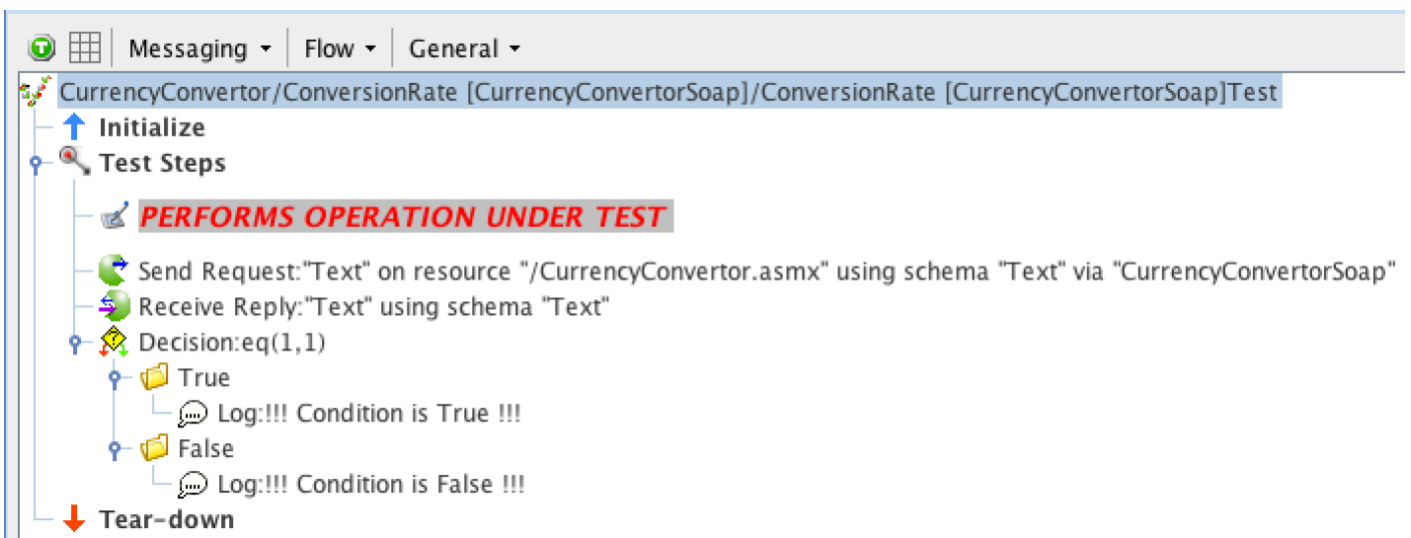
The comment actions enable you to add notes.

```

<HTML>
<Title></Title>
<Body><H3><FONT COLOR=RED bgcolor="silver"><B><I>PERFORMS OPERATION UNDER TEST </I></B></FONT></H3></Body>
</HTML>
  
```

Render comment as HTML

Ok Cancel



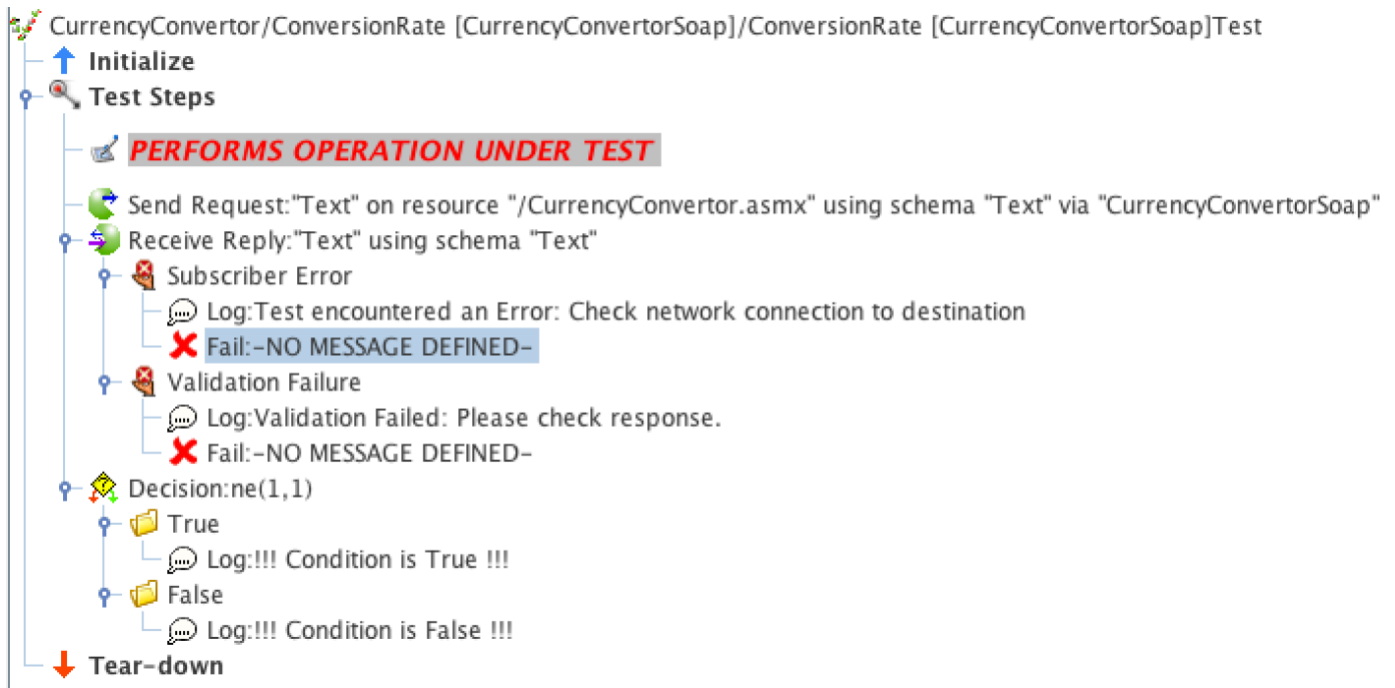


# Exception Handling

Most modern programming languages provide exception handling to react and respond to errors that occur during runtime. In RIT there are possibilities that each action step might fail at runtime. In that case Test stops at the point of error. However we might want to display a message in the console for troubleshooting that the Test failed due to an Unknown error or specific reason. In such case we can add a Failure Path to the test action and add a log message to display in console or to a log file the details of the failed test step.

## Note:

When failure path is added to test step and if the test step fails, error message is displayed and steps within the failure path is executed. After which the execution continues to the next action following the failed step. Also when failure path is used though error message is displayed in the console, test result will always show as Passed unless there is a Fail action encountered after the failure path either as a part of failure path or outside of it. So it is recommended to use Fail action where ever required to have correct test status.



Type	Task	Progress	Status
	versionRate [CurrencyConvertorSoap]Test	100%	Failed

Console

```

<terminated> CurrencyConvertor/ConversionRate [CurrencyConvertorSoap]/ConversionRate [CurrencyConvertorSoap]Test
[04/22/2016, 7:19:15.838 AM] Initializing...
[04/22/2016, 7:19:15.842 AM] Using environment: Harinivass-MacBook-Pro-4
[04/22/2016, 7:19:15.842 AM] - - - - Starting main steps - - - -
[04/22/2016, 7:19:15.853 AM] Send Request:"Text" on resource "/CurrencyConvertor.asmx" using schema "Text" via
"CurrencyConvertorSoap" - Send message
[04/22/2016, 7:19:15.940 AM] [Assertion Failed] Receive Reply:"Text" using schema "Text"
"/text/ConversionRate__OUTPUT__ConversionRateSoapOut/tns:ConversionRateResponse/tns:ConversionRateResult/{}" - The
expected value is not valid.[CRRIT4354E] field can only contain numeric values in the range '4.9E-324' to
'1.7976931348623157E308', eg '4.53', '6'. (Action = "Equality")
[04/22/2016, 7:19:15.940 AM] [Assertion Failed] Receive Reply:"Text" using schema "Text"
"/text/ConversionRate__OUTPUT__ConversionRateSoapOut/tns:ConversionRateResponse/tns:ConversionRateResult/{}" - Expected
value "", found value "-1.0" (Action = "Equality")
[04/22/2016, 7:19:15.940 AM] [Warning] Validation Failed: Please check response.
[04/22/2016, 7:19:15.940 AM] [Failed] Fail:
[04/22/2016, 7:19:15.941 AM] [Failed] 1 iteration(s) completed 1 iteration(s) failed (1)
Logging summary: Info (0), Warnings (1), Errors (0)
Overall status:FAILED

```

Type	Task	Progress	Status
	versionRate [CurrencyConvertorSoap]Test	100%	Failed

Console

```

<terminated> CurrencyConvertor/ConversionRate [CurrencyConvertorSoap]/ConversionRate [CurrencyConvertorSoap]Test
[04/22/2016, 7:19:15.838 AM] Initializing...
[04/22/2016, 7:19:15.842 AM] Using environment: Harinivass-MacBook-Pro-4
[04/22/2016, 7:19:15.842 AM] - - - - Starting main steps - - - -
[04/22/2016, 7:19:15.853 AM] Send Request:"Text" on resource "/CurrencyConvertor.asmx" using schema "Text" via
"CurrencyConvertorSoap" - Send message
[04/22/2016, 7:19:15.940 AM] [Assertion Failed] Receive Reply:"Text" using schema "Text"
"/text/ConversionRate__OUTPUT__ConversionRateSoapOut/tns:ConversionRateResponse/tns:ConversionRateResult/{}" - The
expected value is not valid.[CRRIT4354E] field can only contain numeric values in the range '4.9E-324' to
'1.7976931348623157E308', eg '4.53', '6'. (Action = "Equality")
[04/22/2016, 7:19:15.940 AM] [Assertion Failed] Receive Reply:"Text" using schema "Text"
"/text/ConversionRate__OUTPUT__ConversionRateSoapOut/tns:ConversionRateResponse/tns:ConversionRateResult/{}" - Expected
value "", found value "-1.0" (Action = "Equality")
[04/22/2016, 7:19:15.940 AM] [Warning] Validation Failed: Please check response.
[04/22/2016, 7:19:15.940 AM] [Failed] Fail:
[04/22/2016, 7:19:15.941 AM] [Failed] 1 iteration(s) completed 1 iteration(s) failed (1)
Logging summary: Info (0), Warnings (1), Errors (0)
Overall status:FAILED

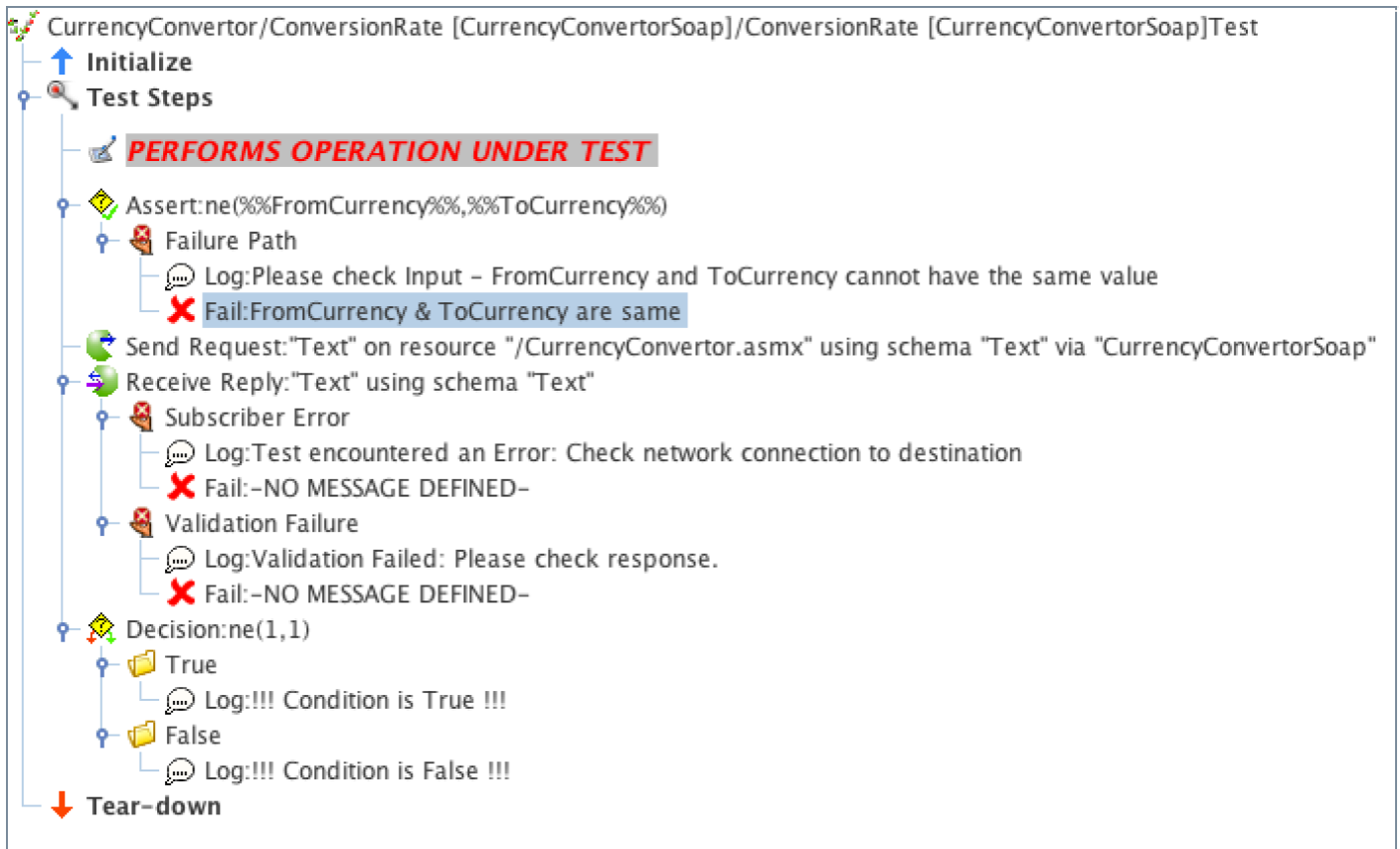
```

## Assertions

In Programming Languages – assertions are true-false statements which is expected to be always true for the execution to proceed. If during runtime, executing assertion statement yields false, the program stops execution at that point and execution does not continue beyond that statement.

### Note:

Assertions can be used as if only Decision Actions. but remember when decision fails executions stops. To have execution continue place the assertion in the action group or add failure path to it.

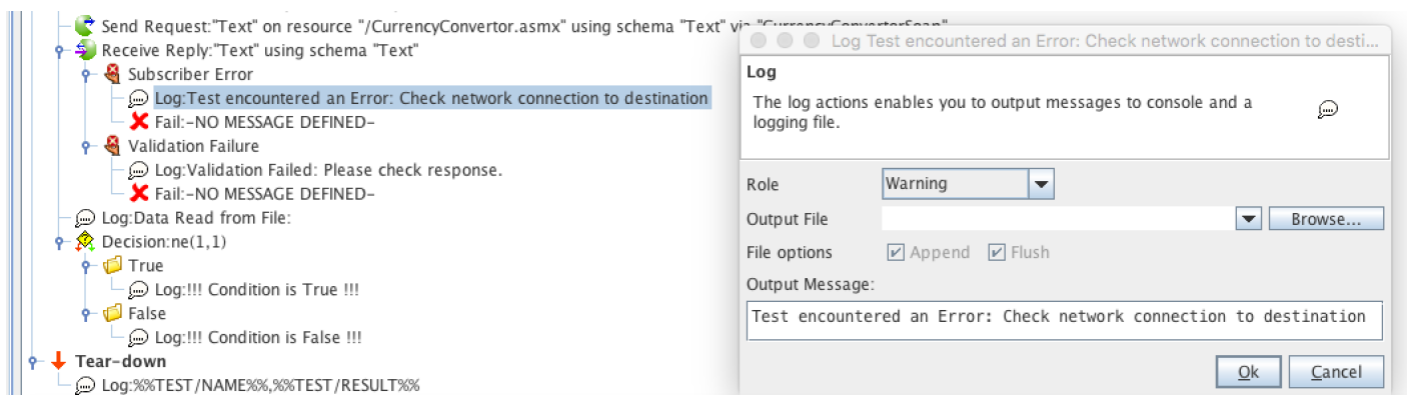


## I/O Functions

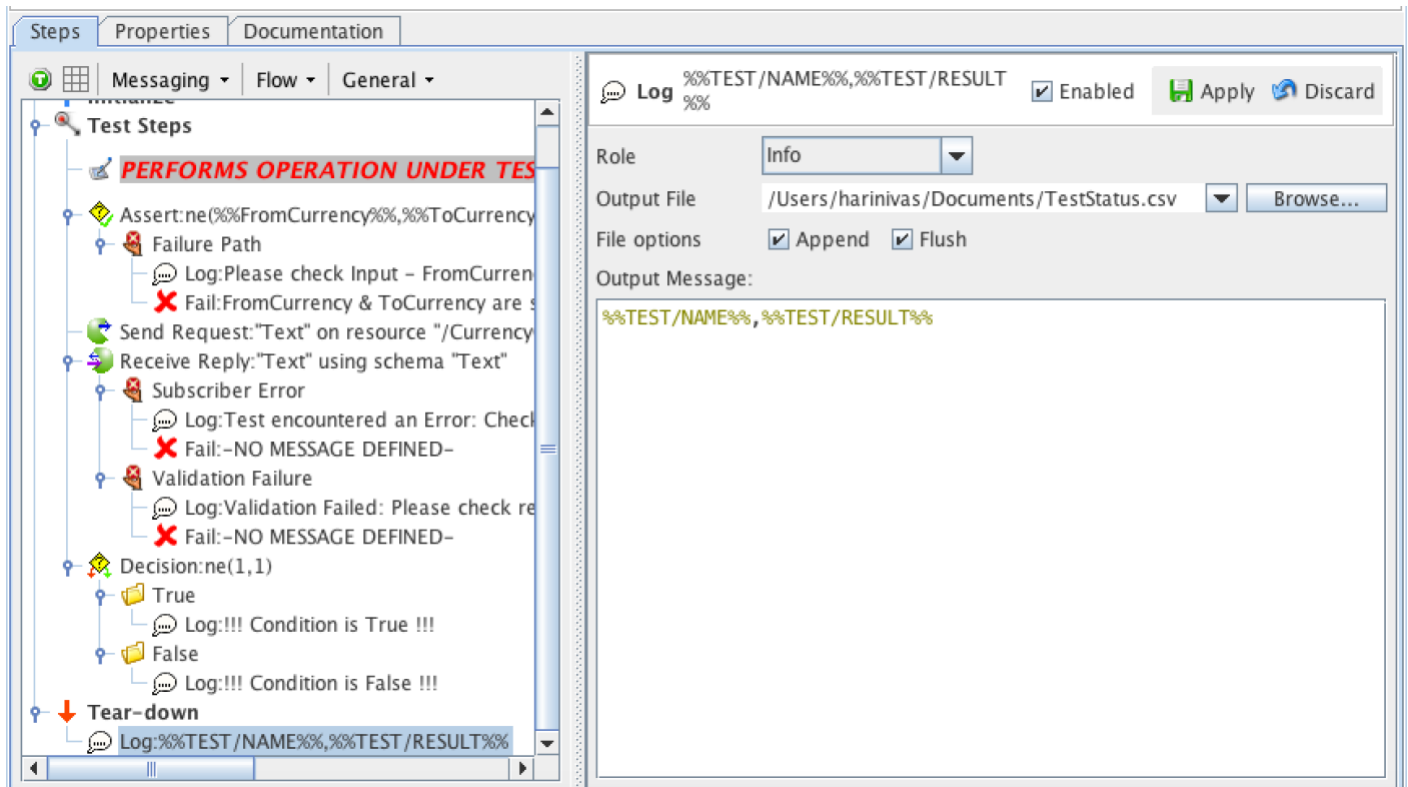
Any programming languages has to deal with data input into the system and output from the system. Similarly RIT provides a couple of ways to deal with input and output. Provides 4 ways –

1. Write contents into Console
2. Write Contents into File
3. Read Contents from File
4. Read data from User at runtime.

**Write contents into Console** Use Log action to display text into Console.

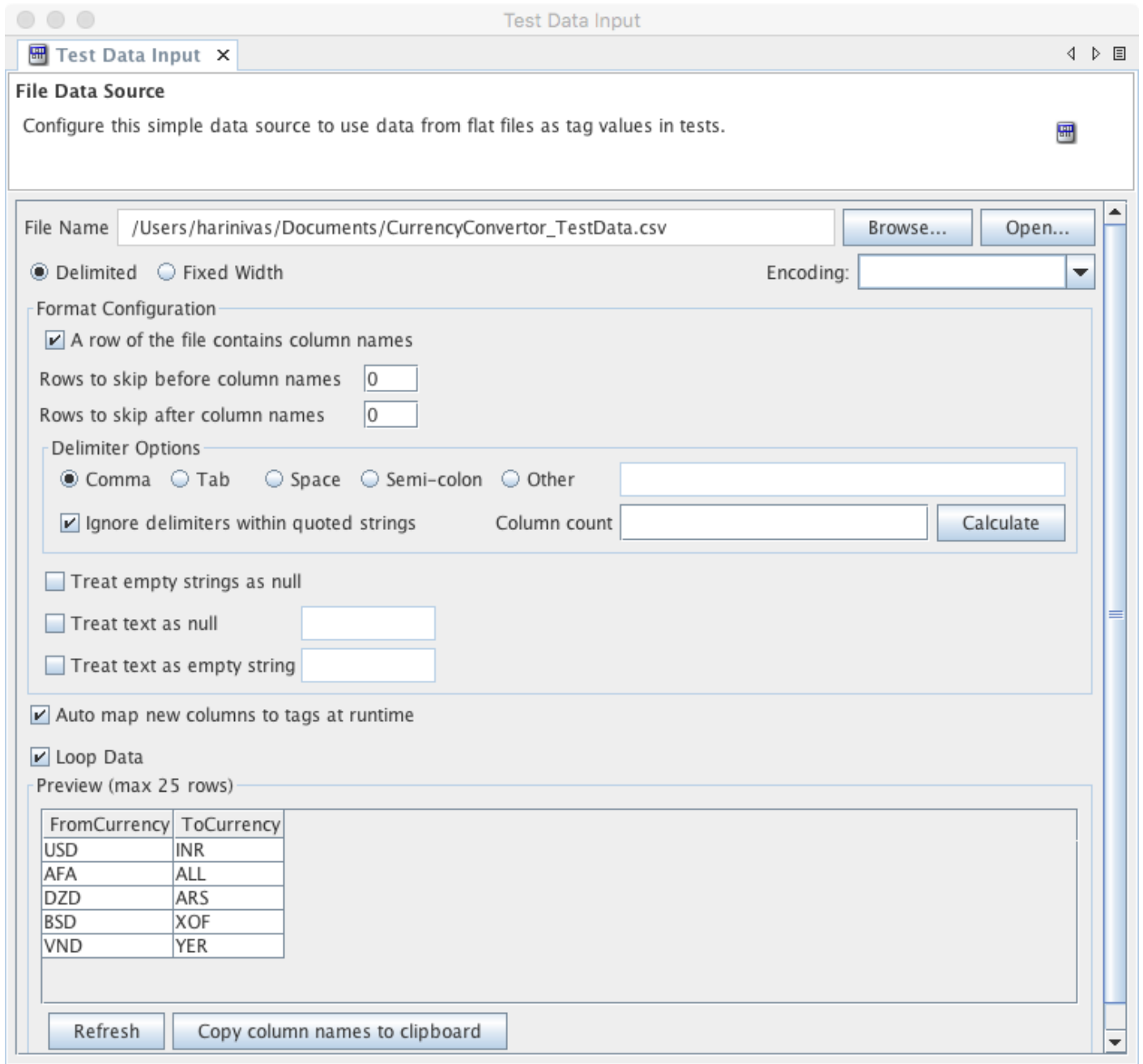


**File I/O – Write to file**RIT provides log action to write contents into a file. As seen in the example below the test status is logged to a file. A log message is added in the Tear-down step and configured to add test name and test result to a CSV file that can be opened outside RIT in a spreadsheet application.



	A	B
1	ConversionRate [CurrencyConvertorSoap]Test	FAILED
2		
3		
4		

**File I/O – Read from file**Reading data from file is not straight forward as writing into it as it requires two RIT concepts – File Data Source and Tag System.**STEP 1:** A file Data source should be created pointing to a CSV file.



**STEP 2:** Click on "Copy column names to clipboard" from bottom of the data source tab.**STEP 3:** Open tag store

Name	Description	Default Value
FromCurrency		
ToCurrency		

**STEP 4:** paste the copied tags.**STEP 5:** Add a Fetch Test Data action and configure it with the created Test data source.

The screenshot shows the SoapUI test plan for 'CurrencyConvertor/ConversionRate [CurrencyConvertorSoap]/ConversionRate [CurrencyConvertorSoap]Test (2)'. The test steps include: Initialize, Test Steps, PERFORMS OPERATION UNDER TEST, Fetch Test Data:from "Test Data Input", Assert:ne(%%FromCurrency%%,%%ToCurrency%%), Failure Path (Log:Please check Input - FromCurrency, Fail:FromCurrency & ToCurrency are not equal), Send Request:"Text" on resource "/CurrencyConvertor/ConversionRate", Receive Reply:"Text" using schema "Text", Subscriber Error (Log:Test encountered an Error: Check the response, Fail:-NO MESSAGE DEFINED-), Validation Failure (Log:Validation Failed: Please check the response, Fail:-NO MESSAGE DEFINED-), and Decision:ne(1,1) (True: Log:!!! Condition is True !!!, False: Log:!!! Condition is False !!!).

The 'Fetch Test Data' dialog box is open, showing the following configuration:

- Dataset: Test Data Input
- Group data by column: (empty)
- Mappings table:
 

Tag name	Data
ToCurrency	ToCurrency
FromCurrency	FromCurrency
- After this row has been mapped, advance to the next row

**STEP 6:** Edit the Send Request message to read value from the Tags – %%FromCurrency%% & %%ToCurrency%%

Send Request "Text" on resource "/CurrencyConvertor.asmx" using schema "Text" via "CurrencyConvertorSoap" [...]

**Send Request**  
Publish a message and wait for a response to be received. This can then be validated accordingly.

Config Value Store

Transport CurrencyConvertorSoap Browse... Formatter HTTP Message

Message Header

HTTP Properties HTTP Headers

Resource Path /CurrencyConvertor.asmx

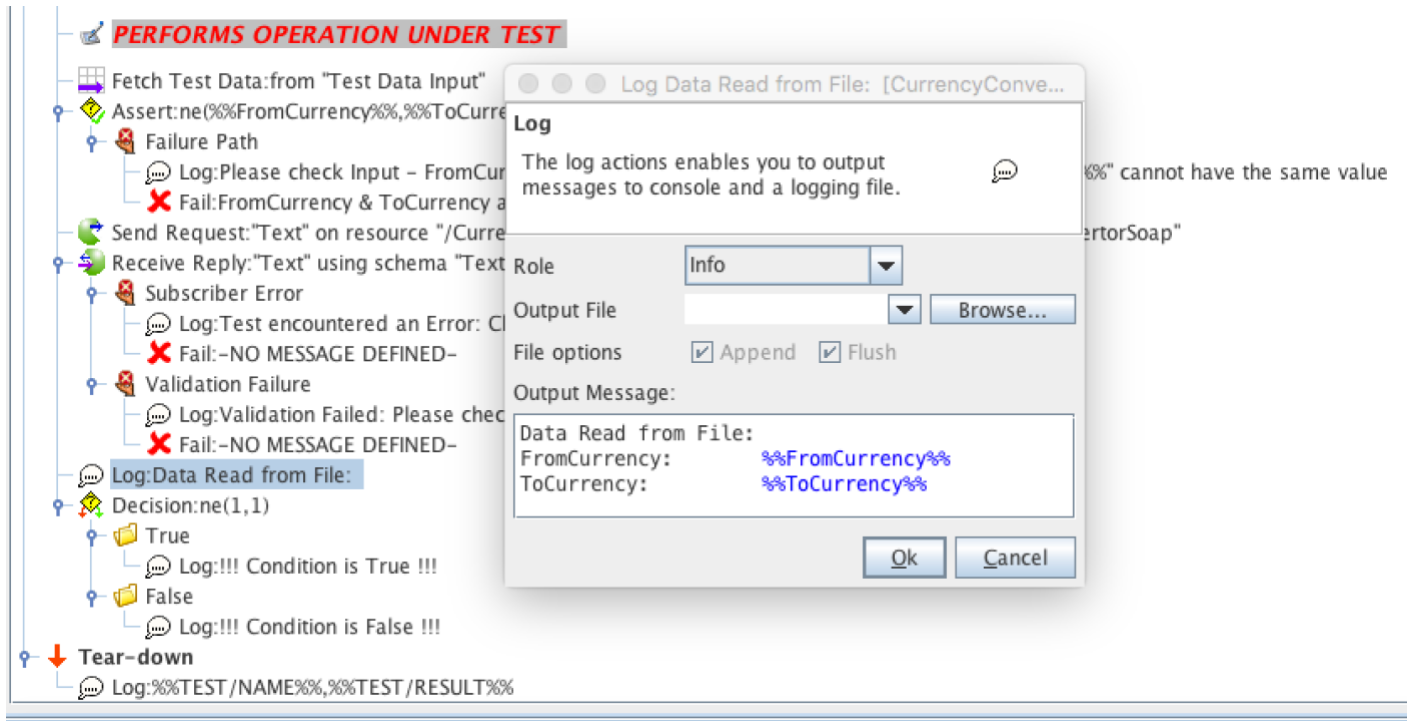
HTTP Method POST  Follow Redirects

Message Type Text

Message	Value	
Text (Message)	Process Children	<input checked="" type="checkbox"/>
text (String) {Document-Literal SOAP}	<ConversionRate__INPUT__ConversionRateSoapIn> <tns:Co	<input checked="" type="checkbox"/>
ConversionRate__INPUT__ConversionRateSoap	Process Children	<input checked="" type="checkbox"/>
tns:ConversionRate (Element)	Process Children	<input checked="" type="checkbox"/>
xmlns:tns (Attribute)	http://www.webserviceX.NET/	<input checked="" type="checkbox"/>
tns:FromCurrency (Element)	Process Children	<input checked="" type="checkbox"/>
(Text)	%%FromCurrency%%	<input checked="" type="checkbox"/>
tns:ToCurrency (Element)	Process Children	<input checked="" type="checkbox"/>
(Text)	%%ToCurrency%%	<input checked="" type="checkbox"/>

Ok Cancel

**STEP 7:** Add log step to display the values read from file



## STEP 8: Save and Run

Type	Task	Progress	Status
	ConversionRate [CurrencyConvertorSoap]Test (2)	100%	Passed

Console

```

<terminated> CurrencyConvertor/ConversionRate [CurrencyConvertorSoap]/ConversionRate [CurrencyConvertorSoap]Test (2)
[04/23/2016, 11:18:06.980 AM] Initializing...
[04/23/2016, 11:18:06.984 AM] Using environment: Harinivass-MacBook-Pro-4
[04/23/2016, 11:18:06.984 AM] - - - - Starting main steps - - - -
[04/23/2016, 11:18:06.986 AM] Fetch Test Data:from "Test Data Input" Opening data source for initial use when no grouping
applied
[04/23/2016, 11:18:06.998 AM] Send Request:"Text" on resource "/CurrencyConvertor.asmx" using schema "Text" via
"CurrencyConvertorSoap" - Send message
[04/23/2016, 11:18:07.089 AM] Receive Reply:"Text" using schema "Text" - Message validation passed
[04/23/2016, 11:18:07.089 AM] Data Read from File:
FromCurrency: USD
ToCurrency: INR
[04/23/2016, 11:18:07.089 AM] !!! Condition is False !!!
[04/23/2016, 11:18:07.089 AM] - - - - Starting teardown - - - -
[04/23/2016, 11:18:07.089 AM] ConversionRate [CurrencyConvertorSoap]Test (2),PASSED
[04/23/2016, 11:18:07.090 AM] [Passed] 1 iteration(s) completed successfully
Logging summary: Info (3), Warnings (0), Errors (0)
Overall status:SUCCESSFUL

```

**Read data from User at runtime** – Like C & Java programming languages have options to read data entered in console RIT has a test action – “User Interaction” to read data from user at runtime.

### Note:

It is a bad idea to read data from user at runtime for testing. In today’s QA environment, automation is used to avoid any manual interaction. So having user to provide data to test at runtime is a bad idea. May be can be used while designing a test but not while finalizing the test.



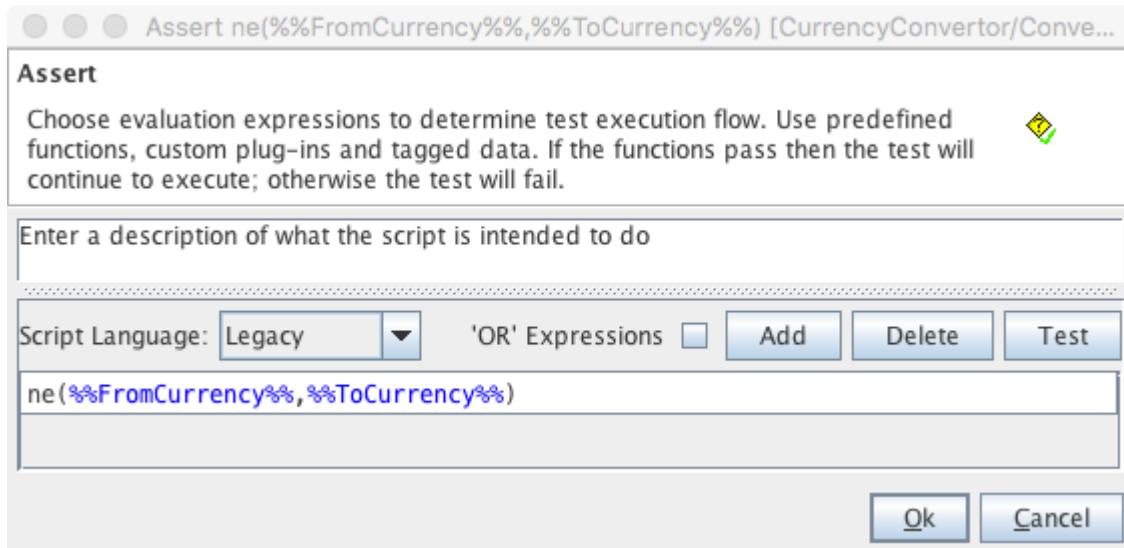
# Library

Every modern programming languages have either built in Library of classes or functions or public library contributed by open source community. RIT supports both. It has built-in Functions and also has provisions for third party to contribute on Functions. Refer table above for list of built-in functions. RIT also provides two ways to build libraries of Function.

1. Build Custom Functions – refer like for ideas on Custom Functions and how to build one – **Demystifying Custom Functions** (<https://harinivasganapathy.wordpress.com/2015/09/06/demystifying-custom-functions/>).
2. Build JavaScript library – Tutorial on Building and Using JavaScript Library – Coming Soon

# String Manipulations

Programs often need to be manipulated to solve specific problems. RIT provides below options and functions to manipulated strings to deal with test automation requirements.**String Interpolation**To reference values stored in tags in log action or sql action or field actions reference tag name in format – %%TagName%%



tns:FromCurrency (Element)	Process Children
(Text)	%%FromCurrency%%
tns:ToCurrency (Element)	Process Children
(Text)	%%ToCurrency%%

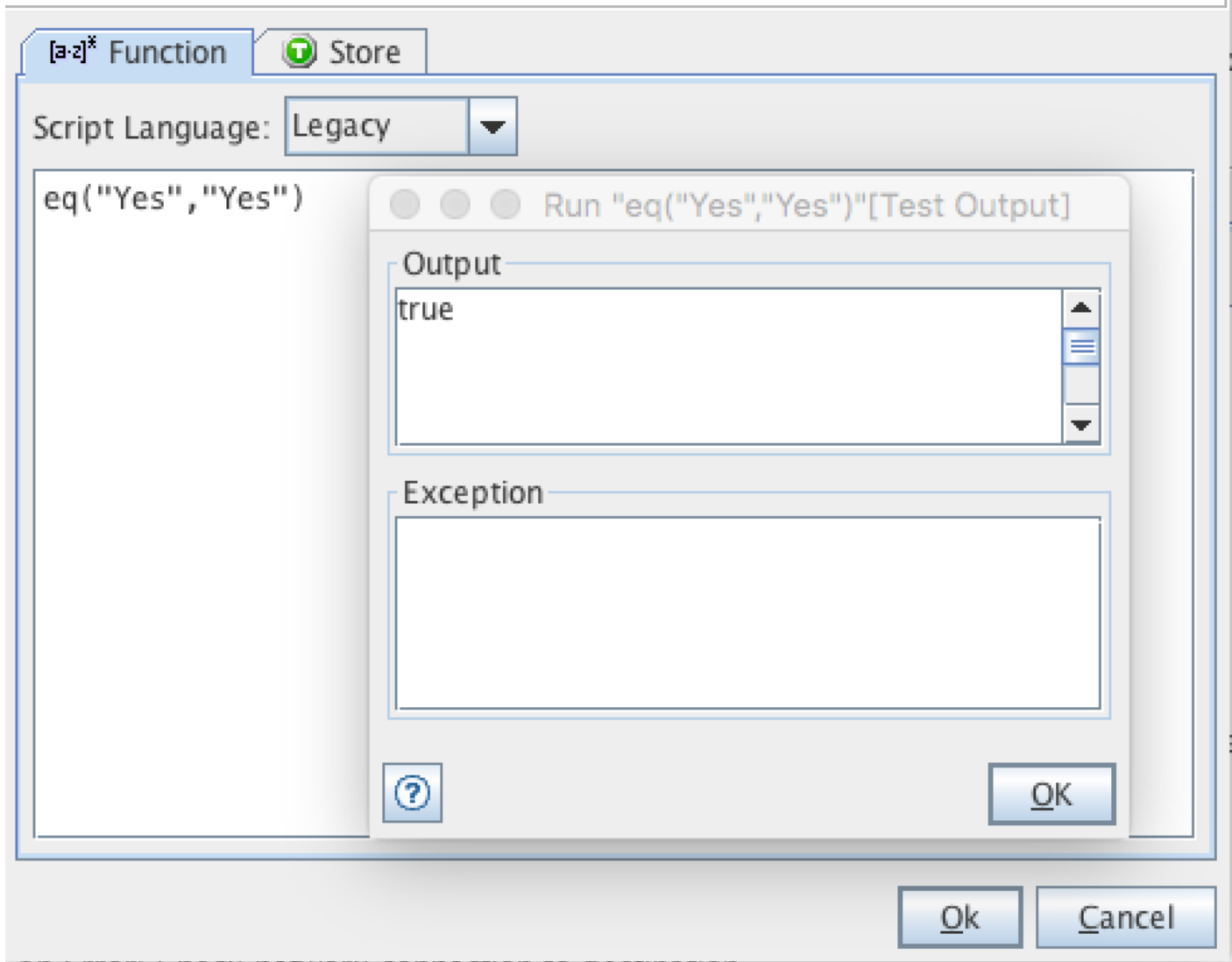
B

**String Comparison** Using the 'eq' Function we can check if two strings are same or not.

## Function

Execute a function and optionally tag the results. Use predefined functions, custom plug-ins and tagged data.

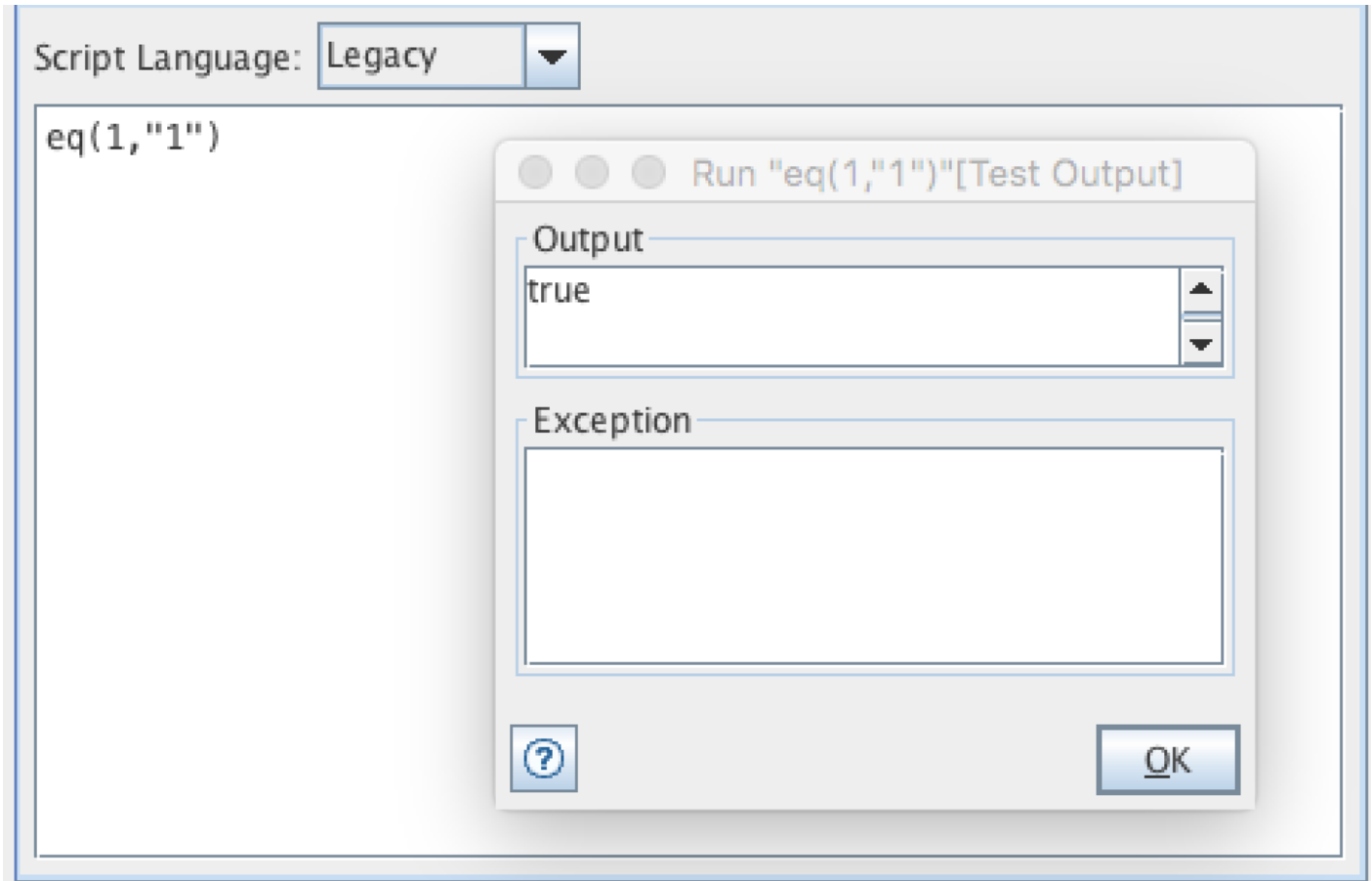
[a-z]\*



Same way we can use 'ne' – Not Equal function to check if two strings are equal.

### *Note:*

The strings used as function parameter need not be enclosed inside double quotes. RIT would automatically consider any parameter as String. Even if we intend to compare numbers and used numbers as parameter, RIT would treat them as String.

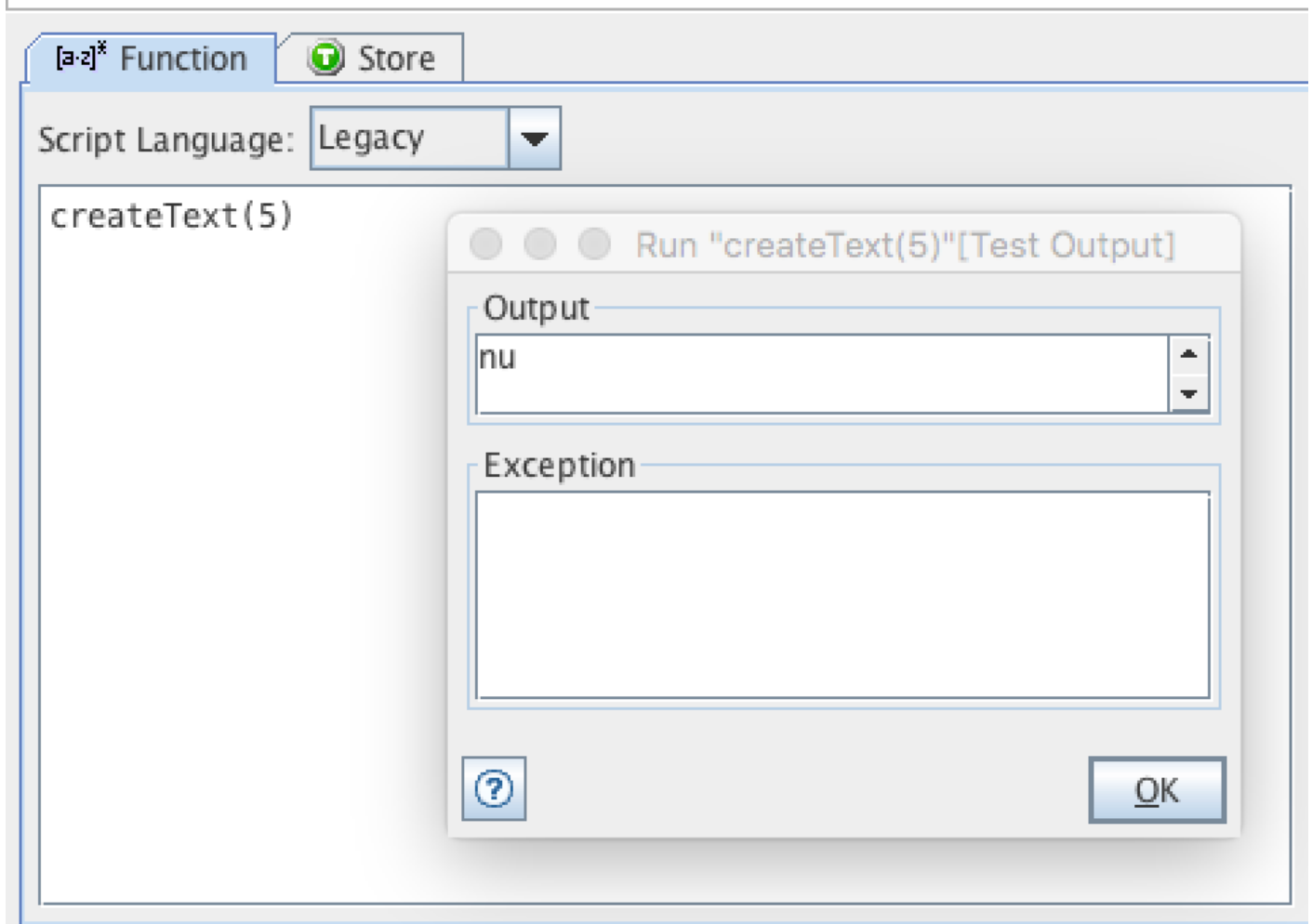


**Create String at runtime** We can use 'CreateText' function to generate random text of length specified as first parameter. Example CreateText(5) would create a random text of size 5 bytes every time it is executed.

## Function

Execute a function and optionally tag the results. Use predefined functions, custom plug-ins and tagged data.

[a-z]\*

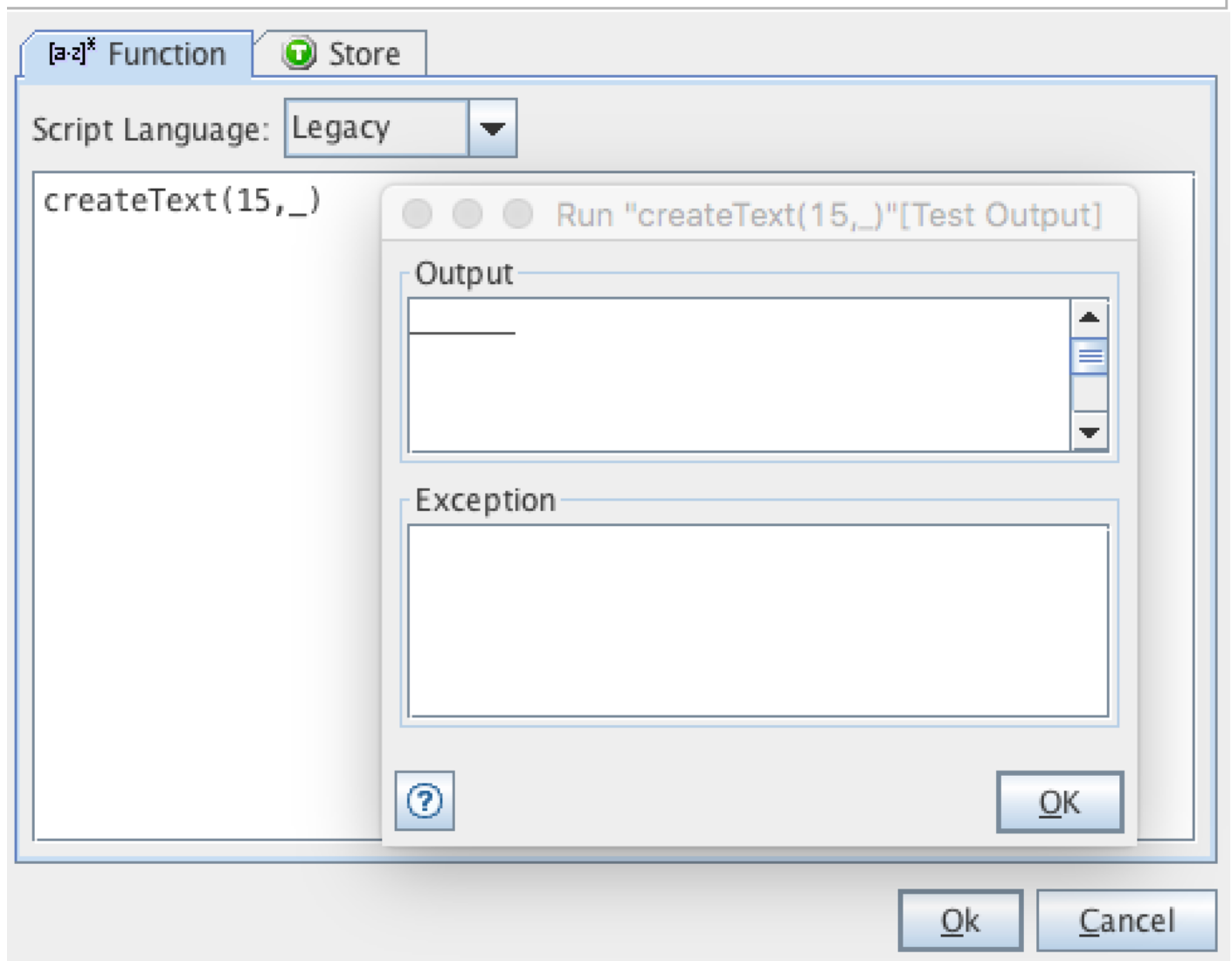


Alternatively we can create a string of certain repeating character using the CreateText function with the second optional parameter mentioning the repeating character. Example CreateText(15,-) creates a string of "\_\_\_\_\_". Do not include the second parameter value within quotes, if given RIT considers it as a part of repeating character.

## Function

Execute a function and optionally tag the results. Use predefined functions, custom plug-ins and tagged data.

[a-z]\*

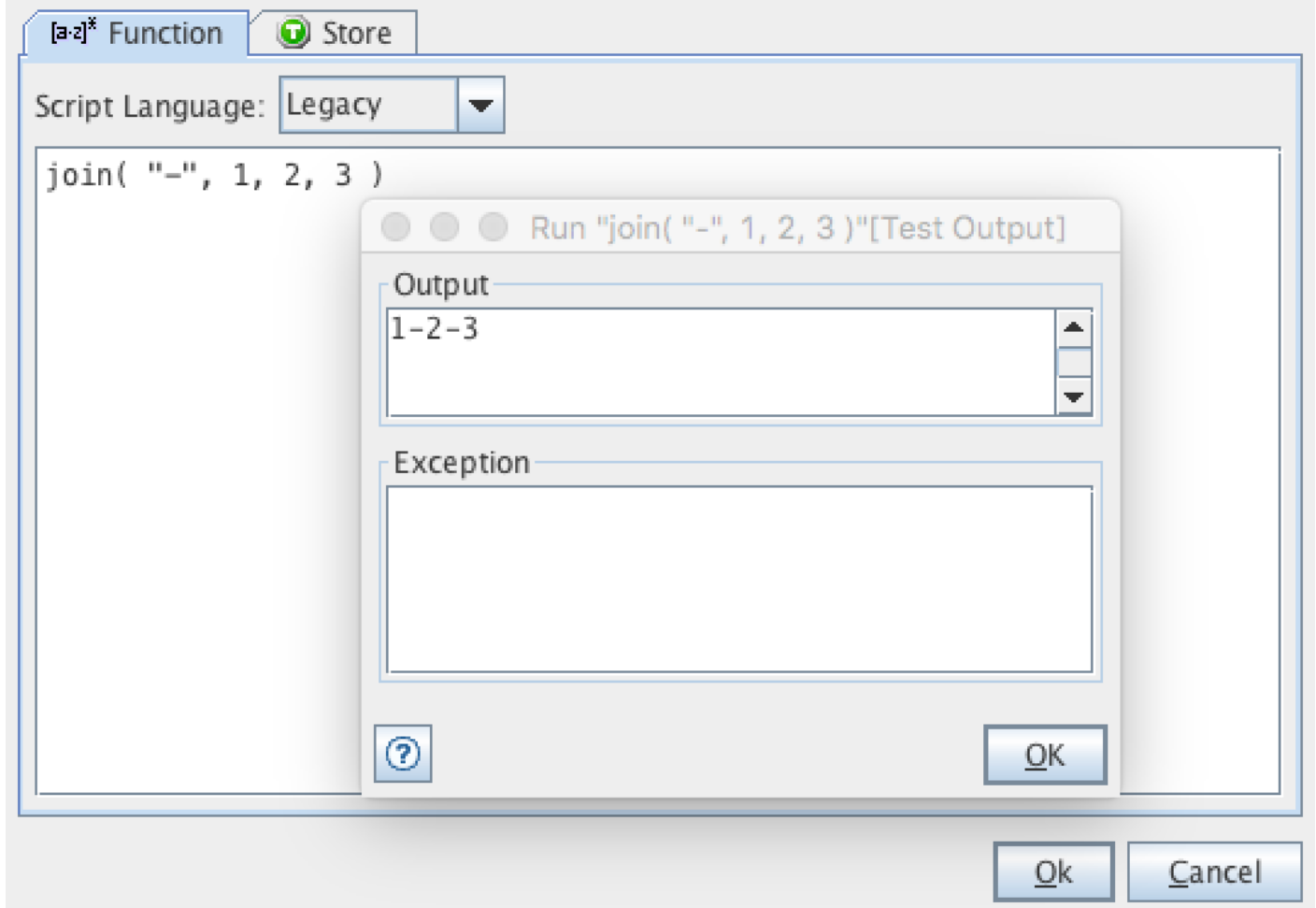


**String Concatenation**RIT has a 'join' function that concatenate 2 or more string into one. First Parameter is the delimiter. Example Join ("-","1,2,3) would return "1-2-3"

## Function

Execute a function and optionally tag the results. Use predefined functions, custom plug-ins and tagged data.

[a-z]\*

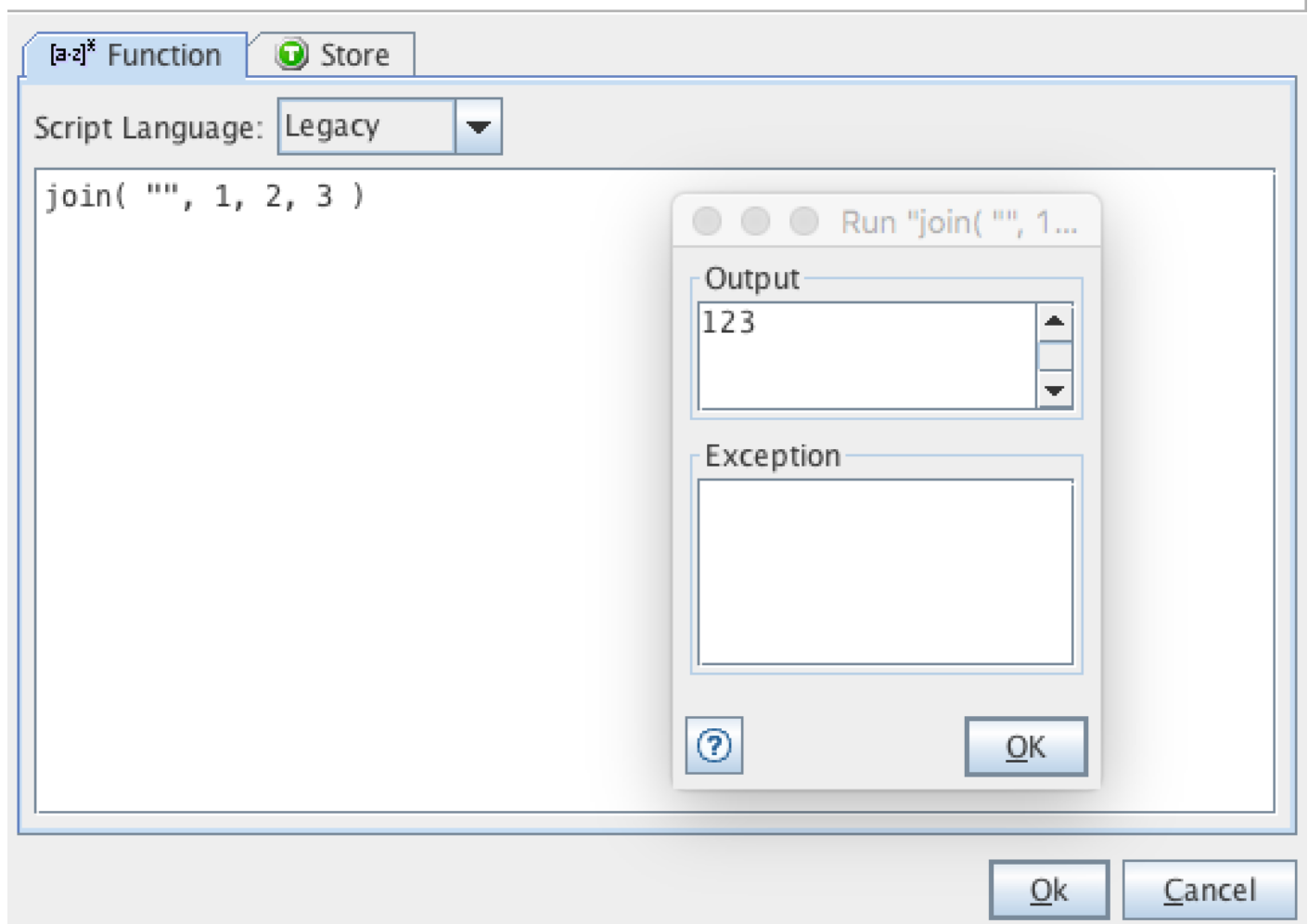


**Note:**For normal style string concatenation without delimiters include empty string for the first parameter. Example Join ("",1,2,3) would return "123"

## Function

Execute a function and optionally tag the results. Use predefined functions, custom plug-ins and tagged data.

[a-z]\*

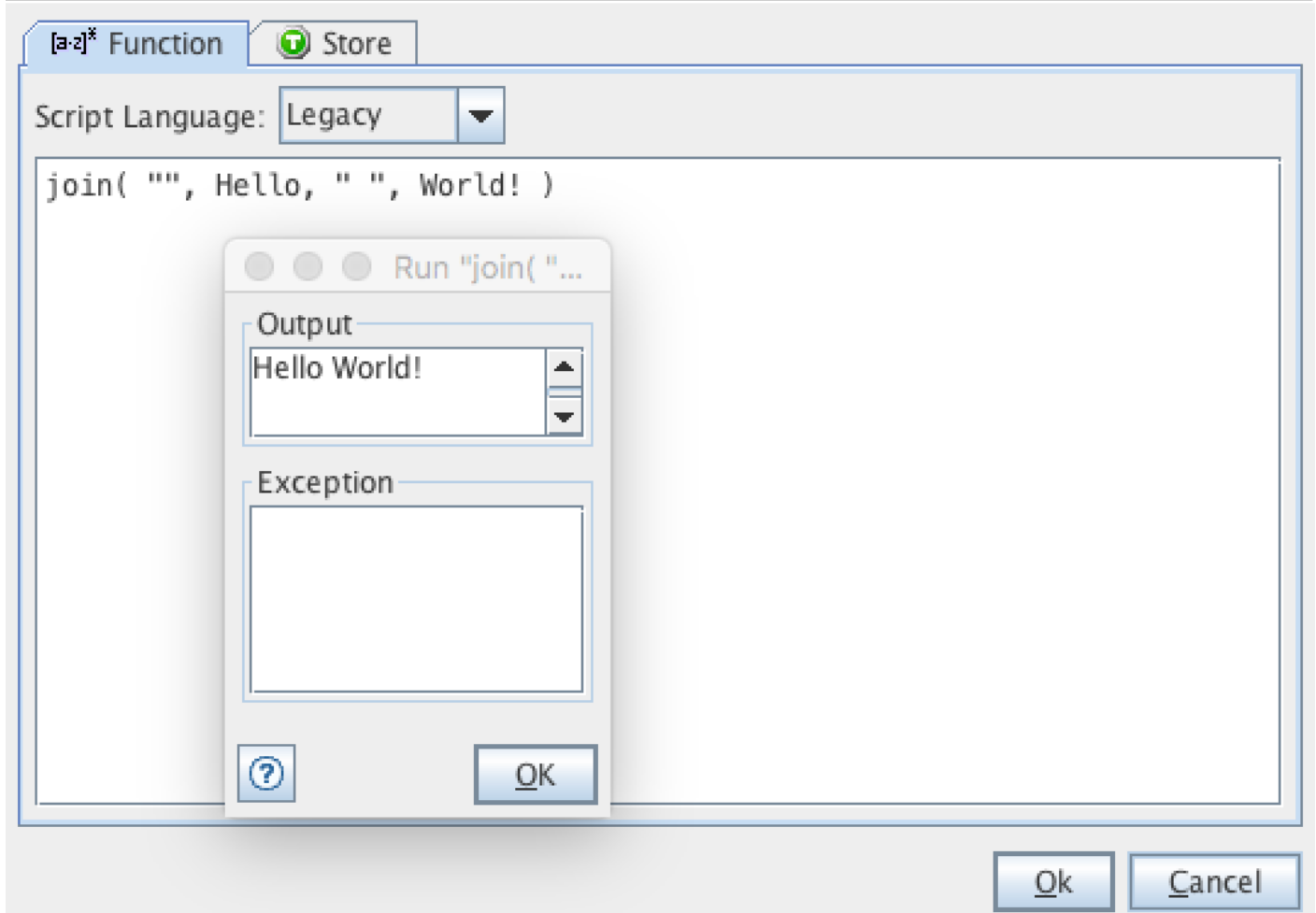


**Note:** If you wish to include space as a part of the String, include it as a parameter within double quotes.

## Function

Execute a function and optionally tag the results. Use predefined functions, custom plug-ins and tagged data.

[a-z]\*



**Getting SubString** RIT does not support direct Function to get substring from a String. However if the string is a delimited string it provides getToken Function to retrieve each part of the String. Example consider the string that represents customer delimited as "FirstName-MiddleName-LastName-Street Address-Apt #-City-State-Zip". to get address part of the customer record delimited by a hyphen use getToken("-",3,"John-Miller-123 Wal Street-456-Iving-TX-34522", true). Result is "123 Wal Street". Note that there is an empty middle name part in the example. Also remember first parameter is the delimiter char that should be enclosed within double quotes, second one is the index that starts with "0", Third one is the actual String from which we are extracting substring and fourth is a boolean that says if empty part of delimited string should be ignored or not.



## Function

Execute a function and optionally tag the results. Use predefined functions, custom plug-ins and tagged data.

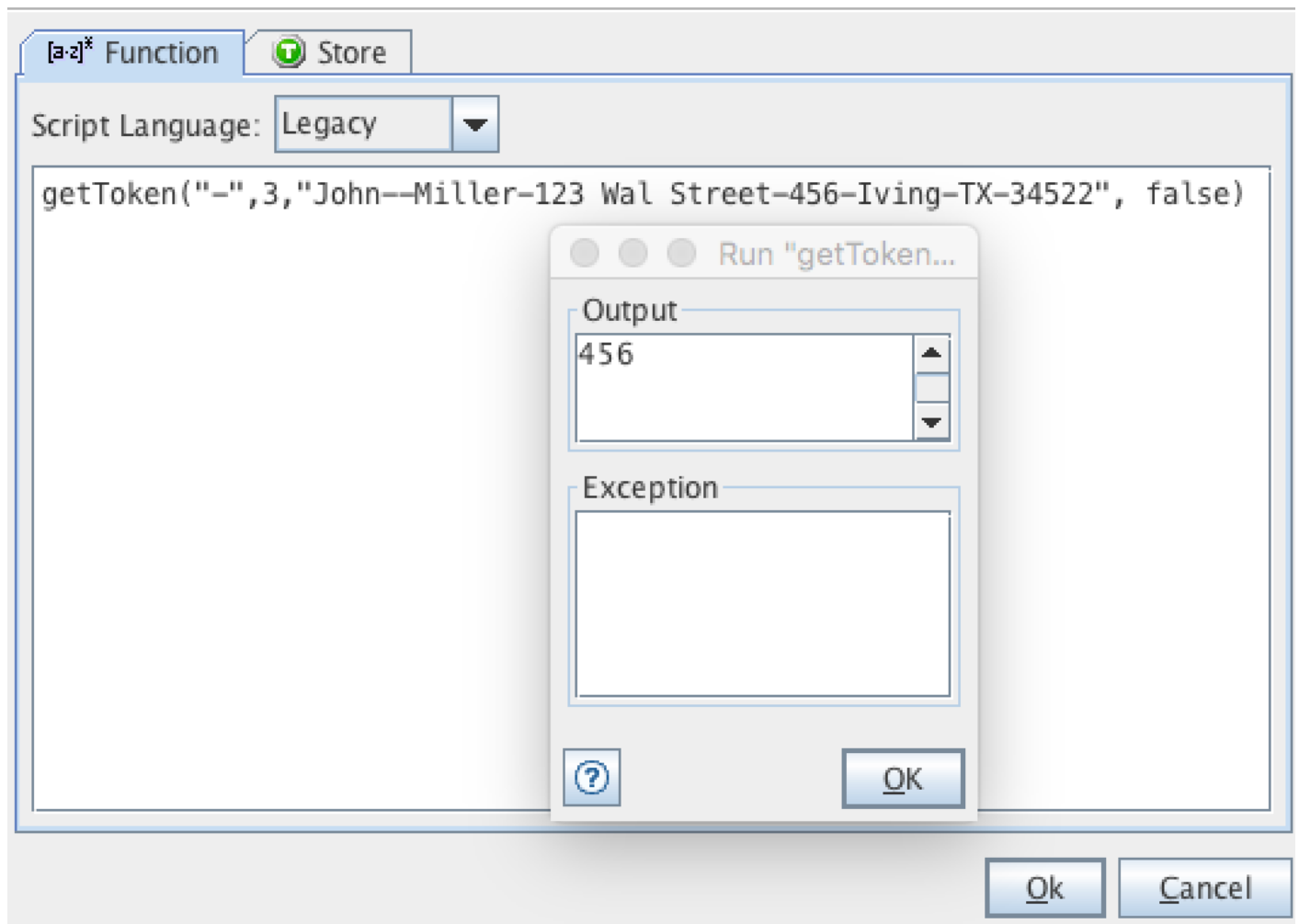
[a-z]\*

The screenshot shows a software interface with a main window and a modal dialog. The main window has two tabs: "[a-z]\* Function" (selected) and "Store". Below the tabs, there is a "Script Language:" dropdown menu set to "Legacy". The main text area contains the code: `getToken("-",3,"John--Miller-123 Wal Street-456-Iving-TX-34522", true)`. A modal dialog titled "Run \"getToken...\"" is open in the center. It has two sections: "Output" and "Exception". The "Output" section contains the text "123 Wal Street". The "Exception" section is empty. At the bottom of the modal dialog are a help button (question mark in a circle) and an "OK" button. At the bottom of the main window are "Ok" and "Cancel" buttons.

if Fourth parameter is false then any empty segment of delimited string is ignored and result is 456.

## Function

Execute a function and optionally tag the results. Use predefined functions, custom plug-ins and tagged data. [a-z]\*



*Note:* If a sub String needs to be extracted from non delimited string, use JavaScript substring() function by selecting ECMAScript from the Script Language dropdown.

*Hope this article was useful to get started with scripting in RIT. Do post your questions or comments.*

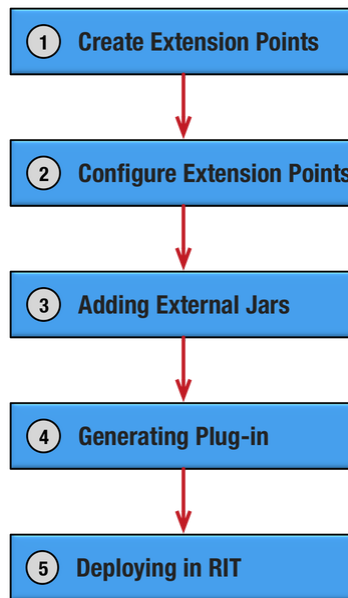
More to follow on RIT.

SEPTEMBER 13, 2015 SEPTEMBER 14, 2015 / HARINIVAS GANAPATHY

## **Demystifying RIT Custom Function – Part 3: Wrapping up for Deployment**

In **Part 1** (<https://harinivasganapathy.wordpress.com/2015/09/06/demystifying-custom-functions/>) we discussed how to set the requirements ready to build a Custom Function using Eclipse plugin development framework. In **Part 2** (<https://harinivasganapathy.wordpress.com/2015/09/10/demystifying-rit-custom-function-part-2-putting-the-gears-together/>) we discussed the architecture of the Function class which is the backbone of Custom Function and how to develop a Custom Function by extending the Custom Function. We are also discussed the rules and considerations of creating a Custom Function java class. In part 3 we will discuss about the requirements and steps to package Custom Function as pluggable jar and deploying it in RIT.

### Steps to deploy Custom Function

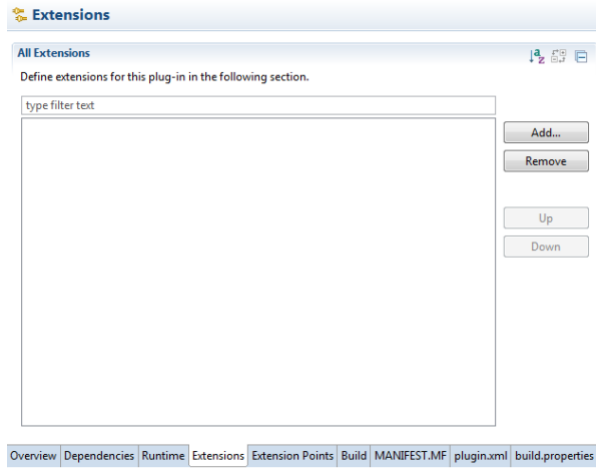


(<https://harinivasganapathy.files.wordpress.com/2015/09/deployment-steps.png>)

#### Create Extension Points

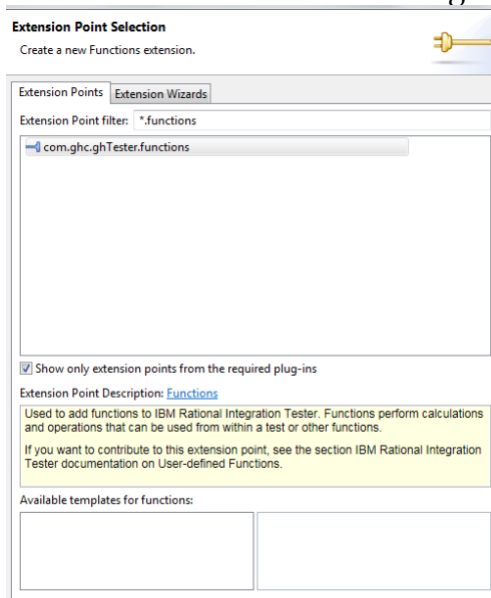
In order to allow 3rd party Functions added to RIT, IBM has created an extension point by means of 'com.ghc.ghTester.functions'. This extension point should be added to the extensions tab in our project's Manifest file.

1. Open plug-in's manifest file by double clicking MANIFEST.MF

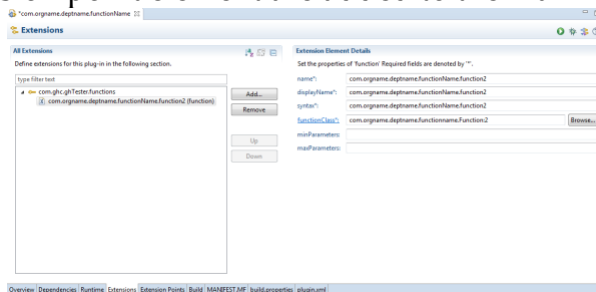


[https://harinivasganapathy.files.wordpress.com/2015/09/new\\_extensions\\_tab.png](https://harinivasganapathy.files.wordpress.com/2015/09/new_extensions_tab.png)

2. In the opened editor select the Extensions tab and click Add.
3. Select com.ghc.ghTester.functions in the New Extension Dialog



4. Click Finish.
5. Extension point and Extension point element are added to the manifest.



<https://harinivasganapathy.files.wordpress.com/2015/09/after-adding-extension.png>

## Configure Extension Points

The Extension Point Element has series of fields profiled with default values. Update the values as per the below as reference –

- name:** Name of the function like *formatDate*
- displayName:** User friendly, readable name that appears in Function menu inside Test function editor. e.g) Format Date
- syntax:** The syntax of the Custom Function that gives an idea for the user. e.g)  
*formatDate(date, inputFormat [, outputFormat])*
- functionClass:** The java class name that we developed in Part 2. Fully qualified name should be mentioned.
- minParameters:** The minimum parameters needed to pass in the function.
- maxParameters:** The maximum parameters needed to pass in the function.

Extension Element Details  
Set the properties of 'function'. Required fields are denoted by '\*'.

name\*: formatDate

displayName\*: Format Date

syntax\*: formatDate(date, inputFormat [, outputFormat])

functionClass\*: com.orgname.deptname.functionname.FormatDate

minParameters: 2

maxParameters: 3

<https://harinivasganapathy.files.wordpress.com/2015/09/extension-element-details.png>

Save the manifest file. Next step will fail if not saved before proceeding.

## Adding External Jars

This step is optional. It is required if the java class developed in Part-2 uses additional external jars. Follow these steps to those external jars to the plug-in jar we created above.

1. Add the Jar to the root of the plug-in project
2. Add the jar to the plug-ins project's build path by –
  1. Right-click the project and select **Properties > Java Build Path > Libraries**.
  2. Click Add Jars and browse for the Jar just added under the plug-in project.
  3. Close the properties dialog.
3. Open the plug-in's manifest file for editing and select the Runtime tab.
4. In the Classpath section, click Add and select the Jar that was copied into the project.
5. Save the manifest file.

## Generating Plug-in

In the previous steps we created a plugin. Now the plugin has to be packaged with the java class developed in part 2. Follow the below steps to package the plugin and create a pluggable jar.

1. **File > Export**.
2. In the Export dialog, select **Plug-in Development > Deployable plug-ins and fragments** from the list of exports.
3. Click **Next** to display the deployable plug-ins and fragment dialog.
4. In the Available Plug-ins and Fragments list, select the plug-in that you want to generate and export.
5. Select the **Directory** option and enter a directory location where the plug-in jar will be generated.
6. Click **Finish**.

7. The output of this process is a directory named plug-ins and jar with the name and version inside the plug-ins directory.

## Deploying in RIT

This is the final step in developing and deploying Custom Function. Follow the below steps to make the Custom Function available under the Functions sub menu inside RIT Tests.

1. Copy the generated plug-in Jar and paste it into the Functions folder under the projects root folder.
2. In RIT select **Tools > Reload Custom Functions** so that all added Custom Functions as jar is available in tests.
3. If needed Select **Tools > View All Functions** to verify that our function is loaded.

We are finally done. Now the Custom Function can be used like any other Function inside Function action editor inside RIT.

*Disclaimer: The information and Infographics produced here is based on years of experience in RIT testing and understanding on foundational information provided in IBM Knowledge Center. The source code re-produced here is taken from the original source and owner – IBM.*

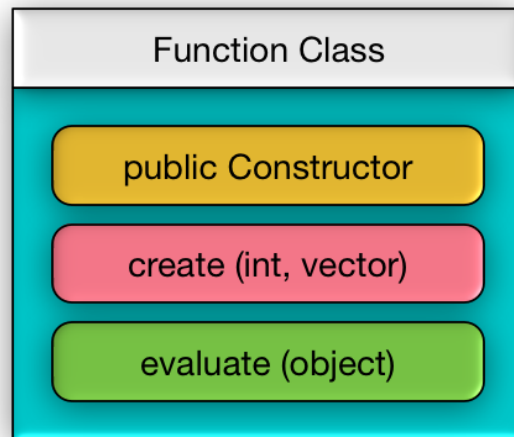
SEPTEMBER 10, 2015SEPTEMBER 14, 2015 / HARINIVAS GANAPATHY

# Demystifying RIT Custom Function – Part 2: Putting the gears together

In **Part 1** (<https://harinivasganapathy.wordpress.com/2015/09/06/demystifying-custom-functions/>) of the series, we discussed the list of pre-requisites and steps to set them up to begin developing Custom Functions. In Part 2 we would explore in detail about understanding the architecture of the Function class which is the backbone for any Functions in RIT.

## Function Class Architecture

Any Function whether it is built-in or custom made – they are subclasses of Function Class available in 'com.ghc.ghTester.expressions' package that we added to the target platform and manifest dependencies in **Part 1** (<https://harinivasganapathy.wordpress.com/2015/09/06/demystifying-custom-functions/>). It is important to understand the design of this Class because RIT lays down few rules that we have to follow for it to recognize, understand, load and execute our Function. Function Class has many methods, but three of them are very important and significant – the public constructor, create and evaluate method.



**public Constructor :** The way functions are designed in RIT mandates a default public constructor with no parameters. This is needed for loading our Custom Function class into RIT JVM. We don't need to add any computation inside this constructor. However it can be used to initialize instance variables.

**create (int, vector) :** This is a factory method that creates the instance of the particular Function with specific arguments we send during runtime. This method is invoked when RIT encounters the Custom Function syntax while evaluating a function action in tests. It is the responsibility of the *create* function to create, initialize and instantiate the specific Function object with matching parameters mentioned in the Function call in tests.

Note that we can design and implement more than one Custom Function inside single Java Class file. And so it is the responsibility of the *create* method to initialize and instantiate the right Function with right parameters.

**evaluate (Object) :** This is the actual method that performs the computation of the Function we design and considered brain of our Custom Function Class. It is invoked by RIT after *create* method returns the second Function instance.

### Rules of extending Function Class

1. Custom Function class should have a public default constructor.
2. Custom Function class should override *create (int, vector)* method.
3. Custom Function class should *evaluate (Object)* method.

### Flow of Control

**Step 1:** When RIT starts up, it creates an instance of our Custom Function class using the default constructor and loads it to RIT JVM. For our reference we call it Instance 1. Once the Function successfully loaded, it will be available for us to use inside function action in tests.

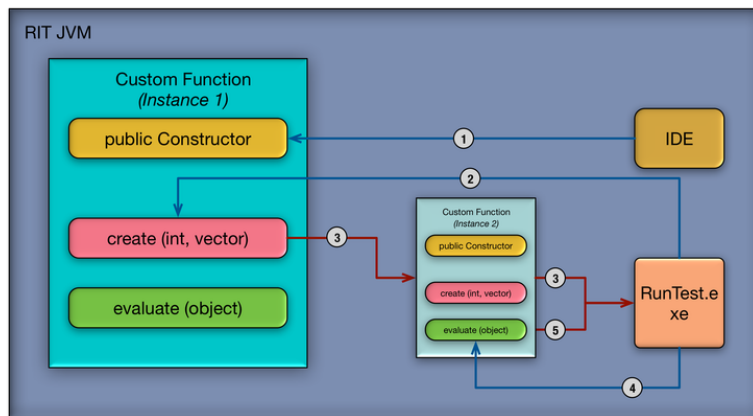
**Step 2:** When RIT encounters the Custom Function syntax, while evaluating the function action in tests, it invokes the *create* method of the Instance 1 and sends two arguments in the call – (1) number of arguments in test function call as *int* and (2) all the arguments as a *vector*. *Create* method

then creates another instance of the Custom Function and initializes it by calling the constructor with matching parameters based on the arguments received.

**Step 3:** Then it returns the newly created instance back to RIT with all data and parameters. For our reference we call this instance as instance 2.

**Step 4:** After receiving the instance 2 Custom Function object, RIT invokes the *evaluate* method of Instance 2 object and sends an Object as argument.

**Step 5:** On receiving the call, *evaluate* method has to discover the String data wrapped as an Function during the constructor call and assign it to the local variables for processing. Based on the received arguments, the evaluate method decides what to do. After performing all the computations it sends back the computed result back to RIT.

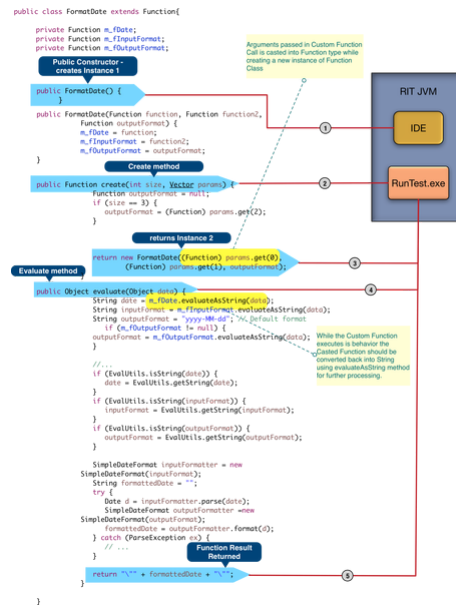


([https://harinivasganapathy.files.wordpress.com/2015/09/blog\\_rit\\_1.png](https://harinivasganapathy.files.wordpress.com/2015/09/blog_rit_1.png))

## Understanding using Code

Let us understand the architecture and flow of control we learnt so far by reviewing a sample code. For this purpose i have taken the example provided by IBM in its product documentation so it would easier when folks use the product documentation for further reference.





[https://harinivasganapathy.files.wordpress.com/2015/09/code\\_control\\_flow1.png](https://harinivasganapathy.files.wordpress.com/2015/09/code_control_flow1.png)

### Understanding the notion: 'Everything is Function'

If you would take another look at the code, if not already noticed, you would realize that everything inside the Custom Function class either takes Function as argument or returns a Function object. Even the data members are type Function. The vector of parameters passed are also casted as Functions. Hence to use them for processing inside *evaluate* method the parameters as Function must be evaluated to discover the String value. For this purpose we use *evaluateAsString(Object)* method available in the Function Class.

### Key Items for Consideration

1. There should be a public no argument constructor.
  2. Class level variables should be created for each possible arguments the Custom Function is designed to take and each of the variables should be declared as type Function.
  3. Class level Function variables must be initialized inside the corresponding constructor with parameters. Remember each parameters in Constructor are Functions so that they can be assigned to corresponding Class Variables.
  4. There should be as many constructors with possible arguments a Custom Function call can have. For e.g. if a Custom Function is designed to take 2 or 3 or 4 arguments, then there should be 3 constructors with 2, 3 & 4 arguments respectively, so that create method would be able to use and instantiate the second instance. Or like in the IBM example single constructors can be used, however optional parameters must be evaluated and check for NULL like below –
- ```
Function outputFormat = null;
if (size == 3) {
    outputFormat = (Function) params.get(2);
}
```

5. *create* method should return only a new Instance of Custom Class with the String arguments stored as vector casted as Function before invoking corresponding constructor.

6. In *evaluate* method the actual String passed in Custom Function call in RIT test must be retrieved from the Function variables using *evaluateAsString* method available in the Function class.

7. *evaluate* method should return a String.

Now that we have got our core Custom Function Java Class ready, next step is to package it as a plug-in Jar and import it in RIT. We cover just these steps in our final Part – 3.

## **Continue to Part 3**

**(<https://harinivasganapathy.wordpress.com/2015/09/13/demystifying-rit-custom-function-part-3-wrapping-up-for-deployment/>)**

SEPTEMBER 6, 2015SEPTEMBER 17, 2015 / HARINIVAS GANAPATHY

# **Demystifying RIT Custom Function – Part 1:** **Gear up**

I had a hard experience reading through the product documentation from IBM on creating Custom Functions in Rational Integration Tester (RIT). After re-reading many times finally got hold of the concept behind extending Functions in RIT. So just thought of sharing my learning in the hope that it would make understanding the underlying concepts more easier than reading through the documentation over and over again.

## **What is a Function?**

Let us take a moment to understand the idea of Function and later dive more deep into creating one our own that can be plugged into RIT. Generally a function is a reusable block of code that takes one or more input (*technically called parameters*) from the test, performs some action (that is abstracted to test designer) on the input data and provides an output that can be used any where within an RIT test case later. RIT provides syntax and required parameters for using every function within its function editor – both legacy and ECMAScript. RIT supports two types of Functions –

- Built-in Functions – like – xpath, round, eq, gt, floor, mod, add, subtract – that are designed by IBM and shipped with every instance of RIT Product.
- Custom Functions – RIT User (*technically test designer*) created Functions that meets his / her custom needs. RIT test designer creates his own for use in more than one place.

## What is a Custom Function?

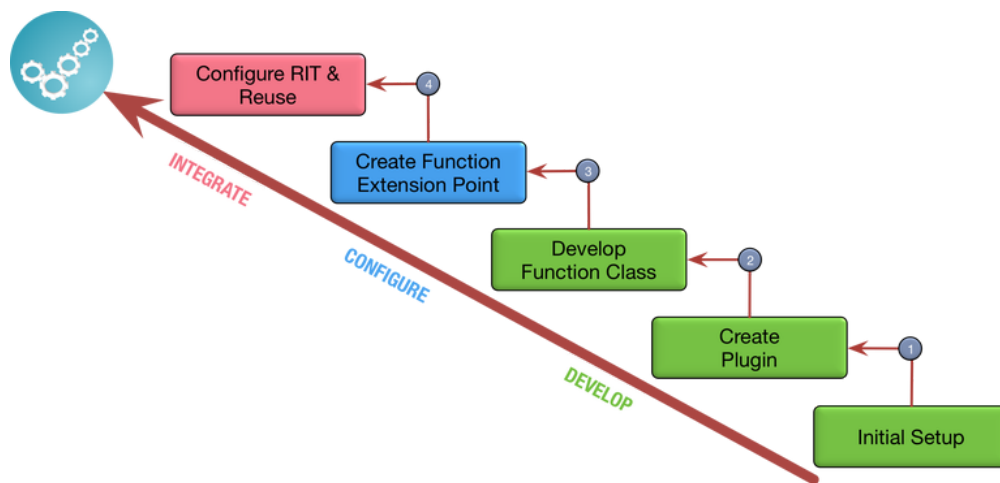
Before taking a detailed view of Functions let us review what is a *Custom Function* at an high abstracted level. A *Custom Function* is a Function that is designed, implemented and used by RIT user (tester or test designer) and not that provided by IBM by default. In addition to the built-in Functions provided by RIT, it also provides capability to create additional Functions by test designers to meet custom needs and testing requirements that can be reused across RIT tests & projects within an organizations.

Function is actually a Class provided by IBM so that it can be extended to create Functions tailored and customized for testers requirement. Hence the name – *Custom Function*. More about this in Part 2

### Steps to develop Custom Functions:

Like every other programming model, developing *Custom Functions* involves 3 steps with each steps involving more additional steps.

1. Design & Develop
2. Configure
3. Integrate



([https://harinivasganapathy.files.wordpress.com/2015/09/process-ladder\\_2.png](https://harinivasganapathy.files.wordpress.com/2015/09/process-ladder_2.png))

### Step 1 – Design & Develop

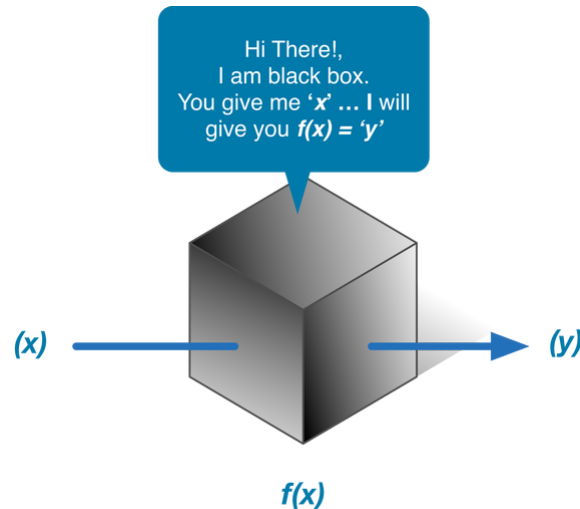
First step is more critical and complex part as it covers the core purpose, design and implementation of the Function. It involves 3 additional steps –

**Step 1.1:** Prepare yourself and complete initial setup.

*Drafting a logical design of the function –*

From test designer (*end-user of the completed plugin*) perspective – Function is a black box, because he doesn't need to know the Function's Implementation – how it process the input to produce the output. All a Function says is "Give me, *so and so* input and i will do *some* computation with the

input to convert it and give you a *certain* output.



So from Function designer perspective he has to have answers to below questions to start creating a Custom Function.

1. What's the purpose of it?
2. What are the minimum and maximum parameters needed to perform its indented operation?
3. What are the Parameters?
4. What should be its output?

Once answers to these questions are collected, we then move into the tools / software requirements –

### **Step 1.2:** Software Requirements

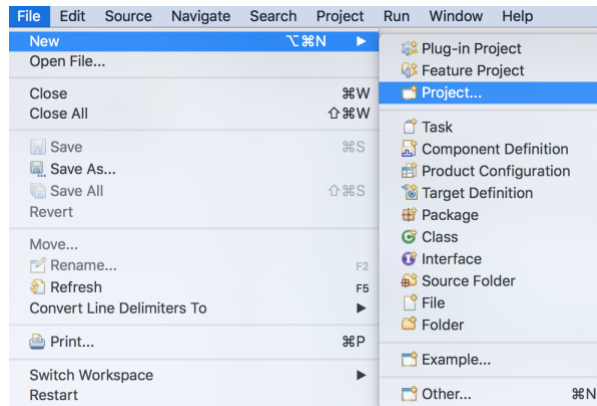
1. RIT installation
2. Eclipse installation – with PDE (Plugin Development Environment Support)

### **Step 2:** Create Plugin Project in Eclipse

RIT is built on top of Java and Eclipse Platform. Hence it has ability to accept plugins developed & packaged as eclipse plugins.

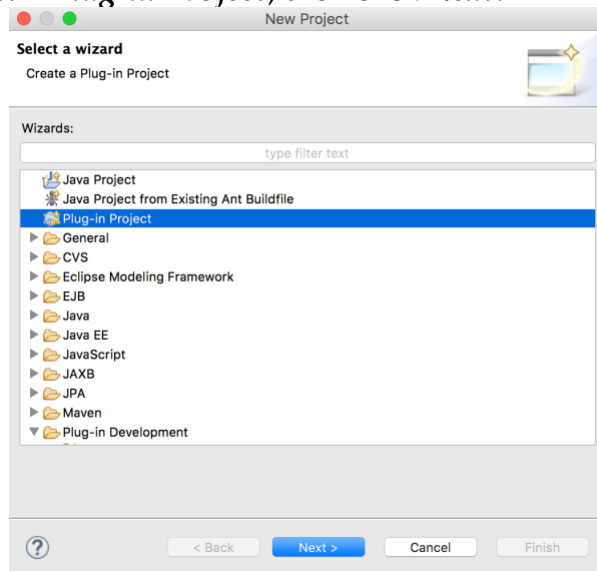
### **To create an Eclipse Plugin Project:**

1. Start / Open Eclipse Project
2. Create an Eclipse Workspace
3. Select *New > Project* from the *File* menu. The New Project wizard is displayed.



([https://harinivasganapathy.files.wordpress.com/2015/09/project\\_selection.png](https://harinivasganapathy.files.wordpress.com/2015/09/project_selection.png))

4. Select *Plug-in Development* > *Plug-in Project*, then click *Next*.



([https://harinivasganapathy.files.wordpress.com/2015/09/plugin\\_project-selection-window.png](https://harinivasganapathy.files.wordpress.com/2015/09/plugin_project-selection-window.png))

5. When the Plugin-in project page is displayed enter a project name, then click Next.

6. Complete the Content Page as per the below information and Click 'Finish'

**Plug-in ID :** Plugin ID is to help RIT differentiate among other plugins created by users.

**Plug-in Version:** Plugins can also have incremental features & bug fixes and each can have a version.

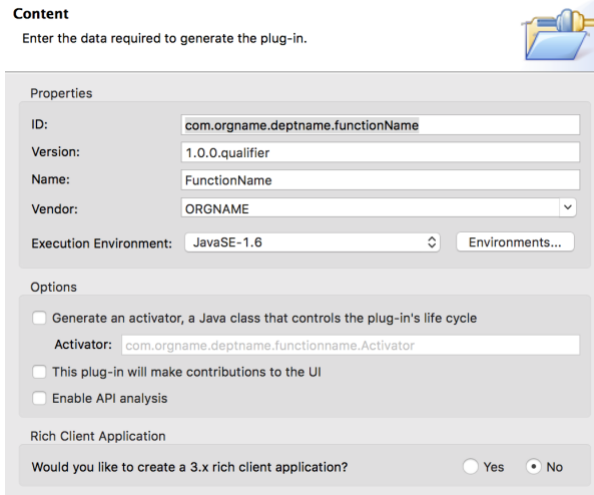
**Plug-in Name:** A short user friendly name that describes the Function's purpose.

**Plug-in Vendor:** Name of the organization or department within organization that owns the plug-in.

**Plug-in Options:** Disable both options.

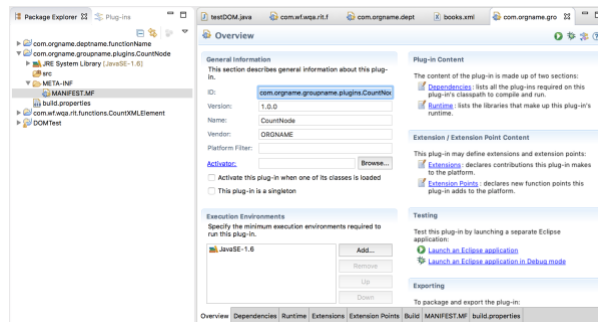
**Rich Client**

**Application:** Select 'No' for this options.



[https://harinivasganapathy.files.wordpress.com/2015/09/plugin\\_prop\\_page.png](https://harinivasganapathy.files.wordpress.com/2015/09/plugin_prop_page.png)

Eclipse opens up the Manifest File associated with the Plug-in development.

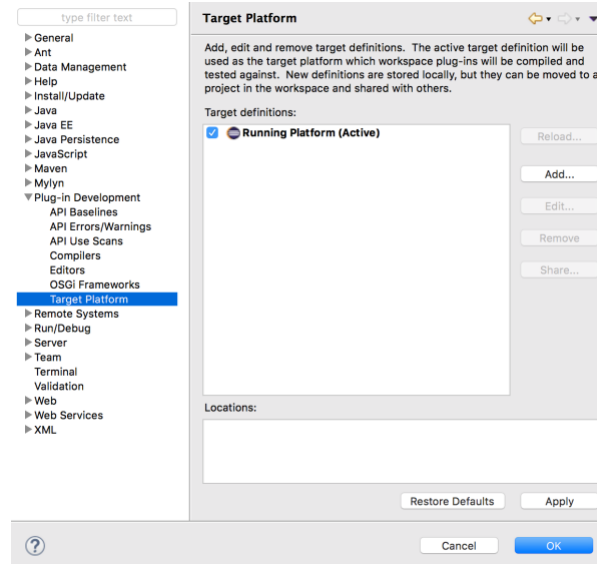


[https://harinivasganapathy.files.wordpress.com/2015/09/manifest\\_file.png](https://harinivasganapathy.files.wordpress.com/2015/09/manifest_file.png)

### Step 3: Setting up the dependencies to RIT

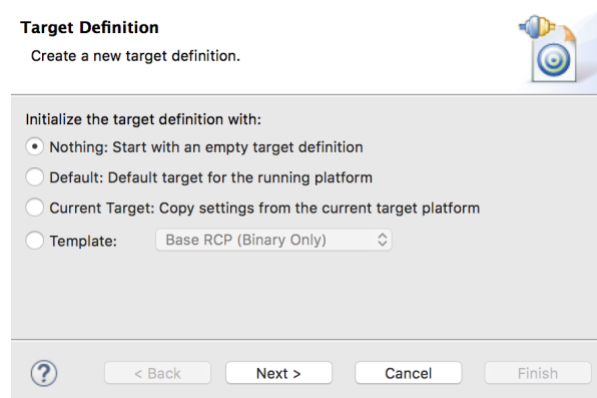
Since we are developing an plug-in that will be used in RIT Environment, We have to let Eclipse know that our plug-ins Target environment is RIT. So that all libraries needed for extending RIT is available during plugin development. This is done by adding below RIT plug-ins as Target Platform in Preferences.

1. Go to **Window > Preferences > Plug-in Development > Target Platform**



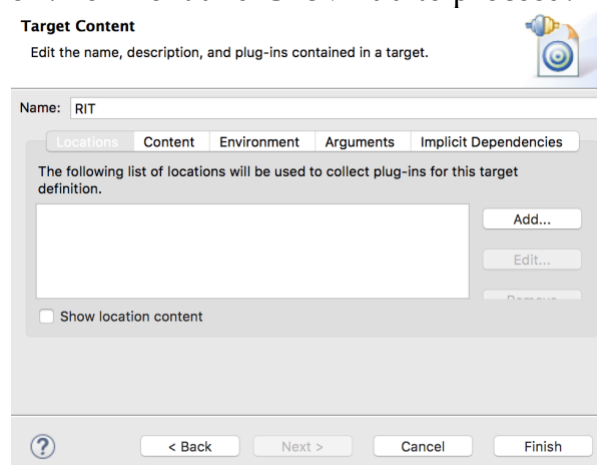
[https://harinivasganapathy.files.wordpress.com/2015/09/target\\_platform\\_page\\_1.png](https://harinivasganapathy.files.wordpress.com/2015/09/target_platform_page_1.png)

2. Click **Add** to an Target Definition to let eclipse know that the plug-in we are developing is for RIT.
3. In the following *Target Definition* Dialog, select “Nothing: Start with an empty target Definition” and Click “Next >”



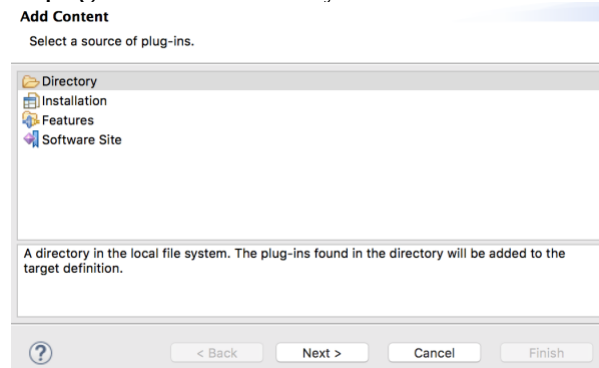
[https://harinivasganapathy.files.wordpress.com/2015/09/target\\_definition.png](https://harinivasganapathy.files.wordpress.com/2015/09/target_definition.png)

4. In the displayed *Target Content* page Give a meaningful name like RIT to signify the Target definition is related to RIT environment and Click **Add** to proceed.



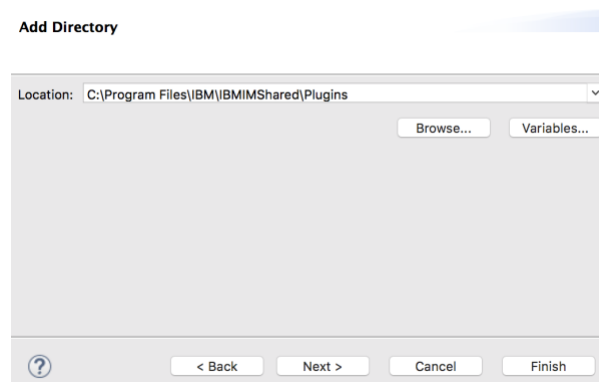
<https://harinivasganapathy.files.wordpress.com/2015/09/targetcontent.png>

5. In the displayed *Add Content* page select **Directory** and click **Next >** to proceed.



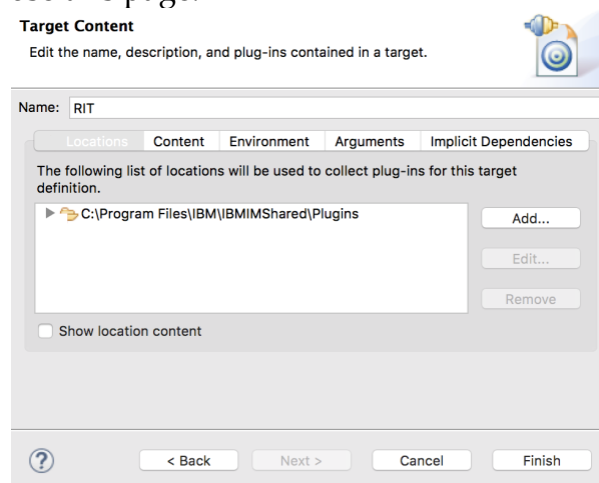
<https://harinivasganapathy.files.wordpress.com/2015/09/add-content.png>

6. In the *Add Directory* page click **Browse** and navigate to the IBM® Rational® Integration Tester installation folder (C:\Program Files\IBM\IBMIMShared\Plugins, by default) and click **Next >**.

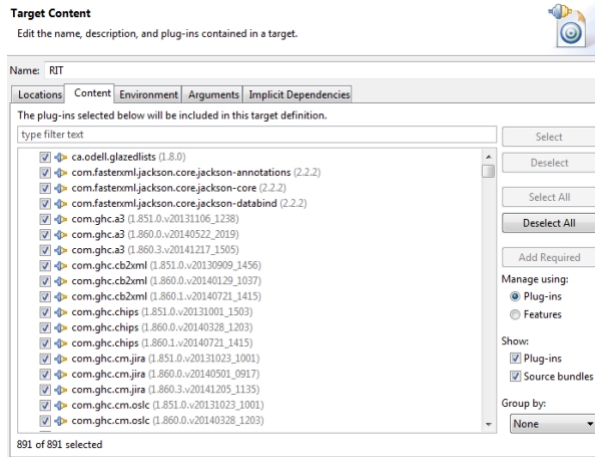


<https://harinivasganapathy.files.wordpress.com/2015/09/add-directory.png>

7. The *Target Content* page will now show the location we added and plugin available in the location. Click **Finish** to Close this page.



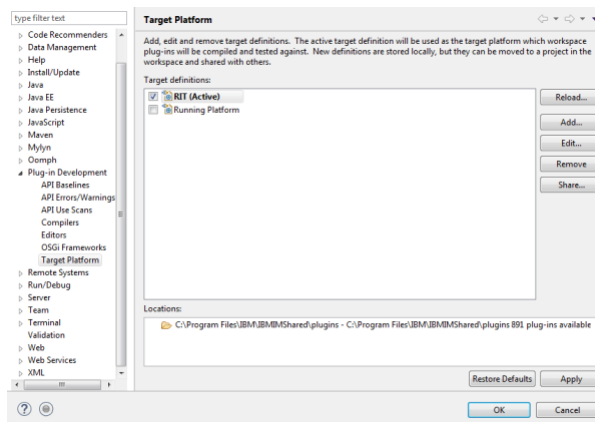




([https://harinivasganapathy.files.wordpress.com/2015/09/target\\_content\\_loc.png](https://harinivasganapathy.files.wordpress.com/2015/09/target_content_loc.png))

8. All of the plug-ins in the Rational Integration Tester folder is read and listed in the Plug-ins tab of the Target Platform preferences.

1. com.ghc.\*
2. com.greenhat.\*
3. com.ibm.greenhat.\*
4. com.ibm.rational.rit.\*

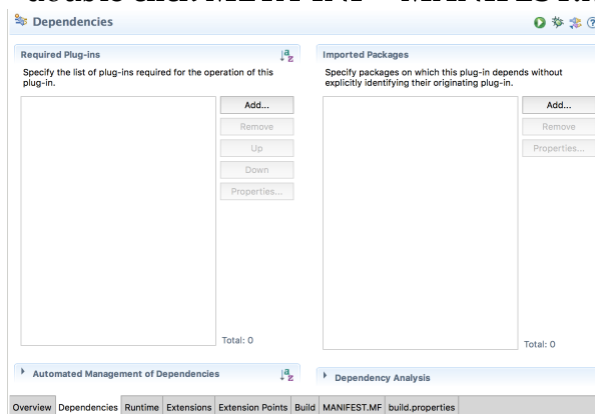


([https://harinivasganapathy.files.wordpress.com/2015/09/preference\\_target\\_platform.png](https://harinivasganapathy.files.wordpress.com/2015/09/preference_target_platform.png))

9. Click OK to close the Preferences dialog.

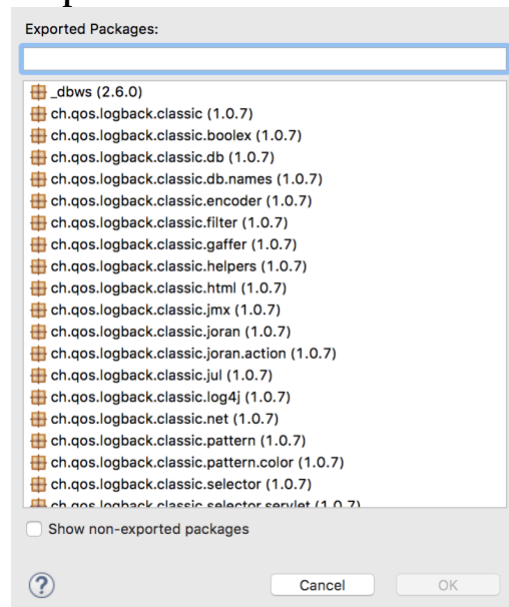
**Step 4:** Setting the dependency in the manifest

1. From the **Project explorer** > double click **META-INF** > **MANIFEST.MF**



[https://harinivasganapathy.files.wordpress.com/2015/09/manifest\\_dependencies.png](https://harinivasganapathy.files.wordpress.com/2015/09/manifest_dependencies.png)

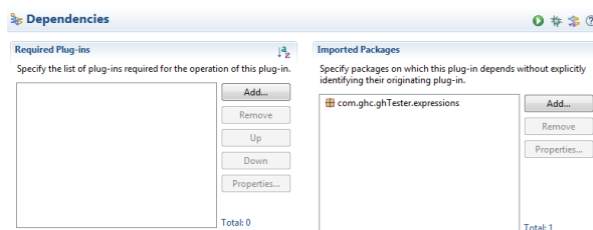
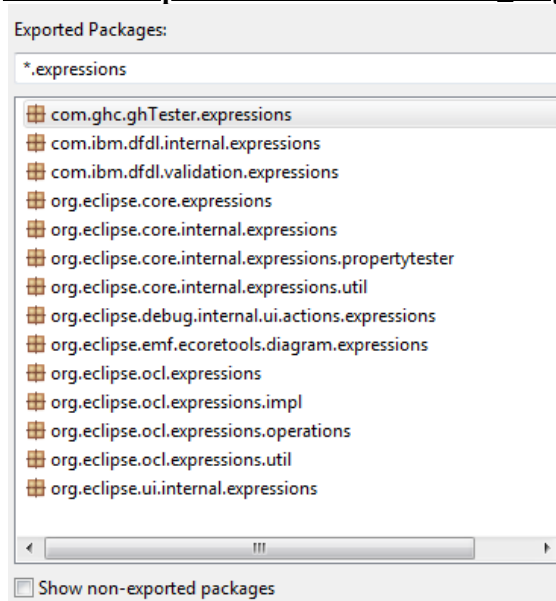
2. In the opened editor select the **Dependencies** tab and click **Add** next to Imported Packages.



<https://harinivasganapathy.files.wordpress.com/2015/09/exported-packages.png>

3. In the displayed The Package Selection dialog filter using *\*.expressions* wildcard and select *com.ghc.ghTester.expressions* and click Ok.

[https://harinivasganapathy.files.wordpress.com/2015/09/after\\_target\\_addition.png](https://harinivasganapathy.files.wordpress.com/2015/09/after_target_addition.png)



[https://harinivasganapathy.files.wordpress.com/2015/09/gh\\_target\\_adding.png](https://harinivasganapathy.files.wordpress.com/2015/09/gh_target_adding.png)

4. Save the Manifest file

Ok we have got all our initial setup ready. This concludes our first part of the journey together. Follow me into the next part of the trail. On our way we'll understand the foundational design of Function Class and implementation details of our Custom Function.

## **Continue to Part 2**

**(<https://harinivasganapathy.wordpress.com/2015/09/10/demystifying-rit-custom-function-part-2-putting-the-gears-together/>)**

Happy Reading!

[Blog at WordPress.com.](#)





