NGA Human Resources

ROBOT FRAMEWORK CODING STYLE GUIDE
Version 1.01

www.ngahr.com

Document Control

Document Information

	INFORMATION
Document Id	
Document Owner	Roberto Galman
Issue Date	13-Apr-2015
Last Saved Date	13-Apr-2015
File Name	Robot_Framework_Coding_StyleGuide.docx

Document History

VERSION	ISSUE DATE	CHANGES
1.00	13-Apr-2015	Initial Doc
1.01	28-Apr-2015	Added Pre-condition and Post-Condition (Section 4.3)

Document Approvals

ROLE	NAME	SIGNATURE	DATE
Project Sponsor			
Project Review Group			
Project Manager			
Quality Manager (if applicable)			
Procurement Manager (if applicable)			
Communications Manager (if applicable)			
Project Office Manager (if applicable)			

TABLE OF CONTENTS

1 INTRODUCTION

This document discusses a set of guidelines for writing **Robot Framework** (RF) Test Scripts including *Naming Conventions*, *Documentation*, *Test Suite/Case Structure*, *User Keywords*, and *Variables*. The purpose of this document is to create a readable, accurate, stable & maintainable RF Test Suite.

2 FOLDER STRUCTURE & FILE ORGANIZATION

Robot Framework Test Data is defined in tabular format using HTML, tab-separated values (TSV), plain text, or reStructuredText (reST) formats.

It is recommended to use **plain text format (.txt) with tabs** used as separator.

2.1 TEST AUTOMATION FRAMEWORK

Robot Framework allows you to create or extend the Test Library to add new functionality. This should be designed such that it can easily be integrated to a Test Suite by importing it. As Robot Framework is written in Python (.py), it is suggested that user library is implemented in Python as well.

It should support:

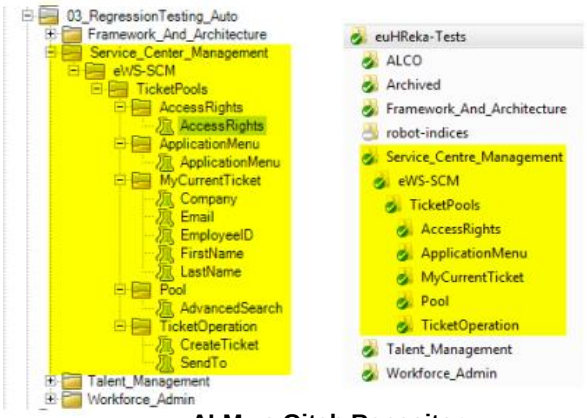
- *Command-line execution with parameterization*
- *Unit Tests available in the repository to be used for quick checks & testing updates*
- *Generates test execution logs with log-levels*

The recommended folder structure should be:

```
+ <PRODUCT>_automationfwk
- - + src
- - - - <Library>
- - - - - */*
- - - - unit tests
- - - sw_dependencies.txt
```

2.2 TEST SUITE

When creating the Test Suite Folder structure to be committed in GitLab, this should be consistent with the structure in ALM. This is for the purpose of traceability & mapping the test cases.



ALM vs Gitab Repository

The Test Suite Folder should follow the hierarchy: **Module > Submodule > Functionality>***. The naming convention should be consistent with ALM. It may follow a **CamelCase format** but for long names, it can also be separated by **underscores** to improve readability. The same convention can be implemented for file names.

```
<Module>_<Submodule>_<Function>_<Scenario/Action>_By_<Role>
```

e.g. FWK_ContextualAction_ContextualActionsDisplay_By_MGR

3 CODE LAYOUT

3.1 INDENTATION & MAX LINE LENGTH

Use **tabs** per indentation level. If there is more data than readily fits the line, you may use ***ellipsis (...)*** to continue the previous line. The **maximum line length is 120**. Limiting the required editor window width makes it possible to have several files open side-by-side, and works well when using code review tools that present the two versions in adjacent columns (PEP-8).

```
** Keywords **
indentation_style
    [Arguments] ${input1}    ${input2}    ${input3}    ${input4}    ${input5}
    ... ${input6}    ${input7}
    log ${input7}
```

3.2 SOURCE FILE ENCODING

Plain Text files are expected to use UTF-8 encoding.

3.3 STRING QUOTES

Always use double-quoted string format. However, when a string contains double quote characters, use a single-quote to avoid backslashes in the string. This improves readability.

```
${DQ_String}    "Double Quoted"
${SQ_DQ_String}    '"Double Quoted"'
${NOT_THIS}    "\"Double Quoted\""
```

3.4 WHITESPACE IN EXPRESSIONS & STATEMENTS

Avoid extra or missing whitespace in the following:

- Whitespace in Conditional expressions

```
    ${Customer} Set Variable if
    ...    '${Password}' == 'EQC-800'    AMO 3M FR
    ...    '${Password}'== 'EQ1-800'    EQY/EQ1
    ...    '${Password}'=='D04-100'    EWS SCM D04
    ...    '${Password}    '=='    EQF-800'    EWS SCM D04
```

- Assigning a keyword return value to a variable

```
** Test Cases **
Indentation
    ${a} =    indentation_style    1    2    3    4    5    6    ${DQ_String}
    ${a}=    indentation_style    1    2    3    4    5    6    ${DQ_String}
```

- Add single space between Test data Table and asterisks

```
*** Settings ***
*** Variables ***
*** Test Cases ***
*** Keywords ***
```

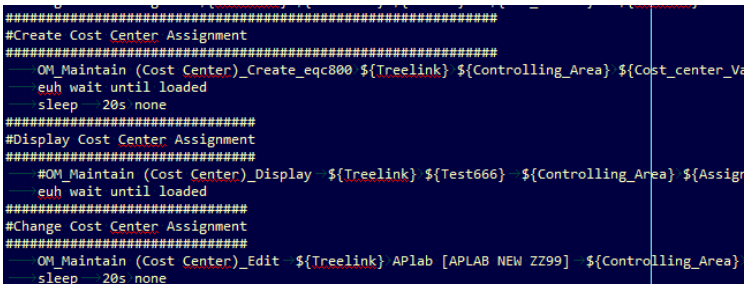
3.5 COMMENTS

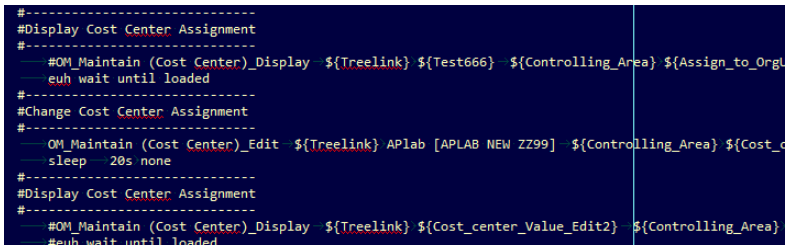
Block comments generally apply to some (or all) code that follows them, and are indented to the same level as that code. Each line of a block comment starts with a # and a single space (PEP-8).

```
** Variables **

${DQ_String}    "Double Quoted"
${SQ_DQ_String}    '"Double Quoted"'
${NOT_THIS}    "\"Double Quoted\""

** Test Cases **
Indentation
    # This is a block comment
    # This is applicable to the succeeding code with the same indent level
    ${a}=    indentation_style    1    2    3    4    5    6    ${DQ_String}
```

- Avoid adding chain of #s to indicate start & end of a block comment. Also, do not use # followed by a series of dash to start/end a block comment. This will make the code look cluttered.

Don’ts:





4 TEST SUITE STRUCTURE

As discussed in Section 2.2, Test Suite Folder & Filename should be consistent with ALM. There should be a 1:1 correspondence between the Test Instances in ALM & the Test Scripts in GitLab.

Hence, there should only be **one Test Case per Test Suite (File)**.

4.1 TEST DATA TABLE NAMES

The Test Data Table should follow the convention:

*** + single whitespace + <tablename> + single whitespace + ***

Where:

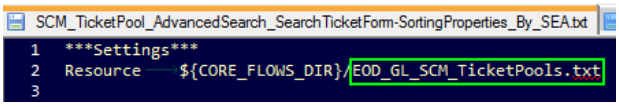
Table name = Settings, Variables, Test Cases, Keywords

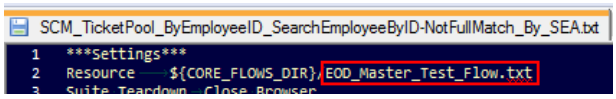
```
*** Settings ***
*** Variables ***
*** Test Cases ***
*** Keywords ***
```

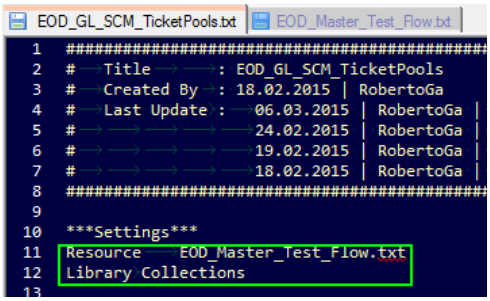
The test scripter should follow the **name, capitalization, spacing** of the test data tables.

4.2 SETTINGS TABLE

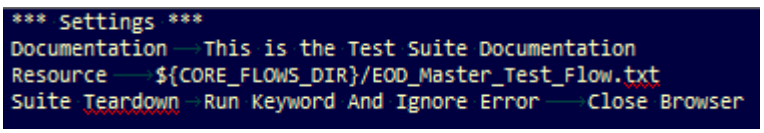
- You should add the Documentation at the Settings table. The description can be picked-up from the test description in ALM.
- Resource setting should only source the required Resource file. Other Test Libraries & Resource Files needed can be sourced in the required Resource file. The master Resource file should only include common/generic resource files used in the test suite.







- Add Suite Teardown > Close Browser to ensure that the current browser will be closed so as not to affect the succeeding tests e.g. Auto-logoff due to inactivity



4.3 PRE-CONDITION AND POST-CONDITION

- Add Preconditions in "Suite Setup" & Cleanup in "Suite Teardown". This will ensure that it will be triggered regardless if the test passed or failed.

- Use "Setup" as Setup keyword & "Cleanup" for Teardown. These keywords are defined in the resource file. Related Test cases using a common resource file should have same Setup & Teardown steps. In case of a different set of tests using the same flow but with different setup/teardown steps, you may use keyword - `Setup_<identifier>, Cleanup_<identifier>.`

```
*** Settings ***
Suite Setup    Setup
Suite Teardown    Cleanup
```

5 TEST CASE STRUCTURE

The TestCase name should be the same as the Test Suitename –i.e. Filename.

5.1 TAGS

Use **[Tags]** in your test case. The tags can be used to select tests for execution & can be set in Jenkins.

```
[Tags] automatedby:<name>
[Tags] status:<wip/obsolete/ci>
[Tags] priority:<high/med/low>
[Tags] testtype:sanity/regression/browser_compatibility
[Tags] system/client:<system/client>
[Tags] module:<module>
[Tags] submodule:<submodule>
[Tags] functionality:<functionality>
```

5.2 ABSTRACTION LEVEL

Use appropriate Abstraction Level. Note that, the Test Case may be reviewed by non-technical person not familiar with the Robot Framework. Ideally, the script implementation can easily be traced from the Detailed Scenario. This means higher abstraction is expected for the Test Case. The lower abstraction can be implemented in the Keywords or Test Library.

- Don't use too low abstraction level in your test case

```
*** Test Cases ***
Valid login
    Open browser    ${LOGIN URL}    ${BROWSER}
    Set Selenium speed    ${DELAY}
    Maximize browser window
    Title should be    Login Page
    Input text    username_field    demo
    Input text    password_field    mode
    Click button    login_button
    Title should be    Welcome Page
    Location should be    ${WELCOME URL}
```

- Don't rely on comments instead of abstraction

```
*** Test Cases ***
Valid login
    # Open browser to login page
    Open browser    ${LOGIN URL}    ${BROWSER}
    Set Selenium speed    ${DELAY}
    Maximize browser window
    Title should be    Login Page
    # Input user name
    Input text    username_field    demo
    # Input password
    Input text    password_field    mode
    # Submit credentials
    Click button    login_button
    # Welcome page should be open
    Title should be    Welcome Page
    Location should be    ${WELCOME URL}
```

Sample Implementation:
(To be discussed)

```
*** Test Cases ***
Valid login
    Open browser to login page
    Input user name    demo
    Input password    mode
    Submit credentials
    Welcome page should be open
```

5.3 TestCase Structure

- Generally has these phases:
  - Preconditions
  - Action
  - Verification
  - Cleanup
- Keyword should describe what a test does
  - Use clear keywords with the appropriate abstraction level
  - Should contain enough information to run manually
- No complex logic
  - No for loops or if/else constructs
  - Test Cases should not look like scripts
- Maximum of 15 steps, preferably less
- Be consistent in the Abstraction Level used. Keep in mind, Customer/Product owner may have to review your test implementation
- Do not test without checks – so basically just the flow was verified
- Avoid Tests with unrelated checks. This is redundant & eats up execution time.
- Avoid dependencies between tests. The test suite should be standalone. The preconditions & cleanup are implemented within the test script.
- Use teardowns for cleanup

6 USER KEYWORDS

6.1 GENERAL

- Name should be descriptive of the function & can easily be mapped in the Test Script Implementation plan
- All variables used in the keyword including the arguments, return & local variable in the implementation should be in **lower case**.
- It may contain programming logic (If/else, for loops)

6.2 COMMENTS

Add comments in your keyword to describe the action/flow if needed. Always keep in mind that others will be maintaining your code, so include in your comments your assumptions, default values, etc.

6.3 USING KEYWORDS

- When using keywords in your Test Case or Keywords table, Robot framework may allow you to use mixed cases – that is, “Close Browser” or ”close Browser” are accepted. **Use the case as it is written in the keyword definition**. If you are using eclipse, the intelliSense will do it for you.
- Use **[Arguments]** , **[Return]** (Follow the case)
- Add explicit waits & Capture page screenshot if needed in the keywords & not in the test cases to hide the technical implementation.

7 VARIABLES

Variables are used to encapsulate long and/or complicated values, and pass values from the command line or keywords.

7.1 NAMING

- Clear but not too long. Do not use generic variable name e.g. ${a}.
- Follow **CamelCase** Format. If name is long, you may use an underscore to improve readability. Do not use space as separator. e.g ${FirstName}
- Use case consistently:
    - Upper Case for Global Variable e.g. ${BROWSER}
    - CamelCase for Suite Variable (Declared in Variables table) e.g. ${JobType}
    - Lower Case for local variable e.g. return values of keywords assign to a variable. e.g. ${tmp}

7.2 ASSIGNING VARIABLE VALUE

- When declaring a Variable in the Variables Table of the Test Suite, omit the equal sign(=).

```
** Variables **
${FirstName}    John
${LastName}  Doe
```

- When assigning the return value of keyword to a variable, use the equal sign(=).

```
** Test Cases **
Using Variables Demo
    ${row}=  Get Row Number  ${LastName}
```

8 REFERENCES

#	REFERENCE	LOCATION
1	NGAHR_Test_Script_Implementation_Plan_Template.dotx	https://drive.google.com/drive/#folders/0B-6qCtvTgLEaQXFuMTloUUQ2T3M/0B-6qCtvTgLEaVnRMeDBPSUtzcjQ/0B-6qCtvTgLEafmRPbjJscjQ3NmlZY0YzNzRwQVNVV1NQZ01tcTJobWtqblFxVGZZZFVFMjA
2	Writing Stable & maintainable RF Test Scripts	https://drive.google.com/drive/#folders/0B-6qCtvTgLEaX1VLdWJnM1dCSFE/0B-6qCtvTgLEaM2tndnFPVGpWZ1U/0B-6qCtvTgLEaLXI1NzVpRl9xUmM
3	Automation PostReg Test Script Cleanup	https://docs.google.com/a/ngahr.com/presentation/d/1pjQrT43Q8DWY0nw12GRTkHHOsfout5Sxt7Bcv90FKlA/edit#slide=id.p25
4	Robot Framework – How to write Good TestCases	https://code.google.com/p/robotframework/wiki/HowToWriteGoodTestCases#Passing_and_returning_values
5	PEP 8 – Style Guide for Python	https://www.python.org/dev/peps/pep-0008/
6	writing_maintainable_automated_acceptance_tests.pdf	http://dhemery.com/pdf/writing_maintainable_automated_acceptance_tests.pdf
7	Robot Framework – Quick Start Guide	http://robotframework.googlecode.com/hg/doc/userguide/RobotFrameworkUserGuide.html?r=2.8.5
8	Robot Framework Do's & Don'ts	http://www.slideshare.net/pekkaklarck/robot-framework-dos-and-donts

9 APPENDIX

9.1 DEFINITIONS

#	NAME	DESCRIPTION

NGA Human Resources is a global leader in helping organizations transform their business-critical HR operations to deliver more effective and efficient people-critical services.

We help our clients become better employers through smarter, more streamlined business processes — to save money, manage employee life cycles, and support globally connected, agile organizations. This is how NGA makes HR work.

What sets us apart is The NGA Advantage. It's a combination of deep HR experience and insight, advanced technology platforms and applications and a global portfolio of flexible service delivery options.

www.ngahr.com

NGA Human Resources