BOSCH
Invented for life

SFA_Test_Automation_Framework (V3.0)

inside.Docupedia Export

Author: Govindaraju T (RBEI/ESD-PP)
Date: 21-Dec-2020 12:01

Table of Contents

1 Overview of Test Automation Framework

1.1 Precondition:

It is assumed that before using this Framework user has basic knowledge or Overview of below Test Script Language / Test Tools

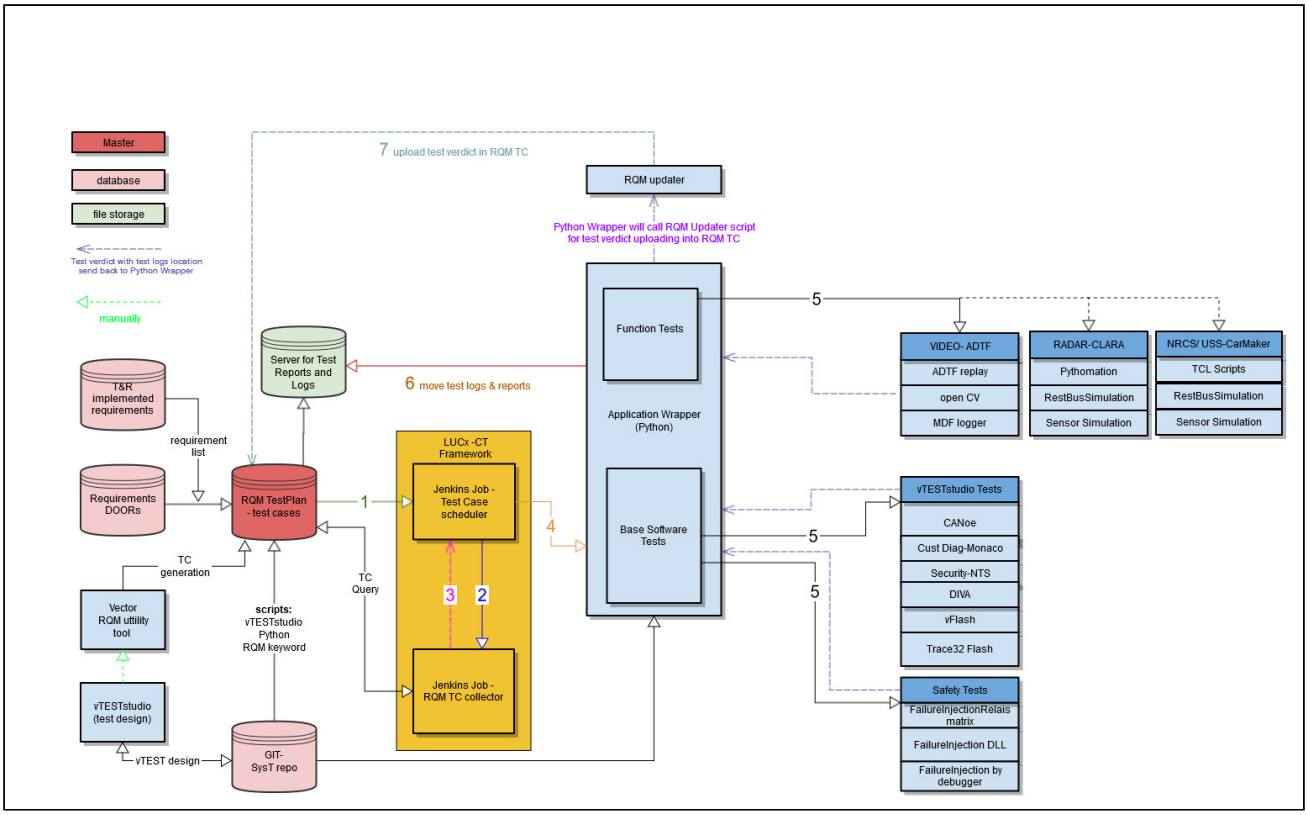
- GIT overview
- Basics of Jenkins
- Basics of Python
- Vector tools like : CANoe, vTESTstudio, CANape, RQM Utility tool etc..
- Basics of CLARA, CarMaker & ADTF HiL tool
- Basics of RQM

1.2 Introduction:

- The objective of this framework is to achieve Complete Automation with one button click by fetching the Test Cases from RQM to Test Execution and Uploading the result back to RQM.
- The Continuous Test Regression Automation Framework is built based on modular approach and developed with Generic libraries which can be re-used across multiple projects with minimum adaptation.
- The Framework is developed in such way that it can integrate easily with other specific test automation tools / solutions with an API (e.g.: ADTF based tests can be integrated easily with one function call under Test_Trigger() function)

1.3 Framework Structure or Architecture:

Below is the architecture of CT(Continous Testing) Regression Automation Framework



2 Python basics necessary for Framework

2.1 Introduction to Python:

„Python is object-oriented:

- „Structure supports such concepts as polymorphism, operation overloading and multiple inheritance

„It's free (open source):

- „Downloading and installing Python is free and easy

„Source code is easily accessible:

- „Free doesn't mean unsupported! Online Python community is huge

„It's portable:

- „Python runs virtually every major platform used today
- „As long as you have a compatible Python interpreter installed, Python programs will run in exactly the same manner, irrespective of platform

„It's powerful:

- „Dynamic typing, Built-in types and tools, Library utilities (e.g. xml.etree.ElementTree, NumPy)
- „Automatic memory management

2.2 Python Data Types/ Parameter:

- Frequently used data types are :

Object type	Example literals/creation
• Numbers	1234, 3.1415, 999L, 3+4j, Decimal
• Strings	'spam', “content"
• Lists	[1, [2, 'three'], 4]
• Dictionaries	{'food': 'spam', 'taste': 'yum'}
• Tuples	(1,'spam', 4, 'U')

- In our Frameowrk we have Intialized all the Parameters related to project specific Path under one file called Parameter_List.py and we advice the user of the framework to adapt the path in one file

2.3 Python Condition and Loops:

- *Frequently used conditions are: if, elif and else*

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the **if** keyword

Example from Python Wrapper :

```
#Launch Canoe
if _checkIfProcessRunning('CANoe64'):
    MyLog.Append("[TEST_TRIGGER][INFO]::: Canoe Instance is running already this will be killed and re-started ")
    print "[TEST_TRIGGER][INFO]::: Canoe Instance is running already this will be killed and re-started"
    try :
        command= 'taskkill /f /im "CANoe64.exe"'
        ret = subprocess.call(command, shell=True, cwd=SCRIPT_DIR, stdout=sys.stdout, stderr=sys.stderr)
    except:
        print "Exception in killing the Canoe Instance"
    CANoeRUN_Set(param.CANoe_cfg_filepath,os.path.join(param.TestUnits_Path,TestCaseInfo['vTESTstudio Test Case Unit']),MyLog)
else:
    print "Inside Else loop"
    CANoeRUN_Set(param.CANoe_cfg_filepath,os.path.join(param.TestUnits_Path,TestCaseInfo['vTESTstudio Test Case Unit']),MyLog)
_Move_CANoe_reports_Files_Set(param.CANoe_cfg_basepath, os.path.join(param.Canoe_log_path,Test_Case_Folder))
```

- *Frequently used Loops are: for, While*
  Python has two primitive loop commands:
  - for loops
  - while loops

  The for loop:
  for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
  This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.
  With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.
  Example from Python Wrapper :

```
#Clear the log contents
ADTF_Log_file_New_Path=param.ADTF_Log_Path+"\\Output_backup.txt"
f = open(ADTF_Result_file_Path)
# Take backup of log file
f2 = open(ADTF_Log_file_New_Path, "a")
for line in f.readlines():
    f2.write("\n")
    f2.write(line)
```

  The While loop:
  With the while loop we can execute a set of statements as long as a condition is true.
  Example:
  Print i as long as i is less than 6:

```
i = 1

while i < 6:
  print(i)
  i += 1
```

2.4 Python functions:

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result

Creating a Function

In Python a function is defined using the def keyword:

Example

```
def my_function():
  print("Hello inside a function")
```

Calling a Function

- To call a function, use the function name followed by parenthesis
- Normaly function is called either inside if __name__ == "__main__": for executing the functions one by one
- Also we call function inside function also to perform specific acton example : Killing existing process, or open file etc.

Example from Python Wrapper :

```
if __name__ == "__main__":
    #clonging of GIT sys_VV_test branch
    GIT_Clone_SytemSystem_Branch_set()
    ## Get the TCF file from RQM through command line arguments.
    o_commandlineArguments = parse_commandline()
    ## Pass command line arguments to local variables...
    TCF_File_Path = o_commandlineArguments.testinput
    #Call TCF_File Reader function to extract Test Case details received from RQM
    TCF_FileReader_Set(TCF_File_Path)
    #Extract TestCase Version number received from RQM
    TC_software_Version=_Extract_Software_Version_TCF_Get()
    #Extract DUT Version number and Compare with TestCase Version_Number and flash the software if needed
    Software_Flashing_Check(TC_software_Version)
```

2.5 Python RegEx:

A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.

RegEx can be used to check if a string contains the specified search pattern.

RegEx Module : Python has a built-in package called `re`, which can be used to work with Regular Expressions, and import the re module using import re

RegEx Functions: The `re` module offers a set of functions that allows us to search a string for a match

Function	Description
findall	Returns a list containing all matches
search	Returns a Match object if there is a match anywhere in the string
split	Returns a list where the string has been split at each match
sub	Replaces one or many matches with a string

Example from Python Wrapper :

```
    #Read any one of the file and get the DUT version number
    with open(UART_Log_list[1],"r")as obj:
        UART_Data=obj.read()
        match=re.search(".*INFO].*Release:(.*)",UART_Data)
        if match:
            SW_Version_Number= match.group(1).strip()
    MyLog.Append("[SW_DOWNLOAD][STATUS]:::Recevied sofware version from DUT is :::{0} :::SUCCESS \n".format(SW_Version_Number))
    print "SUCCESS:::Recevied sofware version from DUT is :::{0} \n".format(SW_Version_Number)
```

Hint: Additional you can refer to following link https://regexr.com/ for exploring more details

2.6 Python Exception handling:

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the `try` statement

Python Try Except

The `try` block lets you test a block of code for errors.

The `except` block lets you handle the error.

The `finally` block lets you execute code, regardless of the result of the try- and except blocks.

Example from Python Wrapper :

```
print "STATUS:::Reading the TCF file:::START"
testCaseInfo_pattern = r'TestCase.*Info.([ A-Za-z0-9_]+)\s*=\s*\"(.+)\"'
testCaseInfo_re = re.compile(testCaseInfo_pattern)
try:
    file=open(tcfFile,'r')
    lines = file.readlines()
    file.close()
except Exception, reason:
    MyLog.AppendException("[TCF_FileReader][STATUS]::: Opening TCF_FileLog_Set in the function :::EXCEPTION:::ERROR[1]")
    MyLog.AppendException(reason)
    print "STATUS::: Opening TCF_FileLog_Set in the function :::EXCEPTION:::ERROR[1]"
    print reason
    sys.exit(1)
```

3 Framework Architecture

3.1 Coding Standards:

Do's	Dont's
Please write the code with Header information contains at least following information • Author Name • Python Version • Run from Windows command line(python ATV_Python_Wraper_Application.py)	Do not start writing the code without code Header information
Immediately after code header, please list all the imports in a good block to ensure better readability	Do not import files in between the code

Make sure all the Objects are initialized immediately after the import in one block for better readability
 	
Example:
 	
`#*****************************************************************`
`#  OBJECT_INTIALIZATION`
`#`
`#*****************************************************************`
`MyLog = Log.LOG(param.logpath+"\\"+'ATV_Python_Wraper.txt')`
	Do not Initialize the objects in between the code

Please document in the code details of the function description and its purpose,
 	
input parameters and return parameters. before defining the functions for better readability
 	
Example:
 	
`#*****************************************************************************************`
`#  FUNCTION BLOCK: FUNCTION_1: Read TCF file`
`# Function name will read TCF file received from Jenkins`
`# INPUT PARAMETERS:`
`#  1) string: TCF File`
`#`
`# OUTPUT PARAMETERS:`
`#  1) Dictionary : Test Case Info`
`#*****************************************************************************************`
	Please do not start defining the functions without documentation of the purpose, input parameters and return parameters

Please follow below naming convention for defining the functions : Software_Flashing_Set-->Do the Action Software_Flashing_Check(String)-->Checking some action should return True or FalseSoftware_Flashing_Get()--->Return the vlaue we need	Do not follow any other name than the recommended ones Examples: 123(), Flashing(), TCF_File() etc...

Do's	Dont's

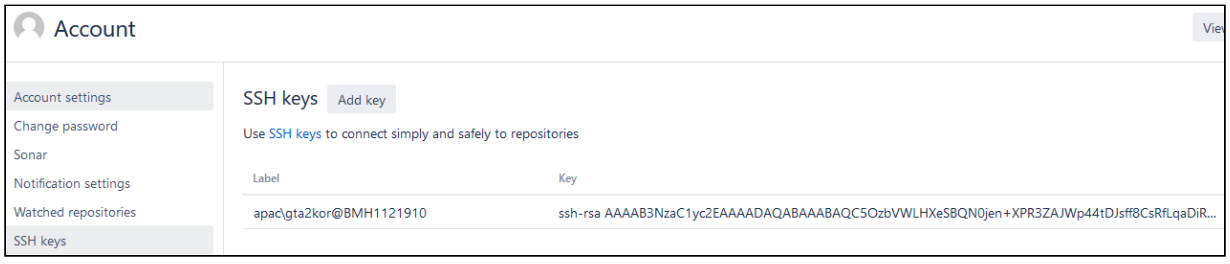
Please use relative path's and define all the obsolete path parameters in a separate file
 	
Example :
 	
Define an absolute parameter like below :
 	
`#Canoe_Updater Params`
`CANoe_cfg_basepath = r'F:\\100_Automation\\ATV_Python_Wraper\\Scripts\\RBS_ATV_PI2003'`
`CANoe_cfg_filepath = r'F:\\100_Automation\\ATV_Python_Wraper\\Scripts\\RBS_ATV_PI2003\\MAIN.cfg'`
 	
Access these Parameters within function in the main python file :
 	
`#Launch Canoe`
`if _checkIfProcessRunning('CANoe64'):`
`MyLog.Append("[TEST_TRIGGER][INFO]::: Canoe Instance is running already this will be killed and re-started ")`
`print "[TEST_TRIGGER][INFO]::: Canoe Instance is running already this will be killed and re-started"`
`try :`
`command= 'taskkill /f /im "CANoe64.exe"'`
`ret = subprocess.call(command, shell=True, cwd=SCRIPT_DIR, stdout=sys.stdout, stderr=sys.stderr)`
`except:`
`print "Exception in killing the Canoe Instance"`
`CANoeRUN_Set(param.CANoe_cfg_filepath,os.path.join(param.TestUnits_Path,TestCaseInfo['vTESTstudio Test Case Unit']),MyLog)`
`else:`
`print "Inside Else loop"`
`CANoeRUN_Set(param.CANoe_cfg_filepath,os.path.join(param.TestUnits_Path,TestCaseInfo['vTESTstudio Test Case Unit']),MyLog)`
`_Move_CANoe_reports_Files_Set(param.CANoe_cfg_basepath, os.path.join(param.Canoe_log_path,Test_Case_Folder))`
	Please do not use Hard coded path inside the main python files and inside the functions and do not use hard coded delays

Use Exception handling as much as possible in the below scenarios and In case of Exception handle it with multiple retry : • File Handling • Any hardware operations • Any other API calls (e.g: RQM connection, ADTF connection etc..)	Do not perform below operation without Exception handling : • File Handling • Any hardware operations • Any other API calls (e.g: RQM connection, ADTF connection etc..)
Push all your sources into GIT Repo as soon as modification is done to avoid data loss	Do not place any source code in local PC

3.2 GIT Introduction and Usage:

GIT First time Installation in new PC :

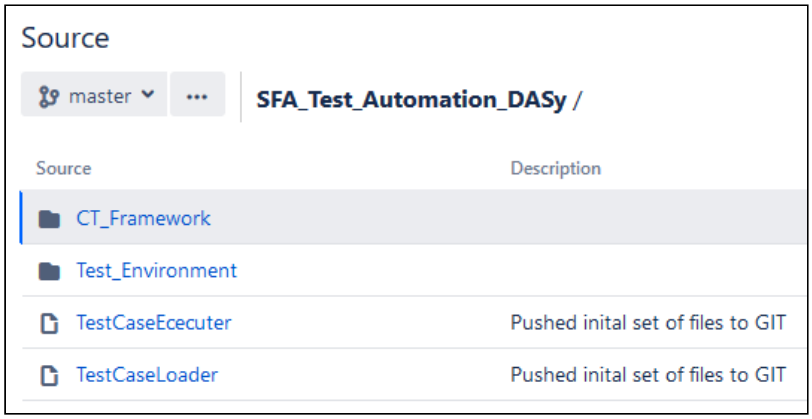
- Make sure you have installed GIT from IT_Workplace_ToolKIT as a software package
- Make sure you have the access to social coding repo : https://sourcecode01.de.bosch.com:7999/sfa_test_automation
- Make sure you have linked SSH key present in your computer from your path : *C:\Users\gta2kor\.ssh* to your profile in social coding shown below



- For more details on GIT please refer to below link
  https://www.atlassian.com/git/tutorials/setting-up-a-repository

GIT Introduction and Usage :

- GIT is used as Software Configuration Management Tool for SFA -Test Automation and our Repo is: https://sourcecode01.de.bosch.com:7999/sfa_test_automation
- Any Product Area Repo shall contain 2 branches called
  - *Develop,*
  - *Master*
- All the automation code will be placed under SFA_Test_Automation_PA(*e.g:DASy*) _Master branch and this branch contains below folder structure



- Under **CT_Framework/Test_Scheduler/scripts** contains all the automation code required to run the framework from a new PC
- Under **Test_Environment** folder, project specific *RBS* and vTESTStudio files (*.vtuexe) will be maintained by project team
  - The folder structure maintined in GIT and PC and Parameterlist.py should be the same
- **TestCaseExecuter** and **TestCaseLoader** files are the groovy scripts required for running the Jenkins jobs

GIT Important commands :

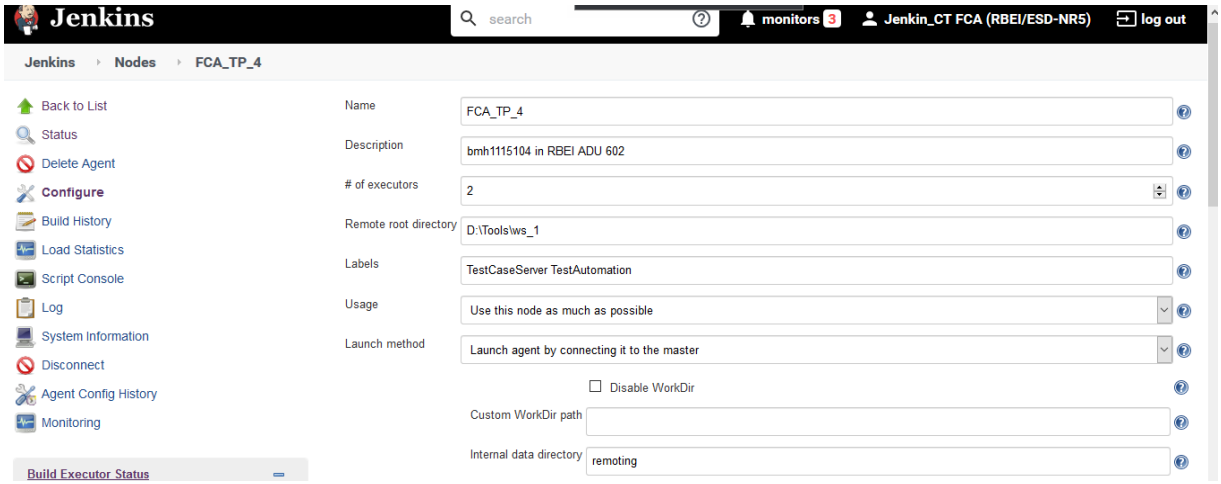
- git clone → ssh://git@sourcecode01.de.bosch.com:7999/sfa_test_automation/sfa_test_automation_dasy.git
- Switch to branch**-->**git clone -branch master/ssh://git@sourcecode01.de.bosch.com:7999/sfa_test_automation/sfa_test_automation_dasy.git
- To see the changes between the PC and GIT server -->git status
- to push the changes from local PC to GIT follow below commands
  - git add filenames (Copy the changed link from git status)
  - git commit -m "Message for committing"
  - git push

3.3 Jenkins configuration and Purpose:

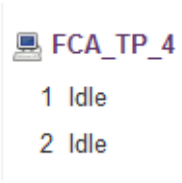
Jenkins Configuration in new PC :

Adding a New Node to Jenkins Server:

- Add a Test_PC as new node in the Jenkins
- Set the option - number of Executors to "2" when both **RQM_Collector** and **Test_Executer Jobs are to be run on the same PC**
- Set the option- number of Executors to "1" when only **Test_Executer Jobs is to be run on the other PC**
- Example of the Jenkins Node configuration:



- Make sure you have the following folder added in the Test_PC (Jenkins_slave) from where you are connecting to Jenkins server
  - D:\Tools\ws_1 and place below folders (Below folder is not a mandatory, some times Jenkins_server itself create a folder called workspace)
- Also place the **"slave-agent.jnlp"** file in the same folder and double click on this file to connect to the Jenkins Server
- After adding the Test_PC to Server make sure on the Jenkins server the slave PC is connected and status shown as IDLE



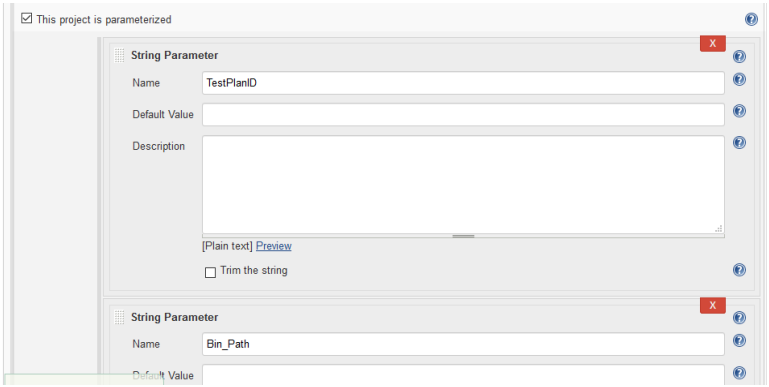
Configuring the Jenkins Job on the Server and linking into PC:

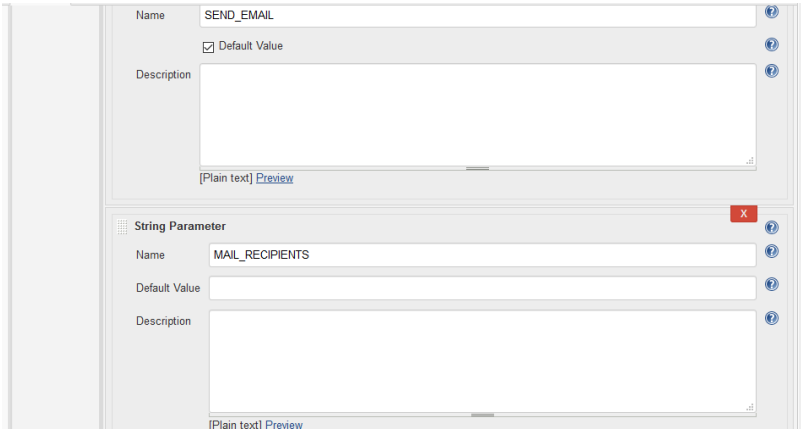
There are 2 Jenkins Jobs are used RQM_Collector and TestExecution

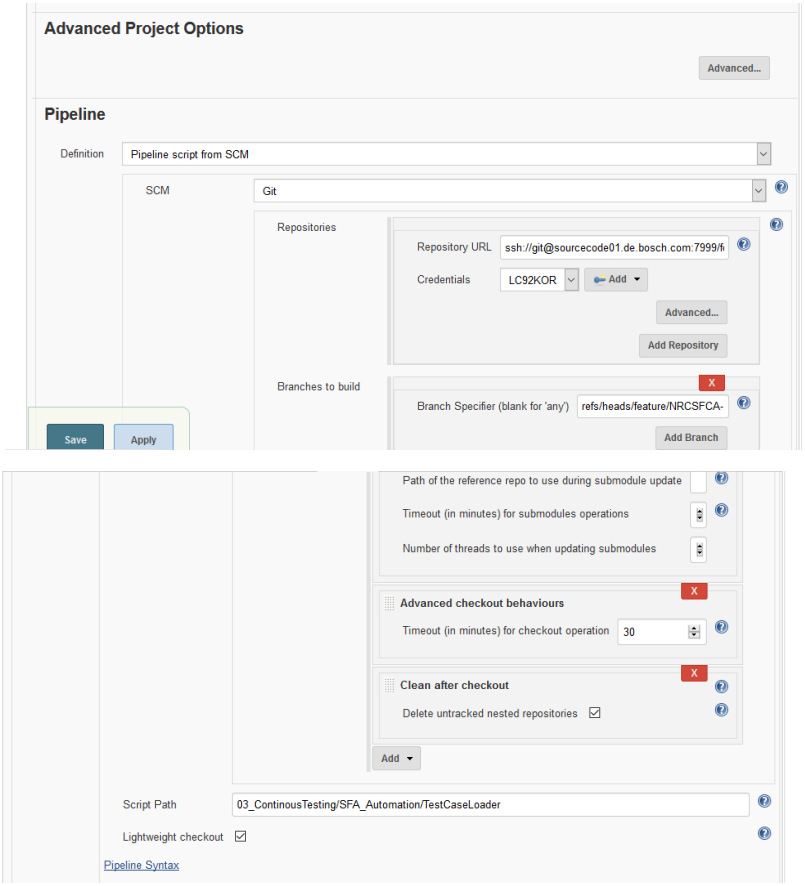
- RQM_Collector will connect to RQM and extract all the TestCase and convert them into .tcf file format and starts TestExecution jobs

Configuring the RQM_Collector:

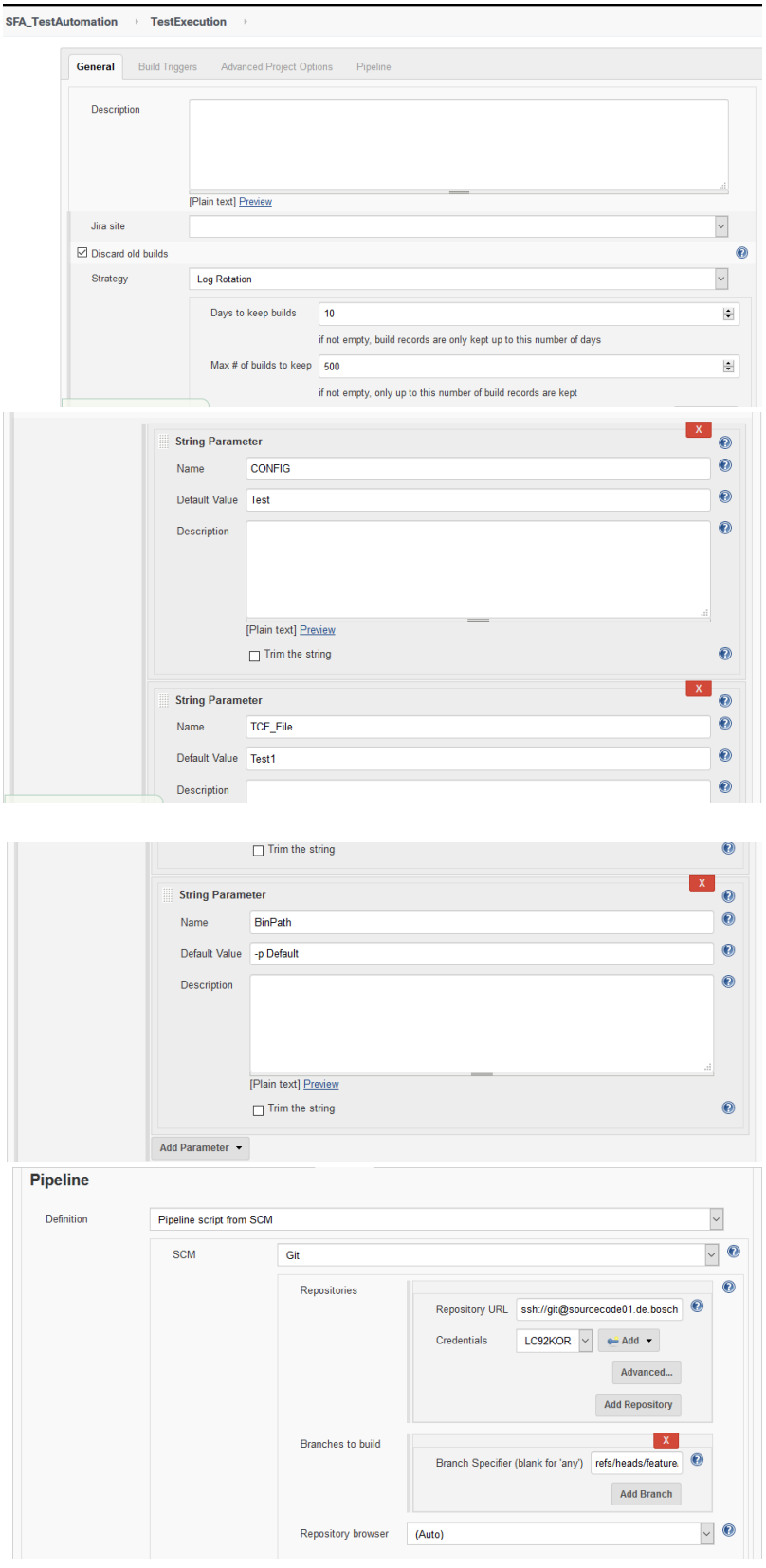
- Add a Job as pipeline job and configure like below and name this Job as **RQM_Collector**
-

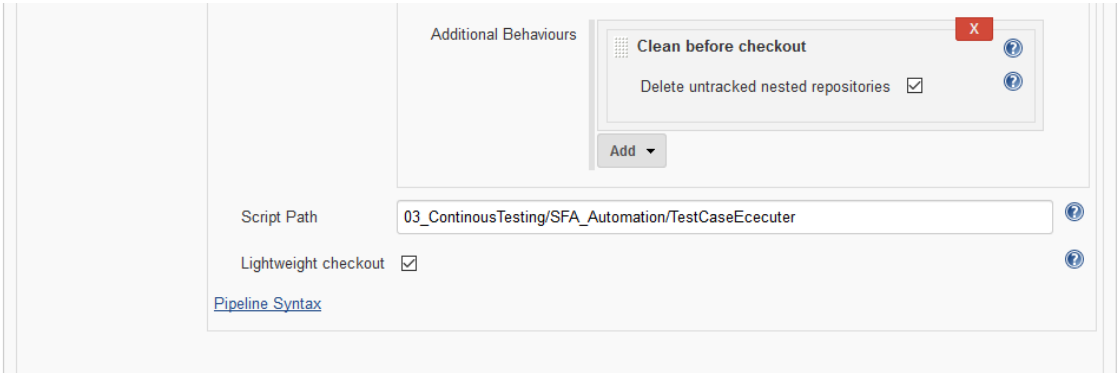






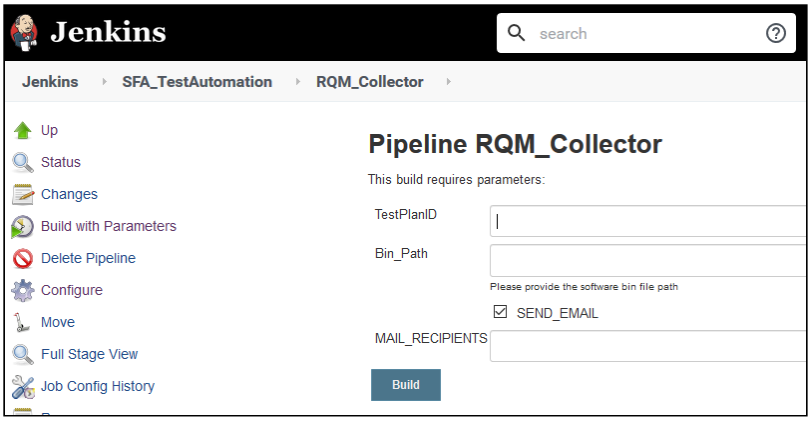
- Add another pipeline Job and and configure like below and name this Job as **TestExecution**

- 

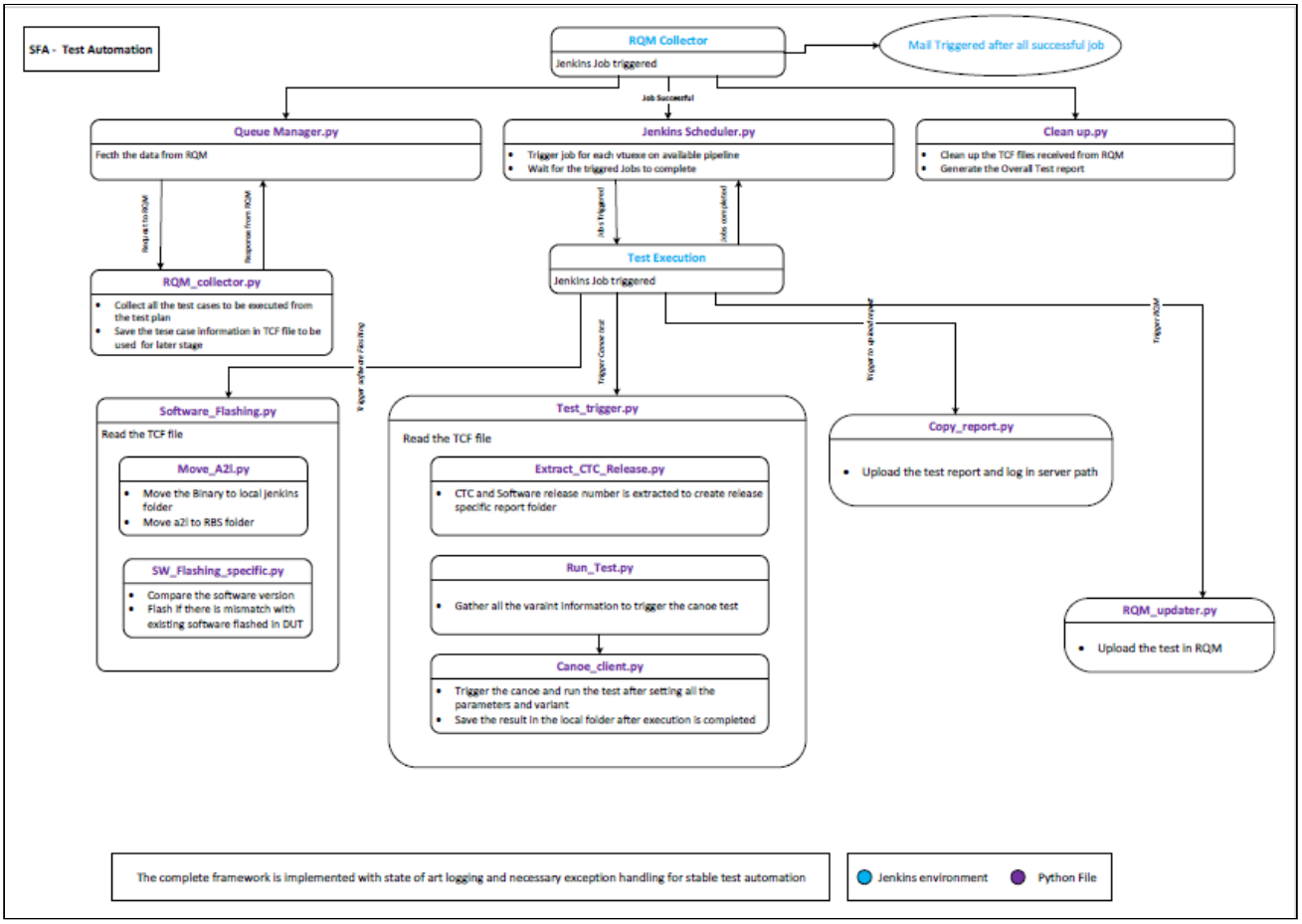


With above points taken care, we are now in a ready state to start Test Execution

- **Go to Jenkins Page and open the job "RQM_Collector"**
  - Navigate to "Build with Parameter" page and enter the below
    - **TestPlanID** --> *Test Plan ID, RQM repo name and Credential (e.g: 23898,"VID_FCA _NRCS2 (qm)",gta2kor)* *Make sure User ID is the same as used on the user_data.txt present in GIT under the folder CT_Framework\RQM_TestCase_Collector)*
    - **Bin_Path** → *Software Binary Path Location*
      (\
      \abtvdfs2.de.bosch.com
      \ismdfs\ism\Archive\CC\Video\NRCS2_FCA\Jenkins_Artifacts\01_NRCS2_FCA_RELEASE\02_Customer_Releas\NRCS2_PP6_FCA_WL75MY21_CVPAM006.6\NRCS2_PP6_FCA_WL75MY21_CVPAM006.6.zip)



3.4 Technical flow(Flowchart) of the SFA-Automation Framework



3.5 Python Wrapper Application details:

- Precondition : Make sure user has the below manditory attrubutes in RQM
  - Test_Iteration, TestCase,test platform, vTESTstudio Test Unit
  - Incase of any project specific attrubute to be added, in that case please add into attribute.text file located in *(..\CT_Framework\RQM_TestCase_Collector)* and do the necessary change in the RQM_Test_Case_Collector.py
  - Make sure the attribute vTESTstudio Test Unit contains the test unit name without ***".vtuexe' and it should be in a following format for manual test cases (e.g.. TestUnit_DIAG_Services)***
- The Python Wrapper is devided into four different modules as Software_Flashing.py, Test_Triger.py, RQM_Updater.py, Copy_Report.py, here onwards in this training document we refer all these modules collectively with specific name called as "Python Wrapper Application"
- The "Python Wrapper Application" will be triggered by **TestExecution** Jenkins Job. this job will execute corresponding Test, will upload the log into Server and will update the result in RQM Automatically.
- **TestExecution** Job will trigger parallely "Python Wrapper Application" to as many Jenkins_Slaves that are connected with the Label "TestAutomation".
  - The first stage of the **TestExecution** will create a temprory file called ***"Received.tcf"***, which contains the test case details received from the **RQM_TestCase_Collector** and this content of Received.tcf file will used in further stages of **TestExecution** job
- The "Python Wrapper Application" is implemented with Exception handling so that Jenkins execution shall not hang .
- Handshake between the Jenkins jobs is implemented in the "Python Wrapper Application" such that the server can keep record of all executed test cases from each of the Jenkin_Slaves along with its status like Failed(0)/ Completed(1)/ Aborted(2)

```
def Concise_Test_Report(build_number):
    try:
        if not os.path.exists(param.MAIN_Server_Report_Path):
            os.makedirs(param.MAIN_Server_Report_Path)

        with open(param.Test_Result_Path,'r') as rls:
            result=rls.read()
            #print(result)

        with open(param.TCF_FileName_Path,'r') as tcf:
            tcf_filename=tcf.read()
            #print(tcf_filename)

        to_report=[]
        to_report.append(tcf_filename)
        to_report.append(build_number)
        to_report.append(result)

        file = open(param.MAIN_Server_Report_Path+'\\Full_Report.txt', "a+")
        for item in to_report:
            file.write(item)
            file.write(' ')

        file.write('\n')
        file.close

    except Exception as reason:
        MyLog.Append("[Concise_Test_Report][STATUS]::: Handshake Result to Main Path:::EXCEPTION:::ERROR[5]")
        MyLog.AppendException(reason)
        print("[Concise_Test_Report][STATUS]::: Handshake Result to Main Path:::EXCEPTION:::ERROR[5]")
        print(reason)
        sys.exit(1)
```

```
TCF_FileReader_Set(param.TCF_File_Path)
build_number=sys.argv[1]
if ('CanoeHIL' in TestCaseInfo['test platform']):
    Copy_Report_Set(param.CANOE_REPORT_FOLDER, param.UPLOAD_FOLDER, param.logpath, build_number)
    Concise_Test_Report(build_number)
    #Move_CANoe_reports_Files_Set(param.CANoe_cfg_basepath, os.path.join(param.Canoe_log_path,Test_Triger_Return[0]))
elif ('ADTF' in TestCaseInfo['test platform'] ):
    Copy_Report_Set(param.source_path_ADTF, param.destination_path, Test_Triger_Return[0], param.logpath+"\\ATV_Python_
```

Function flow of the "Python Wraper Application" is explained below

- **Functional Block 1: Software_Flashing:**
  - Software_Flashing block contains 2 Python files
    - **Software_Flashing.py**
    - **Software_Flashing_Specific.py** :
  - **Software_Flashing.py**
    - Contains Generic function for reading the TCF_File, Checking the Software version received between TestCase vs DUT

```
if __name__ == "__main__":
    try:
        ## Get the TCF file from RQM through command line arguments.
        o_commandlineArguments = parse_commandline()
        ## Pass command line arguments to local variables...
        TCF_File_Path = o_commandlineArguments.testinput
        ## Pass command line arguments to local variables...
        BIN_File_Path = o_commandlineArguments.binpath
        print("BIN_File_Path",BIN_File_Path)

        #Call TCF_File Reader function to extract Test Case details received from RQM
        Extract_TCF_File_Name(TCF_File_Path)
        TCF_FileReader_Set(TCF_File_Path)
        Copy_Binaries_Move_A2L(BIN_File_Path,TestCaseInfo['variant'])
        TC_software_Version=Extract_Software_Version_TCF_Get()
        Software_Flashing_Check(TC_software_Version)
    finally:
        if MyLog:
            del MyLog
```

  - **Software_Flashing_Specific()**

- *Software flashing is project specific function, Please modify Software_Flashing_Get() and Software_Flashing_Set() functions according to the flashing procedure followed in the Project*

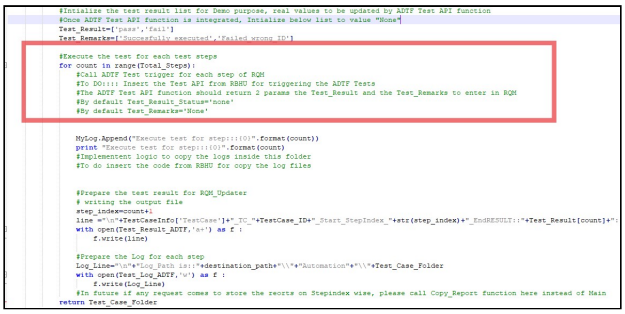
```
def Software_Flashing_Check(Version_Number):
    try:
        #MyLog.Append("Software_Flashing_Check:::START \n")
        #Compare the DUT version and Received version from TC ,
        TC_SW_Version_Number=Version_Number
        DUT_SW_version_Number=Software_Flashing_Get()
        #MyLog.Append("DUT Software Version is :::"+DUT_SW_version_Number+"\n")
        MyLog.Append("[SW_DOWNLOAD][INFO]Software Version received from TestCase is :::"+TC_SW_Version_Number+"\n")
        print "Software Version received from TestCase is :::{0} \n".format(TC_SW_Version_Number)

        #if version mismatch flash the software otherwise continue without flashing
        if DUT_SW_version_Number==TC_SW_Version_Number:
            MyLog.Append("[SW_DOWNLOAD][STATUS]:::Software Version in the DUT is Matched with TC:::SUCCESS\n")
            MyLog.Append("[SW_DOWNLOAD][INFO]:::Test will be continued without Flashing of the software")
            print "STATUS:::Software Version in the DUT is Matched with TC ::: SUCCESS \n"
        else:
            MyLog.Append("[SW_DOWNLOAD][STATUS]:::Software Version in the DUT is not Matched with TC ::: MISMATCH \n")
            print "STATUS:::Software Version in the DUT is not Matched with TC ::: MISMATCH \n"
            MyLog.Append("[SW_DOWNLOAD][INFO]:::Flashing of the TC's software will start and version is :: {0}".format(TC_SW_Version_Number))
            #Start Flashing the Software in case of mismatch
            Software_Flashing_Set(TC_SW_Version_Number)
        #MyLog.Append("Software_Flashing_Check:::END \n")
        #sys.exit(0)
    except Exception, reason:
        MyLog.AppendException("[SW_DOWNLOAD][STATUS]::: Software_Flashing_Check in the function ::: EXCEPTION:::ERROR[2] /n")
        MyLog.AppendException(reason)
        print "STATUS:::Software_Flashing_Check in the function::: EXCEPTION:::ERROR[2]"
        print reason
        sys.exit(1)
    finally:
            return  "SUCCESS"
```

```
if __name__ == "__main__":
    #clonging of GIT sys_VV_test branch
    GIT_Clone_SytemSystem_Branch_set()
    ## Get the TCF file from RQM through command line arguments.
    o_commandlineArguments = parse_commandline()
    ## Pass command line arguments to local variables...
    TCF_File_Path = o_commandlineArguments.testinput
    #Call TCF_File Reader function to extract Test Case details received from RQM
    TCF_FileReader_Set(TCF_File_Path)
    #Extract TestCase Version number received from RQM
    TC_software_Version=_Extract_Software_Version_TCF_Get()
    #Extract DUT Version number and Compare with TestCase Version_Number and flash the software if needed
    Software_Flashing_Check(TC_software_Version)
```

- **Functional Block 2:Test_Triger_Set()**
  - Based on the tTest_Platform attribute received from .tcf file, it will trigger the corresponding function call "Restbus" or "ADTF" etc..
  - For Restbus:
    - It will copy the RBS and *.vtuexe from GIT repo (SFA_Test_Automation_PA(e.g:DASy\Test_Environment\Canoe_RBS) to Test_PC location (Path .\\100_Automation\ATV_Python_Wraper\Scripts\RBS)
    - It will load a configuration with prefix ***TC_*** followed by TestUnit name
      - Make sure before starting the automated test you do not have any testconfiguration loaded in Test configuration view of Canoe
      - Make sure you select the report format to ***HTML/XML in CANoe->Options->General->Test Feature Set***
    - TesCase_log_folder_name will be created in the Test_PC
    - Launch the CANoe and run the corresponding "***.vtuexe**" received from **"*.tcf"** file

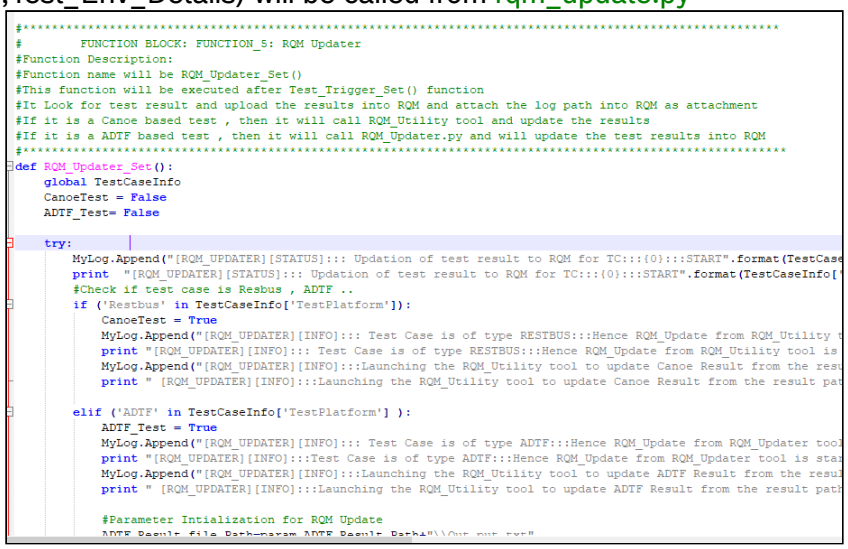
  - For Functional Tests(ADTF/CLARA/CarMaker):
    - A folder with the name **"TestCase_log_folder_name"** will be created automatically to store the logs
      - Within the above folder a text file named **"Out_put.txt"** containing details of TestCase, TestStep and Test Result for uploading to RQM is created automatically.
      - Also another text file named **"Test_log_file_name_For_RQM_Update"** containing test log links from the server, that is to be attached along with the results in RQM is created automatically.
      - The called FunctionTests API shall return 2 Parameter, the first is "Test_Result" and second is "Test_Remarks"

- *Please insert the Function Tests API in the below location*

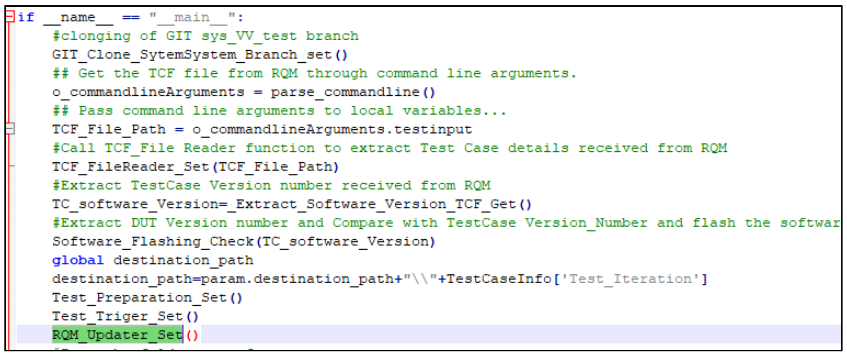


```
#Intialize the test result list for Demo purpose, real values to be updated by ADTF Test API function
#Once ADTF Test API function is integrated, Initialize below list to value "None"
Test_Result=['pass','fail']
Test_Remarks=['Successfully executed','Failed wrong ID']

#Execute the test for each test steps
for count in range(Total_Steps):
    #Call ADTF Test trigger for each step of RQM
    #To DO:::: Insert the Test API from RBHU for triggering the ADTF Tests
    #The ADTF Test API function should return 2 params the Test_Result and the Test_Remarks to enter in RQM
    #By default Test_Result_Status='none'
    #By default Test_Remarks='None'

    MyLog.Append("Execute test for step:::{0}".format(count))
    print "Execute test for step:::{0}".format(count)
    #Implementent logic to copy the logs inside this folder
    #To do insert the code from RBHU for copy the log files


    #Prepare the test result for RQM_Updater
    # writing the output file
    step_index=count+1
    line ="\n"+TestCaseInfo['TestCase']+"_TC_"+TestCase_ID+"_Start_StepIndex_"+str(step_index)+"_EndRESULT::"+Test_Result[count]+"
    with open(Test_Result_ADTF,'a+') as f :
        f.write(line)

    #Prepare the Log for each step
    Log_Line="\n"+Log_Path is::"+destination_path+"\\"+"Automation"+"\\"+Test_Case_Folder
    with open(Test_Log_ADTF,'w') as f :
        f.write(Log_Line)
    #In future if any request comes to store the reorts on Stepindex wise, please call Copy_Report function here instead of Main
return Test_Case_Folder
```

- **Functional Block 3: RQM_Updater_Set()**
  - Check for the type of test to be executed from Test_Platform attribute from .tcf file (Restbus / ADTF)
  - In case of "Rest bus", RQM_Utility from Vector will be launched to create execution record and update the result in RQM
  - In case of Functional Tests, the RQM_Test_Init (ADTF_Result_file_Path,ADTF_Log_file_Path,Test_Iteration_details,Test_Plan_details ,Test_Env_Details) will be called from rqm_update.py



```
#*************************************************************************************************************
#        FUNCTION BLOCK: FUNCTION_5: RQM Updater
#Function Description:
#Function name will be RQM_Updater_Set()
#This function will be executed after Test_Trigger_Set() function
#It Look for test result and upload the results into RQM and attach the log path into RQM as attachment
#If it is a Canoe based test , then it will call RQM_Utility tool and update the results
#If it is a ADTF based test , then it will call RQM_Updater.py and will update the test results into RQM
#*************************************************************************************************************
def RQM_Updater_Set():
    global TestCaseInfo
    CanoeTest = False
    ADTF_Test= False

    try:
        MyLog.Append("[RQM_UPDATER][STATUS::: Updation of test result to RQM for TC:::{0}:::START".format(TestCase
        print  "[RQM_UPDATER][STATUS::: Updation of test result to RQM for TC:::{0}:::START".format(TestCaseInfo['
        #Check if test case is Resbus , ADTF ..
        if ('Restbus' in TestCaseInfo['TestPlatform']):
            CanoeTest = True
            MyLog.Append("[RQM_UPDATER][INFO]::: Test Case is of type RESTBUS:::Hence RQM_Update from RQM_Utility t
            print "[RQM_UPDATER][INFO]::: Test Case is of type RESTBUS:::Hence RQM_Update from RQM_Utility tool is
            MyLog.Append("[RQM_UPDATER][INFO]:::Launching the RQM_Utility tool to update Canoe Result from the rest
            print " [RQM_UPDATER][INFO]:::Launching the RQM_Utility tool to update Canoe Result from the result pat

        elif ('ADTF' in TestCaseInfo['TestPlatform'] ):
            ADTF_Test = True
            MyLog.Append("[RQM_UPDATER][INFO]::: Test Case is of type ADTF:::Hence RQM_Update from RQM_Updater tool
            print "[RQM_UPDATER][INFO]:::Test Case is of type ADTF:::Hence RQM_Update from RQM_Updater tool is star
            MyLog.Append("[RQM_UPDATER][INFO]:::Launching the RQM_Utility tool to update ADTF Result from the resul
            print " [RQM_UPDATER][INFO]:::Launching the RQM_Utility tool to update ADTF Result from the result path

            #Parameter Intialization for RQM Update
            ADTF_Result_file_Path=param.ADTF_Result_Path+"\\Out_put.txt"
```



```
if __name__ == "__main__":
    #clonging of GIT sys_VV_test branch
    GIT_Clone_SytemSystem_Branch_set()
    ## Get the TCF file from RQM through command line arguments.
    o_commandlineArguments = parse_commandline()
    ## Pass command line arguments to local variables...
    TCF_File_Path = o_commandlineArguments.testinput
    #Call TCF_File Reader function to extract Test Case details received from RQM
    TCF_FileReader_Set(TCF_File_Path)
    #Extract TestCase Version number received from RQM
    TC_software_Version=_Extract_Software_Version_TCF_Get()
    #Extract DUT Version number and Compare with TestCase Version_Number and flash the softwar
    Software_Flashing_Check(TC_software_Version)
    global destination_path
    destination_path=param.destination_path+"\\"+TestCaseInfo['Test_Iteration']
    Test_Preparation_Set()
    Test_Triger_Set()
    RQM_Updater_Set()
```

- **Functional Block 4: Copy_Report_Set()**
  - After Test_Case is executed, reports are copied into Server folder (example→ \\abtvdfs2.de.bosch.com\ismdfs\ida\abt\video\ATV_dev\03_V&V_restricted\Test Data\\SoftwareVersion\\Automation)

```
#*****************************************************************************************************
#        FUNCTION BLOCK: FUNCTION_5: Copy Report file
# This function is to copy report from source folder to destination folder
# INPUT PARAMETERS:
#           1) string: Source folder path, Destination folder path, TestCase ID
#
# OUTPUT PARAMETERS:
#           1) Dictionary : Test Case Info
#*****************************************************************************************************

def Copy_Report_Set(source, destination, TC,Python_Wraper_log,UART_Log_Path, ignore=None):

    try:
        # Use Asolute Path
        src = os.path.abspath(source)
        dst = os.path.abspath(destination)
        Python_Wraper_log_Path= os.path.abspath(Python_Wraper_log)
        UART_Log_Path= os.path.abspath(UART_Log_Path)

        #nowTime = time.strftime("%Y-%m-%d-%H-%M-%S", time.localtime())
        src = os.path.join(src+ '\\' + TC)
        dst_folder=dst
        dst = os.path.join(dst+ '\\' + TC)


        MyLog.Append ("[Copy_Report][info]:::Source path --> " + src)
        MyLog.Append ("[Copy_Report][info]:::Destination path --> " + dst )

        #Check if Destination Directory Existing , if not create a new one
        if not os.path.exists(dst_folder):
            MyLog.Append ("[Copy_Report][info]:::Destination Directory is not exisitng we are creating a new one --> " + dst_folder )
            print "[Copy_Report][info]:::Destination Directory is not exisitng we are creating a new one --> {0} " .format(dst_folder)
            os.makedirs(dst_folder)
```

```
if __name__ == "__main__":
    #cloning of GIT sys_VV_test branch
    GIT_Clone_SytemSystem_Branch_set()
    ## Get the TCF file from RQM through command line arguments.
    o_commandlineArguments = parse_commandline()
    ## Pass command line arguments to local variables...
    TCF_File_Path = o_commandlineArguments.testinput
    #Call TCF_File Reader function to extract Test Case details received from RQM
    TCF_FileReader_Set(TCF_File_Path)
    #Extract TestCase Version number received from RQM
    TC_software_Version=_Extract_Software_Version_TCF_Get()
    #Extract DUT Version number and Compare with TestCase Version_Number and flash the software if needed
    Software_Flashing_Check(TC_software_Version)
    global destination_path
    destination_path=param.destination_path+"\\"+TestCaseInfo['Test_Iteration']
    Test_Preparation_Set()
    Test_Triger_Set()
    RQM_Updater_Set()
    #Pass the folder name for report copy
    destination_path=param.destination_path+"\\"+TestCaseInfo['Test_Iteration']
    #Test_Iteration= TestCaseInfo['Test_Iteration']
    #Copy_Report function
    if ('Restbus' in TestCaseInfo['TestPlatform']):
        Copy_Report_Set(param.source_path_Canoe, destination_path, Test_Case_Folder, param.logpath+"\\ATV_Python_Wraper.txt",param.UART_Log_Path,ignore = None)
    elif ('ADTF' in TestCaseInfo['TestPlatform'] ):
        Copy_Report_Set(param.source_path_ADTF, destination_path, Test_Case_Folder, param.logpath+"\\ATV_Python_Wraper.txt",param.UART_Log_Path,ignore = None)
```

4 Framework Installation on New PC

4.1 Python Installation:

- Please install Python 2.7 or 3.8 on the PC under the folder C:\Python27 or C:\Python38
- All required First time installation files are present under the GIT location, please install them
  - SFA_Test_Automation_PA(e.g:DASy)\\CT_Framework\First_Time_Installation_Tools
  - Make sure System Environment Variable is added to installed Python27/ Python38
- Install the below python modules after installing the Python
  - pip install shutil
  - pip install regex
  - pip install psutil
  - pip install requests
  - pip install elementtree
  - pip install http
  - pip install jsonlib
  - ...
- *Note:- If pip install is not working because of proxy settings, please install "cntlm-Bosch-1.0.2-setup.exe" present in the folder sys_vv_test\Continuous_Testing\Automation_Tools\First_Time_Installation_Tools , insert the Bosch domain USerID and password and then proceed with pip install*
- After Installing the Python you can use the Eclipse present under GIT " First_Time_Installation_Tools" as an IDE for Test Script Development

4.2 Vector Tool Installation:

- Make sure necessary Vector tools are installed in the Test_PC

4.3 GIT and Jenkins configuration:

- To set up GIT and Jenkins in a new Test_PC follow below:
  - GIT Configuration explained above in "**GIT Introduction and Usage**" section:
  - Jenkins Configuration explained above in **"Jenkins configuration and Purpose"** section

4.4 Creating Automation_Folder and Executing the Tests (One Time Preparation):

- **Step1: Make sure to adapt the path of the groovy scripts(*TestCaseCollector* and *TestCaseExecutor*)available in the GIT to the valid Drive or Path of the PC**
  - *TestCaseCollector*

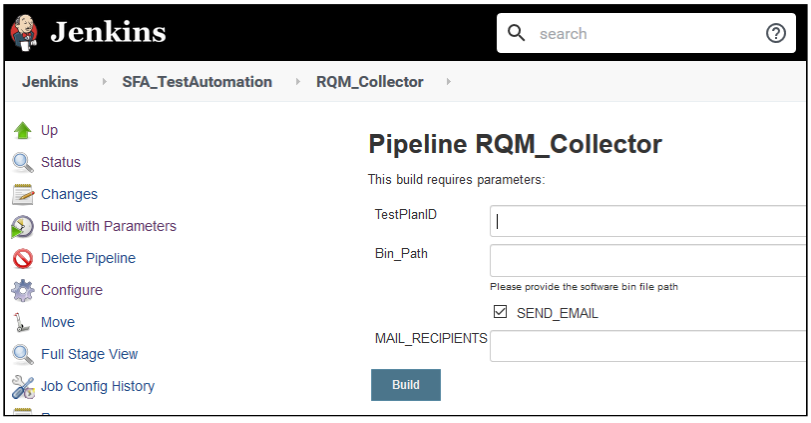
```
pipeline
{
    agent
    {
        node
        {
            label "TestCaseServer"
            customWorkspace 'D:\\Jenkins\\TestCaseServer'
        }
    }
```

- *TestCaseExecuter*

```
pipeline
{
    agent
    {
        node
        {
            label "TestAutomation"
            customWorkspace 'D:\\Jenkins\\TestAutomation'
        }
    }
```

- **Step2: Adapting the Parameter_List.py file**
  - Please adapt the paths in the "Parameter_List.py", present under the folder "Project_repo\CT_Framework\Test_Scheduler\Scripts" and make sure you push these changes in the GIT before starting the Automated Tests
  - Create a copy of "Parameter_List.py" and modify the paths in the copied file, so that framework file is backed up automatically

- **Step3: Start Automated Tests/ Execute the Jenkin Jobs :**
  - Go to Jenkins Page and open the job "**RQM_Collector**"
    - Navigate to "Build with Parameter" page and enter the below
      - **TestPlanID** --> *Test Plan ID, RQM repo name and Credential (e.g: 23898,"VID_ FCA _NRCS2 (qm)",gta2kor)* *Make sure User ID is the same as used on the user_data.txt present in GIT under the folder CT_Framework\RQM_TestCase_Collector)*
      - **Bin_Path** → *Software Binary Path Location* (\\abtvdfs2.de.bosch.com\ismdfs\ism\Archive\CC\Video\NRCS2_FCA\Jenkins_Artifacts\01_NRCS2_FCA_RELEASE\02_Customer_Releas\NRCS2_PP6_FCA_WL75MY21_CVPAM006.6\NRCS2_PP6_FCA_WL75MY21_CVPAM006.6.zip)



HURRAY → SFA-Automation Framework is now Installed and running successfully 🙂🙂