Unittest for `state_machine`

May 15, 2026

Contents

1 Test Information

1.1 Testobject Information

This Module helps implementing state machines.

Information	
Name	state_machine
State	Released
Version	1.0.4
Dependencies	
	Python standard

1.2 Testdata Information

Information	
Version	1.0.4
Testruns	Library test

1.3 Testsystem Information

System Information	
Architecture	64bit
Distribution	Debian GNU/Linux 13 trixie
Hostname	ahorn
Kernel	6.12.86+deb13-amd64 (#1 SMP PREEMPT_DYNAMIC Debian 6.12.86-1 (2026-05-08))
Machine	x86_64
Path	/home/dirk/work/unittest_collection/state_machine/unittest/src
System	Linux
Username	dirk

2 Statistic

2.1 Test-Statistic for testrun: Library test

Number of tests	**21**
Number of successfull tests	**21**
Number of possibly failed tests	**0**
Number of failed tests	**0**
Executionlevel	Full Test (all defined tests)
Time consumption	1.665s

2.2 Coverage Statistic

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

3 Tested Requirements for testrun: Library test

3.1 Initialisation

3.1.1 Initialisation with a defined state (REQ-0005)

Description
It shall be possible to initialise the state machine with a defined state.

Reason for the implementation
The user shall be able to define the start condition (state) of the state machine.

Fitcriterion
Initialisation of the state machine with a specified state. Test the state of the state machine.

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (12)
Start-Time:	2026-05-15 14:45:08,513
Finished-Time:	2026-05-15 14:45:08,514
Time-Consumption	0.001s

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

3.1.2 Last transition information after initialisation (REQ-0006)

Description
The last transition method shall return `__init__` after state machine initialisation.

Reason for the implementation
The user shall be able to differentiate between the initial transition to default state and a normal operation transition.

Fitcriterion
Initialisation of the state machine. Check the last transition information.

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (13)

Start-Time:	2026-05-15 14:45:08,514
Finished-Time:	2026-05-15 14:45:08,515
Time-Consumption	0.000s

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

3.1.3 Previous state information after initialisation (REQ-0007)

Description

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

Reason for the implementation

The user shall be able to differentiate between the initial previous state and the previous state during normal operation.

Fitcriterion

Initialisation of the state machine. Check the last state information.

Testresult

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (14)
Start-Time:	2026-05-15 14:45:08,515
Finished-Time:	2026-05-15 14:45:08,515
Time-Consumption	0.000s

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

3.1.4 Pass all kind of information inside the state machine instance (REQ-0008)

Description

It shall be possible to store all kind of information as instance attributes to the state machine.

Reason for the implementation

The user shall be able to store data or methods to the state machine attributes, which might be helpful changing the state or determining the change conditions.

Fitcriterion

Initialisation of the state machine with a bunch of data. Check that the data is stored in the state machine instance.

Testresult

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (15)
Start-Time:	2026-05-15 14:45:08,515
Finished-Time:	2026-05-15 14:45:08,516
Time-Consumption	0.001s

Testsummary:

Info	Initialising the state machine with state_c
Success	Keyword argument kw_arg_no_1 stored in state_machine is correct (Content 1 and Type is <class 'int'>).
Success	Keyword argument kw_arg_no_2 stored in state_machine is correct (Content True and Type is <class 'bool'>).
Success	Keyword argument kw_arg_no_3 stored in state_machine is correct (Content {'1': 1, '2': 'two'} and Type is <class 'dict'>).
Success	Keyword argument kw_arg_no_4 stored in state_machine is correct (Content <built-in function print> and Type is <class 'builtin_function_or_method'>).

3.2 Transitions

3.2.1 States and transitions (REQ-0017)

Description

The user shall be able to define states and transitions between the states.

Reason for the implementation

A state machine shall be able to change between states in a user defined way under user defined conditions.

Fitcriterion

Create a simple state machine with some states and conditions. Check the state change path as defined by the user.

Testresult

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (18)
Start-Time:	2026-05-15 14:45:08,516
Finished-Time:	2026-05-15 14:45:08,517
Time-Consumption	0.001s

Testsummary:

Info	Initialising state machine with state_a
Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <class 'str'>).
Info	Work routine executed, defined Transitions (from state_a) are: False→state_b (0.0s); True→state_c (0.0s)

Success	State after 1st execution of work method is correct (Content 'state_c' and Type is <class 'str'>).
Info	Work routine executed, defined Transitions (from state_c) are: True→state_b (0.0s); False→state_a (0.0s)
Success	State after 2nd execution of work method is correct (Content 'state_b' and Type is <class 'str'>).
Info	Work routine executed, no transitions defined from state_b (dead end)
Success	State after 3rd execution of work method is correct (Content 'state_b' and Type is <class 'str'>).

3.2.2 Transition timing (REQ-0018)

Description

The user shall be able to define a time a transition condition needs to be active till the transition is performed.

Reason for the implementation

In a lot of cases you might face signal noise or slight fluctuations. To be able to not change states back and for, such a timing can help.

Fitcriterion

Create a state machine with active transition condition and a defined transition time. The transition shall take the defined time. The time shall start over, if the condition was False for at least one cycle.

Testresult

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (19)
Start-Time:	2026-05-15 14:45:08,518
Finished-Time:	2026-05-15 14:45:08,897
Time-Consumption	0.380s

Testsummary:

Info	Initialising state machine with state_a
Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <class 'str'>).
Info	Checking state change every 0.15ms with timeout condition after 0.154s.
Success	State after 1st change is correct (Content 'state_b' and Type is <class 'str'>).
Success	Transition time of 1st change is correct (Content 0.1502072811126709 in [0.148 ... 0.152] and Type is <class 'float'>).
Info	Setting transition condition to false for one cycle after half of the transition time.
Info	Checking state change every 0.15ms with timeout condition after 0.154s.
Success	State after 2nd change is correct (Content 'state_c' and Type is <class 'str'>).
Success	Transition time of 2nd change is correct (Content 0.15016961097717285 in [0.148 ... 0.152] and Type is <class 'float'>).
Success	Previous state duration is correct (Content 0.22746682167053223 in [0.22499999999999998 ... 0.22899999999999998] and Type is <class 'float'>).

3.2.3 Transition prioritisation (REQ-0019)

Description
When two conditions and timings result in two state changes, when executing the worker task, the transition which is longer fulfilled shall win.

Reason for the implementation
If the execution of the worker is to infrequent, the state change shall act like the states haven't been changed, but the execution was directly after the first active state change.

Fitcriterion
Define a state machine with two active state changes, but different transition times. Execute the worker, after both transition times are fulfilled. The resulting state shall be according the first fulfilled transition.

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (20)
Start-Time:	2026-05-15 14:45:08,898
Finished-Time:	2026-05-15 14:45:09,143
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 <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 Interface

3.3.1 This state (REQ-0009)

Description
The user shall be able to get the current state of the state machine.

Reason for the implementation
Implement logic depending on the state machine's state.

Fitcriterion
Initialise a state machine with a specific state and check the result of the method `.this_state`.

Testresult

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (23)
Start-Time:	2026-05-15 14:45:09,144
Finished-Time:	2026-05-15 14:45:09,145
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 (REQ-0010)

Description

The user shall be able to get the information if the state machine is in a specific state (as boolean value).

Reason for the implementation

Implement logic depending on the state machine's state.

Fitcriterion

Initialise a state machine with a specific state and check the result of the method `.this_state_is`.

Testresult

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (24)
Start-Time:	2026-05-15 14:45:09,146
Finished-Time:	2026-05-15 14:45:09,147
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 <class 'bool'>).
Success	Returnvalue of this_state_is(state_b) is correct (Content False and Type is <class 'bool'>).

3.3.3 State duration (REQ-0011)

Description

The user shall be able to get the information how long the state machine is in the current state.

Reason for the implementation

Implement logic depending on the state machine's current state duration.

Fitcriterion
Initialise a state machine with a specific state, wait for a certain time, change the state and check the result of the method `.state_duration`.

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (25)
Start-Time:	2026-05-15 14:45:09,148
Finished-Time:	2026-05-15 14:45:09,400
Time-Consumption	0.252s

Testsummary:	
Info	Running state machine test sequence.
Success	Return Value of this_state_duration() is correct (Content 0.2510814666748047 in [0.2 ... 0.3] and Type is <class 'float'>).

3.3.4 Last transition condition (REQ-0012)

Description
The user shall be able to get the information, which transition condition changed the state to the current state.

Reason for the implementation
Implement logic depending on the last transition condition.

Fitcriterion
Initialise a state machine with a specific state, change the state and check the transition condition information given by the method `.last_transition_condition`.

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (26)
Start-Time:	2026-05-15 14:45:09,400
Finished-Time:	2026-05-15 14:45:09,401
Time-Consumption	0.001s

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

3.3.5 Last transition condition was (REQ-0013)

Description
The user shall be able to get the information, if the last transition condition was the specified one (boolean value).

Reason for the implementation
Implement logic depending on the last transition condition.

Fitcriterion
Initialise a state machine with a specific state, change the state and check the boolean value given by the method `.last_transition_condition_was`.

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (27)
Start-Time:	2026-05-15 14:45:09,402
Finished-Time:	2026-05-15 14:45:09,404
Time-Consumption	0.002s

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

3.3.6 Previous state (REQ-0014)

Description
The user shall be able to get the previous state of the state machine.

Reason for the implementation
Implement logic depending on the previous state machine's state.

Fitcriterion
Initialise a state machine with a specific state, change the state and check the result of the method `.previous_state`.

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (28)

Start-Time:	2026-05-15 14:45:09,404
Finished-Time:	2026-05-15 14:45:09,405
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 (REQ-0015)

Description

The user shall be able to get the information if the previous state was a specific state (as boolean value).

Reason for the implementation

Implement logic depending on the previous state machine's state.

Fitcriterion

Initialise a state machine with a specific state, change the state and check the result of the method `.previous_state_was`.

Testresult

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (29)
Start-Time:	2026-05-15 14:45:09,406
Finished-Time:	2026-05-15 14:45:09,407
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 <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 (REQ-0016)

Description

The user shall be able to get the information how long the state machine was in the previous state.

Reason for the implementation

Implement logic depending on the state machine's previous state duration.

Fitcriterion

Initialise a state machine with a specific state, wait for a certain time, change ste state and check the result of the

method .previous_state_duration.

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (30)
Start-Time:	2026-05-15 14:45:09,408
Finished-Time:	2026-05-15 14:45:10,160
Time-Consumption	0.752s

Testsummary:	
Info	Running state machine test sequence.
Success	Return Value of previous_state_duration() is correct (Content 0.7509286403656006 in [0.7 ... 0.8] and Type is <class 'float'>).

3.4 Callbacks

3.4.1 Callback for a specific transition to a specific target state (REQ-0001)

Description
The user shall be able to register a callback, which is executed with the given arguments and keyword arguments, if the specified state was reached by the specified transition condition.

Reason for the implementation
Trigger actions depending on the state and transition condition.

Fitcriterion
Run through a state change sequence and identify the correct callback execution (e.g. by a sequence counter).

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (33)
Start-Time:	2026-05-15 14:45:10,160
Finished-Time:	2026-05-15 14:45:10,163
Time-Consumption	0.003s

Testsummary:	
Info	Running state machine sequence a(0)→b(1)→a(2)→b(3)→c(4) by conditions a(1), b,(2) b(3), c(4) and incrementing sequence counter before every transition. Registered callback for state b and condition a.
Success	Sequence list is correct (Content [1] and Type is <class 'list'>).

3.4.2 Callback for a specific transition (only) (REQ-0002)

Description
The user shall be able to register a callback, which is executed with the given arguments and keyword arguments, if the state change was reached by the specified transition condition.

Reason for the implementation
Trigger actions depending on the transition condition.

Fitcriterion
Run through a state change sequence and identify the correct callback execution (e.g. by a sequence counter).

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (34)
Start-Time:	2026-05-15 14:45:10,164
Finished-Time:	2026-05-15 14:45:10,166
Time-Consumption	0.003s

Testsummary:	
Info	Running state machine sequence a(0)→b(1)→a(2)→b(3)→c(4) by conditions a(1), b,(2) b(3), c(4) and incrementing sequence counter before every transition. Registered callback for condition b.
Success	Sequence list is correct (Content [2, 3] and Type is <class 'list'>).

3.4.3 Callback for a specific target state (only) (REQ-0003)

Description
The user shall be able to register a callback, which is executed with the given arguments and keyword arguments, if the specified state was reached.

Reason for the implementation
Trigger actions depending on the state.

Fitcriterion
Run through a state change sequence and identify the correct callback execution (e.g. by a sequence counter).

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (35)

Start-Time:	2026-05-15 14:45:10,167
Finished-Time:	2026-05-15 14:45:10,169
Time-Consumption	0.002s

Testsummary:	
Info	Running state machine sequence a(0)→b(1)→a(2)→b(3)→c(4) by conditions a(1), b,(2) b(3), c(4) and incrementing sequence counter before every transition. Registered callback for state a.
Success	Sequence list (state a) is correct (Content [2] and Type is <class 'list'>).

3.4.4 Callback independent from transition and target state (REQ-0004)

Description

The user shall be able to register a callback, which is executed with the given arguments and keyword arguments.

Reason for the implementation

Trigger actions on state changes.

Fitcriterion

Run through a state change sequence and identify the correct callback execution (e.g. by a sequence counter).

Testresult

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (36)
Start-Time:	2026-05-15 14:45:10,170
Finished-Time:	2026-05-15 14:45:10,175
Time-Consumption	0.005s

Testsummary:	
Info	Running state machine sequence a(0)→b(1)→a(2)→b(3)→c(4) by conditions a(1), b,(2) b(3), c(4) and incrementing sequence counter before every transition. Registered callback for every transition.
Success	Sequence list is correct (Content [1, 2, 3, 4] and Type is <class 'list'>).

3.4.5 Multi callback registration (REQ-0021)

Description

The user shall be able to register multiple callback (also with identical state and condtion information).

Reason for the implementation

The user shall be able to register as many callbacks as needed to reach the target functionality.

Fitcriterion
Run through a state change sequence and identify the correct callback execution (e.g. by a sequence counter).

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (37)
Start-Time:	2026-05-15 14:45:10,175
Finished-Time:	2026-05-15 14:45:10,180
Time-Consumption	0.005s

Testsummary:	
Info	Running state machine sequence a(0)→b(1)→a(2)→b(3)→c(4) by conditions a(1), b,(2) b(3), c(4) and incrementing sequence counter before every transition. Registered callback for state b (twice).
Success	Sequence list (state a) is correct (Content [1, 1, 3, 3] and Type is <class 'list'>).

3.4.6 Callback execution order (REQ-0020)

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

Reason for the implementation
The user can control the callback execution order by the registration order.

Fitcriterion
Run through a state change sequence and identify the correct callback execution order.

Testresult
This test was passed with the state: **Success**. See also full trace in section A.1.21!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/tests/__init__.py (38)
Start-Time:	2026-05-15 14:45:10,181
Finished-Time:	2026-05-15 14:45:10,188
Time-Consumption	0.007s

Testsummary:	
Info	Callback registration order: → state_b, → state *, → state_a. Running state machine sequence: a→b→a
Success	Callback execution order: Values and number of submitted values is correct. See detailed log for more information.

A Trace for testrun: Library test

A.1 Tests with status Info (21)

A.1.1 Initialisation with a defined state (REQ-0005)

Description
It shall be possible to initialise the state machine with a defined state.

Reason for the implementation
The user shall be able to define the start condition (state) of the state machine.

Fitcriterion
Initialisation of the state machine with a specified state. Test the state of the state machine.

Testresult
This test was passed with the state: Success.

Info	Initialising the state machine with 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'>)
```

A.1.2 Last transition information after initialisation (REQ-0006)

Description
The last transition method shall return `__init__` after state machine initialisation.

Reason for the implementation
The user shall be able to differentiate between the initial transition to default state and a normal operation transition.

Fitcriterion
Initialisation of the state machine. Check the last transition information.

Testresult
This test was passed with the state: Success.

Info	Initialising the state machine with 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'>)
```

A.1.3 Previous state information after initialisation (REQ-0007)

Description
The previous state method shall return None after state machine initialisation.

Reason for the implementation
The user shall be able to differentiate between the initial previous state and the previous state during normal operation.

Fitcriterion
Initialisation of the state machine. Check the last state information.

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

Info	Initialising the state machine with 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'>)
```

A.1.4 Pass all kind of information inside the state machine instance (REQ-0008)

Description
It shall be possible to store all kind of information as instance attributes to the state machine.

Reason for the implementation
The user shall be able to store data or methods to the state machine attributes, which might be helpful changing the state or determining the change conditions.

Fitcriterion
Initialisation of the state machine with a bunch of data. Check that the data is stored in the state machine instance.

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

Info	Initialising the state machine with state_c

Success	Keyword argument kw_arg_no_1 stored in state_machine is correct (Content 1 and Type is <class 'int'>).

```
Result (Keyword argument kw_arg_no_1 stored in state_machine): 1 (<class 'int'>)
```

```
Expectation (Keyword argument kw_arg_no_1 stored in state_machine): result = 1 (<class 'int'>)
```

Success	Keyword argument kw_arg_no_2 stored in state_machine is correct (Content True and Type is <class 'bool'>).

```
Result (Keyword argument kw_arg_no_2 stored in state_machine): True (<class 'bool'>)
Expectation (Keyword argument kw_arg_no_2 stored in state_machine): result = True (<class
↪  'bool'>)
```

Success	Keyword argument kw_arg_no_3 stored in state_machine is correct (Content {'1': 1, '2': 'two'} and Type is <class 'dict'>).

```
Result (Keyword argument kw_arg_no_3 stored in state_machine): { '1': 1, '2': 'two' } (<class
↪  'dict'>)
Expectation (Keyword argument kw_arg_no_3 stored in state_machine): result = { '1': 1, '2':
↪  'two' } (<class 'dict'>)
```

Success	Keyword argument kw_arg_no_4 stored in state_machine is correct (Content <built-in function print> and Type is <class 'builtin_function_or_method'>).

```
Result (Keyword argument kw_arg_no_4 stored in state_machine): <built-in function print>
↪  (<class 'builtin_function_or_method'>)
Expectation (Keyword argument kw_arg_no_4 stored in state_machine): result = <built-in
↪  function print> (<class 'builtin_function_or_method'>)
```

A.1.5 States and transitions (REQ-0017)

Description

The user shall be able to define states and transitions between the states.

Reason for the implementation

A state machine shall be able to change between states in a user defined way under user defined conditions.

Fitcriterion

Create a simple state machine with some states and conditions. Check the state change path as defined by the user.

Testresult

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

Info	Initialising state machine with 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, defined Transitions (from state_a) are: False→state_b (0.0s); True→state_c (0.0s)

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

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

Info	Work routine executed, defined Transitions (from state_c) are: True→state_b (0.0s); False→state_a (0.0s)

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

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

Info	Work routine executed, no transitions defined from state_b (dead end)

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

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

A.1.6 Transition timing (REQ-0018)

Description

The user shall be able to define a time a transition condition needs to be active till the transition is performed.

Reason for the implementation

In a lot of cases you might face signal noise or slight fluctuations. To be able to not change states back and for, such a timing can help.

Fitcriterion

Create a state machine with active transition condition and a defined transition time. The transition shall take the defined time. The time shall start over, if the condition was False for at least one cycle.

Testresult

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

Info	Initialising state machine with 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	Checking state change every 0.15ms with timeout condition after 0.154s.

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

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

Success	Transition time of 1st change is correct (Content 0.1502072811126709 in [0.148 ... 0.152] and Type is <class 'float'>).

```
Result (Transition time of 1st change): 0.1502072811126709 (<class 'float'>)
Expectation (Transition time of 1st change): 0.148 <= result <= 0.152
```

Info	Setting transition condition to false for one cycle after half of the transition time.

Info	Checking state change every 0.15ms with timeout condition after 0.154s.

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

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

Success	Transition time of 2nd change is correct (Content 0.15016961097717285 in [0.148 ... 0.152] and Type is <class 'float'>).

```
Result (Transition time of 2nd change): 0.15016961097717285 (<class 'float'>)
Expectation (Transition time of 2nd change): 0.148 <= result <= 0.152
```

Success	Previous state duration is correct (Content 0.22746682167053223 in [0.22499999999999998 ... 0.22899999999999998] and Type is <class 'float'>).

```
Result (Previous state duration): 0.22746682167053223 (<class 'float'>)
Expectation (Previous state duration): 0.22499999999999998 <= result <= 0.22899999999999998
```

A.1.7 Transition prioritisation (REQ-0019)

Description

When two conditions and timings result in two state changes, when executing the worker task, the transition which is longer fulfilled shall win.

Reason for the implementation

If the execution of the worker is to infrequent, the state change shall act like the states haven't been changed, but the execution was directly after the first active state change.

Fitcriterion
Define a state machine with two active state changes, but different transition times. Execute the worker, after both transition times are fulfilled. The resulting state shall be according the first fulfilled transition.

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

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 0.061s
Executing method work after 0.121s
Executing method work after 0.182s
```

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'>)
```

A.1.8 This state (REQ-0009)

Description
The user shall be able to get the current state of the state machine.

Reason for the implementation
Implement logic depending on the state machine's state.

Fitcriterion
Initialise a state machine with a specific state and check the result of the method `.this_state`.

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

Info	Initialising the state machine with 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'>)
```

A.1.9 This state is (REQ-0010)

Description
The user shall be able to get the information if the state machine is in a specific state (as boolean value).

Reason for the implementation
Implement logic depending on the state machine's state.

Fitcriterion
Initialise a state machine with a specific state and check the result of the method `.this_state_is`.

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

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'>).

```
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'>)
```

A.1.10 State duration (REQ-0011)

Description
The user shall be able to get the information how long the state machine is in the current state.

Reason for the implementation
Implement logic depending on the state machine's current state duration.

Fitcriterion
Initialise a state machine with a specific state, wait for a certain time, change the state and check the result of the method `.state_duration`.

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

Info	Running state machine test sequence.

```
Waiting for 0.25s
```

Success	Return Value of this_state_duration() is correct (Content 0.2510814666748047 in [0.2 ... 0.3] and Type is <class 'float'>).

```
Result (Return Value of this_state_duration()): 0.2510814666748047 (<class 'float'>)
Expectation (Return Value of this_state_duration()): 0.2 <= result <= 0.3
```

A.1.11 Last transition condition (REQ-0012)

Description
The user shall be able to get the information, which transition condition changed the state to the current state.

Reason for the implementation
Implement logic depending on the last transition condition.

Fitcriterion
Initialise a state machine with a specific state, change the state and check the transition condition information given by the method `.last_transition_condition`.

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

Info	Running state machine test sequence.

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'>)
```

A.1.12 Last transition condition was (REQ-0013)

Description
The user shall be able to get the information, if the last transition condition was the specified one (boolean value).

Reason for the implementation
Implement logic depending on the last transition condition.

Fitcriterion
Initialise a state machine with a specific state, change the state and check the boolean value given by the method `.last_transition_condition_was`.

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

Info	Running state machine test sequence.

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'>)
```

A.1.13 Previous state (REQ-0014)

Description
The user shall be able to get the previous state of the state machine.

Reason for the implementation
Implement logic depending on the previous state machine's state.

Fitcriterion
Initialise a state machine with a specific state, change the state and check the result of the method `.previous_state`.

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

Info	Running state machine test sequence.

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'>)
```

A.1.14 Previous state was (REQ-0015)

Description
The user shall be able to get the information if the previous state was a specific state (as boolean value).

Reason for the implementation
Implement logic depending on the previous state machine's state.

Fitcriterion
Initialise a state machine with a specific state, change the state and check the result of the method `.previous_state_was`.

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

Info	Running state machine test sequence.

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'>)
```

A.1.15 Previous state duration (REQ-0016)

Description
The user shall be able to get the information how long the state machine was in the previous state.

Reason for the implementation
Implement logic depending on the state machine's previous state duration.

Fitcriterion
Initialise a state machine with a specific state, wait for a certain time, change ste state and check the result of the method `.previous_state_duration`.

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

Info	Running state machine test sequence.

```
Waiting for 0.75s
```

Success	Return Value of previous_state_duration() is correct (Content 0.7509286403656006 in [0.7 ... 0.8] and Type is <class 'float'>).

```
Result (Return Value of previous_state_duration()): 0.7509286403656006 (<class 'float'>)
Expectation (Return Value of previous_state_duration()): 0.7 <= result <= 0.8
```

A.1.16 Callback for a specific transition to a specific target state (REQ-0001)

Description
The user shall be able to register a callback, which is executed with the given arguments and keyword arguments, if the specified state was reached by the specified transition condition.

Reason for the implementation
Trigger actions depending on the state and transition condition.

Fitcriterion
Run through a state change sequence and identify the correct callback execution (e.g. by a sequence counter).

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

Info	Running state machine sequence a(0)→b(1)→a(2)→b(3)→c(4) by conditions a(1), b,(2) b(3), c(4) and incrementing sequence counter before every transition. Registered callback for state b and condition a.

```
Increasing state sequence number to 1
Storing sequence counter counter 1 (current state: state_b, last condition: condition_a)
Increasing state sequence number to 2
Increasing state sequence number to 3
Increasing state sequence number to 4
```

Success	Sequence list is correct (Content [1] and Type is <class 'list'>).

```
Result (Sequence list): [ 1 ] (<class 'list'>)
Expectation (Sequence list): result = [ 1 ] (<class 'list'>)
```

A.1.17 Callback for a specific transition (only) (REQ-0002)

Description
The user shall be able to register a callback, which is executed with the given arguments and keyword arguments, if the state change was reached by the specified transition condition.

Reason for the implementation
Trigger actions depending on the transition condition.

Fitcriterion
Run through a state change sequence and identify the correct callback execution (e.g. by a sequence counter).

Testresult
This test was passed with the state: Success.

Info	Running state machine sequence a(0)→b(1)→a(2)→b(3)→c(4) by conditions a(1), b,(2) b(3), c(4) and incrementing sequence counter before every transition. Registered callback for condition b.

```
Increasing state sequence number to 1
Increasing state sequence number to 2
Storing sequence counter counter 2 (current state: state_a, last condition: condition_b)
Increasing state sequence number to 3
Storing sequence counter counter 3 (current state: state_b, last condition: condition_b)
Increasing state sequence number to 4
```

Success	Sequence list is correct (Content [2, 3] and Type is <class 'list'>).

```
Result (Sequence list): [ 2, 3 ] (<class 'list'>)
Expectation (Sequence list): result = [ 2, 3 ] (<class 'list'>)
```

A.1.18 Callback for a specific target state (only) (REQ-0003)

Description
The user shall be able to register a callback, which is executed with the given arguments and keyword arguments, if the specified state was reached.

Reason for the implementation
Trigger actions depending on the state.

Fitcriterion
Run through a state change sequence and identify the correct callback execution (e.g. by a sequence counter).

Testresult
This test was passed with the state: Success.

Info	Running state machine sequence a(0)→b(1)→a(2)→b(3)→c(4) by conditions a(1), b,(2) b(3), c(4) and incrementing sequence counter before every transition. Registered callback for state a.

```
Increasing state sequence number to 1
Increasing state sequence number to 2
```

```
Storing sequence counter counter 2 (current state: state_a, last condition: condition_b)
Increasing state sequence number to 3
Increasing state sequence number to 4
```

Success	Sequence list (state a) is correct (Content [2] and Type is <class 'list'>).

```
Result (Sequence list (state a)): [ 2 ] (<class 'list'>)
Expectation (Sequence list (state a)): result = [ 2 ] (<class 'list'>)
```

A.1.19 Callback independent from transition and target state (REQ-0004)

Description

The user shall be able to register a callback, which is executed with the given arguments and keyword arguments.

Reason for the implementation

Trigger actions on state changes.

Fitcriterion

Run through a state change sequence and identify the correct callback execution (e.g. by a sequence counter).

Testresult

This test was passed with the state: Success.

Info	Running state machine sequence a(0)→b(1)→a(2)→b(3)→c(4) by conditions a(1), b,(2) b(3), c(4) and incrementing sequence counter before every transition. Registered callback for every transition.

```
Increasing state sequence number to 1
Storing sequence counter counter 1 (current state: state_b, last condition: condition_a)
Increasing state sequence number to 2
Storing sequence counter counter 2 (current state: state_a, last condition: condition_b)
Increasing state sequence number to 3
Storing sequence counter counter 3 (current state: state_b, last condition: condition_b)
Increasing state sequence number to 4
Storing sequence counter counter 4 (current state: state_c, last condition: condition_c)
```

Success	Sequence list is correct (Content [1, 2, 3, 4] and Type is <class 'list'>).

```
Result (Sequence list): [ 1, 2, 3, 4 ] (<class 'list'>)
Expectation (Sequence list): result = [ 1, 2, 3, 4 ] (<class 'list'>)
```

A.1.20 Multi callback registration (REQ-0021)

Description

The user shall be able to register multiple callback (also with identical state and condtion information).

Reason for the implementation
The user shall be able to register as many callbacks as needed to reach the target functionality.

Fitcriterion
Run through a state change sequence and identify the correct callback execution (e.g. by a sequence counter).

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

Info	Running state machine sequence a(0)→b(1)→a(2)→b(3)→c(4) by conditions a(1), b,(2) b(3), c(4) and incrementing sequence counter before every transition. Registered callback for state b (twice).

```
Increasing state sequence number to 1
Storing sequence counter counter 1 (current state: state_b, last condition: condition_a)
Storing sequence counter counter 1 (current state: state_b, last condition: condition_a)
Increasing state sequence number to 2
Increasing state sequence number to 3
Storing sequence counter counter 3 (current state: state_b, last condition: condition_b)
Storing sequence counter counter 3 (current state: state_b, last condition: condition_b)
Increasing state sequence number to 4
```

Success	Sequence list (state a) is correct (Content [1, 1, 3, 3] and Type is <class 'list'>).

```
Result (Sequence list (state a)): [ 1, 1, 3, 3 ] (<class 'list'>)
Expectation (Sequence list (state a)): result = [ 1, 1, 3, 3 ] (<class 'list'>)
```

A.1.21 Callback execution order (REQ-0020)

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

Reason for the implementation
The user can control the callback execution order by the registration order.

Fitcriterion
Run through a state change sequence and identify the correct callback execution order.

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

Info	Callback registration order: → state_b, → state *, → state_a. Running state machine sequence: a→b→a

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

```
Result (Callback execution order): [ '-> state_b', '-> state *', '-> state *', '-> state_a' ]
↪  (<class 'list'>)
Expectation (Callback execution order): result = [ '-> state_b', '-> state *', '-> state *',
↪  '-> state_a' ] (<class 'list'>)
Result (Submitted value number 1): '-> state_b' (<class 'str'>)
Expectation (Submitted value number 1): result = '-> state_b' (<class 'str'>)
Submitted value number 1 is correct (Content '-> state_b' and Type is <class 'str'>).
Result (Submitted value number 2): '-> state *' (<class 'str'>)
Expectation (Submitted value number 2): result = '-> state *' (<class 'str'>)
Submitted value number 2 is correct (Content '-> state *' and Type is <class 'str'>).
Result (Submitted value number 3): '-> state *' (<class 'str'>)
Expectation (Submitted value number 3): result = '-> state *' (<class 'str'>)
Submitted value number 3 is correct (Content '-> state *' and Type is <class 'str'>).
Result (Submitted value number 4): '-> state_a' (<class 'str'>)
Expectation (Submitted value number 4): result = '-> state_a' (<class 'str'>)
Submitted value number 4 is correct (Content '-> state_a' and Type is <class 'str'>).
```

B Test-Coverage

B.1 state_machine

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

B.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%

```
"""
state_machine (State Machine)
=============================

**Author:**

* Dirk Alders <sudo-dirk@mount-mockery.de>

**Description:**

    This Module helps implementing state machines.

**Submodules:**

* :class:`state_machine.state_machine`

**Unittest:**

    See also the :download:`unittest <state_machine/_testresults_/unittest.pdf>` documentation.

**Module Documentation:**

"""

```

```
__VERSION__ = "1.0.4"
__DEPENDENCIES__ = ["Python standard"]

import logging
import time

try:
    from config import APP_NAME as ROOT_LOGGER_NAME
except ImportError:
    ROOT_LOGGER_NAME = "root"
logger = logging.getLogger(ROOT_LOGGER_NAME).getChild(__name__)


__DESCRIPTION__ = """This Module helps implementing state machines."""
"""The Module description"""


class state_machine(object):
    """
    :param default_state: The default state which is set on initialisation.
    :param log_lvl: The log level, this Module logs to (see Loging-Levels of Module :mod:`logging`)

    .. note:: Additional keyword parameters well be stored as varibles of the instance (e.g. to give variables or methods for transition condition calculation).

    A state machine class can be created by deriving it from this class. The transitions are defined by overriding the variable `TRANSITIONS`.
    This Variable is a dictionary, where the key is the start-state and the content is a tuple or list of transitions. Each transition is a tuple or list
    including the following information: (condition-method (str), transition-time (number), target_state (str)).

    .. note:: The condition-method needs to be implemented as part of the new class.

    .. note:: It is usefull to define the states as variables of this class.


    **Example:**

    .. literalinclude:: state_machine/_examples_/example.py

    .. literalinclude:: state_machine/_examples_/example.log
    """

    TRANSITIONS = {}

    def __init__(self, default_state, log_lvl, **kwargs):
        self.__state__ = None
        self.__last_transition_condition__ = None
        self.__conditions_start_time__ = {}
        self.__state_change_callbacks__ = {}
        self.__log_lvl__ = log_lvl
        self.__set_state__(default_state, "__init__")
        self.__callback_id__ = 0
        for key in kwargs:
            setattr(self, key, kwargs.get(key))

    def register_state_change_callback(self, state, condition, callback, *args, **kwargs):
```

```
79          """
80          :param state: The target state. The callback will be executed, if the state machine changes to this state. None means all states.
81          :type state: str
82          :param condition: The transition condition. The callback will be executed, if this condition is responsible for the state change. None means all conditions.
83          :type condition: str
84          :param callback: The callback to be executed.
85
86          .. note:: Additional arguments and keyword parameters are supported. These arguments and parameters will be used as arguments and parameters for the callback execution.
87
88          This methods allows to register callbacks which will be executed on state changes.
89          """
90          if state not in self.__state_change_callbacks__:
91              self.__state_change_callbacks__[state] = {}
92          if condition not in self.__state_change_callbacks__[state]:
93              self.__state_change_callbacks__[state][condition] = []
94          self.__state_change_callbacks__[state][condition].append((self.__callback_id__, callback, args, kwargs))
95          self.__callback_id__ += 1
96
97      def this_state(self):
98          """
99          :return: The current state.
100
101         This method returns the current state of the state machine.
102         """
103         return self.__state__
104
105     def this_state_is(self, state):
106         """
107         :param state: The state to be checked
108         :type state: str
109         :return: True if the given state is currently active, else False.
110         :rtype: bool
111
112         This methods returns the boolean information if the state machine is currently in the given state.
113         """
114         return self.__state__ == state
115
116     def this_state_duration(self):
117         """
118         :return: The time how long the current state is active.
119         :rtype: float
120
121         This method returns the time how long the current state is active.
122         """
123         return time.time() - self.__time_stamp_state_change__
124
125     def last_transition_condition(self):
126         """
127         :return: The last transition condition.
128         :rtype: str
129
130         This method returns the last transition condition.
131         """
132         return self.__last_transition_condition__
133
134     def last_transition_condition_was(self, condition):
```

```
135         """
136         :param condition: The condition to be checked
137         :type condition: str
138         :return: True if the given condition was the last transition condition, else False.
139         :rtype: bool
140 
141         This methods returns the boolean information if the last transition condition is equivalent to the given condition.
142         """
143         return self.__last_transition_condition__ == condition
144 
145     def previous_state(self):
146         """
147         :return: The previous state.
148         :rtype: str
149 
150         This method returns the previous state of the state machine.
151         """
152         return self.__prev_state__
153 
154     def previous_state_was(self, state):
155         """
156         :param state: The state to be checked
157         :type state: str
158         :return: True if the given state was previously active, else False.
159         :rtype: bool
160 
161         This methods returns the boolean information if the state machine was previously in the given state.
162         """
163         return self.__prev_state__ == state
164 
165     def previous_state_duration(self):
166         """
167         :return: The time how long the previous state was active.
168         :rtype: float
169 
170         This method returns the time how long the previous state was active.
171         """
172         return self.__prev_state_dt__
173 
174     def __set_state__(self, target_state, condition):
175         logger.log(self.__log_lvl__, "State change (%s): %s -> %s", repr(condition), repr(self.__state__), repr(target_state))
176         timestamp = time.time()
177         self.__prev_state__ = self.__state__
178         if self.__prev_state__ is None:
179             self.__prev_state_dt__ = 0.0
180         else:
181             self.__prev_state_dt__ = timestamp - self.__time_stamp_state_change__
182         self.__state__ = target_state
183         self.__last_transition_condition__ = condition
184         self.__time_stamp_state_change__ = timestamp
185         self.__conditions_start_time__ = {}
186         # Callback collect
187         this_state_change_callbacks = []
188         this_state_change_callbacks.extend(self.__state_change_callbacks__.get(None, {}).get(None, []))
189         this_state_change_callbacks.extend(self.__state_change_callbacks__.get(target_state, {}).get(None, []))
190         this_state_change_callbacks.extend(self.__state_change_callbacks__.get(None, {}).get(condition, []))
191         this_state_change_callbacks.extend(self.__state_change_callbacks__.get(target_state, {}).get(condition, []))
```

```
192         # Callback sorting
193         this_state_change_callbacks.sort()
194         # Callback execution
195         for cid, callback, args, kwargs in this_state_change_callbacks:
196             argkwarg_str = "(" + ", ".join([repr(a) for a in args]) + ") - {" + ", ".join([k + ": " + repr(kwargs[k]) for k in kwargs]) + "}"
197             logger.debug("Executing callback %d - %s.%s - %s", cid, callback.__module__, callback.__name__, argkwarg_str[:75])
198             callback(*args, **kwargs)
199 
200     def work(self):
201         """
202         This Method needs to be executed cyclically to enable the state machine.
203         """
204         tm = time.time()
205         transitions = self.TRANSITIONS.get(self.this_state())
206         if transitions is not None:
207             active_transitions = []
208             cnt = 0
209             for method_name, transition_delay, target_state in transitions:
210                 method = getattr(self, method_name)
211                 if method():
212                     if method_name not in self.__conditions_start_time__:
213                         self.__conditions_start_time__[method_name] = tm
214                     if tm - self.__conditions_start_time__[method_name] >= transition_delay:
215                         active_transitions.append(
216                             (transition_delay - tm + self.__conditions_start_time__[method_name], cnt, target_state, method_name)
217                         )
218                 else:
219                     if method_name in self.__conditions_start_time__:
220                         logger.debug("Resetting transition time from %fs to 0 for %s", tm - self.__conditions_start_time__[method_name], method_name)
221                         self.__conditions_start_time__.pop(method_name)
222                 cnt += 1
223             if len(active_transitions) > 0:
224                 active_transitions.sort()
225                 self.__set_state__(active_transitions[0][2], active_transitions[0][3])
```