systest Documentation

Release 5.8.0

Erik Moqvist

May 27, 2020

Contents

CHAPTER 1

Installation

```
pip install systest
```

CHAPTER 2

Description

Execute a sequence of test cases in serial and/or parallel.

Test cases in a list are executed in serial and test cases in a tuple are executed in parallel, in separate Python threads.

This framework is created with integration/system test in mind. The framework is *not* intended as a replacement for `unittest`, but rather to be a complement to it.

Documentation: http://systest.readthedocs.org/en/latest

CHAPTER 3

Example usage

See the test suite: https://github.com/eerimoq/systest/blob/master/tests/test_systest.py

For example, the sequence below starts with test case `MyTestCase("1")`. When `MyTestCase("1")` has been executed, `MyTestCase("2")` and the list of `MyTestCase("3")` and `MyTestCase("4")` are executed in parallel. When both `MyTestCase("2")` and the list of `MyTestCase("3")` and `MyTestCase("4")` has been executed, `MyTestCase("5")` is executed. Then the sequence ends.

```python
import logging
import systest

LOGGER = logging.getLogger(__name__)

# Define a testcase.
class MyTestCase(systest.TestCase):
    """Test case description.

    """

    def __init__(self, name):
        super(MyTestCase, self).__init__()
        self.name = "my_testcase_" + name

    def run(self):
        LOGGER.info("Hello!")
        self.assert_equal(1, 1)
        self.assert_true(1 == 1)
        self.assert_in(1 in [1, 2])
        self.assert_none(None)

        with self.assert_raises(RuntimeError) as cm:
            raise RuntimeError('foo')

        self.assert_equal(str(cm.exception), 'foo')
```

(continues on next page)

(continued from previous page)

```
main("my_sequence",
     MyTestCase("1"),
     (
         MyTestCase("2"),
         [
             MyTestCase("3"),
             MyTestCase("4")
         ]
     ),
     MyTestCase("5"))
```

The output is:

```
Name: my_sequence
Date: 2016-02-02 18:42:40.446213
Node: erik-VirtualBox
User: erik

---------------------------------------------------------------

Name: my_testcase_1
Description:

    Test case description.

Hello!

my_testcase_1: PASSED in 0m 0s

---------------------------------------------------------------

Name: my_testcase_2
Description:

    Test case description.

Hello!

my_testcase_2: PASSED in 0m 0s

---------------------------------------------------------------

Name: my_testcase_3
Description:

    Test case description.

Hello!

my_testcase_3: PASSED in 0m 0s

---------------------------------------------------------------

Name: my_testcase_4
Description:

    Test case description.
```

(continues on next page)

(continued from previous page)

```

Hello!

my_testcase_4: PASSED in 0m 0s

---------------------------------------------------------------

Name: my_testcase_5
Description:

    Test case description.

Hello!

my_testcase_5: PASSED in 0m 0s

---------------------- Test summary begin ----------------------

[
    [
        my_testcase_1: PASSED,
        (
            my_testcase_2: PASSED,
            [
                my_testcase_3: PASSED,
                my_testcase_4: PASSED
            ]
        ),
        my_testcase_5: PASSED
    ]
]

Execution time: 0m 0s
Result: PASSED (passed: 5, failed: 0, skipped: 0, xpassed: 0, xfailed: 0)

----------------------- Test summary end -----------------------
```

CHAPTER 4

Classes

class systest.**TestCase**(*name=None*)

Base class of a test case executed by the sequencer.

setup()

Called before run().

teardown()

Called after run().

run()

The test case logic. This function is called be the sequencer to execute the test case.

A test case can either pass or fail. It fails if any exception is raised, otherwise it passes.

dry_run()

Called by the sequencer for a dry run execution. Should return the estimated execution time of the test case.

assert_equal(*first*, *second*)

Raise an exception if `first` and `second` are not equal.

assert_not_equal(*first*, *second*)

Raise an exception if `first` and `second` are equal.

assert_text_equal(*first*, *second*)

Raises an exception if `first` and `second` are not equal.

This is equivalent to `assert_equal` except it requires the arguments to be multi-line strings. The description of the failure is presented in the exception as a diff. This is an easier way to determine what has gone wrong in multi-line text.

assert_true(*condition*)

Raise an exception if given condition *condition* is false.

assert_false(*condition*)

Raise an exception if given condition *condition* is true.

assert_in (*member*, *container*)
Raise an exception if given member *member* is not found in given container *container*.

assert_not_in (*member*, *container*)
Raise an exception if given member *member* is found in given container *container*.

assert_is_none (*obj*)
Raise an exception if given object *obj* is not None.

assert_is_not_none (*obj*)
Raise an exception if given object *obj* is None.

assert_greater (*first*, *second*)
Raise an exception if `first` is not greater than `second`.

assert_greater_equal (*first*, *second*)
Raise an exception if `first` is not greater than or equal to `second`.

assert_less (*first*, *second*)
Raise an exception if `first` is not less than `second`.

assert_less_equal (*first*, *second*)
Raise an exception if `first` is not less than or equal to `second`.

assert_raises (*expected_type*, *expected_message=None*)
Raise an exception if no exception of given type(s) or subclass of given type(s) *expected_type* is raised.

assert_none (*obj*)
Raise an exception if given object *obj* is not None.

class systest.**Sequencer** (*name*, *testcase_pattern=None*, *dry_run=False*, *force_serial_execution=False*)

dot_digraph ()
Create a graphviz dot digraph of given test sequence.

The slowest execution path has bold edges.

Use the program `dot` to create an image from the output of this function.

```
dot -Tpng -Gdpi=200 -o mysequence.png mysequence.dot
```

report ()
Print a summary and create a dot graph image.

run (**tests*, ***kwargs*)
Run given testcase(s).

Test cases may be grouped into lists and tuples. All test cases in a list are executed in serial and all test cases in a tuple are executed in parallel. Lists and/or tuples may be used in multiple levels in the sequence.

For example, the sequence below starts with test case Test1. When Test1 has been executed, Test2 and the list of Test3 and Test4 are executed in parallel. When both Test2 and the list of Test3 and Test4 has been executed, Test5 is executed. Then the sequence ends.

```
[
    Test1(),
    (
        Test2(),
        [
            Test3(),
            Test4()
        ]
```

(continues on next page)

(continued from previous page)

```
    ),
    Test5()
]
```

summary()
Compile the test execution summary and return it as a string.

summary_count()
Compile the test execution summary and return it as a string.

summary_json()
Compile the test execution summary and return it as a JSON object.

summary_json_test(*test*)
Returns a test case summary ordered dictionary.

summary_test(*test*, *indent*)
Returns a test case summary line.

`systest.`**main**(*name*, **tests*, ***kwargs*)
Run given tests *tests* from the command line.

Use *console_log_level* to set the console log level.

Use *file_log_level* to set the file log level.

`systest.`**configure_logging**(*filename=None*, *console_log_level=None*, *file_log_level=None*)
Configure the logging module to write output to the console and a file. The file name is *filename-<date>.log* if *filename* is not None, otherwise the file name is `systest-<date>.log`.

Use *console_log_level* to set the console log level. It is `INFO` by default.

Use *file_log_level* to set the file log level. It is `DEBUG` by default.

Index