如何存取目標待測 module

井民全, Jing, mqjing@gmail.com

a.

Layout

```
root
├── src
│   └── mynamespace
│       └── mypkg
│           ├── __init__.py
│           └── PropertyWidget_App.py
```

Code

```
#載入 PropertyWinget_App
from namespace.mypkg.test_foo import PropertyWidget_App
```

b.

```
root
├── src
│   └── mynamespace
│       │
│       └── mypkg
│           ├── __init__.py
│           └── PropertyWidget_App.py
│
├── tests
│   ├── __init__.py
│   └── test_pkg1
│                ├── __init__.py
│                ├── test_PropertyWidget_App.py
│                └── utility.py
└── setup.py
```

有幾種啟動測試的方法

Command
./root/tests/test_pkg $ python3 -m **pytest** test_PropertyWidget_App.py
./root/tests/test_pkg $ **pytest** test_PropertyWidget_App.py
./root # pytest . # 執行所有的測試模組

Code: 範例操作不建議使用: 因為沒有考慮 QApplication 有只能一個 [ref]

```
# test_PropertyWidget_App.py
def test_case_1(qtbot):
        str_title = 'Testing: ' + inspect.currentframe().f_code.co_name
        print(str_title + ' testing... ')
        app = QApplication(sys.argv)
        window = get_window_target(qtbot, str_title)

        def on_timeout_action_1():
            window.widget_target.__line_edit__.clear()
            qtbot.keyClicks(window.widget_target.__line_edit__, '1') # test: 1

        def on_timeout_action_2():
            window.widget_target.__line_edit__.clear()
            qtbot.keyClicks(window.widget_target.__line_edit__, '2') # test: 2

        def on_timeout_close():
            qtbot.mouseClick(window.bt_close, Qt.LeftButton) # close

        QTimer.singleShot(1000, on_timeout_action_1)
        QTimer.singleShot(2000, on_timeout_action_2)
        QTimer.singleShot(3000, on_timeout_close)
        app.exec_()

def get_window_target(qtbot, str_title):
        window = test_PropertyWidget_App()
        window.__init_UI__()
        window.setWindowTitle(str_title)
        qtbot.add_widget(window)
        qtbot.waitUntil(lambda: window.isVisible())
        return window
```

c.

Code: 這個版本把 test_case 包在 TestWidget 類中, 在 __run__ method 設計了連續 time callback 呼叫產生 連續的 actions 並且考慮了 QApplication instance 只能有一個的狀況 [ref].

```
# test_PropertyWidget_App2.py
import sys
import pytestqt.qtbot as qtbot
from pytestqt.qt_compat import qt_api
```

```
from PyQt5 import QtTest
from PyQt5.QtCore import QTimer, Qt
import inspect

class TestWidget():
    app = None # ensure only one QApplication instance

    def __begin__(self, qtbot, str_title):
        self.index = 0
        if TestWidget.app is None:
            TestWidget.app = QApplication(sys.argv)
        window = self.get_window_target(qtbot, str_title)
        return window

    def __run__(self, lst_fun):
        self.index = 0
        # QTime chain invoking
        def on_timeout_fun():
            if self.index < len(lst_fun):
                lst_fun[self.index]()
                QTimer.singleShot(1000, on_timeout_fun)
            self.index = self.index + 1

        QTimer.singleShot(1000, on_timeout_fun)
        self.index = 0
        TestWidget.app.exec_()

    def get_window_target(qtbot, str_title):
            window = test_PropertyWidget_App()
            window.__init_UI__()
            window.setWindowTitle(str_title)
            qtbot.add_widget(window)
            qtbot.waitUntil(lambda: window.isVisible())
            return window


    def test_case_1(self, qtbot):
        str_title = 'Testing: ' + inspect.currentframe().f_code.co_name
        print(str_title + ' testing... ')
        self.window = self.__begin__(qtbot, str_title)

        def fn_action_1():
            self.window.widget_target.__line_edit__.clear()
            qtbot.keyClicks(self.window.widget_target.__line_edit__, '1') # test: 1

        def fn_action_2():
```

```
    self.window.widget_target.__line_edit__.clear()
    qtbot.keyClicks(self.window.widget_target.__line_edit__, '2') # test: 2

def on_timeout_close():
    qtbot.mouseClick(self.window.bt_close, Qt.LeftButton) # close

self.lst_fun = [fn_action_1, fn_action_2, on_timeout_close]
self.__run__(self.lst_fun)
```

d.