Path To QA Experience

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?

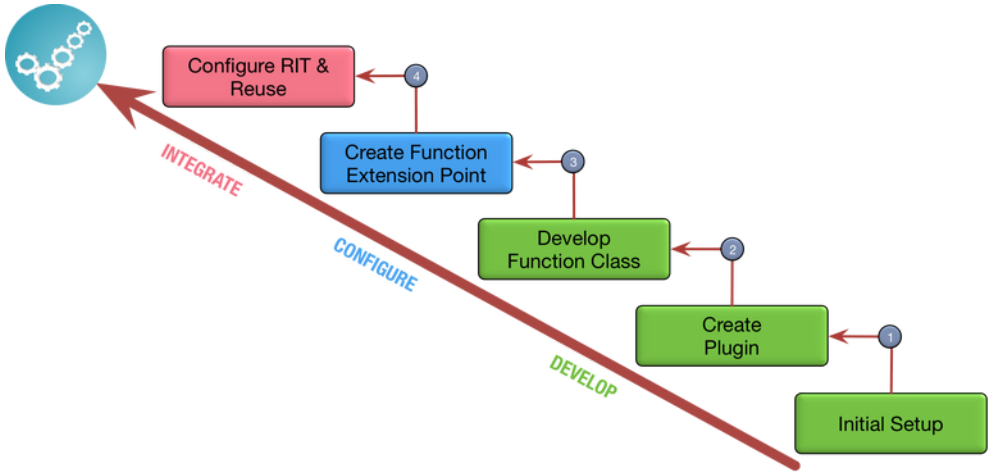
Before taking a detailed view of Functions let us review what is a *Custom Function* at an high abstracted level. A *Custom Function* is a Function that is designed, implemented and used by RIT user (tester or test designer) and not that provided by IBM by default. In addition to the built-in Functions 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)

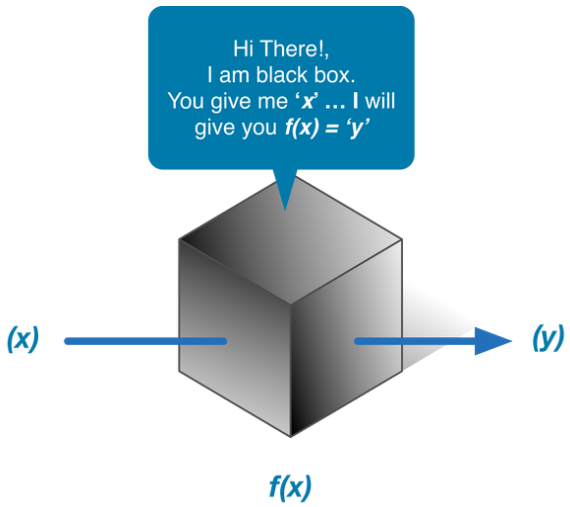
Step 1 – Design & Develop

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

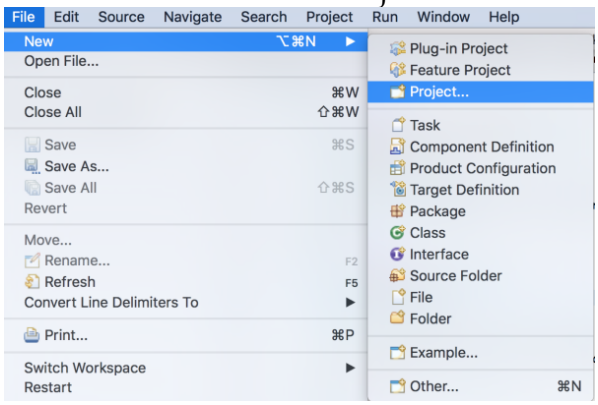
1. RIT installation
2. Eclipse installation – with PDE (Plugin Development Environment Support)

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.

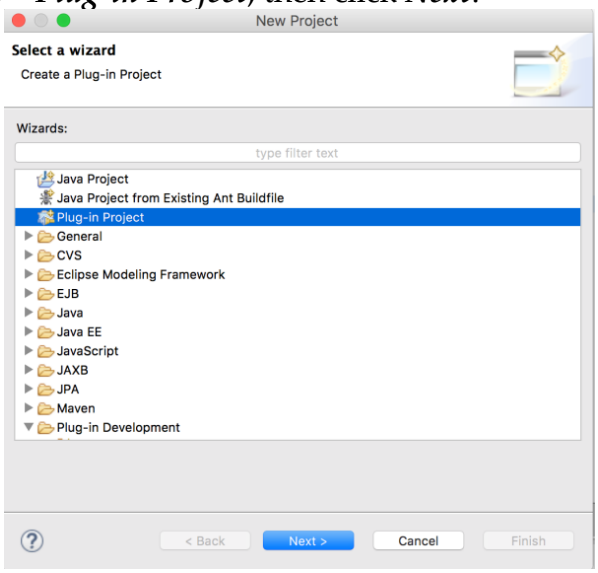
To create an Eclipse Plugin Project:

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)

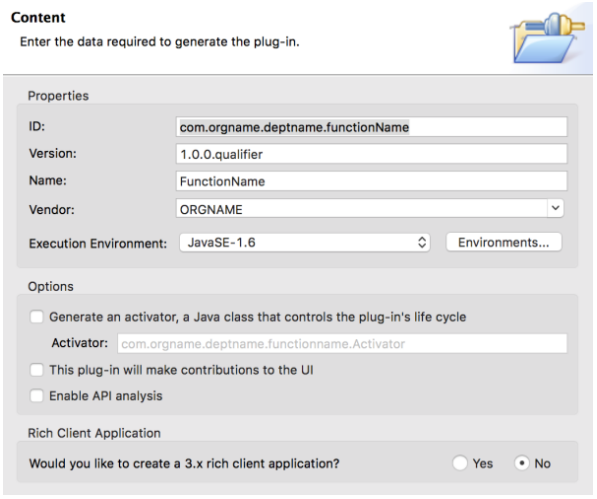
4. Select ***Plug-in Development*** > ***Plug-in Project***, then click ***Next***.



(https://harinivasganapathy.files.wordpress.com/2015/09/plugin_project-selection-window.png)

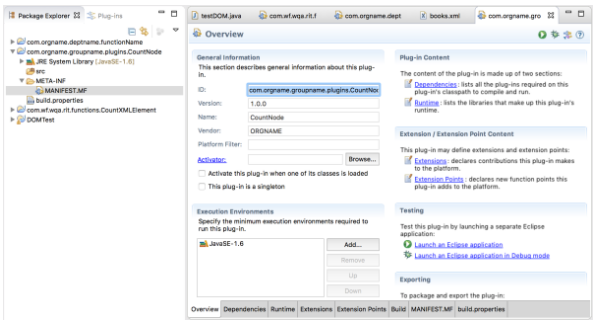
5. When the Plugin-in project page is displayed enter a project name, then click Next.
6. Complete the Content Page as per the below information and Click 'Finish'

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)

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

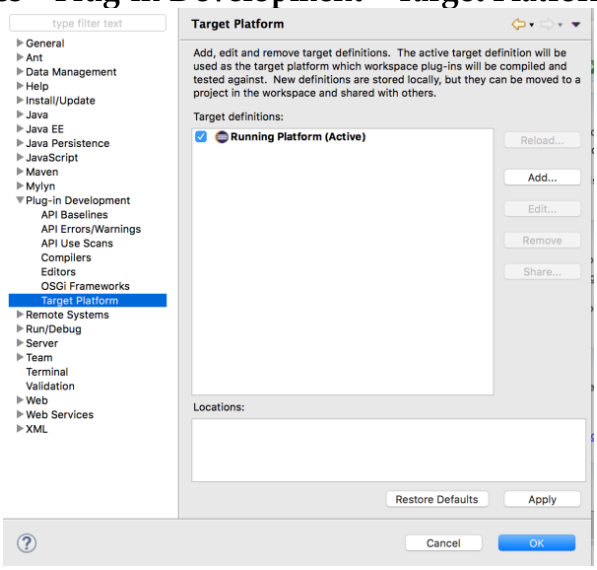


(https://harinivasganapathy.files.wordpress.com/2015/09/manifest_file.png)

Step 3: Setting up the dependencies to RIT

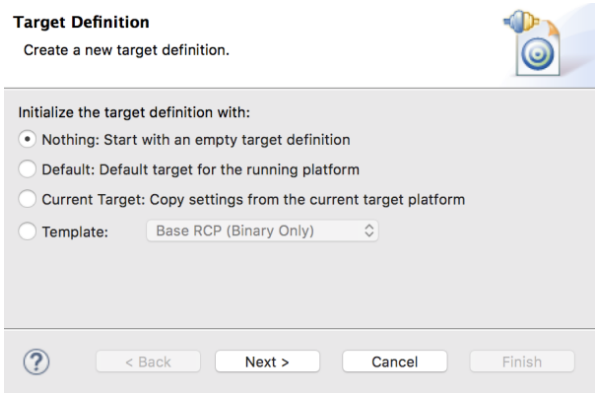
Since we are developing an plug-in that will be used in RIT Environment, We have to let Eclipse know that our plug-ins Target environment is RIT. So that all libraries needed for extending RIT is available during plugin development. This is done by adding below RIT plug-ins as Target Platform in Preferences.

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



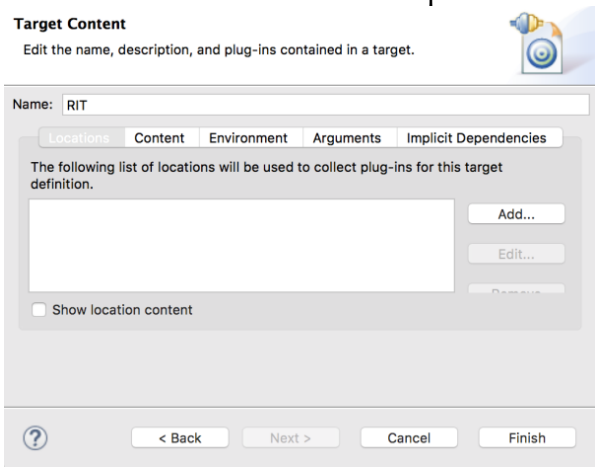
(https://harinivasganapathy.files.wordpress.com/2015/09/target_platform_page_1.png)

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 >**"



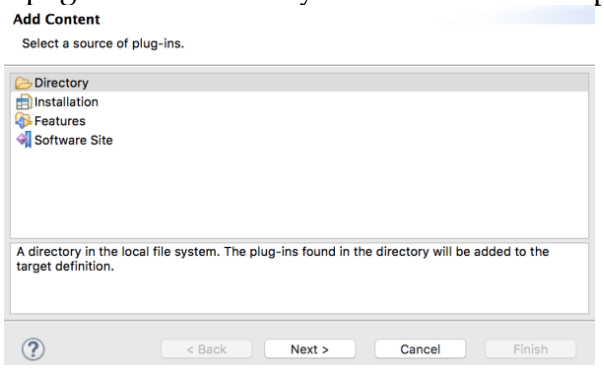
(https://harinivasganapathy.files.wordpress.com/2015/09/target_definition.png)

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.



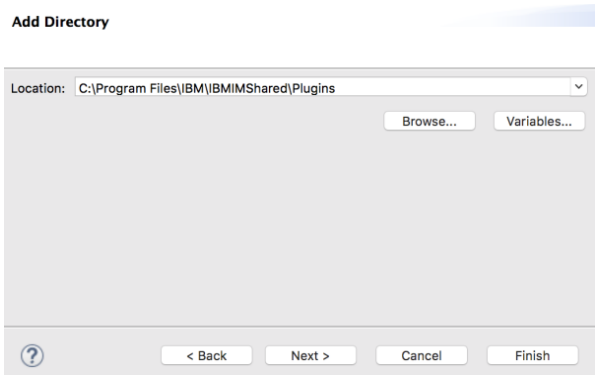
(https://harinivasganapathy.files.wordpress.com/2015/09/targetcontent.png)

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



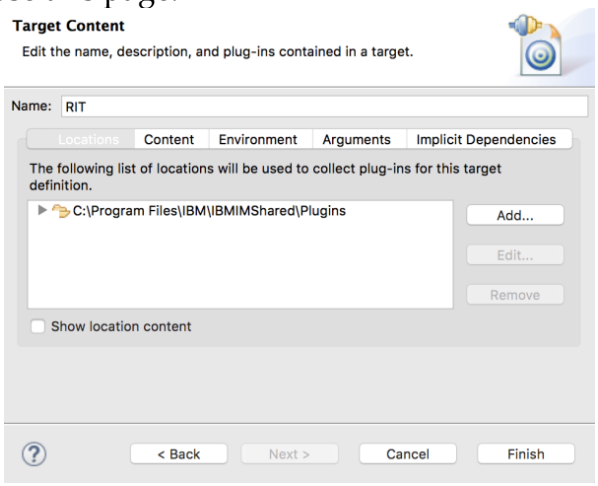
(https://harinivasganapathy.files.wordpress.com/2015/09/add-content.png)

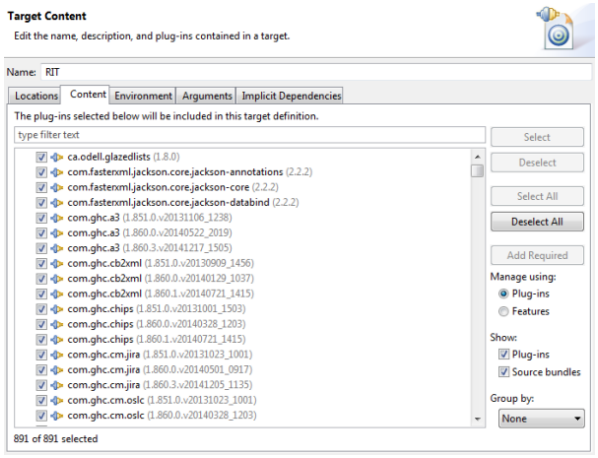
6. In the *Add Directory* page click **Browse** and navigate to the IBM® Rational® Integration Tester installation folder (C:\Program Files\IBM\IBMIMShared\Plugins, by default) and click **Next >**.



(https://harinivasganapathy.files.wordpress.com/2015/09/add-directory.png)

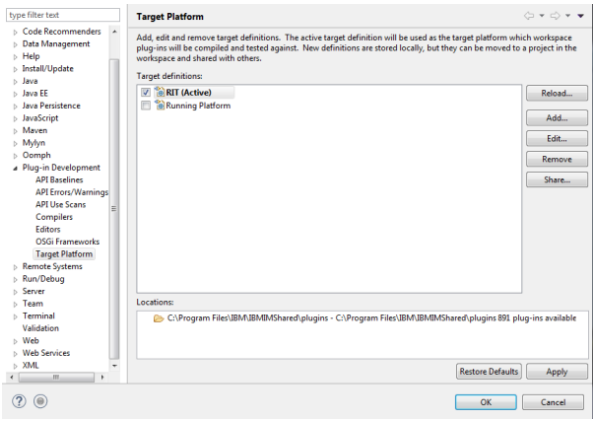
7. The *Target Content* page will now show the location we added and plugin available in the location. Click **Finish** to Close this page.





(https://harinivasganapathy.files.wordpress.com/2015/09/target_content_loc.png)

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.*

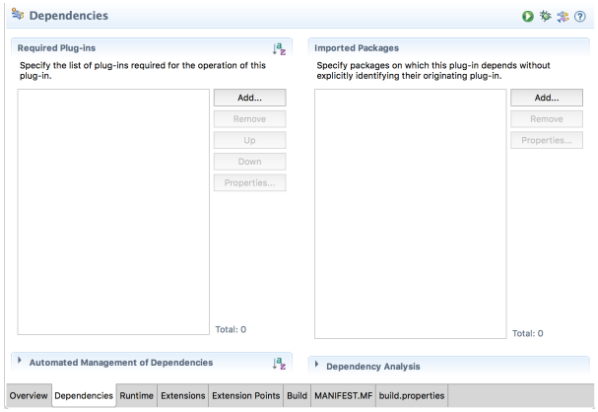


(https://harinivasganapathy.files.wordpress.com/2015/09/preference_target_platform.png)

9. Click **OK** to close the Preferences dialog.

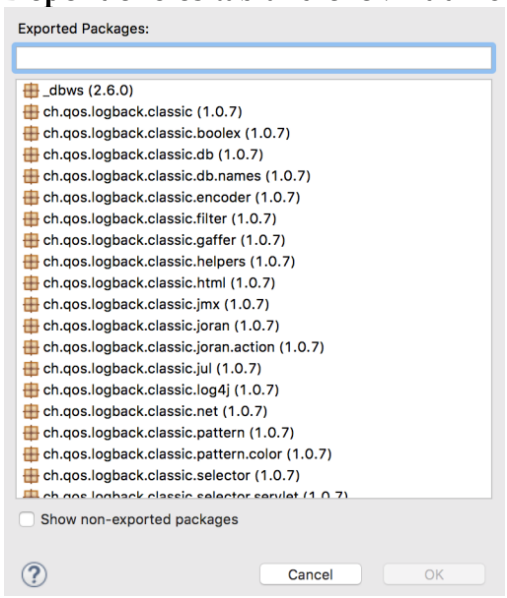
Step 4: Setting the dependency in the manifest

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



(https://harinivasganapathy.files.wordpress.com/2015/09/manifest_dependencies.png)

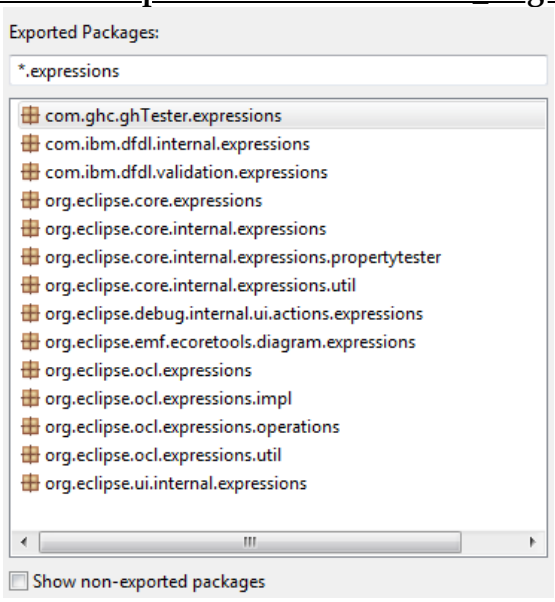
2. In the opened editor select the **Dependencies** tab and click **Add** next to Imported Packages.

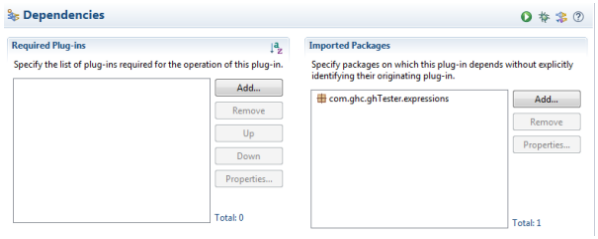


(https://harinivasganapathy.files.wordpress.com/2015/09/exported-packages.png)

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/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!

Posted in **Rational Integration Testing**, **RIT**/Tagged **Custom Functions**, **RIT**/**2 Comments**

2 thoughts on “Demystifying RIT Custom Function – Part 1: Gear up”

1. Pingback: **Demystifying RIT Custom Function – Part 3: Wrapping up for Deployment | Path To My Experience**
2. Pingback: **Rational integration tester – A Visual Scripting Experience | Path To My Experience**