

Unittest for state_machine

June 16, 2020

Contents

1	Test Information	4
1.1	Test Candidate Information	4
1.2	Unittest Information	4
1.3	Test System Information	4
2	Statistic	4
2.1	Test-Statistic for testrun with python 2.7.17 (final)	4
2.2	Test-Statistic for testrun with python 3.6.9 (final)	5
2.3	Coverage Statistic	5
3	Tested Requirements	6
3.1	Module Initialisation	6
3.1.1	Default State	6
3.1.2	Default Last Transition Condtion	6
3.1.3	Default Previous State	7
3.1.4	Additional Keyword Arguments	8
3.2	Transition Changes	9
3.2.1	Transitiondefinition and -flow	9
3.2.2	Transitiontiming	11
3.2.3	Transitionpriorisation	12
3.3	Module Interface	13
3.3.1	This State	13
3.3.2	This State is	14
3.3.3	This State Duration	15
3.3.4	Last Transition Condition	15
3.3.5	Last Transition Condition was	16
3.3.6	Previous State	17
3.3.7	Previous State was	18
3.3.8	Previous State Duration	19
3.4	Transition Callbacks	20

3.4.1	State change callback for a defined transition and targetstate	20
3.4.2	State change callback for a defined transition	21
3.4.3	State change callback for a defined targetstate	22
3.4.4	State change callback for all kind of state changes	23
3.4.5	Execution order of Callbacks	24
A	Trace for testrun with python 2.7.17 (final)	26
A.1	Tests with status Info (20)	26
A.1.1	Default State	26
A.1.2	Default Last Transition Condtion	26
A.1.3	Default Previous State	27
A.1.4	Additional Keyword Arguments	27
A.1.5	Transitiondefinition and -flow	28
A.1.6	Transitiontiming	29
A.1.7	Transitionpriorisation	31
A.1.8	This State	32
A.1.9	This State is	32
A.1.10	This State Duration	33
A.1.11	Last Transition Condition	33
A.1.12	Last Transition Condition was	34
A.1.13	Previous State	35
A.1.14	Previous State was	35
A.1.15	Previous State Duration	36
A.1.16	State change callback for a defined transition and targetstate	36
A.1.17	State change callback for a defined transition	38
A.1.18	State change callback for a defined targetstate	39
A.1.19	State change callback for all kind of state changes	41
A.1.20	Execution order of Callbacks	42

B Trace for testrun with python 3.6.9 (final)	43
B.1 Tests with status Info (20)	43
B.1.1 Default State	43
B.1.2 Default Last Transition Condtion	44
B.1.3 Default Previous State	44
B.1.4 Additional Keyword Arguments	45
B.1.5 Transitiondefinition and -flow	46
B.1.6 Transitiontiming	47
B.1.7 Transitionpriorisation	48
B.1.8 This State	49
B.1.9 This State is	50
B.1.10 This State Duration	50
B.1.11 Last Transition Condition	51
B.1.12 Last Transition Condition was	51
B.1.13 Previous State	52
B.1.14 Previous State was	53
B.1.15 Previous State Duration	53
B.1.16 State change callback for a defined transition and targetstate	54
B.1.17 State change callback for a defined transition	55
B.1.18 State change callback for a defined targetstate	57
B.1.19 State change callback for all kind of state changes	59
B.1.20 Execution order of Callbacks	60
C Test-Coverage	61
C.1 state_machine	61
C.1.1 state_machine.__init__.py	61

1 Test Information

1.1 Test Candidate Information

This Module helps implementing state machines.

Library Information

Name	state_machine
State	Released
Supported Interpreters	python2, python3
Version	9884b22fc1e8af73e10e88e52951e585

Dependencies

1.2 Unittest Information

Unittest Information

Version	04693b5f87703f8bba98048b7730760f
Testruns with	python 2.7.17 (final), python 3.6.9 (final)

1.3 Test System Information

System Information

Architecture	64bit
Distribution	LinuxMint 19.3 tricia
Hostname	ahorn
Kernel	5.3.0-59-generic (#53 18.04.1-Ubuntu SMP Thu Jun 4 14:58:26 UTC 2020)
Machine	x86_64
Path	/user_data/data/dirk/prj/unittest/state_machine/unittest
System	Linux
Username	dirk

2 Statistic

2.1 Test-Statistic for testrun with python 2.7.17 (final)

Number of tests	20
Number of successfull tests	20
Number of possibly failed tests	0
Number of failed tests	0

Executionlevel	Full Test (all defined tests)
Time consumption	1.657s

2.2 Test-Statistic for testrun with python 3.6.9 (final)

Number of tests	20
Number of successfull tests	20
Number of possibly failed tests	0
Number of failed tests	0

Executionlevel	Full Test (all defined tests)
Time consumption	1.636s

2.3 Coverage Statistic

Module- or Filename	Line-Coverage	Branch-Coverage
state_machine	100.0%	100.0%
state_machine.__init__.py	100.0%	

3 Tested Requirements

3.1 Module Initialisation

3.1.1 Default State

Description

The state machine shall start in the state, given while module initialisation.

Reason for the implementation

Creation of a defined state after initialisation.

Fitcriterion

State machine is in the initial state after initialisation.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.1!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (22)
Start-Time:	2020-06-16 09:01:50,021
Finished-Time:	2020-06-16 09:01:50,022
Time-Consumption	0.000s

Testsummary:

Info	Initialising the state machine with state.c
Success	State after initialisation is correct (Content 'state.c' and Type is <type 'str'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.1!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (22)
Start-Time:	2020-06-16 09:01:52,065
Finished-Time:	2020-06-16 09:01:52,066
Time-Consumption	0.000s

Testsummary:

Info	Initialising the state machine with state.c
Success	State after initialisation is correct (Content 'state.c' and Type is <class 'str'>).

3.1.2 Default Last Transition Condition

Description

The state machine shall return the string `__init__` for last transition condition after initialisation.

Reason for the implementation

Creation of a defined state after initialisation.

Fitcriterion

The last transition condition is `__init__` after initialisation.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.2!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (23)
Start-Time:	2020-06-16 09:01:50,022
Finished-Time:	2020-06-16 09:01:50,022
Time-Consumption	0.000s

Testsummary:

Info	Initialising the state machine with state_c
Success	Last transition condition after initialisation is correct (Content ' <code>__init__</code> ' and Type is <code><type 'str'></code>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.2!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (23)
Start-Time:	2020-06-16 09:01:52,066
Finished-Time:	2020-06-16 09:01:52,066
Time-Consumption	0.000s

Testsummary:

Info	Initialising the state machine with state_c
Success	Last transition condition after initialisation is correct (Content ' <code>__init__</code> ' and Type is <code><class 'str'></code>).

3.1.3 Default Previous State

Description

The state machine shall return `None` for previous state after initialisation.

Reason for the implementation

Creation of a defined state after initialisation.

Fitcriterion

The previous state is `None` after initialisation.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.3!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (24)
Start-Time:	2020-06-16 09:01:50,022
Finished-Time:	2020-06-16 09:01:50,023
Time-Consumption	0.000s

Testsummary:

Info	Initialising the state machine with state.c
Success	Last state after initialisation is correct (Content None and Type is <type 'NoneType'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.3!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (24)
Start-Time:	2020-06-16 09:01:52,066
Finished-Time:	2020-06-16 09:01:52,066
Time-Consumption	0.000s

Testsummary:

Info	Initialising the state machine with state.c
Success	Last state after initialisation is correct (Content None and Type is <class 'NoneType'>).

3.1.4 Additional Keyword Arguments

Description

The state machine shall store all given keyword arguments as variables of the classes instance.

Reason for the implementation

Store further information (e.g. for calculation of the transition conditions).

Fitcriterion

At least two given keyword arguments with different types are available after initialisation.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.4!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (25)
Start-Time:	2020-06-16 09:01:50,023
Finished-Time:	2020-06-16 09:01:50,027
Time-Consumption	0.004s

Testsummary:

Info	Initialising the state machine with state.c
Success	Keyword argument kw_arg_no.4 stored in state_machine is correct (Content {'1': 1, '2': 'two'} and Type is <type 'dict'>).
Success	Keyword argument kw_arg_no.1 stored in state_machine is correct (Content 1 and Type is <type 'int'>).
Success	Keyword argument kw_arg_no.3 stored in state_machine is correct (Content True and Type is <type 'bool'>).
Success	Keyword argument kw_arg_no.2 stored in state_machine is correct (Content '2' and Type is <type 'str'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.4!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/_init_.py (25)
Start-Time:	2020-06-16 09:01:52,066
Finished-Time:	2020-06-16 09:01:52,067
Time-Consumption	0.001s

Testsummary:

Info	Initialising the state machine with state.c
Success	Keyword argument kw_arg_no.1 stored in state_machine is correct (Content 1 and Type is <class 'int'>).
Success	Keyword argument kw_arg_no.2 stored in state_machine is correct (Content '2' and Type is <class 'str'>).
Success	Keyword argument kw_arg_no.3 stored in state_machine is correct (Content True and Type is <class 'bool'>).
Success	Keyword argument kw_arg_no.4 stored in state_machine is correct (Content {'1': 1, '2': 'two'} and Type is <class 'dict'>).

3.2 Transition Changes

3.2.1 Transitiondefinition and -flow

Description

The user shall be able to define multiple states and transitions for the state machine. A transition shall have a start state, a target state and a transition condition. The transition condition shall be a method, where the user is able to calculate the condition on demand.

Reason for the implementation

Definition of the transitions for a state machine.

Fitcriterion

The order of at least three state changes is correct.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.5!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (28)
Start-Time:	2020-06-16 09:01:50,027
Finished-Time:	2020-06-16 09:01:50,031
Time-Consumption	0.004s

Testsummary:

Info	Initialising state machine with state_a
Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <type 'str'>).
Info	Work routine executed the 1st time to do the state change. Defined Transitions are: True→state_b (0.0s); False→state_c (0.0s)
Success	State after 1st execution of work method is correct (Content 'state_b' and Type is <type 'str'>).
Info	Work routine executed the 2nd time to do the state change. Defined Transitions are: False→state_a (0.0s); True→state_c (0.0s)
Success	State after 2nd execution of work method is correct (Content 'state_c' and Type is <type 'str'>).
Info	Work routine executed the 3rd time with no effect. No Transitions starting from state_c (dead end)
Success	State after 3rd execution of work method is correct (Content 'state_c' and Type is <type 'str'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.5!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (28)
Start-Time:	2020-06-16 09:01:52,067
Finished-Time:	2020-06-16 09:01:52,068
Time-Consumption	0.001s

Testsummary:

Info	Initialising state machine with state_a
Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <class 'str'>).
Info	Work routine executed the 1st time to do the state change. Defined Transitions are: True→state_b (0.0s); False→state_c (0.0s)
Success	State after 1st execution of work method is correct (Content 'state_b' and Type is <class 'str'>).
Info	Work routine executed the 2nd time to do the state change. Defined Transitions are: False→state_a (0.0s); True→state_c (0.0s)
Success	State after 2nd execution of work method is correct (Content 'state_c' and Type is <class 'str'>).
Info	Work routine executed the 3rd time with no effect. No Transitions starting from state_c (dead end)
Success	State after 3rd execution of work method is correct (Content 'state_c' and Type is <class 'str'>).

3.2.2 Transitiontiming

Description

The user shall be able to define for each transition a transition time. On change of the transition condition to True, the transition timer starts counting the time from 0.0s. After reaching the transition time, the transition gets active.

Reason for the implementation

Robustness of the state changes (e.g. Oscillating conditions shall be ignored).

Fitcriterion

The transition time and the restart of the transition timer by setting the transition condition to False and to True again results in the expected transition timing ($\pm 0.05s$).

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.6!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (29)
Start-Time:	2020-06-16 09:01:50,032
Finished-Time:	2020-06-16 09:01:50,413
Time-Consumption	0.381s

Testsummary:

Info	Initialising state machine with state_a
Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <type 'str'>).
Info	Waiting for 0.160s or state change
Success	State after 1st cycle is correct (Content 'state_b' and Type is <type 'str'>).
Success	Transition time after 1st cycle is correct (Content 0.15041112899780273 in [0.145 ... 0.155] and Type is <type 'float'>).
Info	Waiting for 0.235s or state change
Success	State after 2nd cycle is correct (Content 'state_c' and Type is <type 'str'>).
Success	Transition time after 2nd cycle is correct (Content 0.15091490745544434 in [0.145 ... 0.155] and Type is <type 'float'>).
Success	Previous state duration is correct (Content 0.22620105743408203 in [0.21999999999999997 ... 0.22999999999999998] and Type is <type 'float'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.6!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (29)
Start-Time:	2020-06-16 09:01:52,068
Finished-Time:	2020-06-16 09:01:52,447
Time-Consumption	0.379s

Testsummary:

Info	Initialising state machine with state_a
-------------	---

Success Initial state after Initialisation is correct (Content 'state_a' and Type is <class 'str'>).
Info Waiting for 0.160s or state change
Success State after 1st cycle is correct (Content 'state_b' and Type is <class 'str'>).
Success Transition time after 1st cycle is correct (Content 0.1506061553955078 in [0.145 ... 0.155] and Type is <class 'float'>).
Info Waiting for 0.235s or state change
Success State after 2nd cycle is correct (Content 'state_c' and Type is <class 'str'>).
Success Transition time after 2nd cycle is correct (Content 0.15029168128967285 in [0.145 ... 0.155] and Type is <class 'float'>).
Success Previous state duration is correct (Content 0.22554683685302734 in [0.21999999999999997 ... 0.22999999999999998] and Type is <class 'float'>).

3.2.3 Transitionpriorisation

Description

The state machine shall use the first active transition. If multiple transition are active, the transition with the highest overlap time will be used.

Reason for the implementation

Compensate the weakness of the execution quantisation.

Fitcriterion

At least one transition with at least two active conditions results in the expected state change.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.7!

Testrun: python 2.7.17 (final)
 Caller: /user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (30)
 Start-Time: 2020-06-16 09:01:50,413
 Finished-Time: 2020-06-16 09:01:50,659
 Time-Consumption 0.245s

Testsummary:

Info Initialising state machine with state_a, a transition to state_b after 0.151s and a transition to state_c after 0.150s
Success Initial state after Initialisation is correct (Content 'state_a' and Type is <type 'str'>).
Info Waiting for 0.300s or state change
Success State after 1st cycle is correct (Content 'state_c' and Type is <type 'str'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.7!

Testrun: python 3.6.9 (final)
 Caller: /user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (30)
 Start-Time: 2020-06-16 09:01:52,447

Finished-Time: 2020-06-16 09:01:52,691
 Time-Consumption 0.244s

Testsummary:

Info Initialising state machine with state_a, a transition to state_b after 0.151s and a transition to state_c after 0.150s
Success Initial state after Initialisation is correct (Content 'state_a' and Type is <class 'str'>).
Info Waiting for 0.300s or state change
Success State after 1st cycle is correct (Content 'state_c' and Type is <class 'str'>).

3.3 Module Interface

3.3.1 This State

Description

The Module shall have a method for getting the current state.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returned state fits to the expectation.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.8!

Testrun: python 2.7.17 (final)
 Caller: /user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (33)
 Start-Time: 2020-06-16 09:01:50,659
 Finished-Time: 2020-06-16 09:01:50,660
 Time-Consumption 0.001s

Testsummary:

Info Initialising the state machine with state_c
Success Returnvalue of this_state() is correct (Content 'state_c' and Type is <type 'str'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.8!

Testrun: python 3.6.9 (final)
 Caller: /user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (33)
 Start-Time: 2020-06-16 09:01:52,691
 Finished-Time: 2020-06-16 09:01:52,691
 Time-Consumption 0.000s

Testsummary:

Info Initialising the state machine with state_c
Success Returnvalue of this_state() is correct (Content 'state.c' and Type is <class 'str'>).

3.3.2 This State is

Description

The Module shall have a method for checking if the given state is currently active.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least two calls with different return values fit to the expectation.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.9!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (34)
Start-Time:	2020-06-16 09:01:50,660
Finished-Time:	2020-06-16 09:01:50,661
Time-Consumption	0.001s

Testsummary:

Info Initialising the state machine with state_c
Success Returnvalue of this_state_is(state_c) is correct (Content True and Type is <type 'bool'>).
Success Returnvalue of this_state_is(state_b) is correct (Content False and Type is <type 'bool'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.9!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (34)
Start-Time:	2020-06-16 09:01:52,691
Finished-Time:	2020-06-16 09:01:52,692
Time-Consumption	0.000s

Testsummary:

Info Initialising the state machine with state_c
Success Returnvalue of this_state_is(state_c) is correct (Content True and Type is <class 'bool'>).
Success Returnvalue of this_state_is(state_b) is correct (Content False and Type is <class 'bool'>).

3.3.3 This State Duration

Description

The Module shall have a method for getting the time since the last state change appears.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returned duration fits to the current state duration ($\pm 0.05s$).

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.10!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (35)
Start-Time:	2020-06-16 09:01:50,662
Finished-Time:	2020-06-16 09:01:50,913
Time-Consumption	0.252s

Testsummary:

Info	Running state machine test sequence.
Success	Return Value of this_state_duration() is correct (Content 0.2509438991546631 in [0.2 ... 0.3] and Type is <type 'float'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.10!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (35)
Start-Time:	2020-06-16 09:01:52,692
Finished-Time:	2020-06-16 09:01:52,942
Time-Consumption	0.251s

Testsummary:

Info	Running state machine test sequence.
Success	Return Value of this_state_duration() is correct (Content 0.25031614303588867 in [0.2 ... 0.3] and Type is <class 'float'>).

3.3.4 Last Transition Condition

Description

The Module shall have a method for getting the last transition condition.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returned transition condition fits to the expectation.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.11!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (36)
Start-Time:	2020-06-16 09:01:50,914
Finished-Time:	2020-06-16 09:01:50,914
Time-Consumption	0.001s

Testsummary:

Info	Running state machine test sequence.
Success	Returnvalue of last_transition_condition() is correct (Content 'condition_a' and Type is <type 'str'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.11!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (36)
Start-Time:	2020-06-16 09:01:52,943
Finished-Time:	2020-06-16 09:01:52,943
Time-Consumption	0.000s

Testsummary:

Info	Running state machine test sequence.
Success	Returnvalue of last_transition_condition() is correct (Content 'condition_a' and Type is <class 'str'>).

3.3.5 Last Transition Condition was

Description

The Module shall have a method for checking if the given condition was the last transition condition.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least two calls with different return values fit to the expectation.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.12!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (37)
Start-Time:	2020-06-16 09:01:50,914
Finished-Time:	2020-06-16 09:01:50,915
Time-Consumption	0.001s

Testsummary:

Info	Running state machine test sequence.
Success	Returnvalue of last_transition_condition(condition_a) is correct (Content True and Type is <type 'bool'>).
Success	Returnvalue of last_transition_condition(condition_c) is correct (Content False and Type is <type 'bool'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.12!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (37)
Start-Time:	2020-06-16 09:01:52,943
Finished-Time:	2020-06-16 09:01:52,944
Time-Consumption	0.000s

Testsummary:

Info	Running state machine test sequence.
Success	Returnvalue of last_transition_condition(condition_a) is correct (Content True and Type is <class 'bool'>).
Success	Returnvalue of last_transition_condition(condition_c) is correct (Content False and Type is <class 'bool'>).

3.3.6 Previous State**Description**

The Module shall have a method for getting the previous state.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returned state fits to the expectation.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.13!

Testrun: python 2.7.17 (final)
 Caller: /user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (38)
 Start-Time: 2020-06-16 09:01:50,915
 Finished-Time: 2020-06-16 09:01:50,916
 Time-Consumption 0.001s

Testsummary:

Info Running state machine test sequence.
Success Returnvalue of previous_state() is correct (Content 'state_a' and Type is <type 'str'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.13!

Testrun: python 3.6.9 (final)
 Caller: /user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (38)
 Start-Time: 2020-06-16 09:01:52,944
 Finished-Time: 2020-06-16 09:01:52,944
 Time-Consumption 0.000s

Testsummary:

Info Running state machine test sequence.
Success Returnvalue of previous_state() is correct (Content 'state_a' and Type is <class 'str'>).

3.3.7 Previous State was

Description

The Module shall have a method for checking if the given state was the previous state.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least two calls with different return values fit to the expectation.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.14!

Testrun: python 2.7.17 (final)
 Caller: /user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (39)
 Start-Time: 2020-06-16 09:01:50,916
 Finished-Time: 2020-06-16 09:01:50,917
 Time-Consumption 0.001s

Testsummary:

Info Running state machine test sequence.
Success Returnvalue of previous_state_was(state_a) is correct (Content True and Type is <type 'bool'>).

Success Returnvalue of previous_state_was(state_b) is correct (Content False and Type is <type 'bool'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.14!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/___init___py (39)
Start-Time:	2020-06-16 09:01:52,944
Finished-Time:	2020-06-16 09:01:52,944
Time-Consumption	0.000s

Testsummary:

Info	Running state machine test sequence.
Success	Returnvalue of previous_state_was(state_a) is correct (Content True and Type is <class 'bool'>).
Success	Returnvalue of previous_state_was(state_b) is correct (Content False and Type is <class 'bool'>).

3.3.8 Previous State Duration

Description

The Module shall have a method for getting active time for the previous state.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returned duration fits to the previous state duration ($\pm 0.05s$).

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.15!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/___init___py (40)
Start-Time:	2020-06-16 09:01:50,917
Finished-Time:	2020-06-16 09:01:51,669
Time-Consumption	0.752s

Testsummary:

Info	Running state machine test sequence.
Success	Return Value of previous_state_duration() is correct (Content 0.751147985458374 in [0.7 ... 0.8] and Type is <type 'float'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.15!

Testrun: python 3.6.9 (final)
 Caller: /user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init....py (40)
 Start-Time: 2020-06-16 09:01:52,945
 Finished-Time: 2020-06-16 09:01:53,696
 Time-Consumption 0.751s

Testsummary:

Info Running state machine test sequence.
Success Return Value of previous_state_duration() is correct (Content 0.7510056495666504 in [0.7 ... 0.8] and Type is <class 'float'>).

3.4 Transition Callbacks

3.4.1 State change callback for a defined transition and targetstate

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for a defined set of *transition_condition* and *target_state*.

Reason for the implementation

Triggering state change actions for a specific transition condition and targetstate.

Fitcriterion

Methods are called in the registration order after state change with all user given arguments for the defined transition condition and targetstate and at least for one other condition not.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.16!

Testrun: python 2.7.17 (final)
 Caller: /user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init....py (43)
 Start-Time: 2020-06-16 09:01:51,669
 Finished-Time: 2020-06-16 09:01:51,672
 Time-Consumption 0.003s

Testsummary:

Info Running state machine sequence and storing sequence number for each callback
Success Execution of state machine callback (1) (state_b, condition_a) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.
Success Execution of state machine callback (2) (state_b, condition_a) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.16!

Testrun: python 3.6.9 (final)
 Caller: /user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (43)
 Start-Time: 2020-06-16 09:01:53,696
 Finished-Time: 2020-06-16 09:01:53,697
 Time-Consumption 0.001s

Testsummary:

Info	Running state machine sequence and storing sequence number for each callback
Success	Execution of state machine callback (1) (state_b, condition_a) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.
Success	Execution of state machine callback (2) (state_b, condition_a) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

3.4.2 State change callback for a defined transition

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for a defined *transition_condition* and all *target_states*.

Reason for the implementation

Triggering state change actions for a specific transition condition.

Fitcriterion

Methods are called in the registration order after state change with all user given arguments for the defined transition condition and at least for one other transition condition not.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.17!

Testrun: python 2.7.17 (final)
 Caller: /user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (44)
 Start-Time: 2020-06-16 09:01:51,672
 Finished-Time: 2020-06-16 09:01:51,675
 Time-Consumption 0.003s

Testsummary:

Info	Running state machine sequence and storing sequence number for each callback
Success	Execution of state machine callback (1) (all_transitions, condition_b) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.
Success	Execution of state machine callback (2) (all_transitions, condition_b) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.17!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/___init___py (44)
Start-Time:	2020-06-16 09:01:53,697
Finished-Time:	2020-06-16 09:01:53,699
Time-Consumption	0.001s

Testsummary:

Info	Running state machine sequence and storing sequence number for each callback
Success	Execution of state machine callback (1) (all_transitions, condition_b) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.
Success	Execution of state machine callback (2) (all_transitions, condition_b) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

3.4.3 State change callback for a defined targetstate

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for all *transition_conditions* and a defined *target_state*.

Reason for the implementation

Triggering state change actions for a specific targetstate.

Fitcriterion

Methods are called in the registration order after state change with the defined targetstate and at least for one other targetstate not.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.18!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/___init___py (45)
Start-Time:	2020-06-16 09:01:51,675
Finished-Time:	2020-06-16 09:01:51,679
Time-Consumption	0.003s

Testsummary:

Info	Running state machine sequence and storing sequence number for each callback
Success	Execution of state machine callback (1) (state_b, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.
Success	Execution of state machine callback (2) (state_b, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.18!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (45)
Start-Time:	2020-06-16 09:01:53,699
Finished-Time:	2020-06-16 09:01:53,700
Time-Consumption	0.001s

Testsummary:

Info	Running state machine sequence and storing sequence number for each callback
Success	Execution of state machine callback (1) (state_b, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.
Success	Execution of state machine callback (2) (state_b, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

3.4.4 State change callback for all kind of state changes

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for all transitions.

Reason for the implementation

Triggering state change actions for all transition conditions and targetstates.

Fitcriterion

Methods are called in the registration order after state change.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.19!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (46)
Start-Time:	2020-06-16 09:01:51,679
Finished-Time:	2020-06-16 09:01:51,683
Time-Consumption	0.004s

Testsummary:

Info	Running state machine sequence and storing sequence number for each callback
Success	Execution of state machine callback (1) (all_transitions, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.
Success	Execution of state machine callback (2) (all_transitions, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.19!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (46)
Start-Time:	2020-06-16 09:01:53,701
Finished-Time:	2020-06-16 09:01:53,703
Time-Consumption	0.002s

Testsummary:

Info	Running state machine sequence and storing sequence number for each callback
Success	Execution of state machine callback (1) (all_transitions, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.
Success	Execution of state machine callback (2) (all_transitions, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

3.4.5 Execution order of Callbacks

Description

The callbacks shall be executed in the same order as they had been registered.

Reason for the implementation

User shall have the control about the execution order.

Fitcriterion

A callback with specific targetstate and condition will be executed before a non specific callback if the specific one had been regestered first.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.20!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (47)
Start-Time:	2020-06-16 09:01:51,683
Finished-Time:	2020-06-16 09:01:51,684
Time-Consumption	0.001s

Testsummary:

Success	Callback execution order: Values and number of submitted values is correct. See detailed log for more information.
----------------	--

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.20!

Unittest for state_machine

Testrun: python 3.6.9 (final)
Caller: /user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (47)
Start-Time: 2020-06-16 09:01:53,703
Finished-Time: 2020-06-16 09:01:53,703
Time-Consumption 0.001s

Testsummary:

Success Callback execution order: Values and number of submitted values is correct. See detailed log for more information.

A Trace for testrun with python 2.7.17 (final)

A.1 Tests with status Info (20)

A.1.1 Default State

Description

The state machine shall start in the state, given while module initialisation.

Reason for the implementation

Creation of a defined state after initialisation.

Fitcriterion

State machine is in the initial state after initialisation.

Testresult

This test was passed with the state: **Success**.

Info Initialising the state machine with state_c

```
StateMachine: State change ('__init__'): None -> 'state_c'
```

Success State after initialisation is correct (Content 'state_c' and Type is <type 'str'>).

```
Result (State after initialisation): 'state_c' (<type 'str'>)
```

```
Expectation (State after initialisation): result = 'state_c' (<type 'str'>)
```

A.1.2 Default Last Transition Condition

Description

The state machine shall return the string `__init__` for last transition condition after initialisation.

Reason for the implementation

Creation of a defined state after initialisation.

Fitcriterion

The last transition condition is `__init__` after initialisation.

Testresult

This test was passed with the state: **Success**.

```

Info   Initialising the state machine with state_c
    
```

```

StateMachine: State change ('__init__'): None -> 'state_c'
    
```

```

Success Last transition condition after initialisation is correct (Content '__init__' and Type is <type 'str'>).
    
```

```

Result (Last transition condition after initialisation): '__init__' (<type 'str'>)
Expectation (Last transition condition after initialisation): result = '__init__' (<type
↳ 'str'>)
    
```

A.1.3 Default Previous State

Description

The state machine shall return `None` for previous state after initalisation.

Reason for the implementation

Creation of a defined state after initialisation.

Fitcriterion

The previous state is `None` after initialisation.

Testresult

This test was passed with the state: **Success**.

```

Info   Initialising the state machine with state_c
    
```

```

StateMachine: State change ('__init__'): None -> 'state_c'
    
```

```

Success Last state after initialisation is correct (Content None and Type is <type 'NoneType'>).
    
```

```

Result (Last state after initialisation): None (<type 'NoneType'>)
Expectation (Last state after initialisation): result = None (<type 'NoneType'>)
    
```

A.1.4 Additional Keyword Arguments

Description

The state machine shall store all given keyword arguments as variables of the classes instance.

Reason for the implementation

Store further information (e.g. for calculation of the transition conditions).

Fitcriterion

At least two given keyword arguments with different types are available after initialisation.

Testresult

This test was passed with the state: **Success**.

Info	Initialising the state machine with state_c
<pre>StateMachine: State change ('__init__'): None -> 'state_c'</pre>	
Success	Keyword argument kw_arg_no.4 stored in state_machine is correct (Content {'1': 1, '2': 'two'} and Type is <type 'dict'>).
<pre>Result (Keyword argument kw_arg_no.4 stored in state_machine): { '1': 1, '2': 'two' } (<type 'dict'> ↪ 'dict'>)</pre>	
<pre>Expectation (Keyword argument kw_arg_no.4 stored in state_machine): result = { '1': 1, '2': 'two' } (<type 'dict'> ↪ 'two' } (<type 'dict'>)</pre>	
Success	Keyword argument kw_arg_no.1 stored in state_machine is correct (Content 1 and Type is <type 'int'>).
<pre>Result (Keyword argument kw_arg_no.1 stored in state_machine): 1 (<type 'int'>)</pre>	
<pre>Expectation (Keyword argument kw_arg_no.1 stored in state_machine): result = 1 (<type 'int'>)</pre>	
Success	Keyword argument kw_arg_no.3 stored in state_machine is correct (Content True and Type is <type 'bool'>).
<pre>Result (Keyword argument kw_arg_no.3 stored in state_machine): True (<type 'bool'>)</pre>	
<pre>Expectation (Keyword argument kw_arg_no.3 stored in state_machine): result = True (<type 'bool'> ↪ 'bool'>)</pre>	
Success	Keyword argument kw_arg_no.2 stored in state_machine is correct (Content '2' and Type is <type 'str'>).
<pre>Result (Keyword argument kw_arg_no.2 stored in state_machine): '2' (<type 'str'>)</pre>	
<pre>Expectation (Keyword argument kw_arg_no.2 stored in state_machine): result = '2' (<type 'str'> ↪ 'str'>)</pre>	

A.1.5 Transitiondefinition and -flow

Description

The user shall be able to define multiple states and transitions for the state machine. A transition shall have a start state, a target state and a transition condition. The transition condition shall be a method, where the user is able to calculate the condition on demand.

Reason for the implementation

Definition of the transitions for a state machine.

Fitcriterion

The order of at least three state changes is correct.

Testresult

This test was passed with the state: **Success**.

Info	Initialising state machine with state_a
-------------	---

<code>StateMachine: State change ('__init__'): None -> 'state_a'</code>	
--	--

Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <type 'str'>).
----------------	---

<code>Result (Initial state after Initialisation): 'state_a' (<type 'str'>)</code>	
<code>Expectation (Initial state after Initialisation): result = 'state_a' (<type 'str'>)</code>	

Info	Work routine executed the 1st time to do the state change. Defined Transitions are: True→state_b (0.0s); False→state_c (0.0s)
-------------	---

<code>StateMachine: State change ('condition_true'): 'state_a' -> 'state_b'</code>	
---	--

Success	State after 1st execution of work method is correct (Content 'state_b' and Type is <type 'str'>).
----------------	---

<code>Result (State after 1st execution of work method): 'state_b' (<type 'str'>)</code>	
<code>Expectation (State after 1st execution of work method): result = 'state_b' (<type 'str'>)</code>	

Info	Work routine executed the 2nd time to do the state change. Defined Transitions are: False→state_a (0.0s); True→state_c (0.0s)
-------------	---

<code>StateMachine: State change ('condition_true'): 'state_b' -> 'state_c'</code>	
---	--

Success	State after 2nd execution of work method is correct (Content 'state_c' and Type is <type 'str'>).
----------------	---

<code>Result (State after 2nd execution of work method): 'state_c' (<type 'str'>)</code>	
<code>Expectation (State after 2nd execution of work method): result = 'state_c' (<type 'str'>)</code>	

Info	Work routine executed the 3rd time with no effect. No Transitions starting from state_c (dead end)
-------------	--

Success	State after 3rd execution of work method is correct (Content 'state_c' and Type is <type 'str'>).
----------------	---

<code>Result (State after 3rd execution of work method): 'state_c' (<type 'str'>)</code>	
<code>Expectation (State after 3rd execution of work method): result = 'state_c' (<type 'str'>)</code>	

A.1.6 Transitiontiming

Description

The user shall be able to define for each transition a transition time. On change of the transition condition to True, the transition timer starts counting the time from 0.0s. After reaching the transition time, the transition gets active.

Reason for the implementation

Robustness of the state changes (e.g. Oscillating conditions shall be ignored).

Fitcriterion

The transition time and the restart of the transition timer by setting the transition condition to False and to True again results in the expected transition timing ($\pm 0.05s$).

Testresult

This test was passed with the state: **Success**.

Info Initialising state machine with state_a

StateMachine: State change ('__init__'): None -> 'state_a'

Success Initial state after Initialisation is correct (Content 'state_a' and Type is <type 'str'>).

Result (Initial state after Initialisation): 'state_a' (<type 'str'>)

Expectation (Initial state after Initialisation): result = 'state_a' (<type 'str'>)

Info Waiting for 0.160s or state change

StateMachine: State change ('condition_true'): 'state_a' -> 'state_b'

Success State after 1st cycle is correct (Content 'state_b' and Type is <type 'str'>).

Result (State after 1st cycle): 'state_b' (<type 'str'>)

Expectation (State after 1st cycle): result = 'state_b' (<type 'str'>)

Success Transition time after 1st cycle is correct (Content 0.15041112899780273 in [0.145 ... 0.155] and Type is <type 'float'>).

Result (Transition time after 1st cycle): 0.15041112899780273 (<type 'float'>)

Expectation (Transition time after 1st cycle): 0.145 <= result <= 0.155

Info Waiting for 0.235s or state change

StateMachine: State change ('condition_true'): 'state_b' -> 'state_c'

Success State after 2nd cycle is correct (Content 'state_c' and Type is <type 'str'>).

Result (State after 2nd cycle): 'state_c' (<type 'str'>)

Expectation (State after 2nd cycle): result = 'state_c' (<type 'str'>)

Success Transition time after 2nd cycle is correct (Content 0.15091490745544434 in [0.145 ... 0.155] and Type is <type 'float'>).

Result (Transition time after 2nd cycle): 0.15091490745544434 (<type 'float'>)

Expectation (Transition time after 2nd cycle): 0.145 <= result <= 0.155

Success Previous state duration is correct (Content 0.22620105743408203 in [0.2199999999999997 ... 0.2299999999999998] and Type is <type 'float'>).

Result (Previous state duration): 0.22620105743408203 (<type 'float'>)

Expectation (Previous state duration): 0.2199999999999997 <= result <= 0.2299999999999998

A.1.7 Transitionpriorisation

Description

The state machine shall use the first active transition. If multiple transition are active, the transition with the highest overlap time will be used.

Reason for the implementation

Compensate the weakness of the execution quantisation.

Fitcriterion

At least one transition with at least two active conditions results in the expected state change.

Testresult

This test was passed with the state: **Success**.

Info Initialising state machine with state_a, a transition to state_b after 0.151s and a transition to state_c after 0.150s

StateMachine: State change ('__init__'): None -> 'state_a'

Success Initial state after Initialisation is correct (Content 'state_a' and Type is <type 'str'>).

Result (Initial state after Initialisation): 'state_a' (<type 'str'>)

Expectation (Initial state after Initialisation): result = 'state_a' (<type 'str'>)

Info Waiting for 0.300s or state change

Executing method work after 0.000s

Executing method work after 0.061s

Executing method work after 0.122s

Executing method work after 0.182s

StateMachine: State change ('condition_true'): 'state_a' -> 'state_c'

Success State after 1st cycle is correct (Content 'state_c' and Type is <type 'str'>).

Result (State after 1st cycle): 'state_c' (<type 'str'>)

Expectation (State after 1st cycle): result = 'state_c' (<type 'str'>)

A.1.8 This State

Description

The Module shall have a method for getting the current state.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returned state fits to the expectation.

Testresult

This test was passed with the state: **Success**.

Info Initialising the state machine with state_c

StateMachine: State change ('__init__'): None -> 'state_c'

Success Returnvalue of this_state() is correct (Content 'state_c' and Type is <type 'str'>).

Result (Returnvalue of this_state()): 'state_c' (<type 'str'>)

Expectation (Returnvalue of this_state()): result = 'state_c' (<type 'str'>)

A.1.9 This State is

Description

The Module shall have a method for checking if the given state is currently active.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least two calls with different return values fit to the expectation.

Testresult

This test was passed with the state: **Success**.

Info Initialising the state machine with state_c

StateMachine: State change ('__init__'): None -> 'state_c'

Success Returnvalue of this_state_is(state_c) is correct (Content True and Type is <type 'bool'>).

```
Result (Returnvalue of this_state_is(state_c)): True (<type 'bool'>)
```

```
Expectation (Returnvalue of this_state_is(state_c)): result = True (<type 'bool'>)
```

Success Returnvalue of this_state_is(state_b) is correct (Content False and Type is <type 'bool'>).

```
Result (Returnvalue of this_state_is(state_b)): False (<type 'bool'>)
```

```
Expectation (Returnvalue of this_state_is(state_b)): result = False (<type 'bool'>)
```

A.1.10 This State Duration

Description

The Module shall have a method for getting the time since the last state change appears.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returned duration fits to the current state duration ($\pm 0.05s$).

Testresult

This test was passed with the state: **Success**.

Info Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
```

```
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
```

```
Waiting for 0.25s
```

Success Return Value of this_state_duration() is correct (Content 0.2509438991546631 in [0.2 ... 0.3] and Type is <type 'float'>).

```
Result (Return Value of this_state_duration()): 0.2509438991546631 (<type 'float'>)
```

```
Expectation (Return Value of this_state_duration()): 0.2 <= result <= 0.3
```

A.1.11 Last Transition Condition

Description

The Module shall have a method for getting the last transition condition.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returned transition condition fits to the expectation.

Testresult

This test was passed with the state: **Success**.

Info Running state machine test sequence.

StateMachine: State change ('__init__'): None -> 'state_a'

StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'

Success Returnvalue of last_transition_condition() is correct (Content 'condition_a' and Type is <type 'str'>).

Result (Returnvalue of last_transition_condition()): 'condition_a' (<type 'str'>)

Expectation (Returnvalue of last_transition_condition()): result = 'condition_a' (<type 'str'>)
 ↪ 'str'>)

A.1.12 Last Transition Condition was

Description

The Module shall have a method for checking if the given condition was the last transition condition.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least two calls with different return values fit to the expectation.

Testresult

This test was passed with the state: **Success**.

Info Running state machine test sequence.

StateMachine: State change ('__init__'): None -> 'state_a'

StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'

Success Returnvalue of last_transition_condition(condition_a) is correct (Content True and Type is <type 'bool'>).

Result (Returnvalue of last_transition_condition(condition_a)): True (<type 'bool'>)

Expectation (Returnvalue of last_transition_condition(condition_a)): result = True (<type 'bool'>)
 ↪ 'bool'>)

Success Returnvalue of last_transition_condition(condition_c) is correct (Content False and Type is <type 'bool'>).

Result (Returnvalue of last_transition_condition(condition_c)): False (<type 'bool'>)

Expectation (Returnvalue of last_transition_condition(condition_c)): result = False (<type 'bool'>)
 ↪ 'bool'>)

A.1.13 Previous State

Description

The Module shall have a method for getting the previous state.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returned state fits to the expectation.

Testresult

This test was passed with the state: **Success**.

Info Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
```

```
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
```

Success Returnvalue of previous_state() is correct (Content 'state_a' and Type is <type 'str'>).

```
Result (Returnvalue of previous_state()): 'state_a' (<type 'str'>)
```

```
Expectation (Returnvalue of previous_state()): result = 'state_a' (<type 'str'>)
```

A.1.14 Previous State was

Description

The Module shall have a method for checking if the given state was the previous state.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least two calls with different return values fit to the expectation.

Testresult

This test was passed with the state: **Success**.

Info Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
```

```
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
```

Success Returnvalue of previous_state_was(state_a) is correct (Content True and Type is <type 'bool'>).

```
Result (Returnvalue of previous_state_was(state_a)): True (<type 'bool'>)
```

```
Expectation (Returnvalue of previous_state_was(state_a)): result = True (<type 'bool'>)
```

Success Returnvalue of previous_state_was(state.b) is correct (Content False and Type is <type 'bool'>).

```
Result (Returnvalue of previous_state_was(state_b)): False (<type 'bool'>)
```

```
Expectation (Returnvalue of previous_state_was(state_b)): result = False (<type 'bool'>)
```

A.1.15 Previous State Duration

Description

The Module shall have a method for getting active time for the previous state.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returned duration fits to the previous state duration ($\pm 0.05s$).

Testresult

This test was passed with the state: **Success**.

Info Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
```

```
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
```

```
Waiting for 0.75s
```

```
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
```

Success Return Value of previous_state_duration() is correct (Content 0.751147985458374 in [0.7 ... 0.8] and Type is <type 'float'>).

```
Result (Return Value of previous_state_duration()): 0.751147985458374 (<type 'float'>)
```

```
Expectation (Return Value of previous_state_duration()): 0.7 <= result <= 0.8
```

A.1.16 State change callback for a defined transition and targetstate

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for a defined set of *transition_condition* and *target_state*.

Reason for the implementation

Triggering state change actions for a specific transition condition and targetstate.

Fitcriterion

Methods are called in the registration order after state change with all user given arguments for the defined transition condition and targetstate and at least for one other condition not.

Testresult

This test was passed with the state: **Success**.

Info Running state machine sequence and storing sequence number for each callback

```

StateMachine: State change ('__init__'): None -> 'state_a'
Increasing sequence number to 1 caused by sequence progress
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Executing callback 0 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 2 caused by callback_execution
Executing callback 1 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 3 caused by callback_execution
Increasing sequence number to 4 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
Increasing sequence number to 5 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_a' -> 'state_b'
Increasing sequence number to 6 caused by sequence progress
StateMachine: State change ('condition_c'): 'state_b' -> 'state_c'
    
```

Success Execution of state machine callback (1) (state_b, condition_a) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

```

Result (Execution of state machine callback (1) (state_b, condition_a) identified by a
↳ sequence number): [ 1 ] (<type 'list'>)
Expectation (Execution of state machine callback (1) (state_b, condition_a) identified by a
↳ sequence number): result = [ 1 ] (<type 'list'>)
Result (Submitted value number 1): 1 (<type 'int'>)
Expectation (Submitted value number 1): result = 1 (<type 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <type 'int'>).
    
```

Success Execution of state machine callback (2) (state_b, condition_a) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

```

Result (Execution of state machine callback (2) (state_b, condition_a) identified by a
↳ sequence number): [ 2 ] (<type 'list'>)
Expectation (Execution of state machine callback (2) (state_b, condition_a) identified by a
↳ sequence number): result = [ 2 ] (<type 'list'>)
Result (Submitted value number 1): 2 (<type 'int'>)
Expectation (Submitted value number 1): result = 2 (<type 'int'>)
Submitted value number 1 is correct (Content 2 and Type is <type 'int'>).
    
```

A.1.17 State change callback for a defined transition

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for a defined *transition_condition* and all *target_states*.

Reason for the implementation

Triggering state change actions for a specific transition condition.

Fitcriterion

Methods are called in the registration order after state change with all user given arguments for the defined transition condition and at least for one other transition condition not.

Testresult

This test was passed with the state: **Success**.

Info	Running state machine sequence and storing sequence number for each callback
	StateMachine: State change ('__init__'): None -> 'state_a'
	Increasing sequence number to 1 caused by sequence progress
	StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
	Increasing sequence number to 2 caused by sequence progress
	StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
	Executing callback 0 - tests.test_callbacks.exec_with_counter
	Increasing sequence number to 3 caused by callback_execution
	Executing callback 1 - tests.test_callbacks.exec_with_counter
	Increasing sequence number to 4 caused by callback_execution
	Increasing sequence number to 5 caused by sequence progress
	StateMachine: State change ('condition_b'): 'state_a' -> 'state_b'
	Executing callback 0 - tests.test_callbacks.exec_with_counter
	Increasing sequence number to 6 caused by callback_execution
	Executing callback 1 - tests.test_callbacks.exec_with_counter
	Increasing sequence number to 7 caused by callback_execution
	Increasing sequence number to 8 caused by sequence progress
	StateMachine: State change ('condition_c'): 'state_b' -> 'state_c'
Success	Execution of state machine callback (1) (all_transitions, condition_b) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

```
Result (Execution of state machine callback (1) (all_transitions, condition_b) identified by
↳ a sequence number): [ 2, 5 ] (<type 'list'>)
```

```
Expectation (Execution of state machine callback (1) (all_transitions, condition_b)
↳ identified by a sequence number): result = [ 2, 5 ] (<type 'list'>)
```

```
Result (Submitted value number 1): 2 (<type 'int'>)
```

```
Expectation (Submitted value number 1): result = 2 (<type 'int'>)
```

```
Submitted value number 1 is correct (Content 2 and Type is <type 'int'>).
```

```
Result (Submitted value number 2): 5 (<type 'int'>)
```

```
Expectation (Submitted value number 2): result = 5 (<type 'int'>)
```

```
Submitted value number 2 is correct (Content 5 and Type is <type 'int'>).
```

Success Execution of state machine callback (2) (all_transitions, condition_b) identified by a sequence number:
Values and number of submitted values is correct. See detailed log for more information.

```
Result (Execution of state machine callback (2) (all_transitions, condition_b) identified by
↳ a sequence number): [ 3, 6 ] (<type 'list'>)
```

```
Expectation (Execution of state machine callback (2) (all_transitions, condition_b)
↳ identified by a sequence number): result = [ 3, 6 ] (<type 'list'>)
```

```
Result (Submitted value number 1): 3 (<type 'int'>)
```

```
Expectation (Submitted value number 1): result = 3 (<type 'int'>)
```

```
Submitted value number 1 is correct (Content 3 and Type is <type 'int'>).
```

```
Result (Submitted value number 2): 6 (<type 'int'>)
```

```
Expectation (Submitted value number 2): result = 6 (<type 'int'>)
```

```
Submitted value number 2 is correct (Content 6 and Type is <type 'int'>).
```

A.1.18 State change callback for a defined targetstate

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for all *transition_conditions* and a defined *target_state*.

Reason for the implementation

Triggering state change actions for a specific targetstate.

Fitcriterion

Methods are called in the registration order after state change with the defined targetstate and at least for one other targetstate not.

Testresult

This test was passed with the state: **Success**.

Info Running state machine sequence and storing sequence number for each callback

```

StateMachine: State change ('__init__'): None -> 'state_a'
Increasing sequence number to 1 caused by sequence progress
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Executing callback 0 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 2 caused by callback_execution
Executing callback 1 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 3 caused by callback_execution
Increasing sequence number to 4 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
Increasing sequence number to 5 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_a' -> 'state_b'
Executing callback 0 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 6 caused by callback_execution
Executing callback 1 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 7 caused by callback_execution
Increasing sequence number to 8 caused by sequence progress
StateMachine: State change ('condition_c'): 'state_b' -> 'state_c'
    
```

Success Execution of state machine callback (1) (state_b, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

```

Result (Execution of state machine callback (1) (state_b, all_conditions) identified by a
↪ sequence number): [ 1, 5 ] (<type 'list'>)
Expectation (Execution of state machine callback (1) (state_b, all_conditions) identified by
↪ a sequence number): result = [ 1, 5 ] (<type 'list'>)
Result (Submitted value number 1): 1 (<type 'int'>)
Expectation (Submitted value number 1): result = 1 (<type 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <type 'int'>).
Result (Submitted value number 2): 5 (<type 'int'>)
Expectation (Submitted value number 2): result = 5 (<type 'int'>)
Submitted value number 2 is correct (Content 5 and Type is <type 'int'>).
    
```

Success Execution of state machine callback (2) (state_b, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

```

Result (Execution of state machine callback (2) (state_b, all_conditions) identified by a
↪ sequence number): [ 2, 6 ] (<type 'list'>)
Expectation (Execution of state machine callback (2) (state_b, all_conditions) identified by
↪ a sequence number): result = [ 2, 6 ] (<type 'list'>)
Result (Submitted value number 1): 2 (<type 'int'>)
Expectation (Submitted value number 1): result = 2 (<type 'int'>)
Submitted value number 1 is correct (Content 2 and Type is <type 'int'>).
Result (Submitted value number 2): 6 (<type 'int'>)
Expectation (Submitted value number 2): result = 6 (<type 'int'>)
Submitted value number 2 is correct (Content 6 and Type is <type 'int'>).
    
```

A.1.19 State change callback for all kind of state changes

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for all transitions.

Reason for the implementation

Triggering state change actions for all transition conditions and targetstates.

Fitcriterion

Methods are called in the registration order after state change.

Testresult

This test was passed with the state: **Success**.

Info Running state machine sequence and storing sequence number for each callback

```

StateMachine: State change ('__init__'): None -> 'state_a'
Increasing sequence number to 1 caused by sequence progress
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Executing callback 0 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 2 caused by callback_execution
Executing callback 1 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 3 caused by callback_execution
Increasing sequence number to 4 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
Executing callback 0 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 5 caused by callback_execution
Executing callback 1 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 6 caused by callback_execution
Increasing sequence number to 7 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_a' -> 'state_b'
Executing callback 0 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 8 caused by callback_execution
Executing callback 1 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 9 caused by callback_execution
Increasing sequence number to 10 caused by sequence progress
StateMachine: State change ('condition_c'): 'state_b' -> 'state_c'
Executing callback 0 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 11 caused by callback_execution
Executing callback 1 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 12 caused by callback_execution
    
```

Success Execution of state machine callback (1) (all_transitions, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

```

Result (Execution of state machine callback (1) (all_transitions, all_conditions) identified
↳ by a sequence number): [ 1, 4, 7, 10 ] (<type 'list'>)
Expectation (Execution of state machine callback (1) (all_transitions, all_conditions)
↳ identified by a sequence number): result = [ 1, 4, 7, 10 ] (<type 'list'>)
Result (Submitted value number 1): 1 (<type 'int'>)
Expectation (Submitted value number 1): result = 1 (<type 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <type 'int'>).
Result (Submitted value number 2): 4 (<type 'int'>)
Expectation (Submitted value number 2): result = 4 (<type 'int'>)
Submitted value number 2 is correct (Content 4 and Type is <type 'int'>).
Result (Submitted value number 3): 7 (<type 'int'>)
Expectation (Submitted value number 3): result = 7 (<type 'int'>)
Submitted value number 3 is correct (Content 7 and Type is <type 'int'>).
Result (Submitted value number 4): 10 (<type 'int'>)
Expectation (Submitted value number 4): result = 10 (<type 'int'>)
Submitted value number 4 is correct (Content 10 and Type is <type 'int'>).

```

Success Execution of state machine callback (2) (all_transitions, all_conditions) identified by a sequence number:
 Values and number of submitted values is correct. See detailed log for more information.

```

Result (Execution of state machine callback (2) (all_transitions, all_conditions) identified
↳ by a sequence number): [ 2, 5, 8, 11 ] (<type 'list'>)
Expectation (Execution of state machine callback (2) (all_transitions, all_conditions)
↳ identified by a sequence number): result = [ 2, 5, 8, 11 ] (<type 'list'>)
Result (Submitted value number 1): 2 (<type 'int'>)
Expectation (Submitted value number 1): result = 2 (<type 'int'>)
Submitted value number 1 is correct (Content 2 and Type is <type 'int'>).
Result (Submitted value number 2): 5 (<type 'int'>)
Expectation (Submitted value number 2): result = 5 (<type 'int'>)
Submitted value number 2 is correct (Content 5 and Type is <type 'int'>).
Result (Submitted value number 3): 8 (<type 'int'>)
Expectation (Submitted value number 3): result = 8 (<type 'int'>)
Submitted value number 3 is correct (Content 8 and Type is <type 'int'>).
Result (Submitted value number 4): 11 (<type 'int'>)
Expectation (Submitted value number 4): result = 11 (<type 'int'>)
Submitted value number 4 is correct (Content 11 and Type is <type 'int'>).

```

A.1.20 Execution order of Callbacks

Description

The callbacks shall be executed in the same order as they had been registered.

Reason for the implementation

User shall have the control about the execution order.

Fitcriterion

A callback with specific targetstate and condition will be executed before a non specific callback if the specific one had been registered first.

Testresult

This test was passed with the state: **Success**.

Success Callback execution order: Values and number of submitted values is correct. See detailed log for more information.

```

StateMachine: State change ('__init__'): None -> 'state_a'
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Executing callback 0 - unittest.test.report_value
Executing callback 1 - unittest.test.report_value
Result (Callback execution order): [ 'specific callback', 'nonspecific callback' ] (<type
↳ 'list'>)
Expectation (Callback execution order): result = [ 'specific callback', 'nonspecific
↳ callback' ] (<type 'list'>)
Result (Submitted value number 1): 'specific callback' (<type 'str'>)
Expectation (Submitted value number 1): result = 'specific callback' (<type 'str'>)
Submitted value number 1 is correct (Content 'specific callback' and Type is <type 'str'>).
Result (Submitted value number 2): 'nonspecific callback' (<type 'str'>)
Expectation (Submitted value number 2): result = 'nonspecific callback' (<type 'str'>)
Submitted value number 2 is correct (Content 'nonspecific callback' and Type is <type 'str'>).
    
```

B Trace for testrun with python 3.6.9 (final)

B.1 Tests with status Info (20)

B.1.1 Default State

Description

The state machine shall start in the state, given while module initialisation.

Reason for the implementation

Creation of a defined state after initialisation.

Fitcriterion

State machine is in the initial state after initialisation.

Testresult

This test was passed with the state: **Success**.

```

Info    Initialising the state machine with state_c
    
```

```

StateMachine: State change ('__init__'): None -> 'state_c'
    
```

```

Success State after initialisation is correct (Content 'state_c' and Type is <class 'str'>).
    
```

```

Result (State after initialisation): 'state_c' (<class 'str'>)
Expectation (State after initialisation): result = 'state_c' (<class 'str'>)
    
```

B.1.2 Default Last Transition Condition

Description

The state machine shall return the string `__init__` for last transition condition after initalisation.

Reason for the implementation

Creation of a defined state after initialisation.

Fitcriterion

The last transition condition is `__init__` after initialisation.

Testresult

This test was passed with the state: **Success**.

```

Info    Initialising the state machine with state_c
    
```

```

StateMachine: State change ('__init__'): None -> 'state_c'
    
```

```

Success Last transition condition after initialisation is correct (Content '__init__' and Type is <class 'str'>).
    
```

```

Result (Last transition condition after initialisation): '__init__' (<class 'str'>)
Expectation (Last transition condition after initialisation): result = '__init__' (<class
↳ 'str'>)
    
```

B.1.3 Default Previous State

Description

The state machine shall return `None` for previous state after initalisation.

Reason for the implementation

Creation of a defined state after initialisation.

Fitcriterion

The previous state is None after initialisation.

Testresult

This test was passed with the state: **Success**.

Info	Initialising the state machine with state_c
StateMachine: State change ('__init__'): None -> 'state_c'	
Success	Last state after initialisation is correct (Content None and Type is <class 'NoneType'>).
Result (Last state after initialisation): None (<class 'NoneType'>)	
Expectation (Last state after initialisation): result = None (<class 'NoneType'>)	

B.1.4 Additional Keyword Arguments

Description

The state machine shall store all given keyword arguments as variables of the classes instance.

Reason for the implementation

Store further information (e.g. for calculation of the transition conditions).

Fitcriterion

At least two given keyword arguments with different types are available after initialisation.

Testresult

This test was passed with the state: **Success**.

Info	Initialising the state machine with state_c
StateMachine: State change ('__init__'): None -> 'state_c'	
Success	Keyword argument kw_arg_no.1 stored in state_machine is correct (Content 1 and Type is <class 'int'>).
Result (Keyword argument kw_arg_no_1 stored in state_machine): 1 (<class 'int'>)	
Expectation (Keyword argument kw_arg_no_1 stored in state_machine): result = 1 (<class 'int'>)	
Success	Keyword argument kw_arg_no.2 stored in state_machine is correct (Content '2' and Type is <class 'str'>).
Result (Keyword argument kw_arg_no_2 stored in state_machine): '2' (<class 'str'>)	
Expectation (Keyword argument kw_arg_no_2 stored in state_machine): result = '2' (<class 'str'>)	
Success	Keyword argument kw_arg_no.3 stored in state_machine is correct (Content True and Type is <class 'bool'>).

```
Result (Keyword argument kw_arg_no_3 stored in state_machine): True (<class 'bool'>)
```

```
Expectation (Keyword argument kw_arg_no_3 stored in state_machine): result = True (<class 'bool'>)
```

Success Keyword argument kw_arg_no_4 stored in state_machine is correct (Content {'1': 1, '2': 'two'} and Type is <class 'dict'>).

```
Result (Keyword argument kw_arg_no_4 stored in state_machine): { '1': 1, '2': 'two' } (<class 'dict'>)
```

```
Expectation (Keyword argument kw_arg_no_4 stored in state_machine): result = { '1': 1, '2': 'two' } (<class 'dict'>)
```

B.1.5 Transitiondefinition and -flow

Description

The user shall be able to define multiple states and transitions for the state machine. A transition shall have a start state, a target state and a transition condition. The transition condition shall be a method, where the user is able to calculate the condition on demand.

Reason for the implementation

Definition of the transitions for a state machine.

Fitcriterion

The order of at least three state changes is correct.

Testresult

This test was passed with the state: **Success**.

Info Initialising state machine with state_a

```
StateMachine: State change ('__init__'): None -> 'state_a'
```

Success Initial state after Initialisation is correct (Content 'state_a' and Type is <class 'str'>).

```
Result (Initial state after Initialisation): 'state_a' (<class 'str'>)
```

```
Expectation (Initial state after Initialisation): result = 'state_a' (<class 'str'>)
```

Info Work routine executed the 1st time to do the state change. Defined Transitions are: True→state_b (0.0s); False→state_c (0.0s)

```
StateMachine: State change ('condition_true'): 'state_a' -> 'state_b'
```

Success State after 1st execution of work method is correct (Content 'state_b' and Type is <class 'str'>).

Result (State after 1st execution of work method): 'state_b' (<class 'str'>)

Expectation (State after 1st execution of work method): result = 'state_b' (<class 'str'>)

Info Work routine executed the 2nd time to do the state change. Defined Transitions are: False→state_a (0.0s); True→state_c (0.0s)

StateMachine: State change ('condition_true'): 'state_b' -> 'state_c'

Success State after 2nd execution of work method is correct (Content 'state_c' and Type is <class 'str'>).

Result (State after 2nd execution of work method): 'state_c' (<class 'str'>)

Expectation (State after 2nd execution of work method): result = 'state_c' (<class 'str'>)

Info Work routine executed the 3rd time with no effect. No Transitions starting from state_c (dead end)

Success State after 3rd execution of work method is correct (Content 'state_c' and Type is <class 'str'>).

Result (State after 3rd execution of work method): 'state_c' (<class 'str'>)

Expectation (State after 3rd execution of work method): result = 'state_c' (<class 'str'>)

B.1.6 Transitiontiming

Description

The user shall be able to define for each transition a transition time. On change of the transition condition to True, the transition timer starts counting the time from 0.0s. After reaching the transition time, the transition gets active.

Reason for the implementation

Robustness of the state changes (e.g. Oscillating conditions shall be ignored).

Fitcriterion

The transition time and the restart of the transition timer by setting the transition condition to False and to True again results in the expected transition timing ($\pm 0.05s$).

Testresult

This test was passed with the state: **Success**.

Info Initialising state machine with state_a

StateMachine: State change ('__init__'): None -> 'state_a'

Success Initial state after Initialisation is correct (Content 'state_a' and Type is <class 'str'>).

Result (Initial state after Initialisation): 'state_a' (<class 'str'>)

Expectation (Initial state after Initialisation): result = 'state_a' (<class 'str'>)

Info Waiting for 0.160s or state change

```
StateMachine: State change ('condition_true'): 'state_a' -> 'state_b'
```

Success State after 1st cycle is correct (Content 'state_b' and Type is <class 'str'>).

```
Result (State after 1st cycle): 'state_b' (<class 'str'>)
```

```
Expectation (State after 1st cycle): result = 'state_b' (<class 'str'>)
```

Success Transition time after 1st cycle is correct (Content 0.1506061553955078 in [0.145 ... 0.155] and Type is <class 'float'>).

```
Result (Transition time after 1st cycle): 0.1506061553955078 (<class 'float'>)
```

```
Expectation (Transition time after 1st cycle): 0.145 <= result <= 0.155
```

Info Waiting for 0.235s or state change

```
StateMachine: State change ('condition_true'): 'state_b' -> 'state_c'
```

Success State after 2nd cycle is correct (Content 'state_c' and Type is <class 'str'>).

```
Result (State after 2nd cycle): 'state_c' (<class 'str'>)
```

```
Expectation (State after 2nd cycle): result = 'state_c' (<class 'str'>)
```

Success Transition time after 2nd cycle is correct (Content 0.15029168128967285 in [0.145 ... 0.155] and Type is <class 'float'>).

```
Result (Transition time after 2nd cycle): 0.15029168128967285 (<class 'float'>)
```

```
Expectation (Transition time after 2nd cycle): 0.145 <= result <= 0.155
```

Success Previous state duration is correct (Content 0.22554683685302734 in [0.21999999999999997 ... 0.22999999999999998] and Type is <class 'float'>).

```
Result (Previous state duration): 0.22554683685302734 (<class 'float'>)
```

```
Expectation (Previous state duration): 0.21999999999999997 <= result <= 0.22999999999999998
```

B.1.7 Transitionpriorisation

Description

The state machine shall use the first active transition. If multiple transition are active, the transition with the highest overlap time will be used.

Reason for the implementation

Compensate the weakness of the execution quantisation.

Fitcriterion

At least one transition with at least two active conditions results in the expected state change.

Testresult

This test was passed with the state: **Success**.

Info Initialising state machine with state_a, a transition to state_b after 0.151s and a transition to state_c after 0.150s

StateMachine: State change ('__init__'): None -> 'state_a'

Success Initial state after Initialisation is correct (Content 'state_a' and Type is <class 'str'>).

Result (Initial state after Initialisation): 'state_a' (<class 'str'>)

Expectation (Initial state after Initialisation): result = 'state_a' (<class 'str'>)

Info Waiting for 0.300s or state change

Executing method work after 0.000s

Executing method work after 0.060s

Executing method work after 0.121s

Executing method work after 0.181s

StateMachine: State change ('condition_true'): 'state_a' -> 'state_c'

Success State after 1st cycle is correct (Content 'state_c' and Type is <class 'str'>).

Result (State after 1st cycle): 'state_c' (<class 'str'>)

Expectation (State after 1st cycle): result = 'state_c' (<class 'str'>)

B.1.8 This State

Description

The Module shall have a method for getting the current state.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returned state fits to the expectation.

Testresult

This test was passed with the state: **Success**.

Info Initialising the state machine with state_c

StateMachine: State change ('__init__'): None -> 'state_c'

Success Returnvalue of this_state() is correct (Content 'state_c' and Type is <class 'str'>).

Result (Returnvalue of this_state()): 'state_c' (<class 'str'>)

Expectation (Returnvalue of this_state()): result = 'state_c' (<class 'str'>)

B.1.9 This State is

Description

The Module shall have a method for checking if the given state is currently active.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least two calls with different return values fit to the expectation.

Testresult

This test was passed with the state: **Success**.

Info Initialising the state machine with state_c

```
StateMachine: State change ('__init__'): None -> 'state_c'
```

Success Returnvalue of this_state_is(state_c) is correct (Content True and Type is <class 'bool'>).

```
Result (Returnvalue of this_state_is(state_c)): True (<class 'bool'>)
```

```
Expectation (Returnvalue of this_state_is(state_c)): result = True (<class 'bool'>)
```

Success Returnvalue of this_state_is(state_b) is correct (Content False and Type is <class 'bool'>).

```
Result (Returnvalue of this_state_is(state_b)): False (<class 'bool'>)
```

```
Expectation (Returnvalue of this_state_is(state_b)): result = False (<class 'bool'>)
```

B.1.10 This State Duration

Description

The Module shall have a method for getting the time since the last state change appears.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returned duration fits to the current state duration ($\pm 0.05s$).

Testresult

This test was passed with the state: **Success**.

```

Info    Running state machine test sequence.
-----
StateMachine: State change ('__init__'): None -> 'state_a'
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Waiting for 0.25s
-----
Success Return Value of this_state_duration() is correct (Content 0.25031614303588867 in [0.2 ... 0.3] and Type
           is <class 'float'>).
-----
Result (Return Value of this_state_duration()): 0.25031614303588867 (<class 'float'>)
Expectation (Return Value of this_state_duration()): 0.2 <= result <= 0.3
    
```

B.1.11 Last Transition Condition

Description

The Module shall have a method for getting the last transition condition.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returned transition condition fits to the expectation.

Testresult

This test was passed with the state: **Success**.

```

Info    Running state machine test sequence.
-----
StateMachine: State change ('__init__'): None -> 'state_a'
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
-----
Success Returnvalue of last_transition_condition() is correct (Content 'condition_a' and Type is <class 'str'>).
-----
Result (Returnvalue of last_transition_condition()): 'condition_a' (<class 'str'>)
Expectation (Returnvalue of last_transition_condition()): result = 'condition_a' (<class
↳ 'str'>)
    
```

B.1.12 Last Transition Condition was

Description

The Module shall have a method for checking if the given condition was the last transition condition.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least two calls with different return values fit to the expectation.

Testresult

This test was passed with the state: **Success**.

Info Running state machine test sequence.

StateMachine: State change ('__init__'): None -> 'state_a'

StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'

Success Returnvalue of last_transition_condition(condition_a) is correct (Content True and Type is <class 'bool'>).

Result (Returnvalue of last_transition_condition(condition_a)): True (<class 'bool'>)

Expectation (Returnvalue of last_transition_condition(condition_a)): result = True (<class 'bool'>)
 ↔ 'bool'>)

Success Returnvalue of last_transition_condition(condition_c) is correct (Content False and Type is <class 'bool'>).

Result (Returnvalue of last_transition_condition(condition_c)): False (<class 'bool'>)

Expectation (Returnvalue of last_transition_condition(condition_c)): result = False (<class 'bool'>)
 ↔ 'bool'>)

B.1.13 Previous State

Description

The Module shall have a method for getting the previous state.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returned state fits to the expectation.

Testresult

This test was passed with the state: **Success**.

Info Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
```

```
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
```

Success Returnvalue of previous_state() is correct (Content 'state_a' and Type is <class 'str'>).

```
Result (Returnvalue of previous_state()): 'state_a' (<class 'str'>)
```

```
Expectation (Returnvalue of previous_state()): result = 'state_a' (<class 'str'>)
```

B.1.14 Previous State was

Description

The Module shall have a method for checking if the given state was the previous state.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least two calls with different return values fit to the expectation.

Testresult

This test was passed with the state: **Success**.

Info Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
```

```
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
```

Success Returnvalue of previous_state_was(state.a) is correct (Content True and Type is <class 'bool'>).

```
Result (Returnvalue of previous_state_was(state_a)): True (<class 'bool'>)
```

```
Expectation (Returnvalue of previous_state_was(state_a)): result = True (<class 'bool'>)
```

Success Returnvalue of previous_state_was(state.b) is correct (Content False and Type is <class 'bool'>).

```
Result (Returnvalue of previous_state_was(state_b)): False (<class 'bool'>)
```

```
Expectation (Returnvalue of previous_state_was(state_b)): result = False (<class 'bool'>)
```

B.1.15 Previous State Duration

Description

The Module shall have a method for getting active time for the previous state.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returned duration fits to the previous state duration ($\pm 0.05s$).

Testresult

This test was passed with the state: **Success**.

Info	Running state machine test sequence.
-------------	--------------------------------------

```

StateMachine: State change ('__init__'): None -> 'state_a'
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Waiting for 0.75s
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
    
```

Success	Return Value of previous_state_duration() is correct (Content 0.7510056495666504 in [0.7 ... 0.8] and Type is <class 'float'>).
----------------	---

```

Result (Return Value of previous_state_duration()): 0.7510056495666504 (<class 'float'>)
Expectation (Return Value of previous_state_duration()): 0.7 <= result <= 0.8
    
```

B.1.16 State change callback for a defined transition and targetstate

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for a defined set of *transition_condition* and *target_state*.

Reason for the implementation

Triggering state change actions for a specific transition condition and targetstate.

Fitcriterion

Methods are called in the registration order after state change with all user given arguments for the defined transition condition and targetstate and at least for one other condition not.

Testresult

This test was passed with the state: **Success**.

Info	Running state machine sequence and storing sequence number for each callback
-------------	--

```

StateMachine: State change ('__init__'): None -> 'state_a'
Increasing sequence number to 1 caused by sequence progress
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Executing callback 0 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 2 caused by callback_execution
Executing callback 1 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 3 caused by callback_execution
Increasing sequence number to 4 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
Increasing sequence number to 5 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_a' -> 'state_b'
Increasing sequence number to 6 caused by sequence progress
StateMachine: State change ('condition_c'): 'state_b' -> 'state_c'

```

Success Execution of state machine callback (1) (state_b, condition.a) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

```

Result (Execution of state machine callback (1) (state_b, condition_a) identified by a
↪ sequence number): [ 1 ] (<class 'list'>)
Expectation (Execution of state machine callback (1) (state_b, condition_a) identified by a
↪ sequence number): result = [ 1 ] (<class 'list'>)
Result (Submitted value number 1): 1 (<class 'int'>)
Expectation (Submitted value number 1): result = 1 (<class 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <class 'int'>).

```

Success Execution of state machine callback (2) (state_b, condition.a) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

```

Result (Execution of state machine callback (2) (state_b, condition_a) identified by a
↪ sequence number): [ 2 ] (<class 'list'>)
Expectation (Execution of state machine callback (2) (state_b, condition_a) identified by a
↪ sequence number): result = [ 2 ] (<class 'list'>)
Result (Submitted value number 1): 2 (<class 'int'>)
Expectation (Submitted value number 1): result = 2 (<class 'int'>)
Submitted value number 1 is correct (Content 2 and Type is <class 'int'>).

```

B.1.17 State change callback for a defined transition

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for a defined *transition_condition* and all *target_states*.

Reason for the implementation

Triggering state change actions for a specific transition condition.

Fitcriterion

Methods are called in the registration order after state change with all user given arguments for the defined transition condition and at least for one other transition condition not.

Testresult

This test was passed with the state: **Success**.

Info Running state machine sequence and storing sequence number for each callback

```

StateMachine: State change ('__init__'): None -> 'state_a'
Increasing sequence number to 1 caused by sequence progress
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Increasing sequence number to 2 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
Executing callback 0 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 3 caused by callback_execution
Executing callback 1 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 4 caused by callback_execution
Increasing sequence number to 5 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_a' -> 'state_b'
Executing callback 0 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 6 caused by callback_execution
Executing callback 1 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 7 caused by callback_execution
Increasing sequence number to 8 caused by sequence progress
StateMachine: State change ('condition_c'): 'state_b' -> 'state_c'
    
```

Success Execution of state machine callback (1) (all_transitions, condition_b) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

```

Result (Execution of state machine callback (1) (all_transitions, condition_b) identified by
↪ a sequence number): [ 2, 5 ] (<class 'list'>)
Expectation (Execution of state machine callback (1) (all_transitions, condition_b)
↪ identified by a sequence number): result = [ 2, 5 ] (<class 'list'>)
Result (Submitted value number 1): 2 (<class 'int'>)
Expectation (Submitted value number 1): result = 2 (<class 'int'>)
Submitted value number 1 is correct (Content 2 and Type is <class 'int'>).
Result (Submitted value number 2): 5 (<class 'int'>)
Expectation (Submitted value number 2): result = 5 (<class 'int'>)
Submitted value number 2 is correct (Content 5 and Type is <class 'int'>).
    
```

Success Execution of state machine callback (2) (all_transitions, condition_b) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

```
Result (Execution of state machine callback (2) (all_transitions, condition_b) identified by  
↳ a sequence number): [ 3, 6 ] (<class 'list'>)
```

```
Expectation (Execution of state machine callback (2) (all_transitions, condition_b)  
↳ identified by a sequence number): result = [ 3, 6 ] (<class 'list'>)
```

```
Result (Submitted value number 1): 3 (<class 'int'>)
```

```
Expectation (Submitted value number 1): result = 3 (<class 'int'>)
```

```
Submitted value number 1 is correct (Content 3 and Type is <class 'int'>).
```

```
Result (Submitted value number 2): 6 (<class 'int'>)
```

```
Expectation (Submitted value number 2): result = 6 (<class 'int'>)
```

```
Submitted value number 2 is correct (Content 6 and Type is <class 'int'>).
```

B.1.18 State change callback for a defined targetstate

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for all *transition_conditions* and a defined *target_state*.

Reason for the implementation

Triggering state change actions for a specific targetstate.

Fitcriterion

Methods are called in the registration order after state change with the defined targetstate and at least for one other targetstate not.

Testresult

This test was passed with the state: **Success**.

Info Running state machine sequence and storing sequence number for each callback

```

StateMachine: State change ('__init__'): None -> 'state_a'
Increasing sequence number to 1 caused by sequence progress
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Executing callback 0 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 2 caused by callback_execution
Executing callback 1 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 3 caused by callback_execution
Increasing sequence number to 4 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
Increasing sequence number to 5 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_a' -> 'state_b'
Executing callback 0 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 6 caused by callback_execution
Executing callback 1 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 7 caused by callback_execution
Increasing sequence number to 8 caused by sequence progress
StateMachine: State change ('condition_c'): 'state_b' -> 'state_c'

```

Success Execution of state machine callback (1) (state_b, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

```

Result (Execution of state machine callback (1) (state_b, all_conditions) identified by a
↪ sequence number): [ 1, 5 ] (<class 'list'>)
Expectation (Execution of state machine callback (1) (state_b, all_conditions) identified by
↪ a sequence number): result = [ 1, 5 ] (<class 'list'>)
Result (Submitted value number 1): 1 (<class 'int'>)
Expectation (Submitted value number 1): result = 1 (<class 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <class 'int'>).
Result (Submitted value number 2): 5 (<class 'int'>)
Expectation (Submitted value number 2): result = 5 (<class 'int'>)
Submitted value number 2 is correct (Content 5 and Type is <class 'int'>).

```

Success Execution of state machine callback (2) (state_b, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

```

Result (Execution of state machine callback (2) (state_b, all_conditions) identified by a
↪ sequence number): [ 2, 6 ] (<class 'list'>)
Expectation (Execution of state machine callback (2) (state_b, all_conditions) identified by
↪ a sequence number): result = [ 2, 6 ] (<class 'list'>)
Result (Submitted value number 1): 2 (<class 'int'>)
Expectation (Submitted value number 1): result = 2 (<class 'int'>)
Submitted value number 1 is correct (Content 2 and Type is <class 'int'>).
Result (Submitted value number 2): 6 (<class 'int'>)
Expectation (Submitted value number 2): result = 6 (<class 'int'>)
Submitted value number 2 is correct (Content 6 and Type is <class 'int'>).

```

B.1.19 State change callback for all kind of state changes

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for all transitions.

Reason for the implementation

Triggering state change actions for all transition conditions and targetstates.

Fitcriterion

Methods are called in the registration order after state change.

Testresult

This test was passed with the state: **Success**.

Info	Running state machine sequence and storing sequence number for each callback
StateMachine: State change ('__init__'): None -> 'state_a'	
Increasing sequence number to 1 caused by sequence progress	
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'	
Executing callback 0 - tests.test_callbacks.exec_with_counter	
Increasing sequence number to 2 caused by callback_execution	
Executing callback 1 - tests.test_callbacks.exec_with_counter	
Increasing sequence number to 3 caused by callback_execution	
Increasing sequence number to 4 caused by sequence progress	
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'	
Executing callback 0 - tests.test_callbacks.exec_with_counter	
Increasing sequence number to 5 caused by callback_execution	
Executing callback 1 - tests.test_callbacks.exec_with_counter	
Increasing sequence number to 6 caused by callback_execution	
Increasing sequence number to 7 caused by sequence progress	
StateMachine: State change ('condition_b'): 'state_a' -> 'state_b'	
Executing callback 0 - tests.test_callbacks.exec_with_counter	
Increasing sequence number to 8 caused by callback_execution	
Executing callback 1 - tests.test_callbacks.exec_with_counter	
Increasing sequence number to 9 caused by callback_execution	
Increasing sequence number to 10 caused by sequence progress	
StateMachine: State change ('condition_c'): 'state_b' -> 'state_c'	
Executing callback 0 - tests.test_callbacks.exec_with_counter	
Increasing sequence number to 11 caused by callback_execution	
Executing callback 1 - tests.test_callbacks.exec_with_counter	
Increasing sequence number to 12 caused by callback_execution	
Success	Execution of state machine callback (1) (all_transitions, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

```

Result (Execution of state machine callback (1) (all_transitions, all_conditions) identified
↳ by a sequence number): [ 1, 4, 7, 10 ] (<class 'list'>)
Expectation (Execution of state machine callback (1) (all_transitions, all_conditions)
↳ identified by a sequence number): result = [ 1, 4, 7, 10 ] (<class 'list'>)
Result (Submitted value number 1): 1 (<class 'int'>)
Expectation (Submitted value number 1): result = 1 (<class 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <class 'int'>).
Result (Submitted value number 2): 4 (<class 'int'>)
Expectation (Submitted value number 2): result = 4 (<class 'int'>)
Submitted value number 2 is correct (Content 4 and Type is <class 'int'>).
Result (Submitted value number 3): 7 (<class 'int'>)
Expectation (Submitted value number 3): result = 7 (<class 'int'>)
Submitted value number 3 is correct (Content 7 and Type is <class 'int'>).
Result (Submitted value number 4): 10 (<class 'int'>)
Expectation (Submitted value number 4): result = 10 (<class 'int'>)
Submitted value number 4 is correct (Content 10 and Type is <class 'int'>).

```

Success Execution of state machine callback (2) (all_transitions, all_conditions) identified by a sequence number:
 Values and number of submitted values is correct. See detailed log for more information.

```

Result (Execution of state machine callback (2) (all_transitions, all_conditions) identified
↳ by a sequence number): [ 2, 5, 8, 11 ] (<class 'list'>)
Expectation (Execution of state machine callback (2) (all_transitions, all_conditions)
↳ identified by a sequence number): result = [ 2, 5, 8, 11 ] (<class 'list'>)
Result (Submitted value number 1): 2 (<class 'int'>)
Expectation (Submitted value number 1): result = 2 (<class 'int'>)
Submitted value number 1 is correct (Content 2 and Type is <class 'int'>).
Result (Submitted value number 2): 5 (<class 'int'>)
Expectation (Submitted value number 2): result = 5 (<class 'int'>)
Submitted value number 2 is correct (Content 5 and Type is <class 'int'>).
Result (Submitted value number 3): 8 (<class 'int'>)
Expectation (Submitted value number 3): result = 8 (<class 'int'>)
Submitted value number 3 is correct (Content 8 and Type is <class 'int'>).
Result (Submitted value number 4): 11 (<class 'int'>)
Expectation (Submitted value number 4): result = 11 (<class 'int'>)
Submitted value number 4 is correct (Content 11 and Type is <class 'int'>).

```

B.1.20 Execution order of Callbacks

Description

The callbacks shall be executed in the same order as they had been registered.

Reason for the implementation

User shall have the control about the execution order.

Fitcriterion

A callback with specific targetstate and condition will be executed before a non specific callback if the specific one had been registered first.

Testresult

This test was passed with the state: **Success**.

Success Callback execution order: Values and number of submitted values is correct. See detailed log for more information.

```

StateMachine: State change ('__init__'): None -> 'state_a'
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Executing callback 0 - unittest.test.report_value
Executing callback 1 - unittest.test.report_value
Result (Callback execution order): [ 'specific callback', 'nonspecific callback' ] (<class 'list'>)
Expectation (Callback execution order): result = [ 'specific callback', 'nonspecific callback' ] (<class 'list'>)
Result (Submitted value number 1): 'specific callback' (<class 'str'>)
Expectation (Submitted value number 1): result = 'specific callback' (<class 'str'>)
Submitted value number 1 is correct (Content 'specific callback' and Type is <class 'str'>).
Result (Submitted value number 2): 'nonspecific callback' (<class 'str'>)
Expectation (Submitted value number 2): result = 'nonspecific callback' (<class 'str'>)
Submitted value number 2 is correct (Content 'nonspecific callback' and Type is <class 'str'>).
    
```

C Test-Coverage

C.1 state_machine

The line coverage for state_machine was 100.0%
 The branch coverage for state_machine was 100.0%

C.1.1 state_machine.__init__.py

The line coverage for state_machine.__init__.py was 100.0%
 The branch coverage for state_machine.__init__.py was 100.0%

```

1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 #
4 """
    
```

Unittest for state_machine

```
5 state_machine (State Machine)
6 =====
7
8 **Author:**
9
10 * Dirk Alders <sudo-dirk@mount-mockery.de>
11
12 **Description:**
13
14     This Module helps implementing state machines.
15
16 **Submodules:**
17
18 * :class:`state_machine.state_machine`
19
20 **Unittest:**
21
22     See also the :download:`unittest <state_machine/_testresults_/unittest.pdf>` documentation.
23
24 **Module Documentation:**
25
26 """
27 __DEPENDENCIES__ = []
28
29 import logging
30 import time
31
32
33 logger_name = 'STATE_MACHINE'
34 logger = logging.getLogger(logger_name)
35
36
37 __INTERPRETER__ = (2, 3)
38 """The supported Interpreter-Versions"""
39 __DESCRIPTION__ = """This Module helps implementing state machines."""
40 """The Module description"""
41
42
43 class state_machine(object):
44     """
45     :param default_state: The default state which is set on initialisation.
46     :param log_lvl: The log level, this Module logs to (see Logging-Levels of Module :mod:`logging`
47     `)
48
49     .. note:: Additional keyword parameters will be stored as variables of the instance (e.g. to
50     give variables or methods for transition condition calculation).
51
52     A state machine class can be created by deriving it from this class. The transitions are
53     defined by overriding the variable `TRANSITIONS`.
54     This Variable is a dictionary, where the key is the start-state and the content is a tuple or
55     list of transitions. Each transition is a tuple or list
56     including the following information: (condition-method (str), transition-time (number),
57     target_state (str)).
58
59     .. note:: The condition-method needs to be implemented as part of the new class.
60
61     .. note:: It is usefull to define the states as variables of this class.
62
63     """
64
65     **Example:**
```

Unittest for state_machine

```
61 .. literalinclude:: ../examples/example.py
62
63 .. literalinclude:: ../examples/example.log
64 """
65 TRANSITIONS = {}
66 LOG_PREFIX = 'StateMachine:'
67
68 def __init__(self, default_state, log_lvl, **kwargs):
69     self.__state__ = None
70     self.__last_transition_condition__ = None
71     self.__conditions_start_time__ = {}
72     self.__state_change_callbacks__ = {}
73     self.__log_lvl__ = log_lvl
74     self.__set_state__(default_state, '__init__')
75     self.__callback_id__ = 0
76     for key in kwargs:
77         setattr(self, key, kwargs.get(key))
78
79 def register_state_change_callback(self, state, condition, callback, *args, **kwargs):
80     """
81     :param state: The target state. The callback will be executed, if the state machine
82     changes to this state. None means all states.
83     :type state: str
84     :param condition: The transition condition. The callback will be executed, if this
85     condition is responsible for the state change. None means all conditions.
86     :type condition: str
87     :param callback: The callback to be executed.
88
89     .. note:: Additional arguments and keyword parameters are supported. These arguments and
90     parameters will be used as arguments and parameters for the callback execution.
91
92     This methods allows to register callbacks which will be executed on state changes.
93     """
94     if state not in self.__state_change_callbacks__:
95         self.__state_change_callbacks__[state] = {}
96     if condition not in self.__state_change_callbacks__[state]:
97         self.__state_change_callbacks__[state][condition] = []
98     self.__state_change_callbacks__[state][condition].append((self.__callback_id__, callback,
99     args, kwargs))
100     self.__callback_id__ += 1
101
102 def this_state(self):
103     """
104     :return: The current state.
105
106     This method returns the current state of the state machine.
107     """
108     return self.__state__
109
110 def this_state_is(self, state):
111     """
112     :param state: The state to be checked
113     :type state: str
114     :return: True if the given state is currently active, else False.
115     :rtype: bool
116
117     This methods returns the boolean information if the state machine is currently in the
118     given state.
119     """
120     return self.__state__ == state
121
122 def this_state_duration(self):
```

Unittest for state_machine

```
118     """
119     :return: The time how long the current state is active.
120     :rtype: float
121
122     This method returns the time how long the current state is active.
123     """
124     return time.time() - self.__time_stamp_state_change__
125
126 def last_transition_condition(self):
127     """
128     :return: The last transition condition.
129     :rtype: str
130
131     This method returns the last transition condition.
132     """
133     return self.__last_transition_condition__
134
135 def last_transition_condition_was(self, condition):
136     """
137     :param condition: The condition to be checked
138     :type condition: str
139     :return: True if the given condition was the last transition condition, else False.
140     :rtype: bool
141
142     This methods returns the boolean information if the last transition condition is
143     equivalent to the given condition.
144     """
145     return self.__last_transition_condition__ == condition
146
147 def previous_state(self):
148     """
149     :return: The previous state.
150     :rtype: str
151
152     This method returns the previous state of the state machine.
153     """
154     return self.__prev_state__
155
156 def previous_state_was(self, state):
157     """
158     :param state: The state to be checked
159     :type state: str
160     :return: True if the given state was previously active, else False.
161     :rtype: bool
162
163     This methods returns the boolean information if the state machine was previously in the
164     given state.
165     """
166     return self.__prev_state__ == state
167
168 def previous_state_duration(self):
169     """
170     :return: The time how long the previous state was active.
171     :rtype: float
172
173     This method returns the time how long the previous state was active.
174     """
175     return self.__prev_state_dt__
```

Unittest for state_machine

```

175 def __set_state__(self, target_state, condition):
176     logger.log(self.__log_lvl__, "%s State change (%s): %s -> %s", self.LOG_PREFIX, repr(
condition), repr(self.__state__), repr(target_state))
177     timestamp = time.time()
178     self.__prev_state__ = self.__state__
179     if self.__prev_state__ is None:
180         self.__prev_state_dt__ = 0.
181     else:
182         self.__prev_state_dt__ = timestamp - self.__time_stamp_state_change__
183         self.__state__ = target_state
184         self.__last_transition_condition__ = condition
185         self.__time_stamp_state_change__ = timestamp
186         self.__conditions_start_time__ = {}
187         # TODO: Execute callbacks in the same order as registration (by sorting after
identification)
188         this_state_change_callbacks = self.__state_change_callbacks__.get(None, {}).get(None, [])
189         this_state_change_callbacks.extend(self.__state_change_callbacks__.get(target_state, {}).
get(None, []))
190         this_state_change_callbacks.extend(self.__state_change_callbacks__.get(None, {}).get(
condition, []))
191         this_state_change_callbacks.extend(self.__state_change_callbacks__.get(target_state, {}).
get(condition, []))
192         # Callback execution
193         this_state_change_callbacks.sort()
194         for cid, callback, args, kwargs in this_state_change_callbacks:
195             logger.debug('Executing callback %d - %s.%s', cid, callback.__module__, callback.
__name__)
196             callback(*args, **kwargs)
197
198 def work(self):
199     """
200     This Method needs to be executed cyclicly to enable the state machine.
201     """
202     tm = time.time()
203     transitions = self.TRANSITIONS.get(self.this_state())
204     if transitions is not None:
205         active_transitions = []
206         cnt = 0
207         for method_name, transition_delay, target_state in transitions:
208             method = getattr(self, method_name)
209             if method():
210                 if method_name not in self.__conditions_start_time__:
211                     self.__conditions_start_time__[method_name] = tm
212                 if tm - self.__conditions_start_time__[method_name] >= transition_delay:
213                     active_transitions.append((transition_delay - tm + self.
__conditions_start_time__[method_name], cnt, target_state, method_name))
214             else:
215                 self.__conditions_start_time__[method_name] = tm
216                 cnt += 1
217         if len(active_transitions) > 0:
218             active_transitions.sort()
219             self.__set_state__(active_transitions[0][2], active_transitions[0][3])

```