Robot Framework Development Guide

Introduction

General Guidelines and Structure

Robot Framework itself provides a good set of basic guidelines for writing good tests. You can read it here. The rest of this generally guide generally assumes you are familiar with this, as well as writing Robot Framework and python.

ONAP basically follows the structure represented at this link and the image linked below

blocked URL

The robot framework layer is managed for you and you should install whatever the setup.sh file has as any other version could cause issues with compatibility.

ONAP Robot Project Structure

Folder Structure

```
`-- robot
    |-- assets
    |   |-- keys
    |   |-- templates
    |   |-- aai
    |   |-- vid
    |   `-- etc
    |-- library - any python libraries needed to run tests are installed here (you dont edit these directly!)
    |   `-- eteutils - this is the robot code we write in the python testing utils project aka robotframework-
onap. Put any python code libraries we write in there, but do not edit in here
    |-- resources
    |   |-- aai
    |   |-- vid
    |   `-- etc
    `-- testsuites - put any robot test suites we write in here
```

Assets

You should put anything you need as input like json files, cert keys, heat templates and templates. You should include subfolders for each component.

Two special folders are keys (put any private keys used in testing in here, you should NOT include keys used anywhere but testing, because they are here they are not secure.) and templates (put any json templates in here)

Resources

You should put any robot resource files aka reusable keywords we write in here, you should include subfolders for each component. These should be written in as reusable a manner as possible as they are meant to be keywords that provide a specific functionality, and if that functionality needs enhancement it should be added to the main keyword. Resources should be a useful library for all the test cases to call upon!

Tag Structure

Robot uses tags to separate out test cases to run. below are the tags we use

- health - use this for test cases that perform a health check of the environment
- core, small, medium - use these for test cases that require a certain level of ecomp
- dev-<helm-component> - use this tag to filter test cases that should be run only if this component exists
- api - this tag is for tests that use the msb api
- csit_<component>_<whatever> - this tag is used to define specific tests. this lets you run certain tests in the csit phase of your project.

Branching Structure

Repository Name: testsuite

See Configuring Gerrit

Contributing

What to Put Where

Component Level Interactions

Code was originally written all in robot. In order to increase re-usability the target is to write this level of items in python code. This code is located in https://gerrit.onap.org/r/gitweb?p=testsuite/python-testing-utils.git;a=tree;f=robotframework-onap;h=113e26102f4e9a3e729d46f0f1936d08de02045b;hb=refs/heads/master. However not all code has been ported there, but if it has you should be putting your changes there. If the component has not been ported, it will be in /robot/resources/<component>/whatever.robot or /robot/resources/<component>_<whatever>.robot and you can add/edit your items there. If you are making a new component, it should be created in the python library.

Testcase specific Keywords

Items that are not testcases and have some usability only within a single testsuite can we put in /robot/resources/

Testsuites and Testcases

All tests should be in /robot/testsuites. Since there are various reasons for splitting logically grouped items into separate files, please put them in a folder so that robot will show them together in the hierarchy. Certain tests like demos, orchestration, healthcheck and post init checks are already existing and should be put in their respective files

Writing Good Robot Tests

Read The Literature

You should follow the industry best practices for Robot Framework, a few good resources for that are

How To Write Good Test Cases

Testsuite Specific Items

- Avoid passing command line argument global variables to scripts
  - Use environment variables instead or get necessary data in the testsuite. Variables passed that way are specific to the test script and usually lead to logic being written in the script to get them that is better suited to being in the robot script. For example, rather then doing a curl to get a value and pass it to Robot, do the equivalent get call in the Robot Script.
- Avoid setting or reading global variables set by scripts.
  - Global state is very hard to set and also will most likely break when multiple tests are running in parallel.

- Don't place `*** Keywords ***` sections in testsuite files.
  - Files with `*** Test Cases ***` in them should not contain `*** Keywords ***` since it is then very hard to reuse those keywords. You never know who will want to use them, perhaps even outside the project. Any keywords can be simply moved to the `resources` directory.
- Do not use Evaluate
  - Check if there is relevant keyword provided in available libraries.
  - Check if somewhere else a keyword is provided that does what you need
  - If it really does not exist, you can write a python function based keyword in robotframework-onap that can be tested
- Do not use Sleep
  - Sleep guarantees that your test will be at-least that slow. If you are waiting for something consider using 'Wait Until Keyword Succeeds' or use a for loop and something like 'Run Keyword And Return If'.