

# Unittest for state\_machine

January 14, 2021

# Contents

<b>1</b>	<b>Test Information</b>	<b>4</b>
1.1	Test Candidate Information . . . . .	4
1.2	Unittest Information . . . . .	4
1.3	Test System Information . . . . .	4
<b>2</b>	<b>Statistic</b>	<b>4</b>
2.1	Test-Statistic for testrun with python 2.7.18 (final) . . . . .	4
2.2	Test-Statistic for testrun with python 3.8.5 (final) . . . . .	5
2.3	Coverage Statistic . . . . .	5
<b>3</b>	<b>Tested Requirements</b>	<b>6</b>
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
<b>A</b>	<b>Trace for testrun with python 2.7.18 (final)</b>	<b>26</b>
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 . . . . .	40
A.1.20	Execution order of Callbacks . . . . .	42

<b>B Trace for testrun with python 3.8.5 (final)</b>	<b>44</b>
B.1 Tests with status Info (20)	44
B.1.1 Default State	44
B.1.2 Default Last Transition Condtion	44
B.1.3 Default Previous State	45
B.1.4 Additional Keyword Arguments	45
B.1.5 Transitiondefinition and -flow	46
B.1.6 Transitiontiming	47
B.1.7 Transitionpriorisation	49
B.1.8 This State	50
B.1.9 This State is	50
B.1.10 This State Duration	51
B.1.11 Last Transition Condition	51
B.1.12 Last Transition Condition was	52
B.1.13 Previous State	53
B.1.14 Previous State was	53
B.1.15 Previous State Duration	54
B.1.16 State change callback for a defined transition and targetstate	54
B.1.17 State change callback for a defined transition	56
B.1.18 State change callback for a defined targetstate	57
B.1.19 State change callback for all kind of state changes	58
B.1.20 Execution order of Callbacks	60
<b>C Test-Coverage</b>	<b>62</b>
C.1 state_machine	62
C.1.1 state_machine.__init__.py	62

# 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	f0888b702a36dcafb48cab5d887f4dd

---

Dependencies	
--------------	--

---

## 1.2 Unittest Information

---

Unittest Information	
Version	88eb21720b062b30078e96dd6204ccdd
Testruns with	python 2.7.18 (final), python 3.8.5 (final)

---

## 1.3 Test System Information

---

System Information	
Architecture	64bit
Distribution	Linux Mint 20.1 ulyssa
Hostname	ahorn
Kernel	5.4.0-60-generic (#67-Ubuntu SMP Tue Jan 5 18:31:36 UTC 2021)
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 2.7.18 (final)

---

Number of tests	<b>20</b>
Number of successfull tests	<b>20</b>
Number of possibly failed tests	<b>0</b>
Number of failed tests	<b>0</b>

---

Executionlevel	Full Test (all defined tests)
Time consumption	1.656s

---

## 2.2 Test-Statistic for testrun with python 3.8.5 (final)

---

Number of tests	<b>20</b>
Number of successfull tests	<b>20</b>
Number of possibly failed tests	<b>0</b>
Number of failed tests	<b>0</b>

---

Executionlevel	Full Test (all defined tests)
Time consumption	1.650s

---

## 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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (22)
Start-Time:	2021-01-14 01:07:00,075
Finished-Time:	2021-01-14 01:07:00,075
Time-Consumption	0.000s

---

**Testsummary:**

---

<b>Info</b>	Initialising the state machine with state.c
<b>Success</b>	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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (22)
Start-Time:	2021-01-14 01:07:02,144
Finished-Time:	2021-01-14 01:07:02,145
Time-Consumption	0.001s

---

**Testsummary:**

---

<b>Info</b>	Initialising the state machine with state.c
<b>Success</b>	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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (23)
Start-Time:	2021-01-14 01:07:00,075
Finished-Time:	2021-01-14 01:07:00,076
Time-Consumption	0.000s

---

**Testsummary:**

---

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

---

**Testresult**

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

---

Testrun:	python 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (23)
Start-Time:	2021-01-14 01:07:02,145
Finished-Time:	2021-01-14 01:07:02,145
Time-Consumption	0.000s

---

**Testsummary:**

---

<b>Info</b>	Initialising the state machine with state_c
<b>Success</b>	Last transition condition after initialisation is correct (Content ' <code>__init__</code> ' and Type is <code>&lt;class 'str'&gt;</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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/___init___py (24)
Start-Time:	2021-01-14 01:07:00,076
Finished-Time:	2021-01-14 01:07:00,076
Time-Consumption	0.000s

---

**Testsummary:**

---

<b>Info</b>	Initialising the state machine with state.c
<b>Success</b>	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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/___init___py (24)
Start-Time:	2021-01-14 01:07:02,145
Finished-Time:	2021-01-14 01:07:02,145
Time-Consumption	0.000s

---

**Testsummary:**

---

<b>Info</b>	Initialising the state machine with state.c
<b>Success</b>	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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/___init___py (25)
Start-Time:	2021-01-14 01:07:00,076
Finished-Time:	2021-01-14 01:07:00,077
Time-Consumption	0.001s

---

**Testsummary:**

---

<b>Info</b>	Initialising the state machine with state.c
<b>Success</b>	Keyword argument kw_arg_no.4 stored in state_machine is correct (Content {'1': 1, '2': 'two'} and Type is <type 'dict'>).
<b>Success</b>	Keyword argument kw_arg_no.1 stored in state_machine is correct (Content 1 and Type is <type 'int'>).
<b>Success</b>	Keyword argument kw_arg_no.3 stored in state_machine is correct (Content True and Type is <type 'bool'>).
<b>Success</b>	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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init....py (25)
Start-Time:	2021-01-14 01:07:02,146
Finished-Time:	2021-01-14 01:07:02,146
Time-Consumption	0.001s

---

**Testsummary:**

---

<b>Info</b>	Initialising the state machine with state.c
<b>Success</b>	Keyword argument kw_arg_no.1 stored in state_machine is correct (Content 1 and Type is <class 'int'>).
<b>Success</b>	Keyword argument kw_arg_no.2 stored in state_machine is correct (Content '2' and Type is <class 'str'>).
<b>Success</b>	Keyword argument kw_arg_no.3 stored in state_machine is correct (Content True and Type is <class 'bool'>).
<b>Success</b>	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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (28)
Start-Time:	2021-01-14 01:07:00,077
Finished-Time:	2021-01-14 01:07:00,078
Time-Consumption	0.001s

---

**Testsummary:**

---

<b>Info</b>	Initialising state machine with state_a
<b>Success</b>	Initial state after Initialisation is correct (Content 'state_a' and Type is <type 'str'>).
<b>Info</b>	Work routine executed the 1st time to do the state change. Defined Transitions are: True→state_b (0.0s); False→state_c (0.0s)
<b>Success</b>	State after 1st execution of work method is correct (Content 'state_b' and Type is <type 'str'>).
<b>Info</b>	Work routine executed the 2nd time to do the state change. Defined Transitions are: False→state_a (0.0s); True→state_c (0.0s)
<b>Success</b>	State after 2nd execution of work method is correct (Content 'state_c' and Type is <type 'str'>).
<b>Info</b>	Work routine executed the 3rd time with no effect. No Transitions starting from state_c (dead end)
<b>Success</b>	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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (28)
Start-Time:	2021-01-14 01:07:02,146
Finished-Time:	2021-01-14 01:07:02,147
Time-Consumption	0.001s

---

**Testsummary:**

---

<b>Info</b>	Initialising state machine with state_a
<b>Success</b>	Initial state after Initialisation is correct (Content 'state_a' and Type is <class 'str'>).
<b>Info</b>	Work routine executed the 1st time to do the state change. Defined Transitions are: True→state_b (0.0s); False→state_c (0.0s)
<b>Success</b>	State after 1st execution of work method is correct (Content 'state_b' and Type is <class 'str'>).
<b>Info</b>	Work routine executed the 2nd time to do the state change. Defined Transitions are: False→state_a (0.0s); True→state_c (0.0s)
<b>Success</b>	State after 2nd execution of work method is correct (Content 'state_c' and Type is <class 'str'>).
<b>Info</b>	Work routine executed the 3rd time with no effect. No Transitions starting from state_c (dead end)
<b>Success</b>	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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (29)
Start-Time:	2021-01-14 01:07:00,078
Finished-Time:	2021-01-14 01:07:00,459
Time-Consumption	0.381s

---

**Testsummary:**

---

<b>Info</b>	Initialising state machine with state_a
<b>Success</b>	Initial state after Initialisation is correct (Content 'state_a' and Type is <type 'str'>).
<b>Info</b>	Waiting for 0.160s or state change
<b>Success</b>	State after 1st cycle is correct (Content 'state_b' and Type is <type 'str'>).
<b>Success</b>	Transition time after 1st cycle is correct (Content 0.15072107315063477 in [0.145 ... 0.155] and Type is <type 'float'>).
<b>Info</b>	Waiting for 0.235s or state change
<b>Success</b>	State after 2nd cycle is correct (Content 'state_c' and Type is <type 'str'>).
<b>Success</b>	Transition time after 2nd cycle is correct (Content 0.15036487579345703 in [0.145 ... 0.155] and Type is <type 'float'>).
<b>Success</b>	Previous state duration is correct (Content 0.22558188438415527 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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (29)
Start-Time:	2021-01-14 01:07:02,147
Finished-Time:	2021-01-14 01:07:02,526
Time-Consumption	0.379s

---

**Testsummary:**

---

<b>Info</b>	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 0.15063071250915527 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 0.15029430389404297 in [0.145 ... 0.155] and Type is <class 'float'>).  
**Success** Previous state duration is correct (Content 0.22559070587158203 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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (30)
Start-Time:	2021-01-14 01:07:00,459
Finished-Time:	2021-01-14 01:07:00,704
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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (30)
Start-Time:	2021-01-14 01:07:02,526

Finished-Time: 2021-01-14 01:07:02,769  
 Time-Consumption 0.243s

**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 2.7.18 (final)  
 Caller: /user\_data/data/dirk/prj/unittest/state\_machine/unittest/src/tests/\_\_init\_\_.py (33)  
 Start-Time: 2021-01-14 01:07:00,705  
 Finished-Time: 2021-01-14 01:07:00,706  
 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 3.8.5 (final)  
 Caller: /user\_data/data/dirk/prj/unittest/state\_machine/unittest/src/tests/\_\_init\_\_.py (33)  
 Start-Time: 2021-01-14 01:07:02,769  
 Finished-Time: 2021-01-14 01:07:02,770  
 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 <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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (34)
Start-Time:	2021-01-14 01:07:00,706
Finished-Time:	2021-01-14 01:07:00,708
Time-Consumption	0.002s

---

#### 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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (34)
Start-Time:	2021-01-14 01:07:02,770
Finished-Time:	2021-01-14 01:07:02,771
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 <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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/___init___py (35)
Start-Time:	2021-01-14 01:07:00,709
Finished-Time:	2021-01-14 01:07:00,961
Time-Consumption	0.252s

---

**Testsummary:**

---

<b>Info</b>	Running state machine test sequence.
<b>Success</b>	Return Value of this_state_duration() is correct (Content 0.25125598907470703 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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/___init___py (35)
Start-Time:	2021-01-14 01:07:02,771
Finished-Time:	2021-01-14 01:07:03,023
Time-Consumption	0.252s

---

**Testsummary:**

---

<b>Info</b>	Running state machine test sequence.
<b>Success</b>	Return Value of this_state_duration() is correct (Content 0.25093817710876465 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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (36)
Start-Time:	2021-01-14 01:07:00,961
Finished-Time:	2021-01-14 01:07:00,963
Time-Consumption	0.001s

---

**Testsummary:**

---

<b>Info</b>	Running state machine test sequence.
<b>Success</b>	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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (36)
Start-Time:	2021-01-14 01:07:03,023
Finished-Time:	2021-01-14 01:07:03,024
Time-Consumption	0.001s

---

**Testsummary:**

---

<b>Info</b>	Running state machine test sequence.
<b>Success</b>	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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (37)
Start-Time:	2021-01-14 01:07:00,963
Finished-Time:	2021-01-14 01:07:00,965
Time-Consumption	0.002s

---

**Testsummary:**

---

<b>Info</b>	Running state machine test sequence.
<b>Success</b>	Returnvalue of last_transition_condition(condition_a) is correct (Content True and Type is <type 'bool'>).
<b>Success</b>	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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (37)
Start-Time:	2021-01-14 01:07:03,024
Finished-Time:	2021-01-14 01:07:03,026
Time-Consumption	0.001s

---

**Testsummary:**

---

<b>Info</b>	Running state machine test sequence.
<b>Success</b>	Returnvalue of last_transition_condition(condition_a) is correct (Content True and Type is <class 'bool'>).
<b>Success</b>	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 2.7.18 (final)  
 Caller: /user\_data/data/dirk/prj/unittest/state\_machine/unittest/src/tests/...init...py (38)  
 Start-Time: 2021-01-14 01:07:00,966  
 Finished-Time: 2021-01-14 01:07:00,968  
 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 3.8.5 (final)  
 Caller: /user\_data/data/dirk/prj/unittest/state\_machine/unittest/src/tests/...init...py (38)  
 Start-Time: 2021-01-14 01:07:03,026  
 Finished-Time: 2021-01-14 01:07:03,027  
 Time-Consumption 0.001s

---

**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 2.7.18 (final)  
 Caller: /user\_data/data/dirk/prj/unittest/state\_machine/unittest/src/tests/...init...py (39)  
 Start-Time: 2021-01-14 01:07:00,968  
 Finished-Time: 2021-01-14 01:07:00,970  
 Time-Consumption 0.002s

---

**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 3.8.5 (final)  
 Caller: /user\_data/data/dirk/prj/unittest/state\_machine/unittest/src/tests/\_\_init\_\_.py (39)  
 Start-Time: 2021-01-14 01:07:03,027  
 Finished-Time: 2021-01-14 01:07:03,028  
 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 <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 2.7.18 (final)  
 Caller: /user\_data/data/dirk/prj/unittest/state\_machine/unittest/src/tests/\_\_init\_\_.py (40)  
 Start-Time: 2021-01-14 01:07:00,970  
 Finished-Time: 2021-01-14 01:07:01,723  
 Time-Consumption 0.752s

---

### Testsummary:

**Info** Running state machine test sequence.  
**Success** Return Value of previous\_state\_duration() is correct (Content 0.7513411045074463 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 3.8.5 (final)  
 Caller: /user\_data/data/dirk/prj/unittest/state\_machine/unittest/src/tests/...init....py (40)  
 Start-Time: 2021-01-14 01:07:03,029  
 Finished-Time: 2021-01-14 01:07:03,781  
 Time-Consumption 0.753s

---

**Testsummary:**

---

**Info** Running state machine test sequence.  
**Success** Return Value of previous\_state\_duration() is correct (Content 0.7514586448669434 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 2.7.18 (final)  
 Caller: /user\_data/data/dirk/prj/unittest/state\_machine/unittest/src/tests/...init....py (43)  
 Start-Time: 2021-01-14 01:07:01,723  
 Finished-Time: 2021-01-14 01:07:01,726  
 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 3.8.5 (final)  
 Caller: /user\_data/data/dirk/prj/unittest/state\_machine/unittest/src/tests/...init...py (43)  
 Start-Time: 2021-01-14 01:07:03,782  
 Finished-Time: 2021-01-14 01:07:03,786  
 Time-Consumption 0.004s

---

**Testsummary:**

---

<b>Info</b>	Running state machine sequence and storing sequence number for each callback
<b>Success</b>	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.
<b>Success</b>	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 2.7.18 (final)  
 Caller: /user\_data/data/dirk/prj/unittest/state\_machine/unittest/src/tests/...init...py (44)  
 Start-Time: 2021-01-14 01:07:01,726  
 Finished-Time: 2021-01-14 01:07:01,729  
 Time-Consumption 0.003s

---

**Testsummary:**

---

<b>Info</b>	Running state machine sequence and storing sequence number for each callback
<b>Success</b>	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.
<b>Success</b>	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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/___init___py (44)
Start-Time:	2021-01-14 01:07:03,786
Finished-Time:	2021-01-14 01:07:03,789
Time-Consumption	0.003s

---

**Testsummary:**

---

<b>Info</b>	Running state machine sequence and storing sequence number for each callback
<b>Success</b>	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.
<b>Success</b>	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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/___init___py (45)
Start-Time:	2021-01-14 01:07:01,729
Finished-Time:	2021-01-14 01:07:01,733
Time-Consumption	0.003s

---

**Testsummary:**

---

<b>Info</b>	Running state machine sequence and storing sequence number for each callback
<b>Success</b>	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.
<b>Success</b>	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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (45)
Start-Time:	2021-01-14 01:07:03,790
Finished-Time:	2021-01-14 01:07:03,793
Time-Consumption	0.003s

---

**Testsummary:**

---

<b>Info</b>	Running state machine sequence and storing sequence number for each callback
<b>Success</b>	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.
<b>Success</b>	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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/...init...py (46)
Start-Time:	2021-01-14 01:07:01,733
Finished-Time:	2021-01-14 01:07:01,735
Time-Consumption	0.003s

---

**Testsummary:**

---

<b>Info</b>	Running state machine sequence and storing sequence number for each callback
<b>Success</b>	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.
<b>Success</b>	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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (46)
Start-Time:	2021-01-14 01:07:03,793
Finished-Time:	2021-01-14 01:07:03,797
Time-Consumption	0.004s

---

**Testsummary:**

---

<b>Info</b>	Running state machine sequence and storing sequence number for each callback
<b>Success</b>	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.
<b>Success</b>	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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/state_machine/unittest/src/tests/__init__.py (47)
Start-Time:	2021-01-14 01:07:01,736
Finished-Time:	2021-01-14 01:07:01,737
Time-Consumption	0.001s

---

**Testsummary:**

---

<b>Success</b>	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 3.8.5 (final)  
Caller: /user\_data/data/dirk/prj/unittest/state\_machine/unittest/src/tests/\_init\_.py (47)  
Start-Time: 2021-01-14 01:07:03,797  
Finished-Time: 2021-01-14 01:07:03,798  
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 2.7.18 (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**.

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

---

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** 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 <type 'str'>).

---

Result (State after 1st execution of work method): 'state\_b' (<type 'str'>)

Expectation (State after 1st execution of work method): result = 'state\_b' (<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)

---

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 <type 'str'>).

---

Result (State after 2nd execution of work method): 'state\_c' (<type 'str'>)

Expectation (State after 2nd execution of work method): result = 'state\_c' (<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'>).

---

Result (State after 3rd execution of work method): 'state\_c' (<type 'str'>)

Expectation (State after 3rd execution of work method): result = 'state\_c' (<type 'str'>)

---

### 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 0.15072107315063477 in [0.145 ... 0.155] and Type is <type 'float'>).

---

Result (Transition time after 1st cycle): 0.15072107315063477 (<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 0.15036487579345703 in [0.145 ... 0.155] and Type is <type 'float'>).

---

Result (Transition time after 2nd cycle): 0.15036487579345703 (<type 'float'>)

---

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

**Success** Previous state duration is correct (Content 0.22558188438415527 in [0.2199999999999997 ... 0.2299999999999998] and Type is <type 'float'>).

Result (Previous state duration): 0.22558188438415527 (<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 0.060s

Executing method work after 0.121s

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 0.25125598907470703 in [0.2 ... 0.3] and Type is <type 'float'>).

Result (Return Value of this\_state\_duration()): 0.25125598907470703 (<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 0.7513411045074463 in [0.7 ... 0.8] and Type is <type 'float'>).

---

Result (Return Value of previous\_state\_duration()): 0.7513411045074463 (<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 2 - unittest.test.report_value
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
Executing callback 1 - unittest.test.report_value
Executing callback 2 - unittest.test.report_value
Result (Callback execution order): [ 'specific callback for reaching state_b', 'nonspecific
↳ callback', 'specific callback for reaching state_a', 'nonspecific callback' ] (<type
↳ 'list'>)
Expectation (Callback execution order): result = [ 'specific callback for reaching state_b',
↳ 'nonspecific callback', 'specific callback for reaching state_a', 'nonspecific callback'
↳ ] (<type 'list'>)
Result (Submitted value number 1): 'specific callback for reaching state_b' (<type 'str'>)
Expectation (Submitted value number 1): result = 'specific callback for reaching state_b'
↳ (<type 'str'>)
Submitted value number 1 is correct (Content 'specific callback for reaching state_b' 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'>).
Result (Submitted value number 3): 'specific callback for reaching state_a' (<type 'str'>)
Expectation (Submitted value number 3): result = 'specific callback for reaching state_a'
↳ (<type 'str'>)
Submitted value number 3 is correct (Content 'specific callback for reaching state_a' and
↳ Type is <type 'str'>).
Result (Submitted value number 4): 'nonspecific callback' (<type 'str'>)
Expectation (Submitted value number 4): result = 'nonspecific callback' (<type 'str'>)
Submitted value number 4 is correct (Content 'nonspecific callback' and Type is <type 'str'>).

```

## B Trace for testrun with python 3.8.5 (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 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 <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**.

<b>Info</b>	Initialising the state machine with state_c
StateMachine: State change ('__init__'): None -> 'state_c'	
<b>Success</b>	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'>)	
<b>Success</b>	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'>)	
<b>Success</b>	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'>)	
<b>Success</b>	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 0.15063071250915527 in [0.145 ... 0.155] and Type is <class 'float'>).

---

Result (Transition time after 1st cycle): 0.15063071250915527 (<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 0.15029430389404297 in [0.145 ... 0.155] and Type is <class 'float'>).

---

Result (Transition time after 2nd cycle): 0.15029430389404297 (<class 'float'>)

---

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

**Success** Previous state duration is correct (Content 0.22559070587158203 in [0.2199999999999997 ... 0.2299999999999998] and Type is <class 'float'>).

Result (Previous state duration): 0.22559070587158203 (<class 'float'>)

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

### 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 0.25093817710876465 in [0.2 ... 0.3] and Type is <class 'float'>).

Result (Return Value of this\_state\_duration()): 0.25093817710876465 (<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'>)
```

---

**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'>)
```

---

### 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 0.7514586448669434 in [0.7 ... 0.8] and Type is <class 'float'>).

---

Result (Return Value of previous\_state\_duration()): 0.7514586448669434 (<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**.

---

<b>Info</b>	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
    
```

---

<b>Success</b>	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 2 - unittest.test.report_value
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
Executing callback 1 - unittest.test.report_value
Executing callback 2 - unittest.test.report_value
Result (Callback execution order): [ 'specific callback for reaching state_b', 'nonspecific
↳ callback', 'specific callback for reaching state_a', 'nonspecific callback' ] (<class
↳ 'list'>)
Expectation (Callback execution order): result = [ 'specific callback for reaching state_b',
↳ 'nonspecific callback', 'specific callback for reaching state_a', 'nonspecific callback'
↳ ] (<class 'list'>)
Result (Submitted value number 1): 'specific callback for reaching state_b' (<class 'str'>)
Expectation (Submitted value number 1): result = 'specific callback for reaching state_b'
↳ (<class 'str'>)
Submitted value number 1 is correct (Content 'specific callback for reaching state_b' 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'>).
Result (Submitted value number 3): 'specific callback for reaching state_a' (<class 'str'>)
Expectation (Submitted value number 3): result = 'specific callback for reaching state_a'
↳ (<class 'str'>)
Submitted value number 3 is correct (Content 'specific callback for reaching state_a' and
↳ Type is <class 'str'>).
Result (Submitted value number 4): 'nonspecific callback' (<class 'str'>)
Expectation (Submitted value number 4): result = 'nonspecific callback' (<class 'str'>)
Submitted value number 4 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 """
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 try:
34     from config import APP_NAME as ROOT_LOGGER_NAME
35 except ImportError:
36     ROOT_LOGGER_NAME = 'root'
37 logger = logging.getLogger(ROOT_LOGGER_NAME).getChild(__name__)
38
39
40 __INTERPRETER__ = (2, 3)
41 """The supported Interpreter-Versions"""
42 __DESCRIPTION__ = """This Module helps implementing state machines."""
43 """The Module description"""
44
45
46 class state_machine(object):

```

## Unittest for state\_machine

```
47 """
48 :param default_state: The default state which is set on initialisation.
49 :param log_lvl: The log level, this Module logs to (see Logging—Levels of Module :mod:`logging`
50 ``)
51 .. note:: Additional keyword parameters will be stored as variables of the instance (e.g. to
52 give variables or methods for transition condition calculation).
53
54 A state machine class can be created by deriving it from this class. The transitions are
55 defined by overriding the variable `TRANSITIONS`.
56 This Variable is a dictionary, where the key is the start-state and the content is a tuple or
57 list of transitions. Each transition is a tuple or list
58 including the following information: (condition-method (str), transition-time (number),
59 target_state (str)).
60
61 .. note:: The condition-method needs to be implemented as part of the new class.
62 .. note:: It is useful to define the states as variables of this class.
63
64 **Example:**
65
66 .. literalinclude:: state_machine/_examples_/example.py
67 .. literalinclude:: state_machine/_examples_/example.log
68 """
69 TRANSITIONS = {}
70 LOG_PREFIX = 'StateMachine:'
71
72 def __init__(self, default_state, log_lvl, **kwargs):
73     self.__state__ = None
74     self.__last_transition_condition__ = None
75     self.__conditions_start_time__ = {}
76     self.__state_change_callbacks__ = {}
77     self.__log_lvl__ = log_lvl
78     self.__set_state__(default_state, '__init__')
79     self.__callback_id__ = 0
80     for key in kwargs:
81         setattr(self, key, kwargs.get(key))
82
83 def register_state_change_callback(self, state, condition, callback, *args, **kwargs):
84     """
85     :param state: The target state. The callback will be executed, if the state machine
86     changes to this state. None means all states.
87     :type state: str
88     :param condition: The transition condition. The callback will be executed, if this
89     condition is responsible for the state change. None means all conditions.
90     :type condition: str
91     :param callback: The callback to be executed.
92
93     .. note:: Additional arguments and keyword parameters are supported. These arguments and
94     parameters will be used as arguments and parameters for the callback execution.
95
96     This methods allows to register callbacks which will be executed on state changes.
97     """
98     if state not in self.__state_change_callbacks__:
99         self.__state_change_callbacks__[state] = {}
100     if condition not in self.__state_change_callbacks__[state]:
101         self.__state_change_callbacks__[state][condition] = []
102     self.__state_change_callbacks__[state][condition].append((self.__callback_id__, callback,
103     args, kwargs))
104     self.__callback_id__ += 1
```



## Unittest for state\_machine

```
100
101 def this_state(self):
102     """
103     :return: The current state.
104
105     This method returns the current state of the state machine.
106     """
107     return self.__state__
108
109 def this_state_is(self, state):
110     """
111     :param state: The state to be checked
112     :type state: str
113     :return: True if the given state is currently active, else False.
114     :rtype: bool
115
116     This methods returns the boolean information if the state machine is currently in the
117     given state.
118     """
119     return self.__state__ == state
120
121 def this_state_duration(self):
122     """
123     :return: The time how long the current state is active.
124     :rtype: float
125
126     This method returns the time how long the current state is active.
127     """
128     return time.time() - self.__time_stamp_state_change__
129
130 def last_transition_condition(self):
131     """
132     :return: The last transition condition.
133     :rtype: str
134
135     This method returns the last transition condition.
136     """
137     return self.__last_transition_condition__
138
139 def last_transition_condition_was(self, condition):
140     """
141     :param condition: The condition to be checked
142     :type condition: str
143     :return: True if the given condition was the last transition condition, else False.
144     :rtype: bool
145
146     This methods returns the boolean information if the last transition condition is
147     equivalent to the given condition.
148     """
149     return self.__last_transition_condition__ == condition
150
151 def previous_state(self):
152     """
153     :return: The previous state.
154     :rtype: str
155
156     This method returns the previous state of the state machine.
157     """
158     return self.__prev_state__
159
160 def previous_state_was(self, state):
```

## Unittest for state\_machine

```

159     """
160     :param state: The state to be checked
161     :type state: str
162     :return: True if the given state was previously active, else False.
163     :rtype: bool
164
165     This methods returns the boolean information if the state machine was previously in the
166     given state.
167     """
168     return self.__prev_state__ == state
169
170 def previous_state_duration(self):
171     """
172     :return: The time how long the previous state was active.
173     :rtype: float
174
175     This method returns the time how long the previous state was active.
176     """
177     return self.__prev_state_dt__
178
179 def __set_state__(self, target_state, condition):
180     logger.log(self.__log_lvl__, "%s State change (%s): %s -> %s", self.LOG_PREFIX, repr(
181     condition), repr(self.__state__), repr(target_state))
182     timestamp = time.time()
183     self.__prev_state__ = self.__state__
184     if self.__prev_state__ is None:
185         self.__prev_state_dt__ = 0.
186     else:
187         self.__prev_state_dt__ = timestamp - self.__time_stamp_state_change__
188         self.__state__ = target_state
189         self.__last_transition_condition__ = condition
190         self.__time_stamp_state_change__ = timestamp
191         self.__conditions_start_time__ = {}
192     # Callback collect
193     this_state_change_callbacks = []
194     this_state_change_callbacks.extend(self.__state_change_callbacks__.get(None, {}).get(None
195     , []))
196     this_state_change_callbacks.extend(self.__state_change_callbacks__.get(target_state, {}).
197     get(None, []))
198     this_state_change_callbacks.extend(self.__state_change_callbacks__.get(None, {}).get(
199     condition, []))
200     this_state_change_callbacks.extend(self.__state_change_callbacks__.get(target_state, {}).
201     get(condition, []))
202     # Callback sorting
203     this_state_change_callbacks.sort()
204     # Callback execution
205     for cid, callback, args, kwargs in this_state_change_callbacks:
206         logger.debug('Executing callback %d - %s.%s', cid, callback.__module__, callback.
207         __name__)
208         callback(*args, **kwargs)
209
210 def work(self):
211     """
212     This Method needs to be executed cyclicly to enable the state machine.
213     """
214     tm = time.time()
215     transitions = self.TRANSITIONS.get(self.this_state())
216     if transitions is not None:
217         active_transitions = []
218         cnt = 0
219         for method_name, transition_delay, target_state in transitions:
220             method = getattr(self, method_name)

```

## Unittest for state\_machine

```
214         if method():
215             if method_name not in self.__conditions_start_time__:
216                 self.__conditions_start_time__[method_name] = tm
217             if tm - self.__conditions_start_time__[method_name] >= transition_delay:
218                 active_transitions.append((transition_delay - tm + self.
219                 __conditions_start_time__[method_name], cnt, target_state, method_name))
219         else:
220             self.__conditions_start_time__[method_name] = tm
221             cnt += 1
222         if len(active_transitions) > 0:
223             active_transitions.sort()
224             self.__set_state__(active_transitions[0][2], active_transitions[0][3])
```