class osbk_devices.sensor_base.**SensorBase**(*name: str*, *read_interval: float*, *msg_interface: MsgType = <class 'osbk_interfaces.msg._osbk_float_value.OSBKFloa*

Abstract base class for sensor implementations.

Nodes that implement a specific sensor should inherit from this base class and overwrite the `read_sensor()` function. This Class also inherits from Node.

Parameters

- **read_interval** (*ROS parameter*) – ROS parameter to change the poll-rate for sensor readings
- **publish_topic** (*str*) – topic name, the sensors readings are published to, defaults to `'[node_name]/value'` test
- **publish_service_name** (*str*) – name of the service to publish sensor readings
- **msg_interface** (*MsgType*) – the msg-interface this sensor uses to publish its readings
- **publisher** (*Publisher*) – ROS publisher for sending the readings
- **publish_service** (*Service*) – the service to request a publish of sensor readings
- **publish_timer** (*Timer*) – ROS timer to schedule publishing of sensor-readings

publish_reading() → None

Publish what `read_sensor()` returns.

Return type None

read_sensor() → MsgType

Abstract method that returns a sensor reading.

This should be overridden for specific hardware implementation.

Returns an instance of the MsgType specified in `self.msg_interface`

class osbk_devices.actuator_base.**ActuatorBase**(*name: str*, *continuous: bool = True*, *status_poll_interval: int = 1*)

Abstract base class for actuator implementations.

Nodes that implement a specific actuator should inherit from this base class and overwrite the `set_actuator()` and `poll_status()` function. This class also inherits

from Node.

Parameters

- **service_name** (*str*) – name of the service this actuator can be controlled with
- **srv** (*Service*) – the service registered with ROS, its callback function calls set_actuator()
- **poll_timer** (*Timer*) – the timer to trigger publishing of current status
- **publisher** (*Publisher*) – a publisher to publish current actuator status periodically

poll_status() → MsgType
Abstract method that should retrieves the current actuator status.

set_actuator(*setpoint: SrvTypeRequest*) → SrvTypeResponse
Abstract method that sets the actuator to a new setpoint.

This method should be overridden by specific hardware implementation.

Parameters **setpoint** (*SrvTypeRequest*) – the Request sent to self.srv

Returns confirmation of the setpoint

Return type SrvTypeResponse

OSBK-Operation

class osbk_operation.actuator_state_machine.**ActuatorEntry**(*name: str, service_name: str, service_type: SrvType, topic_name: str, topic_type: MsgType*)
A class that bundles the necessary information to control an actuator.

Parameters

- **name** (*str*) – a name to identify this actuator
- **service_name** (*str*) – name of the service the actuator-node provides
- **service_type** (*SrvType*) – the ROS-service-type used

- **topic_name** (*str*) – name of the topic on which the actuator-node publishes its status
- **topic_type** (*MsgType*) – the ROS-message definition used
- **current_state** (*SrvType*) – stores the last published message on the actuator-nodes topic
- **service** (*Client*) – the service-client to command the actuator-node
- **topic** (*Subscription*) – the subscription to listen to the actuator-nodes topic

class osbk_operation.actuator_state_machine.**ActuatorStateMachine**(*name: str, states: List[osbk_operation.util in- iti- al_state: osbk_operation.utility.S tran- siti- ons: List[osbk_operation.util ac- tua- tors: List[osbk_operation.act up- da- te_interval: int = 1*)

A node that manages multiple actuator-nodes according to a finite statemachine.

The statemachine is defined by a list of `State`- and `Transition`-objects. The actuator-setpoints defined in each state are sent to the according actuator in `actuators`.

Parameters

- **states** (*List [State]*) – the states making up a statemachine
- **initial_state** (State) – the state to start statemachine-execution in
- **current_state** (State) – the state that is currently active

- **transitions** (*List [Transition]*) – a list of transitions that connect the states of a statemachine
- **actuators** (*List [ActuatorEntry]*) – a list of ActuatorEntry-objects to interact with the actuator-nodes controlled by this statemachine
- **update_interval** (*int*) – the interval in seconds with which the current state is checked for possible transitions and the actuators receive commands
- **update_timer** (*Timer*) – the timer that triggers the update

class osbk_operation.utility.**State**(*name: str*, *actuator_states: dict*, *final: bool = False*)

Represents a system-state as collection of actuator states.

Designed to be used in an object of the ActuatorStateMachine-class.

Parameters

- **name** (*str*) – a name for the state
- **actuator_states** (*dict*) – a dictionary to map the names of ActuatorEntry-objects to the setpoint for the actuator
- **final** (*bool*) – wether this state should be a finite one in the statemachine
- **possible_transitions** (*List [Transition]*) – a list of Transition-objects which have this state as starting state

check_exit_conditions() → osbk_operation.utility.Transition

Return the first Transition with a satisfied condition.

class osbk_operation.utility.**Transition**(*start: osbk_operation.utility.State*, *end: osbk_operation.utility.State*, *timed: bool = True*, *time: int = 5000*, *condition: Callable[[], bool] = <function _default_condition>*, *action: Callable[[], None] = <function _default_action>*)

Represents a transition between two states.

Designed to be used in an ActuatorStateMachine-object.

Parameters

- **start** (State) – the starting state of this transition
- **end** (State) – the state this transition leads to
- **timed** (*bool*) – wether this Transition should be taken after a given time

- **time** (*int*) – if `timed` is true this controls the duration in seconds after which this transition is triggered
- **timer** (*Timer*) – the timer to trigger this transition. it should be created by an `ActuatorStateMachine`-node
- **condition** (*Callable[[], bool]*) – a `Callable`-object which should return True when the transition can be taken
- **action** (*Callable[[], None]*) – a `Callable`-object which gets called when this transition gets taken
- **active** (*bool*) – a boolean to activate or deactivate this transition

force_take() → osbk_operation.utility.State

Take the transition without checking its condition.

Executes the `action` and returns the end-state.

Returns the state this transition leads to

Return type *State*

take() → osbk_operation.utility.State

Take the transition if the condition is fulfilled.

Executes the `action` and returns the end-state if `condition` returns True.

Returns the state this transition leads to or None

Return type *State*

Meso-Control-Package

class meso_control_pkg.meso_state_machine.**MesoStateMachine**

A node that imlpements the statemachine for controlling a mesocosm-tankpair.

This node extends the `ActuatorStateMachine`-node.

Parameters

- **step_publisher** (*Publisher*) – publishes the current state-number
- **modbus_write_client** (*Client*) – service-client to write the current state-number to the SPS via the `ModbusTcpNode`
- **change_mode_service** (*Service*) – a service to change the operating mode (automatic cylce, measure tank a, measure tank b, drain, stop) of this statemachine
- **current_mode_publisher** (*Publisher*) – publishes the currently active operating mode
- **publish_status_timer** (*Timer*) – the timer to trigger the publishing of the active operating mode

- **current_mode** (*str*) – a string to represent the currently active operating mode
- **last_step_number** (*int*) – the index of the curently active state

class meso_control_pkg.tide_sim.**TideSim**

A node for controlling the tide platform based on real tide levels.

This node uses pegelonline.wsv.de to retrieve current tide levels at List (Sylt) and sends a proportional height to the stepper-motor-control on the SPS.

Parameters

- **tide_a_client** (*Client*) – interacts with the node controlling the tide-platform in tank A
- **tide_b_client** (*Client*) – interacts with the node controlling the tide-platform in tank B
- **mode_publisher_a** (*Publisher*) – publishes the currently active mode for tide control in tank A (automatic or manual)
- **mode_service_a** (*Service*) – can be used to set the mode for tide control in tank A
- **auto_tide_setting_a** (*bool*) – wether tide level in tank A is controlled automatically
- **mode_publisher_b** (*Publisher*) – publishes the currently active mode for tide control in tank B
- **mode_service_b** (*Service*) – can be used to set the mode for tide control in tank B
- **auto_tide_setting_b** (*bool*) – wether tide level in tank B is controlled automatically
- **update_timer** (*Timer*) – triggers the update of the automatic tide control
- **mode_publish_timer** (*Timer*) – triggers the publishing of the currently active modes

publish_mode()

Publish the current modes of tide-control in tank A and B.

set_auto_tide_a(*request: osbk_interfaces.srv._discrete_actuator_control.DiscreteActuatorCon*
response: osbk_interfaces.srv._discrete_actuator_control.DiscreteActuatorCo
→ osbk_interfaces.srv._discrete_actuator_control.DiscreteActuatorControl_

Service callback to set the mode for tide-control in tank A.

set_auto_tide_b(*request: osbk_interfaces.srv._discrete_actuator_control.DiscreteActuatorCon*
response: osbk_interfaces.srv._discrete_actuator_control.DiscreteActuatorCo
→ osbk_interfaces.srv._discrete_actuator_control.DiscreteActuatorControl_

Service callback to set the mode for tide-control in tank A.

update()
Update the tide levels of the tanks according to real values.

class meso_control_pkg.sps_binary_actuator_node.**SpsBinaryActuator**
A node for a binary actuator controlled by an SPS, connected via Modbus.

Uses the ModbusTcpNode.

Parameters

- **last_state** (*int*) – the last state received from the ModbusTcpNode, either 0 or 1
- **modbus_subscription** (*Subscription*) – the subscriber listening to the ModbusTcpNode
- **write_client** (*Client*) – the client for writing to the SPS via the ModbusTcpNode

poll_status() → osbk_interfaces.msg._discrete_actuator_state.DiscreteActuatorState
Return the last transmitted actuator-state by the SPS.

Returns a message object containing the actuator-state

Return type DiscreteActuatorState

set_actuator(*setpoint: osbk_interfaces.srv._discrete_actuator_control.DiscreteActuatorContro*
→ osbk_interfaces.srv._discrete_actuator_control.DiscreteActuatorControl_Res
Send a new setpoint to the actuator.

Parameters **setpoint** (*DiscreteActuatorControl.Request*) – a service-request-object that contains the new setpoint

Returns a service-response-object to confirm the request

Return type DiscreteActuatorControl.Response

class meso_control_pkg.sps_discrete_actuator_node.**SpsDiscreteActuator**
A node for a discrete actuator controlled by an SPS, connected via Modbus.

Uses the ModbusTcpNode.

Parameters

- **last_state** (*int*) – the last state received from the ModbusTcpNode
- **modbus_subscription** (*Subscription*) – the subscriber listening to the ModbusTcpNode
- **write_client** (*Client*) – the client for writing to the SPS via the ModbusTcpNode

poll_status() → osbk_interfaces.msg._discrete_actuator_state.DiscreteActuatorState
Return the last transmitted actuator-state by the SPS.

Returns a message object containing the actuator-state

Return type DiscreteActuatorState

set_actuator(*setpoint: osbk_interfaces.srv._discrete_actuator_control.DiscreteActuatorContro*
→ osbk_interfaces.srv._discrete_actuator_control.DiscreteActuatorControl_Res
Send a new setpoint to the actuator.

Parameters **setpoint** (*DiscreteActuatorControl.Request*) – a service-request-object that contains the new setpoint

Returns a service-response-object to confirm the request

Return type DiscreteActuatorControl.Response

class meso_control_pkg.sps_continuous_actuator_node.**SpsContinuousActuator**
A node for a continuous actuator controlled by an SPS, connected via Modbus.

Uses the ModbusTcpNode.

Parameters

- **last_state** (*float*) – the last state received from the ModbusTcpNode
- **modbus_subscription** (*Subscription*) – the subscriber listening to the ModbusTcpNode
- **write_client** (*Client*) – the client for writing to the SPS via the ModbusTcpNode

poll_status() → osbk_interfaces.msg._continuous_actuator_state.ContinuousActuatorState
Return the last transmitted actuator-state by the SPS.

Returns a message object containing the actuator-state

Return type ContinuousActuatorState

set_actuator(*setpoint: osbk_interfaces.srv._continuous_actuator_control.ContinuousActuator*
→ osbk_interfaces.srv._continuous_actuator_control.ContinuousActuatorContro
Send a new setpoint to the actuator.

Parameters **setpoint** (*ContinuousActuatorControl.Request*) – a service-request-object that contains the new setpoint

Returns a service-response-object to confirm the request

Return type ContinuousActuatorControl.Response

class meso_control_pkg.water_sensor_node.**WaterSensor**
Node for collecting and publishing the data from the water-sensor.

This node extends the SensorBase-node nad uses the ModbusTcpNode.

Parameters

- **last_reading** (*List[float]*) – last transmitted sensor-readings from the SPS.

- **modbus_subscriber** (*Subscription*) – subscribes to the topic where the content of the SPS-feedback-values are published

modbus_handle(*msg: osbk_interfaces.msg._osbk_string_value.OSBKStringValue*)
Subscription-callback to store the transmitted sensor-values.

read_sensor()
Return the last sensor-reading as ROS-message to be published.

class meso_control_pkg.modbus_tcp_node.**ModbusTcpNode**
A node that implements the modbus communication to the SPS.

The feedback-registers are published as a JSON-formatted string. The write-registers can be written individually via a service.

Python Module Index

m

o