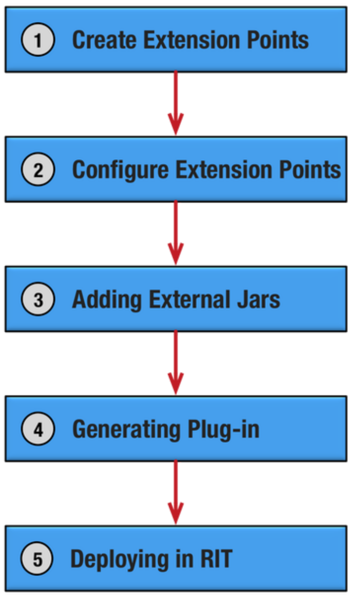
**SEPTEMBER 13, 2015SEPTEMBER 14, 2015** ⁄ **HARINIVAS GANAPATHY**

# 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**

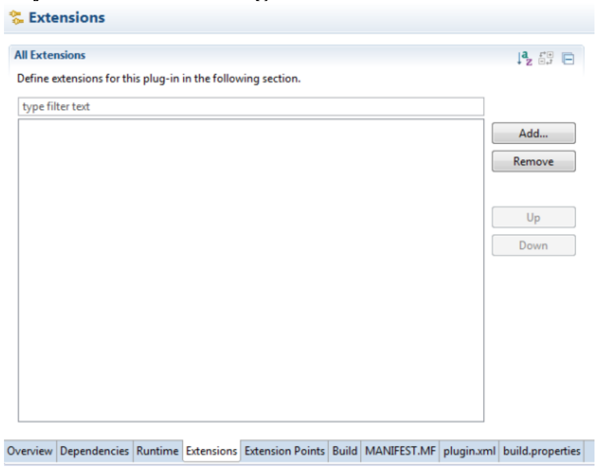


**(https://harinivasganapathy.files.wordpress.com/2015/09/deployment-steps.png)**

**Create Extension Points**

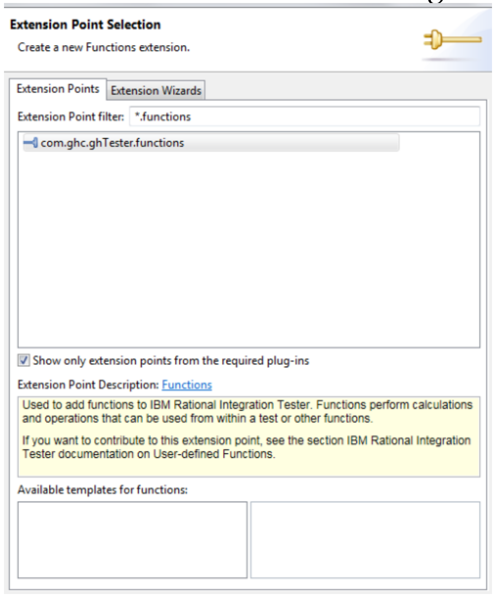
In order to allow 3rd party Functions added to RIT, IBM has created an extension point by means of 'com.ghc.ghTester.functions'. This extension point should be added to the extensions tab in our project's Manifest file.

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



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

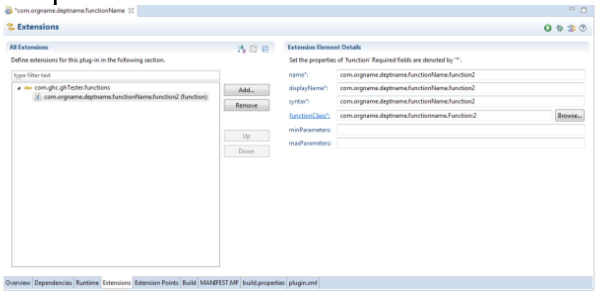
2. In the opened editor select the Extensions tab and click Add.
3. Select com.ghc.ghTester.functions in the New Extension Dialog


Extension Point Selection
Create a new Functions extension.
Extension Points
Extension Wizards
Extension Point filter:
*.functions
com.ghc.ghTester.functions
Show only extension points from the required plug-ins
Extension Point Description: Functions
Available templates for functions:

Used to add functions to IBM Rational Integration Tester. Functions perform calculations and operations that can be used from within a test or other functions.

If you want to contribute to this extension point, see the section IBM Rational Integration Tester documentation on User-defined Functions.

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


*com.orgname.deptname.functionName
Extensions
All Extensions
Define extensions for this plug-in in the following section.
type filter text
com.ghc.ghTester.functions
com.orgname.deptname.functionName.function2 (function)
Add...
Remove
Up
Down
Extension Element Details
Set the properties of 'function' Required fields are denoted by '*'.
Browse...
Overview
Dependencies
Runtime
Extensions
Extension Points
Build
MANIFEST.MF
build.properties
plugin.xml

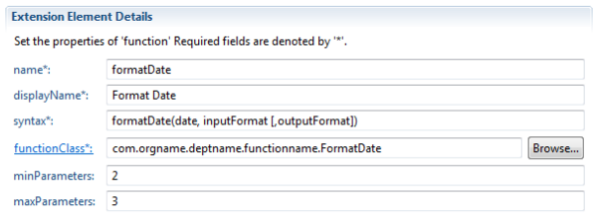
name*: com.orgname.deptname.functionName.function2
displayName*: com.orgname.deptname.functionName.function2
syntax*: com.orgname.deptname.functionName.function2
functionClass*: com.orgname.deptname.functionname.Function2
minParameters:
maxParameters:

**(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.



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.***

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

## 2 thoughts on "Demystifying RIT Custom Function – Part 3: Wrapping up for Deployment"

1. Pingback: **Demystifying RIT Custom Function – Part 2: Putting the gears together | Path To My Experience**
2.

   **rekha** says:
   **June 13, 2017 at 5:08 pm**
   Hi,,
   Thank you for sharing the detailed steps,
   I tried the same,
   I can see the function available in RIT/Allfunctions .
   The issue is -when I Am calling the function from ECMA script – it is giving me error/exception that incorrect arguments .
   Canu you pls, get your code here where you r calling function.

   **REPLY**