Unittest for state_machine

August 14, 2025

Contents

1 Test Information

1.1 Test Candidate Information

This Module helps implementing state machines.

Library Information	
Name	state_machine
State	Released
Supported Interpreters	python3
Version	6ba47253e81e9a0edcd7690c51c05d3d
Dependencies	

1.2 Unittest Information

Unittest Information	
Version	e6f5e3b6cb9ae84eee10254379ddd104
Testruns with	python 3.13.5 (final)

1.3 Test System Information

System Information	
Architecture	64bit
Distribution	Debian GNU/Linux 13 trixie
Hostname	ahorn
Kernel	6.12.38+deb13-amd64 (#1 SMP PREEMPT_DYNAMIC Debian 6.12.38-1 (2025-07-16))
Machine	x86_64
Path	/home/dirk/work/unittest_collection/state_machine
System	Linux
Username	dirk

2 Statistic

2.1 Test-Statistic for testrun with python 3.13.5 (final)

Number of tests	**20**
Number of successfull tests	**20**
Number of possibly failed tests	**0**
Number of failed tests	**0**
Executionlevel	Full Test (all defined tests)
Time consumption	1.651s

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

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 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:18,453
Finished-Time:	2025-08-14 22:47:18,454
Time-Consumption	0.000s
Testsummary:	
Info	Initialising the state machine with state_c
Success	State after initialisation is correct (Content 'state_c' and Type is <class 'str'>).

3.1.2 Default Last Transition Condtion

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

Reason for the implementation
Creation of a defined state after initialisation.

Fitcriterion
The last transition condition is `__init__` after initialisation.

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

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)

Start-Time:	2025-08-14 22:47:18,454
Finished-Time:	2025-08-14 22:47:18,454
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 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**. See also full trace in section A.1.3!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:18,454
Finished-Time:	2025-08-14 22:47:18,454
Time-Consumption	0.000s
Testsummary:	
Info	Initialising the state machine with state_c
Success	Last state after initialisation is correct (Content None and Type is <class 'NoneType'>).

3.1.4 Additional Keyword Arguments

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

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

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

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

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:18,454
Finished-Time:	2025-08-14 22:47:18,455
Time-Consumption	0.001s
Testsummary:	
Info	Initialising the state machine with state_c
Success	Keyword argument kw_arg_no_1 stored in state_machine is correct (Content 1 and Type is <class 'int'>).
Success	Keyword argument kw_arg_no_2 stored in state_machine is correct (Content '2' and Type is <class 'str'>).
Success	Keyword argument kw_arg_no_3 stored in state_machine is correct (Content True and Type is <class 'bool'>).
Success	Keyword argument kw_arg_no_4 stored in state_machine is correct (Content {'1': 1, '2': 'two'} and Type is <class 'dict'>).

3.2 Transition Changes

3.2.1 Transitiondefinition and -flow

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

Reason for the implementation
Definition of the transitions for a state machine.

Fitcriterion
The order of at least three state changes is correct.

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

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:18,455
Finished-Time:	2025-08-14 22:47:18,456
Time-Consumption	0.001s
Testsummary:	
Info	Initialising state machine with state_a
Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <class 'str'>).

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

3.2.2 Transitiontiming

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

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

Fitcriterion
The transition time and the restart of the transion timer by setting the transition condition to `False` and to `True` again results in the expected transition timing (±0.05s).

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

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:18,456
Finished-Time:	2025-08-14 22:47:18,833
Time-Consumption	0.377s

Testsummary:

Info	Initialising state machine with state_a
Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <class 'str'>).
Info	Waiting for 0.160s or state change
Success	State after 1st cycle is correct (Content 'state_b' and Type is <class 'str'>).
Success	Transition time after 1st cycle is correct (Content 0.1504039764404297 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 0.1502220630645752 in [0.145 ... 0.155] and Type is <class 'float'>).
Success	Previous state duration is correct (Content 0.22541022300720215 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 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:18,833
Finished-Time:	2025-08-14 22:47:19,078
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 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 returend state fits to the expecation.

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

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:19,078
Finished-Time:	2025-08-14 22:47:19,080
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 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:19,080
Finished-Time:	2025-08-14 22:47:19,082
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 (± 0.05s).

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

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:19,082
Finished-Time:	2025-08-14 22:47:19,334
Time-Consumption	0.252s
Testsummary:	
Info	Running state machine test sequence.
Success	Return Value of this_state_duration() is correct (Content 0.25122594833374023 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 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:19,335
Finished-Time:	2025-08-14 22:47:19,336
Time-Consumption	0.002s
Testsummary:	
Info	Running state machine test sequence.
Success	Returnvalue of last_transition_condition() is correct (Content 'condition_a' and Type is <class 'str'>).

3.3.5 Last Transition Condition was

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

Reason for the implementation
Comfortable user interface.

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

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

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:19,337
Finished-Time:	2025-08-14 22:47:19,339
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

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

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least one returend state fits to the expecation.

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

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:19,340

Finished-Time:	2025-08-14 22:47:19,341
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 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:19,342
Finished-Time:	2025-08-14 22:47:19,344
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

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 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:19,345
Finished-Time:	2025-08-14 22:47:20,097
Time-Consumption	0.753s
Testsummary:	
Info	Running state machine test sequence.
Success	Return Value of previous_state_duration() is correct (Content 0.7510614395141602 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 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:20,098
Finished-Time:	2025-08-14 22:47:20,102
Time-Consumption	0.004s
Testsummary:	
Info	Running state machine sequence and storing sequence number for each callback
Success	Execution of state machine callback (1) (state_b, condition_a) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.
Success	Execution of state machine callback (2) (state_b, condition_a) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

3.4.2 State change callback for a defined transition

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

Reason for the implementation
Triggering state change actions for a specific transition condition.

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

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

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:20,102
Finished-Time:	2025-08-14 22:47:20,104
Time-Consumption	0.002s
Testsummary:	
Info	Running state machine sequence and storing sequence number for each callback
Success	Execution of state machine callback (1) (all_transitions, condition_b) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.
Success	Execution of state machine callback (2) (all_transitions, condition_b) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

3.4.3 State change callback for a defined targetstate

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

Reason for the implementation
Triggering state change actions for a specific targetstate.

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

Testresult

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

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:20,104
Finished-Time:	2025-08-14 22:47:20,106
Time-Consumption	0.002s
Testsummary:	
Info	Running state machine sequence and storing sequence number for each callback
Success	Execution of state machine callback (1) (state_b, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.
Success	Execution of state machine callback (2) (state_b, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

3.4.4 State change callback for all kind of state changes

Description

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

Reason for the implementation

Triggering state change actions for all transition conditions and targetstates.

Fitcriterion

Methods are called in the registration order after state change.

Testresult

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

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:20,107
Finished-Time:	2025-08-14 22:47:20,109
Time-Consumption	0.003s
Testsummary:	
Info	Running state machine sequence and storing sequence number for each callback
Success	Execution of state machine callback (1) (all_transitions, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.
Success	Execution of state machine callback (2) (all_transitions, all_conditions) identified by a sequence number: Values and number of submitted values is correct. See detailed log for more information.

3.4.5 Execution order of Callbacks

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

Reason for the implementation
User shall have the control about the execution order.

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

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

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/state_machine/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-14 22:47:20,110
Finished-Time:	2025-08-14 22:47:20,111
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 3.13.5 (final)

A.1 Tests with status Info (20)

A.1.1 REQ-0005

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'>)
```

A.1.2 REQ-0006

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'>)
```

A.1.3 REQ-0007

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'>)
```

A.1.4 REQ-0008

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

Info	Initialising the state machine with state_c

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

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

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

Success	Keyword argument kw_arg_no_2 stored in state_machine is correct (Content '2' and Type is <class 'str'>).

```
Result (Keyword argument kw_arg_no_2 stored in state_machine): '2' (<class 'str'>)
Expectation (Keyword argument kw_arg_no_2 stored in state_machine): result = '2' (<class
↪  'str'>)
```

Success	Keyword argument kw_arg_no_3 stored in state_machine is correct (Content True and Type is <class 'bool'>).

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

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

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

A.1.5 REQ-0017

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'>)
```

A.1.6 REQ-0018

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 0.1504039764404297 in [0.145 ... 0.155] and Type is <class 'float'>).

```
Result (Transition time after 1st cycle): 0.1504039764404297 (<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 0.1502220630645752 in [0.145 ... 0.155] and Type is <class 'float'>).

```
Result (Transition time after 2nd cycle): 0.1502220630645752 (<class 'float'>)
Expectation (Transition time after 2nd cycle): 0.145 <= result <= 0.155
```

Success Previous state duration is correct (Content 0.22541022300720215 in [0.21999999999999997 ... 0.22999999999999998] and Type is <class 'float'>).

```
Result (Previous state duration): 0.22541022300720215 (<class 'float'>)
Expectation (Previous state duration): 0.21999999999999997 <= result <= 0.22999999999999998
```

A.1.7 REQ-0019

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 0.182s
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'>)
```

A.1.8 REQ-0009

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'>)
```

A.1.9 REQ-0010

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'>)
```

A.1.10 REQ-0011

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 0.25122594833374023 in [0.2 ... 0.3] and Type is <class 'float'>).

```
Result (Return Value of this_state_duration()): 0.25122594833374023 (<class 'float'>)
Expectation (Return Value of this_state_duration()): 0.2 <= result <= 0.3
```

A.1.11 REQ-0012

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'>)
```

A.1.12 REQ-0013

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'>)
```

A.1.13 REQ-0014

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'>)
```

A.1.14 REQ-0015

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'>)
```

A.1.15 REQ-0016

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 0.7510614395141602 in [0.7 ... 0.8] and Type is <class 'float'>).

```
Result (Return Value of previous_state_duration()): 0.7510614395141602 (<class 'float'>)
Expectation (Return Value of previous_state_duration()): 0.7 <= result <= 0.8
```

A.1.16 REQ-0001

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'>).
```

A.1.17 REQ-0002

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'>).
```

A.1.18 REQ-0003

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'>).
```

A.1.19 REQ-0004

Testresult

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

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

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

```
Increasing sequence number to 3 caused by callback_execution
Increasing sequence number to 4 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
Executing callback 0 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 5 caused by callback_execution
Executing callback 1 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 6 caused by callback_execution
Increasing sequence number to 7 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_a' -> 'state_b'
Executing callback 0 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 8 caused by callback_execution
Executing callback 1 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 9 caused by callback_execution
Increasing sequence number to 10 caused by sequence progress
StateMachine: State change ('condition_c'): 'state_b' -> 'state_c'
Executing callback 0 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 11 caused by callback_execution
Executing callback 1 - tests.test_callbacks.exec_with_counter
Increasing sequence number to 12 caused by callback_execution
```

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

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

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

```
Result (Execution of state machine callback (2) (all_transitions, all_conditions) identified
↪  by a sequence number): [ 2, 5, 8, 11 ] (<class 'list'>)
```

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

A.1.20 REQ-0020

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'>).
```

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%

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
"""
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:**

"""
__DEPENDENCIES__ = []

```

```python
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__)


__INTERPRETER__ = (3, )
"""The supported Interpreter-Versions"""
__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 = {}
    LOG_PREFIX = 'StateMachine:'

    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):
        """
        :param state: The target state. The callback will be executed, if the state machine changes to this state. None means all states.
        :type state: str
        :param condition: The transition condition. The callback will be executed, if this condition is responsible for the state change. None means all conditions.
        :type condition: str
        :param callback: The callback to be executed.

        .. note:: Additional arguments and keyword parameters are supported. These arguments and parameters will be used as arguments and parameters for the callback execution.

        This methods allows to register callbacks which will be executed on state changes.
        """
        if state not in self.__state_change_callbacks__:
            self.__state_change_callbacks__[state] = {}
        if condition not in self.__state_change_callbacks__[state]:
            self.__state_change_callbacks__[state][condition] = []
        self.__state_change_callbacks__[state][condition].append((self.__callback_id__, callback, args, kwargs))
        self.__callback_id__ += 1

    def this_state(self):
        """
        :return: The current state.

        This method returns the current state of the state machine.
        """
        return self.__state__

    def this_state_is(self, state):
        """
        :param state: The state to be checked
        :type state: str
        :return: True if the given state is currently active, else False.
        :rtype: bool

        This methods returns the boolean information if the state machine is currently in the given state.
        """
        return self.__state__ == state

    def this_state_duration(self):
        """
        :return: The time how long the current state is active.
        :rtype: float

        This method returns the time how long the current state is active.
        """
        return time.time() - self.__time_stamp_state_change__

    def last_transition_condition(self):
        """
        :return: The last transition condition.
        :rtype: str

        This method returns the last transition condition.
        """
        return self.__last_transition_condition__

    def last_transition_condition_was(self, condition):
        """
        :param condition: The condition to be checked
        :type condition: str
        :return: True if the given condition was the last transition condition, else False.
        :rtype: bool

        This methods returns the boolean information if the last transition condition is equivalent to the given condition.
        """
        return self.__last_transition_condition__ == condition

    def previous_state(self):
        """
        :return: The previous state.
        :rtype: str

        This method returns the previous state of the state machine.
        """
        return self.__prev_state__

    def previous_state_was(self, state):
        """
        :param state: The state to be checked
        :type state: str
        :return: True if the given state was previously active, else False.
        :rtype: bool

        This methods returns the boolean information if the state machine was previously in the given state.
        """
        return self.__prev_state__ == state

    def previous_state_duration(self):
        """
        :return: The time how long the previous state was active.
        :rtype: float

        This method returns the time how long the previous state was active.
        """
        return self.__prev_state_dt__

    def __set_state__(self, target_state, condition):
        logger.log(self.__log_lvl__, "%s State change (%s): %s -> %s", self.LOG_PREFIX, repr(condition), repr(self.__state__), repr(target_state))
        timestamp = time.time()
        self.__prev_state__ = self.__state__
        if self.__prev_state__ is None:
            self.__prev_state_dt__ = 0.
        else:
            self.__prev_state_dt__ = timestamp - self.__time_stamp_state_change__
        self.__state__ = target_state
        self.__last_transition_condition__ = condition
        self.__time_stamp_state_change__ = timestamp
        self.__conditions_start_time__ = {}
        # Callback collect
        this_state_change_callbacks = []
        this_state_change_callbacks.extend(self.__state_change_callbacks__.get(None, {}).get(None, []))
        this_state_change_callbacks.extend(self.__state_change_callbacks__.get(target_state, {}).get(None, []))
        this_state_change_callbacks.extend(self.__state_change_callbacks__.get(None, {}).get(condition, []))
        this_state_change_callbacks.extend(self.__state_change_callbacks__.get(target_state, {}).get(condition, []))
```

```python
        # Callback sorting
        this_state_change_callbacks.sort()
        # Callback execution
        for cid, callback, args, kwargs in this_state_change_callbacks:
            logger.debug('Executing callback %d - %s.%s', cid, callback.__module__, callback.__name__)
            callback(*args, **kwargs)

    def work(self):
        """
        This Method needs to be executed cyclicly to enable the state machine.
        """
        tm = time.time()
        transitions = self.TRANSITIONS.get(self.this_state())
        if transitions is not None:
            active_transitions = []
            cnt = 0
            for method_name, transition_delay, target_state in transitions:
                method = getattr(self, method_name)
                if method():
                    if method_name not in self.__conditions_start_time__:
                        self.__conditions_start_time__[method_name] = tm
                    if tm - self.__conditions_start_time__[method_name] >= transition_delay:
                        active_transitions.append((transition_delay - tm + self.__conditions_start_time__[method_name], cnt, target_state, method_name))
                else:
                    self.__conditions_start_time__[method_name] = tm
                cnt += 1
            if len(active_transitions) > 0:
                active_transitions.sort()
                self.__set_state__(active_transitions[0][2], active_transitions[0][3])
```