Rossum Plugin User Manual

Table of Contents

This document exmplais Rossum folder structure and how to write and use Rossum TFW plugins.

1. Rossum folder structure

Rossum folder structure is designed such that user can update rossum core code without disturbing the customer specific code.

```
rossum_core  -------------> Top level folder (Git repo clone)
 ├──── core  ----------------> Rossum code (non customer code)
 │     ├──── plugins  ---------> Logging, reporting & other plugins
 │     ├──── rpc  -------------> Whatever can't be added as plugin for ex. `c` code.
 │     └──── testcase_module  -> Core TFW code like base class, plugin support etc..
 ├──── docs  ----------------> Rossum Documentation
 ├──── rossum_customer  -----> Customer specific code (Git repo clone)
 │     ├──── plugins  ---------> Custom reporting plugin, etc..
 │     ├──── rpc  -------------> Customer specific 'c' code.
 │     └──── testcase_module  -> Code to extend TFW Core code like base class, etc..
 │     └──── user_testcases  --> Customer test cases
 └──── user_testcases  ------> Rossum core sample test cases
 ├──── run-rossum.py  -------> Rossum TFW launcher
 ├──── set_vars.sh  ---------> Rossum Installer
 ├──── README.md  -----------> README file
```

2. Rossum plugins module

We use `pluggy` python module to add plugin support to Rossum, pluggy is a minimalist production ready plugin system which is used by pytest, tox and several other projects.

User can use `Rossum Plugin Template` to write Rossum plugin file and save it at rossum_core/core/plugins folder. Rossum will use all plugins in that folder automatically.

3. How to write Rossum plugin

Read comments in code to understand how plugin works.

```
# ##########################
# # Rossum Plugin Template
# ##########################

from plugin_module import hookimpl


class feature_plugin(object):
    '''
    Class name and class function names are all
    part of template and must not be changed.
    '''

    @hookimpl
    def service_start_hook(self, evars):
        '''
        User code in this function is executed in run_rossum.py
        before any testcases are even imported.

        Provides access to evars variable and expects user to return Bool
        '''
        # ## User code Start
        print("feature_cmd_prefix - Print from service_start_hook")
        return True
        # ## User code End

    @hookimpl
    def service_end_hook(self, evars, report):
        '''
        User code in this function is executed in run_rossum.py
        after all testcases execution ends.

        Provides access to evars & report variable and expects user to return Bool
        '''
        # ## User code Start
        print("feature_cmd_prefix - Print from service_end_hook")
        return True
        # ## User code End

    @hookimpl
    def argparse_hook(self, parser):
        '''
        User code in this function is executed in test_case.py baseclass.
        Plugin specific user arguments can be added here.

        Provides user parser object and expects user to return it after adding
arguments.
        '''
        # ## User code Start
        parser.add_argument(
            '--cmd-prefix', dest='cmd_prefix',
            choices=['echo', 'gdb', 'valgrind'],
            help='Cmd prefix for running DUT binary')
        # ## User code End

        return parser

    @hookimpl
    def pre_setup_hook(self, selfo):
        '''
        User code in this function is executed in each test case's pre_setup function

        Provides access to testcase object ie. selfo and expects user to return Bool
        '''
        # ## User code Start
        print("feature_cmd_prefix - Print from pre_setup hook")
        if selfo.evars.interact or selfo.evars.cmd_prefix:
            ret_val = selfo.debug_test_binary()
        else:
            ret_val = True

        return ret_val
        # ## User code End

    @hookimpl
    def post_teardown_hook(self, selfo):
        '''
        User code in this function is executed in each test case's port_teardown
function

        Provides access to testcase object ie. selfo and expects user to return Bool
        '''
        # ## User code Start
        print("feature_cmd_prefix - Print from post_teardown hook")
        return True
        # ## User code End
```