ventilator

Release 0.0.0

jonny saunders et al

Aug 14, 2020

HARDWARE:

1 Hardware **1**
- 1.1 Mechanical Diagram . . . 1
  - 1.1.1 Sensors | Hardware . . . 1
    - 1.1.1.1 Overview . . . 1
    - 1.1.1.2 Sensor PCB . . . 1
    - 1.1.1.3 Flow sensor . . . 5
    - 1.1.1.4 Pressure sensors . . . 5
    - 1.1.1.5 Oxygen sensor . . . 5
- 1.2 Flow actuators . . . 5
- 1.3 Sensors . . . 5
- 1.4 Safety Components . . . 6
- 1.5 Tubing and Adapters . . . 6
- 1.6 Bill of Materials (need to think about what goes in this table, probably separate BoMs into tables by category, but here's a sample table) . . . 6

2 common module **7**
- 2.1 message . . . 7
- 2.2 values . . . 9

3 controller module **13**

4 coordinator module **19**
- 4.1 Submodules . . . 19
- 4.2 coordinator . . . 19
- 4.3 ipc . . . 21
- 4.4 process_manager . . . 21

5 gui **23**
- 5.1 Program Diagram . . . 23
- 5.2 Design Requirements . . . 23
- 5.3 UI Notes & Todo . . . 23
- 5.4 Jonny Questions . . . 26
  - 5.4.1 jonny todo . . . 26
- 5.5 GUI Object Documentation . . . 26
  - 5.5.1 Control . . . 30
  - 5.5.2 Monitor . . . 30
  - 5.5.3 Plot . . . 32
  - 5.5.4 Status Bar . . . 32
  - 5.5.5 Components . . . 35
    - 5.5.5.1 GUI Stylesheets . . . 38

5.5.5.2 GUI Alarm Manager . . . 39

6 io module **41**
6.1 Submodules . . . 41
6.2 vent.io.devices module . . . 41
6.3 vent.io.iobase module . . . 41
6.4 Module contents . . . 41

7 Requirements **43**

8 Datasheets & Manuals **45**
8.1 Manuals . . . 45
8.2 Other Reference Material . . . 45

9 Specs **47**

10 Changelog **49**
10.1 Version 0.0 . . . 49
10.1.1 v0.0.2 (April xxth, 2020) . . . 49
10.1.2 v0.0.1 (April 12th, 2020) . . . 49
10.1.3 v0.0.0 (April 12th, 2020) . . . 49

11 Building the Docs **51**
11.1 Local Build . . . 51

12 h1 Heading 8-) **53**
12.1 h2 Heading . . . 53
12.1.1 h3 Heading . . . 53
12.1.1.1 h4 Heading . . . 53
12.2 Horizontal Rules . . . 53
12.3 Emphasis . . . 53
12.4 Blockquotes . . . 53
12.5 Lists . . . 54
12.6 Code . . . 54
12.7 Links . . . 55
12.8 Images . . . 57

13 Indices and tables **59**

Python Module Index **61**

Index **63**

CHAPTER

ONE

HARDWARE

1.1 Mechanical Diagram

1.1.1 Sensors | Hardware

1.1.1.1 Overview

The TigerVent has four main sensors: 1. oxygen sensor (O2S) 2. proximal pressure sensor (PS1) 3. expiratory pressure sensor (PS2) 4. expiratory flow sensor (FS1)

These materials interface with a modular sensor PCB that can be reconfigured for part substitution. The nominal design assumes both pressure sensors and the oxygen sensor have analog voltage outputs, and interface with the controller via I2C link with a 16-bit, 4 channel ADC (ADS1115). The expiratory flow sensor (SFM3300 or equivalent) uses a direct I2C interface, but can be replaced by a commercial spirometer and an additional differential pressure sensor.

1.1.1.2 Sensor PCB

Schematic

Bill of Materials

- – Ref
  - – Part
  - – Description
  - – Datasheet
- – U1
  - – Amphenol 1 PSI-D1-4V-MINI
  - – Analog output differential pressure sensor
  - – /DS-0103-Rev-A-1499253.pdf <- not sure best way to do this
- – U3
  - – Amphenol 1 PSI-D1-4V-MINI differential pressure sensor
  - – Analog output differential pressure sensor
  - – above
- – U2

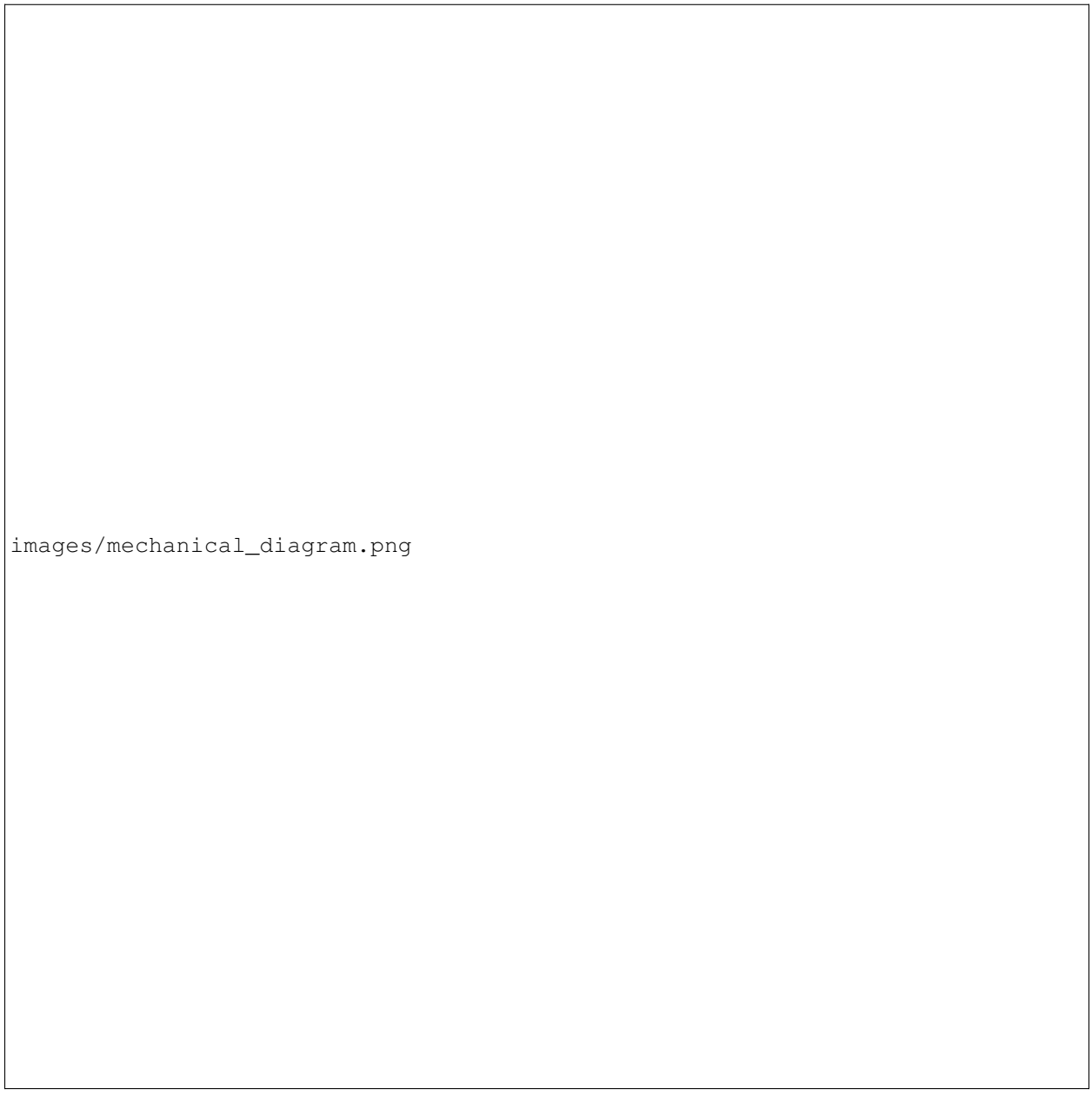

Fig. 1: Schematic diagram of main mechanical components

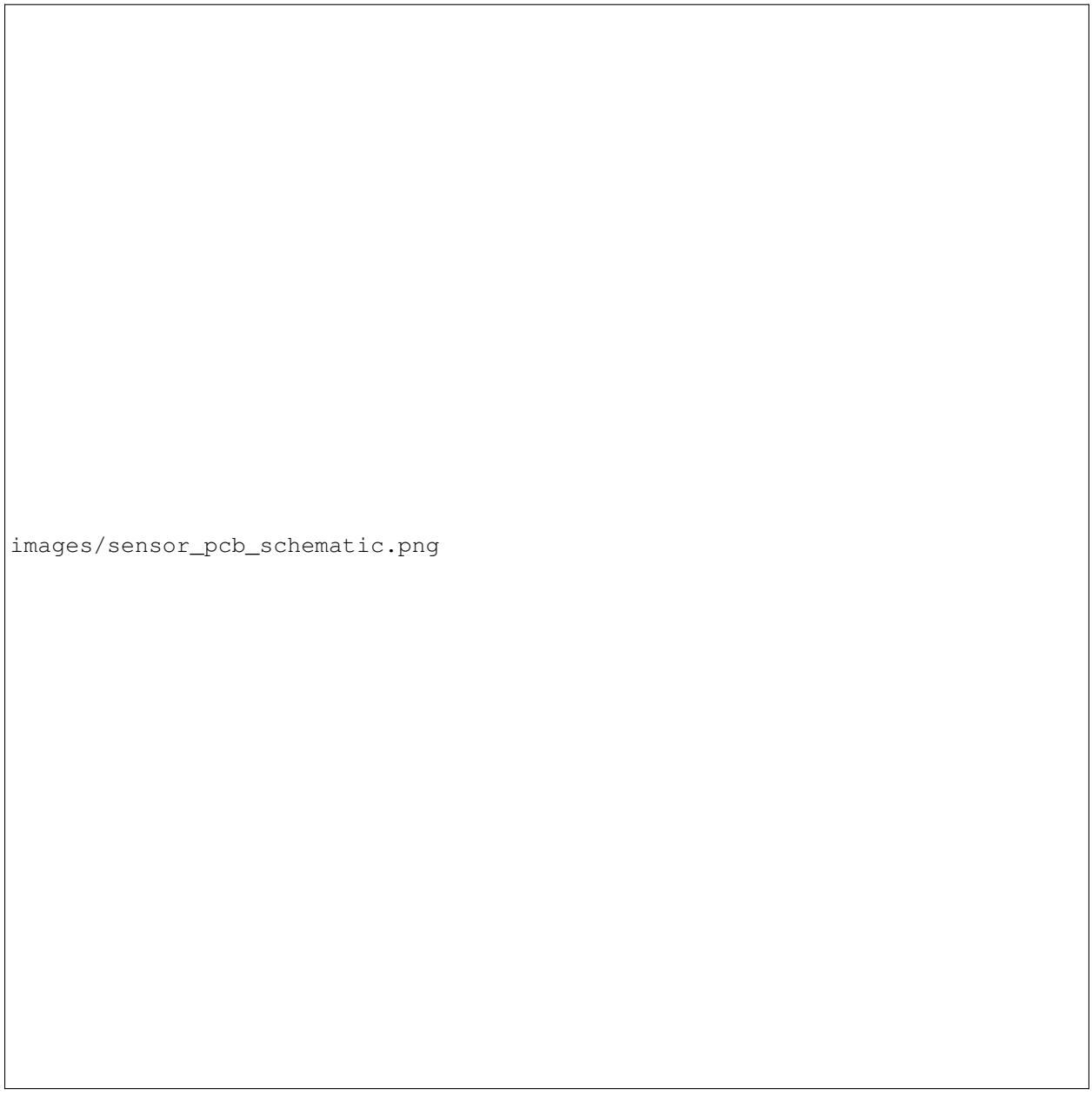

Fig. 2: Electrical schematic for sensor board

  - Adafruit 4-channel ADS1115 ADC breakout
  - Supply ADC to RPi to read analog sensors
  - /adafruit-4-channel-adc-breakouts.pdf
- – U4
  - INA126 instrumentation amplifier, DIP-8
  - Instrumentation amplifier to boost oxygen sensor output
  - /ina126.pdf
- – J1
  - 01x02 2.54 mm pin header
  - Breakout for alert pin from ADS1115 ADC
  - none
- – J2
  - 02x04 2.54 mm pin header
  - Jumpers to select I2C address for ADC
  - none
- – J3
  - 40 pin RPi hat connector
  - Extends RPi GPIO pins to the board
  - (to be inserted)
- – J4
  - 01x02 2.54 mm 90 degree pin header
  - For direct connection to oxygen sensor output
  - none
- – J5
  - 01x04 2.54 mm 90 degree pin header pin header
  - For I2C connection to SFM3300 flow meter
  - none
- – J6
  - 01x03 2.54 mm 90 degree pin header pin header
  - Connector to use an additional analog output (ADS1115 input A3).
  - none
- – R1
  - 1-2.7 k resistor
  - Optional I2C pullup resistor (RPi already has 1.8k pullups)
  - none
- – R2

  - 1-2.7 k resistor
  - Connector to use an additional analog output (RPi already has 1.8k pullups).
  - none
- – R3
  - 0.1-100k resistor
  - R_G that sets gain for the INA126 instrumentation amplifier (U4). G = 5 + 80k/R_G
  - none

1.1.1.3 Flow sensor

Document D-lite alternative

1.1.1.4 Pressure sensors

Just use any other analog voltage output (0-4 V) sensor

1.1.1.5 Oxygen sensor

Explanation of interface circuit and some alts

- Expiratory flow sensor (FS1)

1.2 Flow actuators

- Actuator PCB/overview (link to PCB with BoM, schematic, layout, etc.)
- Proportional solenoid valve (V1) (link to doc with crit specs, driving circuit, part spec, datasheet, alternatives, etc.)
- Expiratory valve (V2) (link to doc with crit specs, driving circuit, part spec, datasheet, etc.)

1.3 Sensors

- Sensor PCB/overview (link to PCB with BoM, schematic, layout, etc.)
- Oxygen sensor (O2S) (link to doc with crit specs, interface circuit, part spec, datasheet, alternatives, etc.)
- Proximal pressure sensor (PS1)
- Expiratory pressure sensor (PS2)
- Expiratory flow sensor (FS1)

1.4 Safety Components

- 50 psi, high pressure relief valve (PRV1)
- Safety check valve (CV)
- 70 cmH2O patient-side pressure relief valve (PRV2)
- Filters (F1, F2)
- PEEP valve (PEEP) (include the design bifurcation in this module description)

1.5 Tubing and Adapters

- Manifold 1
- Manifold 2
- Mounting Bracket 1… etc.

1.6 Bill of Materials (need to think about what goes in this table, probably separate BoMs into tables by category, but here's a sample table)

Ref	Name	Part	Description
V1	Inspiratory on/off valve	red hat process valve	completely cut off flow if required
PRV1	High pressure relief valve	Sets to 50 psi	regulates upstream pressure to 50 psi
CV	Inspiratory check valve	valve stat here	In case of emergency power loss, allows patient to continue taking breaths from air
PRV2	Maximum pressure valve	…	Sets absolute maximum pressure at patient side to 53 cm H2O
F1/F2	Filters	HEPA filters?	Keeps the system's sensors from becoming contaminated
O2S	Oxygen sensor	Sensiron …	Checks FiO2 level
PS1/PS2	Pressure sensors	mini4v	Uses gas takeoffs to measure pressure at each desired point
FS1	Flow sensor	Sensiron flow sensor	Measures expiratory flow to calculate tidal volume
M1/M2	Manifolds	3D printed parts	Hubs to connect multiple components in one place
V3	Expiratory on/off valve	Festo Electrical Air Directional Control Valve, 3/2 flow, Normally Closed, 8 mm Push-to-Connect	Opens to initiation the expiratory cycle
PEEP	PEEP backpressure valve	PEEP valve	Sets PEEP on expiratory cycle!

CHAPTER

TWO

COMMON MODULE

2.1 message

Classes

`Alarm`(alarm_name, is_active, severity, ...)	**param alarm_name**
`AlarmSeverity`(value)	An enumeration.
`ControlSetting`(name, value, min_value, ...)	TODO: if enum is hard to use, we may just use a predefined set, e.g.
`Error`(errnum, err_str, timestamp)	
`SensorValueNew`(name, value, timestamp, ...)	
`SensorValues`([timestamp, loop_counter])	**param **kwargs** sensor readings, must be in `vent.values.SENSOR.keys`

class vent.common.message.**SensorValueNew**(*name*, *value*, *timestamp*, *loop_counter*)

Bases: object

class vent.common.message.**SensorValues**(*timestamp=None*, *loop_counter=None*, ***kwargs*)

Bases: object

Parameters ****kwargs** – sensor readings, must be in `vent.values.SENSOR.keys`

Methods

`__init__`([timestamp, loop_counter])	**param **kwargs** sensor readings, must be in `vent.values.SENSOR.keys`
`to_dict`()	

__init__(*timestamp=None*, *loop_counter=None*, ***kwargs*)

Parameters ****kwargs** – sensor readings, must be in `vent.values.SENSOR.keys`

to_dict()

class vent.common.message.**ControlSetting**(*name*, *value*, *min_value*, *max_value*, *timestamp*)

Bases: object

TODO: if enum is hard to use, we may just use a predefined set, e.g. {'PIP', 'PEEP', ...} :param name: enum belong to ValueName :param value: :param min_value: :param max_value: :param timestamp: **Methods**

__init__(name, value, min_value, max_value, ...)	TODO: if enum is hard to use, we may just use a predefined set, e.g.

__init__(*name*, *value*, *min_value*, *max_value*, *timestamp*)
TODO: if enum is hard to use, we may just use a predefined set, e.g. {'PIP', 'PEEP', ...} :param name: enum belong to ValueName :param value: :param min_value: :param max_value: :param timestamp:

class vent.common.message.**AlarmSeverity**(*value*)
Bases: enum.Enum

An enumeration. **Attributes**

RED	int([x]) -> integer
ORANGE	int([x]) -> integer
YELLOW	int([x]) -> integer
OFF	int([x]) -> integer

RED = 3

ORANGE = 2

YELLOW = 1

OFF = 0

class vent.common.message.**Alarm**(*alarm_name*, *is_active*, *severity*, *alarm_start_time*, *alarm_end_time*, *value=None*, *message=None*)
Bases: object

Parameters

- **alarm_name** –
- **is_active** –
- **severity** –
- **alarm_start_time** –
- **alarm_end_time** –
- **value** (*int, float*) – optional - numerical value that generated the alarm
- **message** (*str*) – optional - override default text generated by *AlarmManager*

Methods

__init__(alarm_name, is_active, severity, ...)	**param alarm_name**

Attributes

id_counter	itertools.count: used to generate unique IDs for each alarm

id_counter = count(0)
used to generate unique IDs for each alarm

Type itertools.count

__init__(*alarm_name*, *is_active*, *severity*, *alarm_start_time*, *alarm_end_time*, *value=None*, *message=None*)

Parameters

- **alarm_name** –
- **is_active** –
- **severity** –
- **alarm_start_time** –
- **alarm_end_time** –
- **value** (*int, float*) – optional - numerical value that generated the alarm
- **message** (*str*) – optional - override default text generated by *AlarmManager*

class vent.common.message.**Error**(*errnum*, *err_str*, *timestamp*)
Bases: object

2.2 values

Parameterization of variables and values

Data

CONTROL	Values to control but not monitor.
LIMITS	Values that are dependent on other values:
SENSOR	Values to monitor but not control.

Classes

Value(name, units, abs_range, safe_range, ...)	Definition of a value.
ValueName(value)	An enumeration.

class vent.common.values.**ValueName**(*value*)
Bases: enum.Enum

An enumeration. **Attributes**

PIP	int([x]) -> integer
PIP_TIME	int([x]) -> integer
PEEP	int([x]) -> integer
PEEP_TIME	int([x]) -> integer
BREATHS_PER_MINUTE	int([x]) -> integer

continues on next page

Table 9 – continued from previous page

INSPIRATION_TIME_SEC	int([x]) -> integer
IE_RATIO	int([x]) -> integer
FIO2	int([x]) -> integer
TEMP	int([x]) -> integer
HUMIDITY	int([x]) -> integer
VTE	int([x]) -> integer
PRESSURE	int([x]) -> integer

PIP = 1

PIP_TIME = 2

PEEP = 3

PEEP_TIME = 4

BREATHS_PER_MINUTE = 5

INSPIRATION_TIME_SEC = 6

IE_RATIO = 7

FIO2 = 8

TEMP = 9

HUMIDITY = 10

VTE = 11

PRESSURE = 12

class vent.common.values.**Value**(*name: str*, *units: str*, *abs_range: tuple*, *safe_range: tuple*, *decimals: int*, *default: (<class 'int'>, <class 'float'>) = None*)

Bases: object

Definition of a value.

Used by the GUI and control module to set defaults.

Parameters

- **name** (*str*) – Human-readable name of the value
- **units** (*str*) – Human-readable description of units
- **abs_range** (*tuple*) – tuple of ints or floats setting the logical limit of the value, eg. a percent between 0 and 100, (0, 100)
- **safe_range** (*tuple*) – tuple of ints or floats setting the safe ranges of the value,

  note:

  ```
  this is not the same thing as the user-set alarm values,
  though the user-set alarm values are initialized as ``safe_range``.
  ```

- **decimals** (*int*) – the number of decimals of precision used when displaying the value

Methods

__init__(name, units, abs_range, safe_range, ...)	Definition of a value.
to_dict()	

Attributes

abs_range
decimals
default
name
safe_range

__init__(*name: str*, *units: str*, *abs_range: tuple*, *safe_range: tuple*, *decimals: int*, *default: (<class 'int'>, <class 'float'>) = None*)

Definition of a value.

Used by the GUI and control module to set defaults.

Parameters

- **name** (*str*) – Human-readable name of the value
- **units** (*str*) – Human-readable description of units
- **abs_range** (*tuple*) – tuple of ints or floats setting the logical limit of the value, eg. a percent between 0 and 100, (0, 100)
- **safe_range** (*tuple*) – tuple of ints or floats setting the safe ranges of the value,

  note:

  ```
  this is not the same thing as the user-set alarm values,
  though the user-set alarm values are initialized as ``safe_
  ↪range``.
  ```

- **decimals** (*int*) – the number of decimals of precision used when displaying the value

property name

property abs_range

property safe_range

property decimals

property default

to_dict()

vent.common.values.**SENSOR = OrderedDict([(<ValueName.FIO2: 8>, <vent.common.values.Value**

Values to monitor but not control.

Used to set alarms for out-of-bounds sensor values. These should be sent from the control module and not computed.:

```
{
    'name' (str): Human readable name,
    'units' (str): units string, (like degrees or %),
    'abs_range' (tuple): absolute possible range of values,
    'safe_range' (tuple): range outside of which a warning will be raised,
    'decimals' (int): The number of decimals of precision this number should be␣
↪displayed with
}
```

vent.common.values.**CONTROL = OrderedDict([(<ValueName.PIP: 1>, <vent.common.values.Value ob**

Values to control but not monitor.

Sent to control module to control operation of ventilator.:

```
{
    'name' (str):  Human readable name,
    'units' (str): units string, (like degrees or %),
    'abs_range' (tuple): absolute possible range of values,
    'safe_range' (tuple): range outside of which a warning will be raised,
    'default' (int, float): the default value of the parameter,
    'decimals' (int): The number of decimals of precision this number should be␣
↪displayed with
}
```

vent.common.values.**LIMITS = {}**

Values that are dependent on other values:

```
{
    "dependent_value": (
        ['value_1', 'value_2'],
        callable_returning_boolean
    }
}
```

Where the first argument in the tuple is a list of the values that will be given as argument to the `callable_returning_boolean` which will return whether (`True`) or not (`False`) a value is allowed.

CHAPTER

THREE

CONTROLLER MODULE

Classes

Balloon_Simulator(leak)	This is a imple physics simulator for inflating a balloon.
ControlModuleBase()	This is an abstract class for controlling simulation and hardware.
ControlModuleDevice()	
ControlModuleSimulator()	

Functions

get_control_module([sim_mode])	

class vent.controller.control_module.**ControlModuleBase**

Bases: object

This is an abstract class for controlling simulation and hardware.

1. **All internal variables fall in three classes, denoted by the beginning of the variable:**
   - "COPY_varname": These are copies (see 1.) that are regularly sync'ed with internal variables.
   - "__varname": These are variables only used in the ControlModuleBase-Class
   - "_varname": These are variables used in derived classes.
2. **Internal variables should only to be accessed though the set_ and get_ functions.** These functions act on COPIES of internal variables ("__" and "_"), that are sync'd every few iterations. How often this is done is adjusted by the variable self._NUMBER_CONTROLL_LOOPS_UNTIL_UPDATE. To avoid multiple threads manipulating the same variables at the same time, every manipulation of "**COPY_**" is surrounded by a thread lock.

Methods

__analyze_last_waveform()	This goes through the last waveform, and updates VTE, PEEP, PIP, PIP_TIME, I_PHASE, FIRST_PEEP and BPM.
__calculate_control_signal_in()	
__get_PID_error(ytarget, yis, dt)	
__get_pressure_derivative(dt)	
__test_critical_levels(min, max, value, name)	This tests whether a variable is within bounds.

continues on next page

Table 3 – continued from previous page

__update_alarms()	This goes through the values obtained from the last waveform, and updates alarms.
_PID_update(dt)	This instantiates the control algorithms.
_alarm_to_COPY()	
_controls_from_COPY()	
_get_control_signal_in()	This is the PID controlled signal on the inspiratory side
_get_control_signal_out()	This is the control signal (open/close) on the expiratory side
_initialize_set_to_COPY()	
_sensor_to_COPY()	
_start_mainloop()	
do_pid_control()	
do_state_control()	
get_active_alarms()	
get_alarms()	
get_control(control_setting_name)	Gets values of the COPY of the control settings.
get_logged_alarms()	
get_past_waveforms()	
get_sensors()	
heartbeat()	only used for fiddling
is_running()	
set_control(control_setting)	Updates the entry of COPY contained in the control settings
start()	
stop()	

Attributes

running

Public Methods: get_sensors(): Returns a copy of the current sensor values. get_alarms(): Returns a List of all alarms, active and logged get_active_alarms(): Returns a Dictionary of all currently active alarms. get_logged_alarms(): Returns a List of logged alarms, up to maximum lengh of self._RINGBUFFER_SIZE get_control(ControlSetting): Sets a controll-setting. Is updated at latest within self._NUMBER_CONTROLL_LOOPS_UNTIL_UPDATE get_past_waveforms(): Returns a List of waveforms of pressure and volume during at the last N breath cycles, N<self._RINGBUFFER_SIZE, AND clears this archive. start(): Starts the main-loop of the controller stop(): Stops the main-loop of the controller

_initialize_set_to_COPY()

_alarm_to_COPY()

_sensor_to_COPY()

_controls_from_COPY()

__test_critical_levels(*min*, *max*, *value*, *name*)

This tests whether a variable is within bounds. If it is, and an alarm existed, then the "alarm_end_time" is set. If it is NOT, a new alarm is generated and appendede to the alarm-list. Input:

min: minimum value (e.g. 2) max: maximum value (e.g. 5) value: test value (e.g. 3) name: parameter type (e.g. "PIP", "PEEP" etc.)

__analyze_last_waveform()
This goes through the last waveform, and updates VTE, PEEP, PIP, PIP_TIME, I_PHASE, FIRST_PEEP and BPM.

__update_alarms()
This goes through the values obtained from the last waveform, and updates alarms.

get_sensors() → vent.common.message.SensorValues

get_alarms() → List[vent.common.message.Alarm]

get_active_alarms()

get_logged_alarms() → List[vent.common.message.Alarm]

set_control(*control_setting: vent.common.message.ControlSetting*)
Updates the entry of COPY contained in the control settings

get_control(*control_setting_name:* vent.common.values.ValueName) → vent.common.message.ControlSetting
Gets values of the COPY of the control settings.

__get_PID_error(*ytarget*, *yis*, *dt*)

__calculate_control_signal_in()

_get_control_signal_in()
This is the PID controlled signal on the inspiratory side

_get_control_signal_out()
This is the control signal (open/close) on the expiratory side

__get_pressure_derivative(*dt*)

_PID_update(*dt*)
This instantiates the control algorithms. During the breathing cycle, it goes through the four states:

1) Rise to PIP
2) Sustain PIP pressure
3) Quick fall to PEEP
4) Sustaint PEEP pressure

Once the cycle is complete, it checks the cycle for any alarms, and starts a new one. A record of pressure/volume waveforms is kept in self.__cycle_waveform_archive

dt: Time since last update in seconds

get_past_waveforms()

_start_mainloop()

start()

stop()

is_running()

do_pid_control()

do_state_control()

heartbeat()
only used for fiddling

property running

class vent.controller.control_module.**ControlModuleDevice**
Bases: vent.controller.control_module.ControlModuleBase **Methods**

_sensor_to_COPY()	
_start_mainloop()	

_sensor_to_COPY()

_start_mainloop()

class vent.controller.control_module.**Balloon_Simulator**(*leak*)
Bases: object

This is a imple physics simulator for inflating a balloon. For math, see https://en.wikipedia.org/wiki/Two-balloon_experiment **Methods**

OUupdate(variable, dt, mu, sigma, tau)	This is a simple function to produce an OU process.
get_pressure()	
get_volume()	
set_flow_in(Qin, dt)	
set_flow_out(Qout, dt)	
update(dt)	

get_pressure()

get_volume()

set_flow_in(*Qin*, *dt*)

set_flow_out(*Qout*, *dt*)

update(*dt*)

OUupdate(*variable*, *dt*, *mu*, *sigma*, *tau*)
This is a simple function to produce an OU process. It is used as model for fluctuations in measurement variables. inputs: variable: float value at previous time step dt : timestep mu : mean sigma : noise amplitude tau : time scale returns: new_variable : value of "variable" at next time step

class vent.controller.control_module.**ControlModuleSimulator**
Bases: vent.controller.control_module.ControlModuleBase **Methods**

__SimulatedPropValve(x, dt)	This simulates the action of a proportional valve.
__SimulatedSolenoid(x)	Depending on x, set flow to a binary value.
_sensor_to_COPY()	
_start_mainloop()	

__SimulatedPropValve(*x*, *dt*)
This simulates the action of a proportional valve. Flow-current-curve eye-balled from the datasheet of SMC PVQ31-5G-23-01N https://www.ocpneumatics.com/content/pdfs/PVQ.pdf

x: Input current [mA] dt: Time since last setting in seconds [for the LP filter]

__SimulatedSolenoid(*x*)
Depending on x, set flow to a binary value. Here: flow is either 0 or 1l/sec

_sensor_to_COPY()

_start_mainloop()

vent.controller.control_module.**get_control_module**(*sim_mode=False*)

CHAPTER

FOUR

COORDINATOR MODULE

4.1 Submodules

4.2 coordinator

Classes

CoordinatorBase([sim_mode])	
CoordinatorLocal([sim_mode])	
	param sim_mode
CoordinatorRemote([sim_mode])	

Functions

get_coordinator([single_process, sim_mode])	

class vent.coordinator.coordinator.**CoordinatorBase**(*sim_mode=False*)

Bases: object **Methods**

clear_logged_alarms()	
get_active_alarms()	
get_control(control_setting_name)	
get_logged_alarms()	
get_sensors()	
is_running()	
set_control(control_setting)	
start()	
stop()	

get_sensors() → vent.common.message.SensorValues

get_active_alarms() → Dict[str, vent.common.message.Alarm]

get_logged_alarms() → List[vent.common.message.Alarm]

clear_logged_alarms()

set_control(*control_setting: vent.common.message.ControlSetting*)

get_control(*control_setting_name:* vent.common.values.ValueName) → vent.common.message.ControlSetting

start()

is_running() → bool

stop()

class vent.coordinator.coordinator.**CoordinatorLocal**(*sim_mode=False*)

Bases: vent.coordinator.coordinator.CoordinatorBase

Parameters **sim_mode** –

Methods

__init__([sim_mode])	**param sim_mode**
clear_logged_alarms()	
get_active_alarms()	
get_control(control_setting_name)	
get_logged_alarms()	
get_sensors()	
is_running()	Test whether the whole system is running
set_control(control_setting)	
start()	Start the coordinator.
stop()	Stop the coordinator.

_is_running

.set() when thread should stop

Type threading.Event

__init__(*sim_mode=False*)

Parameters **sim_mode** –

_is_running

.set() when thread should stop

Type threading.Event

get_sensors() → vent.common.message.SensorValues

get_active_alarms() → Dict[str, vent.common.message.Alarm]

get_logged_alarms() → List[vent.common.message.Alarm]

clear_logged_alarms()

set_control(*control_setting: vent.common.message.ControlSetting*)

get_control(*control_setting_name:* vent.common.values.ValueName) → vent.common.message.ControlSetting

start()

Start the coordinator. This does a soft start (not allocating a process).

is_running() → bool

Test whether the whole system is running

stop()

Stop the coordinator. This does a soft stop (not kill a process)

class vent.coordinator.coordinator.**CoordinatorRemote**(*sim_mode=False*)
Bases: vent.coordinator.coordinator.CoordinatorBase **Methods**

clear_logged_alarms()	
get_active_alarms()	
get_control(control_setting_name)	
get_logged_alarms()	
get_sensors()	
is_running()	Test whether the whole system is running
set_control(control_setting)	
start()	Start the coordinator.
stop()	Stop the coordinator.

get_sensors() → vent.common.message.SensorValues

get_active_alarms() → Dict[str, vent.common.message.Alarm]

get_logged_alarms() → List[vent.common.message.Alarm]

clear_logged_alarms()

set_control(*control_setting: vent.common.message.ControlSetting*)

get_control(*control_setting_name:* vent.common.values.ValueName) → vent.common.message.ControlSetting

start()
Start the coordinator. This does a soft start (not allocating a process).

is_running() → bool
Test whether the whole system is running

stop()
Stop the coordinator. This does a soft stop (not kill a process)

vent.coordinator.coordinator.**get_coordinator**(*single_process=False*, *sim_mode=False*) → vent.coordinator.coordinator.CoordinatorBase

4.3 ipc

4.4 process_manager

Classes

ProcessManager(sim_mode[, startCommandLine, ...])	

class vent.coordinator.process_manager.**ProcessManager**(*sim_mode*, *startCommandLine=None*, *maxHeartbeatInterval=None*)
Bases: object **Methods**

heartbeat(timestamp)	

continues on next page

Table 7 – continued from previous page

restart_process()
start_process()
try_stop_process()

start_process()

try_stop_process()

restart_process()

heartbeat(*timestamp*)

CHAPTER

FIVE

GUI

5.1 Program Diagram

5.2 Design Requirements

- Display Values
  - Value name, units, absolute range, recommended range, default range
  - VTE
  - FiO2
  - Humidity
  - Temperature
- Plots
- Controlled Values
  - PIP
  - PEEP
  - Inspiratory Time
- Value Dependencies

5.3 UI Notes & Todo

- Jonny add notes from helpful RT in discord!!!
- Top status Bar
  - Start/stop button
  - Status indicator - a clock that increments with heartbeats, or some other visual indicator that things are alright
  - Status bar - most recent alarm or notification w/ means of clearing
  - Override to give 100% oxygen and silence all alarms
- API
  - Two queues, input and output. Read from socket and put directly into queue.

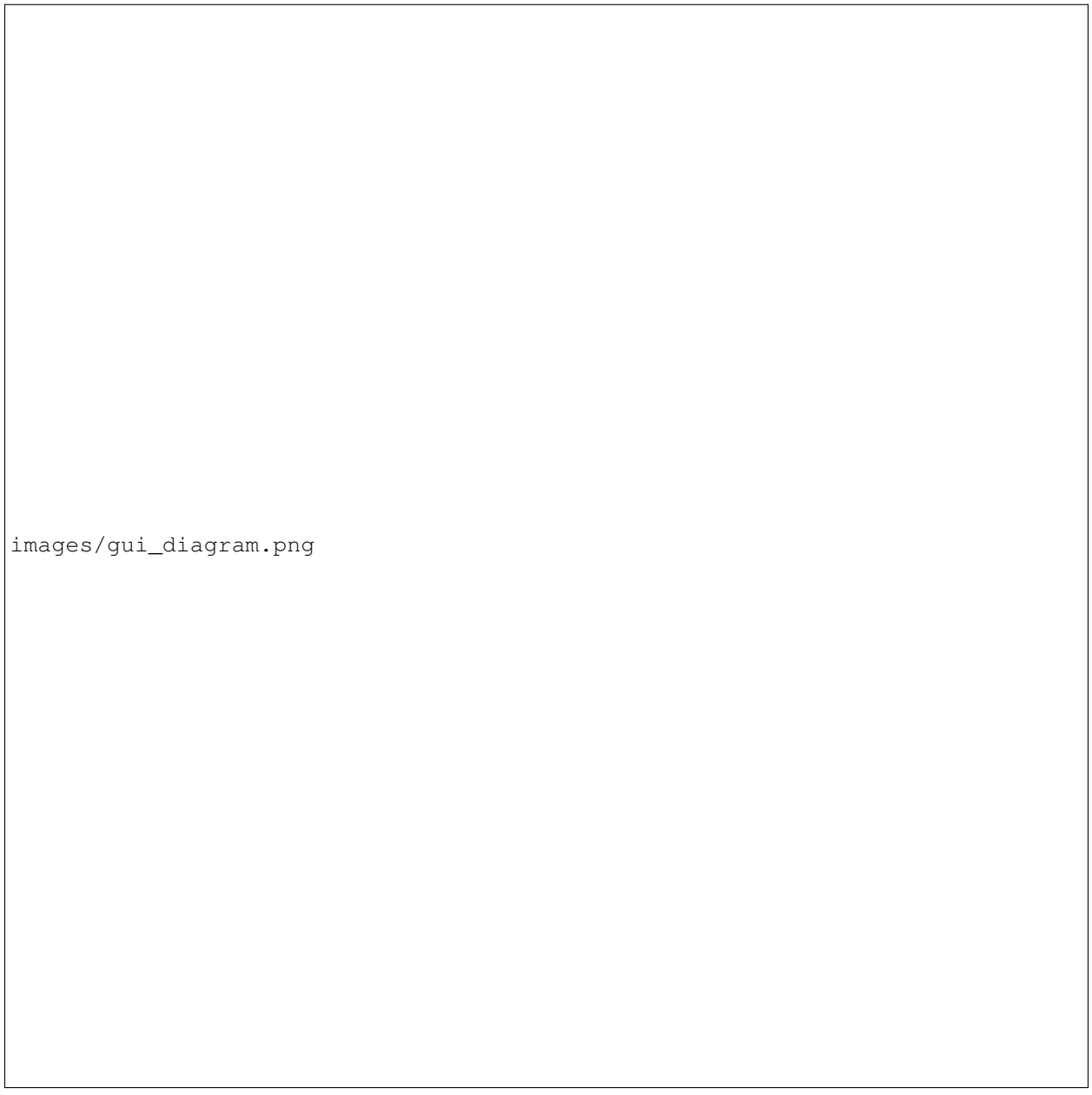

Fig. 1: Schematic diagram of major UI components and signals

  - Input, receive (timestamp, key, value) messages where key and value are names of variables and their values
  - Output, send same format
- Menus
  - Trigger some testing/calibration routine
  - Log/alarm viewer
  - Wizard to set values?
  - save/load values
- Alarms
  - Multiple levels
  - Silenced/reset
  - Logging
  - Sounds?
- General
  - Reduce space given to waveforms
  - Clearer grouping & titling for display values & controls
  - Collapsible range setting
  - Ability to declare dependencies between values
    - Limits - one value's range logically depends on another
    - Derived - one value is computed from another/others
  - Monitored values should have defaults, warning range, and absolute range
  - Two classes of monitored values – ones with limits and ones without. There seem to be lots and lots of observed values, but only some need limits. might want to make larger drawer of smaller displayed values that don't need controls
  - Save/load parameters. Autosave, and autorestore if saved <5m ago, otherwise init from defaults.
  - Implement timed updates to plots to limit resource usage
  - Make class for setting values
  - Possible plots
    - Pressure vs. flow
    - flow vs volume
    - volume vs time
- Performance
  - Cache drawText() calls in range selector by drawing to pixmap

5.4 Jonny Questions

- Which alarm sounds to use?
- Final final final breakdown on values and ranges plzzz

5.4.1 jonny todo

- use loop_counter to check on controller advancement
- implement single values list with properties 'controllable' vs not.

5.5 GUI Object Documentation

Classes

Vent_Gui(coordinator[, update_period])	The Main GUI window.

Functions

launch_gui(coordinator)	

class vent.gui.main.**Vent_Gui**(*coordinator*, *update_period=0.1*)

The Main GUI window.

Only one instance can be created at a time. Uses `set_gui_instance()` to store a reference to itself. after initialization, use *get_gui_instance* to retrieve a reference.

Attributes

CONTROL	see `gui.defaults.CONTROL`
MONITOR	see `gui.defaults.SENSOR`
PLOTS	see `gui.defaults.PLOTS`
alarm_state	
alarms_updated(*args, **kwargs)	`PySide2.QtCore.Signal` emitted whenever alarms are updated.
control_width	int([x]) -> integer
gui_closing(*args, **kwargs)	`PySide2.QtCore.Signal` emitted when the GUI is closing.
main_height	int([x]) -> integer
monitor_width	int([x]) -> integer
plot_width	int([x]) -> integer
status_height	int([x]) -> integer
total_height	computed from `status_height+main_height`
total_width	computed from `monitor_width+plot_width+control_width`
update_period	

Methods

__init__(coordinator[, update_period])	The Main GUI window.
alarm_state_changed(state)	
closeEvent(event)	Emit *gui_closing* and close!
handle_alarm(alarm)	
handle_cleared_alarm(alarm)	
init_controls()	on startup, set controls in coordinator to ensure init state is synchronized
init_ui()	Create the UI components for the ventilator screen
init_ui_controls()	
init_ui_monitor()	
init_ui_plots()	
init_ui_signals()	Connect Qt signals and slots
init_ui_status_bar()	
set_plot_duration(dur)	
set_value(new_value[, value_name])	Set a control value with the `coordinator`
update_gui()	
update_value(value_name, new_value)	**param value_name** Name of key in *Vent_Gui.monitor* and *Vent_Gui.plots* to update

monitor
Dictionary mapping *values.SENSOR* keys to `widgets.Monitor_Value` objects

Type dict

plots
Dictionary mapping `gui.PLOT` keys to `widgets.Plot` objects

Type dict

controls
Dictionary mapping *values.CONTROL* keys to `widgets.Control` objects

Type dict

coordinator
Some coordinator object that we use to communicate with the controller

Type `vent.coordinator.coordinator.CoordinatorBase`

control_module
Reference to the control module, retrieved from coordinator

Type `vent.controller.control_module.ControlModuleBase`

start_time
Start time as returned by `time.time()`

Type float

update_period
The global delay between redraws of the GUI (seconds)

Type float

alarm_manager

Type *AlarmManager*

Parameters

- **update_period** (*float*) – The global delay between redraws of the GUI (seconds)
- **test** (*bool*) – Whether the monitored values and plots should be fed sine waves for visual testing.

gui_closing(**args*, ***kwargs*) **= <PySide2.QtCore.Signal object>**
PySide2.QtCore.Signal emitted when the GUI is closing.

alarms_updated(**args*, ***kwargs*) **= <PySide2.QtCore.Signal object>**
PySide2.QtCore.Signal emitted whenever alarms are updated.

Returns the result of self.coordinator.get_active_alarms, so will emit an empty dict if there are no active alarms.

MONITOR = OrderedDict([(<ValueName.FIO2: 8>, <vent.common.values.Value object>), (<Va
see gui.defaults.SENSOR

CONTROL = OrderedDict([(<ValueName.PIP: 1>, <vent.common.values.Value object>), (<Valu
see gui.defaults.CONTROL

PLOTS = OrderedDict([(<ValueName.PRESSURE: 12>, {'name': 'Pressure', 'units': 'mmH2O
see gui.defaults.PLOTS

monitor_width = 2

plot_width = 4

control_width = 2

total_width = 8
computed from monitor_width+plot_width+control_width

status_height = 1

main_height = 5

total_height = 6
computed from status_height+main_height

__init__(*coordinator*, *update_period=0.1*)
The Main GUI window.

Only one instance can be created at a time. Uses set_gui_instance() to store a reference to itself. after initialization, use *get_gui_instance* to retrieve a reference.

monitor
Dictionary mapping *values.SENSOR* keys to widgets.Monitor_Value objects
Type dict

plots
Dictionary mapping gui.PLOT keys to widgets.Plot objects
Type dict

controls
Dictionary mapping *values.CONTROL* keys to widgets.Control objects
Type dict

coordinator
Some coordinator object that we use to communicate with the controller
Type vent.coordinator.coordinator.CoordinatorBase

control_module
Reference to the control module, retrieved from coordinator
Type vent.controller.control_module.ControlModuleBase

start_time
Start time as returned by time.time()
Type float

update_period
The global delay between redraws of the GUI (seconds)
Type float

alarm_manager
Type *AlarmManager*

Parameters

- **update_period** (*float*) – The global delay between redraws of the GUI (seconds)
- **test** (*bool*) – Whether the monitored values and plots should be fed sine waves for visual testing.

property update_period

init_controls()
on startup, set controls in coordinator to ensure init state is synchronized

set_value(*new_value*, *value_name=None*)
Set a control value with the coordinator

Parameters **new_value** (*float*) – Som

update_gui()

update_value(*value_name*, *new_value*)

Parameters

- **value_name** (*str*) – Name of key in *Vent_Gui.monitor* and *Vent_Gui.plots* to update
- **new_value** (*int, float*) – New value to display/plot

init_ui()
Create the UI components for the ventilator screen

init_ui_status_bar()

init_ui_monitor()

init_ui_plots()

init_ui_controls()

staticMetaObject = <PySide2.QtCore.QMetaObject object>

init_ui_signals()
Connect Qt signals and slots

handle_alarm(*alarm*)

handle_cleared_alarm(*alarm*)

property alarm_state

alarm_state_changed(*state*)

set_plot_duration(*dur*)

closeEvent(*event*)
Emit *gui_closing* and close!

vent.gui.main.**launch_gui**(*coordinator*)

5.5.1 Control

Classes

Control(value)

class vent.gui.widgets.control.**Control**(*value*)

Methods

init_ui()
toggle_control(state)
update_value(new_value)

value_changed(**args*, ***kwargs*) **= <PySide2.QtCore.Signal object>**

init_ui()

toggle_control(*state*)

update_value(*new_value*)

staticMetaObject = <PySide2.QtCore.QMetaObject object>

5.5.2 Monitor

Classes

Monitor(value[, update_period, enum_name]) **param value**

class vent.gui.widgets.monitor.**Monitor**(*value*, *update_period=0.5*, *enum_name=None*)

Parameters

- **value** (Value) –
- **update_period** (*float*) – update period of monitor in s

Methods

__init__(value[, update_period, enum_name]) **param value**

_limits_changed(val)
check_alarm([value])
init_ui()

continues on next page

Table 8 – continued from previous page

set_alarm(alarm)	Simple wrapper to set alarm state from a qt signal
timed_update()	
toggle_alarm()	
toggle_control(state)	
update_boxes(new_values)	
update_limits(new_limits)	
update_value(new_value)	

Attributes

alarm_state

alarm(**args*, ***kwargs*) **= <PySide2.QtCore.Signal object>**

limits_changed(**args*, ***kwargs*) **= <PySide2.QtCore.Signal object>**

__init__(*value*, *update_period=0.5*, *enum_name=None*)

Parameters

- **value** (Value) –
- **update_period** (*float*) – update period of monitor in s

init_ui()

toggle_control(*state*)

update_boxes(*new_values*)

update_value(*new_value*)

update_limits(*new_limits*)

timed_update()

_limits_changed(*val*)

property alarm_state

set_alarm(*alarm*)

Simple wrapper to set alarm state from a qt signal

Parameters **alarm** (*bool*) – Whether to set as alarm state or not

toggle_alarm()

check_alarm(*value=None*)

staticMetaObject = <PySide2.QtCore.QMetaObject object>

5.5.3 Plot

Data

PLOT_FREQ	Update frequency of Plot s in Hz
PLOT_TIMER	A QTimer that updates :class:`.TimedPlotCurveItem`s

Classes

Plot(name[, buffer_size, plot_duration, ...])	When initializing PlotWidget, *parent* and *background* are passed to GraphicsWidget.__init__() and all others are passed to PlotItem.__init__().

vent.gui.widgets.plot.**PLOT_TIMER = None**
A QTimer that updates :class:`.TimedPlotCurveItem`s

vent.gui.widgets.plot.**PLOT_FREQ = 5**
Update frequency of Plot s in Hz

class vent.gui.widgets.plot.**Plot**(*name*, *buffer_size=4092*, *plot_duration=5*, *abs_range=None*, *safe_range=None*, *color=None*, *units=''*, ***kwargs*)
When initializing PlotWidget, *parent* and *background* are passed to GraphicsWidget.__init__() and all others are passed to PlotItem.__init__(). **Methods**

_safe_limits_changed(val)	
set_duration(dur)	
set_safe_limits(limits)	
update_value(new_value)	new_value: (timestamp from time.time(), value)

limits_changed(**args*, ***kwargs*) **= <PySide2.QtCore.Signal object>**

set_duration(*dur*)

staticMetaObject = <PySide2.QtCore.QMetaObject object>

update_value(*new_value*)
new_value: (timestamp from time.time(), value)

_safe_limits_changed(*val*)

set_safe_limits(*limits*)

5.5.4 Status Bar

Classes

HeartBeat([update_interval, timeout_dur])	**param update_interval** How often to do the heartbeat, in ms
Message_Display()	
Power_Button()	

continues on next page

Table 13 – continued from previous page

Status_Bar()	• Start/stop button

class vent.gui.widgets.status_bar.**Status_Bar**

- Start/stop button
- **Status indicator - a clock that increments with heartbeats,** or some other visual indicator that things are alright
- Status bar - most recent alarm or notification w/ means of clearing
- Override to give 100% oxygen and silence all alarms

Methods

init_ui()

init_ui()

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class vent.gui.widgets.status_bar.**Message_Display**

Attributes

alarm_level

Methods

clear_message()	
draw_state([state])	
init_ui()	
make_icons()	
update_message(alarm)	**param alarm**

message_cleared(**args*, ***kwargs*) **= <PySide2.QtCore.Signal object>**

level_changed(**args*, ***kwargs*) **= <PySide2.QtCore.Signal object>**

make_icons()

init_ui()

draw_state(*state=None*)

update_message(*alarm*)

Parameters **alarm** (Alarm) –

clear_message()

property alarm_level

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class vent.gui.widgets.status_bar.**HeartBeat**(*update_interval=100*, *timeout_dur=5000*)

Parameters

- **update_interval** (*int*) – How often to do the heartbeat, in ms
- **timeout** (*int*) – how long to wait before hearing from control process

Methods

__init__([update_interval, timeout_dur])	**param update_interval** How often to do the heartbeat, in ms
_heartbeat()	Called every (update_interval) milliseconds to set the text of the timer.
beatheart(heartbeat_time)	
check_timeout()	
init_ui()	
set_indicator([state])	
start_timer([update_interval])	**param update_interval** How often (in ms) the timer should be updated.
stop_timer()	you can read the sign ya punk

timeout(**args*, ***kwargs*) **= <PySide2.QtCore.Signal object>**

heartbeat(**args*, ***kwargs*) **= <PySide2.QtCore.Signal object>**

__init__(*update_interval=100*, *timeout_dur=5000*)

Parameters

- **update_interval** (*int*) – How often to do the heartbeat, in ms
- **timeout** (*int*) – how long to wait before hearing from control process

init_ui()

check_timeout()

set_indicator(*state=None*)

start_timer(*update_interval=None*)

Parameters **update_interval** (*float*) – How often (in ms) the timer should be updated.

stop_timer()
you can read the sign ya punk

beatheart(*heartbeat_time*)

_heartbeat()
Called every (update_interval) milliseconds to set the text of the timer.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class vent.gui.widgets.status_bar.**Power_Button**

Methods

init_ui()	

init_ui()

staticMetaObject = <PySide2.QtCore.QMetaObject object>

5.5.5 Components

Classes

DoubleSlider([decimals])	Slider capable of representing floats
EditableLabel([parent])	Editable label
KeyPressHandler	Custom key press handler
QHLine([parent, color])	with respct to https://stackoverflow.com/a/51057516
RangeSlider(abs_range, safe_range[, decimals])	Slider with two handles that sets a range

class vent.gui.widgets.components.**DoubleSlider**(*decimals=1*, **args*, ***kargs*)

Slider capable of representing floats

Ripped off from and https://stackoverflow.com/a/50300848 ,

Thank you!!! **Methods**

_maximum()
_minimum()
_singleStep()
emitDoubleValueChanged()
maximum(self)
minimum(self)
setDecimals(decimals)
setMaximum(self, arg__1)
setMinimum(self, arg__1)
setSingleStep(self, arg__1)
setValue(self, arg__1)
singleStep(self)
value(self)

doubleValueChanged(**args*, ***kwargs*) **= <PySide2.QtCore.Signal object>**

setDecimals(*decimals*)

emitDoubleValueChanged()

value(*self*) → int

setMinimum(*self*, *arg__1: int*)

setMaximum(*self*, *arg__1: int*)

minimum(*self*) → int

_minimum()

maximum(*self*) → int

_maximum()

setSingleStep(*self*, *arg__1: int*)

singleStep(*self*) → int

_singleStep()

setValue(*self*, *arg__1: int*)

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class vent.gui.widgets.components.**RangeSlider**(*abs_range*, *safe_range*, *decimals=1*, **args*, ***kwargs*)

Slider with two handles that sets a range

Parameters

- **abs_range** (*tuple*) – absolute range of slider
- **safe_range** (*tuple*) – default set values for handles of slider
- **decimals** (*int*) – number of decimals of precision
- ***args** –
- ****kwargs** –

Methods

__pick(pt)	
__pixelPosToRangeValue(pos)	
__init__(abs_range, safe_range[, decimals])	Slider with two handles that sets a range
generate_labels()	Generate the text labels for the slider.
mouseMoveEvent(self, ev)	
mousePressEvent(self, ev)	
paintEvent(self, ev)	
resizeEvent(self, event)	
setHigh(high)	
setLow(low)	
setValue(value) **param value** (low, high) to set	
value(self)	

Attributes

high	
low	
valueChanged(*args, **kwargs)	(tuple): (low, high) set range of floats

valueChanged(**args*, ***kwargs*) **= <PySide2.QtCore.Signal object>**

(low, high) set range of floats

Type (tuple)

__init__(*abs_range*, *safe_range*, *decimals=1*, **args*, ***kwargs*)

Slider with two handles that sets a range

Parameters

- **abs_range** (*tuple*) – absolute range of slider
- **safe_range** (*tuple*) – default set values for handles of slider
- **decimals** (*int*) – number of decimals of precision
- ***args** –

- ****kwargs** –

property low

property high

setLow(*low*)

setHigh(*high*)

setValue(*value*)

Parameters **value** (*tuple*) – (low, high) to set

value(*self*) → int

generate_labels()

Generate the text labels for the slider.

Called on init and on resizeEvent

paintEvent(*self*, *ev: PySide2.QtGui.QPaintEvent*)

mousePressEvent(*self*, *ev: PySide2.QtGui.QMouseEvent*)

mouseMoveEvent(*self*, *ev: PySide2.QtGui.QMouseEvent*)

__pick(*pt*)

__pixelPosToRangeValue(*pos*)

resizeEvent(*self*, *event: PySide2.QtGui.QResizeEvent*)

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class vent.gui.widgets.components.**KeyPressHandler**

Custom key press handler https://gist.github.com/mfessenden/baa2b87b8addb0b60e54a11c1da48046 **Methods**

eventFilter(self, watched, event)	

escapePressed(**args*, ***kwargs*) **= <PySide2.QtCore.Signal object>**

returnPressed(**args*, ***kwargs*) **= <PySide2.QtCore.Signal object>**

eventFilter(*self*, *watched: PySide2.QtCore.QObject*, *event: PySide2.QtCore.QEvent*) → bool

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class vent.gui.widgets.components.**EditableLabel**(*parent=None*, ***kwargs*)

Editable label https://gist.github.com/mfessenden/baa2b87b8addb0b60e54a11c1da48046 **Methods**

create_signals()	
escapePressedAction()	Escape event handler
labelPressedEvent(event)	Set editable if the left mouse button is clicked
labelUpdatedAction()	Indicates the widget text has been updated
returnPressedAction()	Return/enter event handler
setLabelEditableAction()	Action to make the widget editable
setText(text)	Standard QLabel text setter
text()	Standard QLabel text getter

textChanged(**args*, ***kwargs*) **= <PySide2.QtCore.Signal object>**

create_signals()

text()
Standard QLabel text getter

setText(*text*)
Standard QLabel text setter

labelPressedEvent(*event*)
Set editable if the left mouse button is clicked

setLabelEditableAction()
Action to make the widget editable

labelUpdatedAction()
Indicates the widget text has been updated

returnPressedAction()
Return/enter event handler

escapePressedAction()
Escape event handler

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class vent.gui.widgets.components.**QHLine**(*parent=None*, *color='#FFFFFF'*)
with respct to https://stackoverflow.com/a/51057516 **Methods**

setColor(color)

staticMetaObject = <PySide2.QtCore.QMetaObject object>

setColor(*color*)

5.5.5.1 GUI Stylesheets

Data

MONITOR_UPDATE_INTERVAL	(float): inter-update interval (seconds) for *Monitor*

Functions

set_dark_palette(app)	Apply Dark Theme to the Qt application instance.

vent.gui.styles.**MONITOR_UPDATE_INTERVAL = 0.5**
inter-update interval (seconds) for *Monitor*

Type (float)

vent.gui.styles.**set_dark_palette**(*app*)
Apply Dark Theme to the Qt application instance.

borrowed from https://github.com/gmarull/qtmodern/blob/master/qtmodern/styles.py

Args: app (QApplication): QApplication instance.

5.5.5.2 GUI Alarm Manager

Classes

AlarmManager()

Functions

get_alarm_manager()

vent.gui.alarm_manager.**get_alarm_manager**()

class vent.gui.alarm_manager.**AlarmManager**

Methods

monitor_alarm(alarm)	Parse a tentative alarm from a monitor – we should have already gotten an alarm from the controller, so this largely serves as a double check.
parse_message(alarm)	If an alarm doesn't have a `message` attr, make one for it.
update_alarms(alarms)	

new_alarm(**args*, ***kwargs*) **= <PySide2.QtCore.Signal object>**

update_alarms(*alarms*)

monitor_alarm(*alarm*)

Parse a tentative alarm from a monitor – we should have already gotten an alarm from the controller, so this largely serves as a double check.

Doesn't use the `Alarm` class because creating a new alarm increments the counter.

Parameters **alarm** (*tuple*) – (monitor_name, monitor_value, timestamp)

parse_message(*alarm*)

If an alarm doesn't have a `message` attr, make one for it.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

CHAPTER

SIX

IO MODULE

6.1 Submodules

6.2 vent.io.devices module

A module for ventilator hardware device drivers

6.3 vent.io.iobase module

6.4 Module contents

CHAPTER

SEVEN

REQUIREMENTS

CHAPTER

EIGHT

DATASHEETS & MANUALS

8.1 Manuals

- Hamilton T1 Quick Guide

8.2 Other Reference Material

- Hamilton UI Simulator

CHAPTER

NINE

SPECS

CHAPTER

TEN

CHANGELOG

10.1 Version 0.0

10.1.1 v0.0.2 (April xxth, 2020)

- Refactored gui into a module, splitting `widgets`, `styles`, and `defaults`.

10.1.2 v0.0.1 (April 12th, 2020)

- Added changelog
- Moved requirements for building docs to *requirements_docs.txt* so regular program reqs are a bit lighter.
- added autosummaries
- added additional resources & documentation files, with examples for adding external files like pdfs

10.1.3 v0.0.0 (April 12th, 2020)

Example of a changelog entry!!!

- We fixed this
- and this
- and this

Warning: but we didn't do this thing

Todo: and we still have to do this other thing.

CHAPTER

ELEVEN

BUILDING THE DOCS

A very brief summary...

- Docs are configured to be built from `_docs` into `docs`.
- The main page is `index.rst` which links to the existing modules
- To add a new page, you can create a new `.rst` file if you are writing with Restructuredtext , or a *.md* file if you are writing with markdown.

11.1 Local Build

- `pip install -r requirements.txt`
- `cd _docs`
- `make html`

Documentation will be generated into `docs`

CHAPTER

TWELVE

H1 HEADING 8-)

12.1 h2 Heading

12.1.1 h3 Heading

12.1.1.1 h4 Heading

h5 Heading

h6 Heading

12.2 Horizontal Rules

12.3 Emphasis

This is bold text

This is bold text

This is italic text

This is italic text

12.4 Blockquotes

> Blockquotes can also be nested...
>
> > ...by using additional greater-than signs right next to each other...
> >
> > > ...or with spaces between arrows.

12.5 Lists

Unordered

- Create a list by starting a line with +, −, or ⋆
- Sub-lists are made by indenting 2 spaces:
  - Marker character change forces new list start:
    - Ac tristique libero volutpat at
    - Facilisis in pretium nisl aliquet
    - Nulla volutpat aliquam velit
- Very easy!

Ordered

1. Lorem ipsum dolor sit amet
2. Consectetur adipiscing elit
3. Integer molestie lorem at massa
4. You can use sequential numbers...
5. ...or keep all the numbers as `1.`

12.6 Code

Inline `code`

Indented code

```
// Some comments
line 1 of code
line 2 of code
line 3 of code
```

Block code "fences"

```
Sample text here...
```

Syntax highlighting

```
var foo = function (bar) {
  return bar++;
};

console.log(foo(5));
```

12.7 Links

link text

link with title

12.8 Images

Minion

Like links, Images also have a footnote style syntax

Alt text

With a reference later in the document defining the URL location:

CHAPTER

THIRTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

V

vent.common.values, 9
vent.gui.alarm_manager, 39
vent.gui.main, 26
vent.gui.widgets.monitor, 30

INDEX

Symbols

_PID_update() (*vent.controller.control_module.ControlModuleBase method*), 15
__SimulatedPropValve() (*vent.controller.control_module.ControlModuleSimulator method*), 16
__SimulatedSolenoid() (*vent.controller.control_module.ControlModuleSimulator method*), 16
__analyze_last_waveform() (*vent.controller.control_module.ControlModuleBase method*), 14
__calculate_control_signal_in() (*vent.controller.control_module.ControlModuleBase method*), 15
__get_PID_error() (*vent.controller.control_module.ControlModuleBase method*), 15
__get_pressure_derivative() (*vent.controller.control_module.ControlModuleBase method*), 15
__init__() (*vent.common.message.Alarm method*), 9
__init__() (*vent.common.message.ControlSetting method*), 8
__init__() (*vent.common.message.SensorValues method*), 7
__init__() (*vent.common.values.Value method*), 11
__init__() (*vent.coordinator.coordinator.CoordinatorLocal method*), 20
__init__() (*vent.gui.main.Vent_Gui method*), 28
__init__() (*vent.gui.widgets.components.RangeSlider method*), 36
__init__() (*vent.gui.widgets.monitor.Monitor method*), 31
__init__() (*vent.gui.widgets.status_bar.HeartBeat method*), 34
__pick() (*vent.gui.widgets.components.RangeSlider method*), 37
__pixelPosToRangeValue() (*vent.gui.widgets.components.RangeSlider method*), 37
__test_critical_levels() (*vent.controller.control_module.ControlModuleBase method*), 14
__update_alarms() (*vent.controller.control_module.ControlModuleBase method*), 15
_alarm_to_COPY() (*vent.controller.control_module.ControlModuleBase method*), 14
_controls_from_COPY() (*vent.controller.control_module.ControlModuleBase method*), 14
_get_control_signal_in() (*vent.controller.control_module.ControlModuleBase method*), 15
_get_control_signal_out() (*vent.controller.control_module.ControlModuleBase method*), 15
_heartbeat() (*vent.gui.widgets.status_bar.HeartBeat method*), 34
_initialize_set_to_COPY() (*vent.controller.control_module.ControlModuleBase method*), 14
_is_running (*vent.coordinator.coordinator.CoordinatorLocal attribute*), 20
_limits_changed() (*vent.gui.widgets.monitor.Monitor method*), 31
_maximum() (*vent.gui.widgets.components.DoubleSlider method*), 35
_minimum() (*vent.gui.widgets.components.DoubleSlider method*), 35
_safe_limits_changed() (*vent.gui.widgets.plot.Plot method*), 32
_sensor_to_COPY() (*vent.controller.control_module.ControlModuleBase method*), 14
_sensor_to_COPY() (*vent.controller.control_module.ControlModuleDevice method*), 16
_sensor_to_COPY() (*vent.controller.control_module.ControlModuleSimulator method*), 16
_singleStep() (*vent.gui.widgets.components.DoubleSlider method*), 35

_start_mainloop() (*vent.controller.control_module.ControlModuleBase method*), 15
_start_mainloop() (*vent.controller.control_module.ControlModuleDevice method*), 16
_start_mainloop() (*vent.controller.control_module.ControlModuleSimulator method*), 16

A

abs_range() (*vent.common.values.Value property*), 11
Alarm (*class in vent.common.message*), 8
alarm (*vent.gui.widgets.monitor.Monitor attribute*), 31
alarm_level() (*vent.gui.widgets.status_bar.Message_Display property*), 33
alarm_manager (*vent.gui.main.Vent_Gui attribute*), 27, 29
alarm_state() (*vent.gui.main.Vent_Gui property*), 29
alarm_state() (*vent.gui.widgets.monitor.Monitor property*), 31
alarm_state_changed() (*vent.gui.main.Vent_Gui method*), 29
AlarmManager (*class in vent.gui.alarm_manager*), 39
alarms_updated (*vent.gui.main.Vent_Gui attribute*), 28
AlarmSeverity (*class in vent.common.message*), 8

B

Balloon_Simulator (*class in vent.controller.control_module*), 16
beatheart() (*vent.gui.widgets.status_bar.HeartBeat method*), 34
BREATHS_PER_MINUTE (*vent.common.values.ValueName attribute*), 10

C

check_alarm() (*vent.gui.widgets.monitor.Monitor method*), 31
check_timeout() (*vent.gui.widgets.status_bar.HeartBeat method*), 34
clear_logged_alarms() (*vent.coordinator.coordinator.CoordinatorBase method*), 19
clear_logged_alarms() (*vent.coordinator.coordinator.CoordinatorLocal method*), 20
clear_logged_alarms() (*vent.coordinator.coordinator.CoordinatorRemote method*), 21
clear_message() (*vent.gui.widgets.status_bar.Message_Display method*), 33
closeEvent() (*vent.gui.main.Vent_Gui method*), 30
Control (*class in vent.gui.widgets.control*), 30
CONTROL (*in module vent.common.values*), 11
CONTROL (*vent.gui.main.Vent_Gui attribute*), 28
control_module (*vent.gui.main.Vent_Gui attribute*), 27, 28
control_width (*vent.gui.main.Vent_Gui attribute*), 28
ControlModuleBase (*class in vent.controller.control_module*), 13
ControlModuleDevice (*class in vent.controller.control_module*), 15
ControlModuleSimulator (*class in vent.controller.control_module*), 16
controls (*vent.gui.main.Vent_Gui attribute*), 27, 28
ControlSetting (*class in vent.common.message*), 7
coordinator (*vent.gui.main.Vent_Gui attribute*), 27, 28
CoordinatorBase (*class in vent.coordinator.coordinator*), 19
CoordinatorLocal (*class in vent.coordinator.coordinator*), 20
CoordinatorRemote (*class in vent.coordinator.coordinator*), 20
create_signals() (*vent.gui.widgets.components.EditableLabel method*), 37

D

decimals() (*vent.common.values.Value property*), 11
default() (*vent.common.values.Value property*), 11
do_pid_control() (*vent.controller.control_module.ControlModuleBase method*), 15
do_state_control() (*vent.controller.control_module.ControlModuleBase method*), 15
DoubleSlider (*class in vent.gui.widgets.components*), 35
doubleValueChanged (*vent.gui.widgets.components.DoubleSlider attribute*), 35
draw_state() (*vent.gui.widgets.status_bar.Message_Display method*), 33

E

EditableLabel (*class in vent.gui.widgets.components*), 37
emitDoubleValueChanged() (*vent.gui.widgets.components.DoubleSlider method*), 35
Error (*class in vent.common.message*), 9
escapePressed (*vent.gui.widgets.components.KeyPressHandler attribute*), 37
escapePressedAction() (*vent.gui.widgets.components.EditableLabel*

method), 38
eventFilter() (*vent.gui.widgets.components.KeyPressHandler method*), 37

F

FIO2 (*vent.common.values.ValueName attribute*), 10

G

generate_labels() (*vent.gui.widgets.components.RangeSlider method*), 37
get_active_alarms() (*vent.controller.control_module.ControlModuleBase method*), 15
get_active_alarms() (*vent.coordinator.coordinator.CoordinatorBase method*), 19
get_active_alarms() (*vent.coordinator.coordinator.CoordinatorLocal method*), 20
get_active_alarms() (*vent.coordinator.coordinator.CoordinatorRemote method*), 21
get_alarm_manager() (*in module vent.gui.alarm_manager*), 39
get_alarms() (*vent.controller.control_module.ControlModuleBase method*), 15
get_control() (*vent.controller.control_module.ControlModuleBase method*), 15
get_control() (*vent.coordinator.coordinator.CoordinatorBase method*), 19
get_control() (*vent.coordinator.coordinator.CoordinatorLocal method*), 20
get_control() (*vent.coordinator.coordinator.CoordinatorRemote method*), 21
get_control_module() (*in module vent.controller.control_module*), 16
get_coordinator() (*in module vent.coordinator.coordinator*), 21
get_logged_alarms() (*vent.controller.control_module.ControlModuleBase method*), 15
get_logged_alarms() (*vent.coordinator.coordinator.CoordinatorBase method*), 19
get_logged_alarms() (*vent.coordinator.coordinator.CoordinatorLocal method*), 20
get_logged_alarms() (*vent.coordinator.coordinator.CoordinatorRemote method*), 21
get_past_waveforms() (*vent.controller.control_module.ControlModuleBase method*), 15
get_pressure() (*vent.controller.control_module.Balloon_Simulator method*), 16
get_sensors() (*vent.controller.control_module.ControlModuleBase method*), 15
get_sensors() (*vent.coordinator.coordinator.CoordinatorBase method*), 19
get_sensors() (*vent.coordinator.coordinator.CoordinatorLocal method*), 20
get_sensors() (*vent.coordinator.coordinator.CoordinatorRemote method*), 21
get_volume() (*vent.controller.control_module.Balloon_Simulator method*), 16
gui_closing (*vent.gui.main.Vent_Gui attribute*), 28

H

handle_alarm() (*vent.gui.main.Vent_Gui method*), 29
handle_cleared_alarm() (*vent.gui.main.Vent_Gui method*), 29
HeartBeat (*class in vent.gui.widgets.status_bar*), 33
heartbeat (*vent.gui.widgets.status_bar.HeartBeat attribute*), 34
heartbeat() (*vent.controller.control_module.ControlModuleBase method*), 15
heartbeat() (*vent.coordinator.process_manager.ProcessManager method*), 22
high() (*vent.gui.widgets.components.RangeSlider property*), 37
HUMIDITY (*vent.common.values.ValueName attribute*), 10

I

id_counter (*vent.common.message.Alarm attribute*), 9
IE_RATIO (*vent.common.values.ValueName attribute*), 10
init_controls() (*vent.gui.main.Vent_Gui method*), 29
init_ui() (*vent.gui.main.Vent_Gui method*), 29
init_ui() (*vent.gui.widgets.control.Control method*), 30
init_ui() (*vent.gui.widgets.monitor.Monitor method*), 31
init_ui() (*vent.gui.widgets.status_bar.HeartBeat method*), 34
init_ui() (*vent.gui.widgets.status_bar.Message_Display method*), 33
init_ui() (*vent.gui.widgets.status_bar.Power_Button method*), 34
init_ui() (*vent.gui.widgets.status_bar.Status_Bar method*), 33
init_ui_controls() (*vent.gui.main.Vent_Gui method*), 29

init_ui_monitor() (*vent.gui.main.Vent_Gui method*), 29
init_ui_plots() (*vent.gui.main.Vent_Gui method*), 29
init_ui_signals() (*vent.gui.main.Vent_Gui method*), 29
init_ui_status_bar() (*vent.gui.main.Vent_Gui method*), 29
INSPIRATION_TIME_SEC (*vent.common.values.ValueName attribute*), 10
is_running() (*vent.controller.control_module.ControlModuleBase method*), 15
is_running() (*vent.coordinator.coordinator.CoordinatorBase method*), 20
is_running() (*vent.coordinator.coordinator.CoordinatorLocal method*), 20
is_running() (*vent.coordinator.coordinator.CoordinatorRemote method*), 21

K

KeyPressHandler (*class in vent.gui.widgets.components*), 37

L

labelPressedEvent() (*vent.gui.widgets.components.EditableLabel method*), 38
labelUpdatedAction() (*vent.gui.widgets.components.EditableLabel method*), 38
launch_gui() (*in module vent.gui.main*), 30
level_changed (*vent.gui.widgets.status_bar.Message_Display attribute*), 33
LIMITS (*in module vent.common.values*), 12
limits_changed (*vent.gui.widgets.monitor.Monitor attribute*), 31
limits_changed (*vent.gui.widgets.plot.Plot attribute*), 32
low() (*vent.gui.widgets.components.RangeSlider property*), 37

M

main_height (*vent.gui.main.Vent_Gui attribute*), 28
make_icons() (*vent.gui.widgets.status_bar.Message_Display method*), 33
maximum() (*vent.gui.widgets.components.DoubleSlider method*), 35
message_cleared (*vent.gui.widgets.status_bar.Message_Display attribute*), 33
Message_Display (*class in vent.gui.widgets.status_bar*), 33
minimum() (*vent.gui.widgets.components.DoubleSlider method*), 35
module
vent.common.message, 7
vent.common.values, 9
vent.controller.control_module, 13
vent.coordinator.coordinator, 19
vent.coordinator.process_manager, 21
vent.gui.alarm_manager, 39
vent.gui.main, 26
vent.gui.styles, 38
vent.gui.widgets.components, 35
vent.gui.widgets.control, 30
vent.gui.widgets.monitor, 30
vent.gui.widgets.plot, 32
vent.gui.widgets.status_bar, 32
vent.io, 41
vent.io.devices, 41
Monitor (*class in vent.gui.widgets.monitor*), 30
MONITOR (*vent.gui.main.Vent_Gui attribute*), 28
monitor (*vent.gui.main.Vent_Gui attribute*), 27, 28
monitor_alarm() (*vent.gui.alarm_manager.AlarmManager method*), 39
MONITOR_UPDATE_INTERVAL (*in module vent.gui.styles*), 38
monitor_width (*vent.gui.main.Vent_Gui attribute*), 28
mouseMoveEvent() (*vent.gui.widgets.components.RangeSlider method*), 37
mousePressEvent() (*vent.gui.widgets.components.RangeSlider method*), 37

N

name() (*vent.common.values.Value property*), 11
new_alarm (*vent.gui.alarm_manager.AlarmManager attribute*), 39

O

OFF (*vent.common.message.AlarmSeverity attribute*), 8
ORANGE (*vent.common.message.AlarmSeverity attribute*), 8
OUupdate() (*vent.controller.control_module.Balloon_Simulator method*), 16

P

paintEvent() (*vent.gui.widgets.components.RangeSlider method*), 37
parse_message() (*vent.gui.alarm_manager.AlarmManager method*), 39
PEEP (*vent.common.values.ValueName attribute*), 10
PEEP_TIME (*vent.common.values.ValueName attribute*), 10
PIP (*vent.common.values.ValueName attribute*), 9
PIP_TIME (*vent.common.values.ValueName attribute*), 10
Plot (*class in vent.gui.widgets.plot*), 32

PLOT_FREQ (*in module vent.gui.widgets.plot*), 32
PLOT_TIMER (*in module vent.gui.widgets.plot*), 32
plot_width (*vent.gui.main.Vent_Gui attribute*), 28
PLOTS (*vent.gui.main.Vent_Gui attribute*), 28
plots (*vent.gui.main.Vent_Gui attribute*), 27, 28
Power_Button (*class in vent.gui.widgets.status_bar*), 34
PRESSURE (*vent.common.values.ValueName attribute*), 10
ProcessManager (*class in vent.coordinator.process_manager*), 21

Q

QHLine (*class in vent.gui.widgets.components*), 38

R

RangeSlider (*class in vent.gui.widgets.components*), 36
RED (*vent.common.message.AlarmSeverity attribute*), 8
resizeEvent() (*vent.gui.widgets.components.RangeSlider method*), 37
restart_process() (*vent.coordinator.process_manager.ProcessManager method*), 22
returnPressed (*vent.gui.widgets.components.KeyPressHandler attribute*), 37
returnPressedAction() (*vent.gui.widgets.components.EditableLabel method*), 38
running() (*vent.controller.control_module.ControlModuleBase property*), 15

S

safe_range() (*vent.common.values.Value property*), 11
SENSOR (*in module vent.common.values*), 11
SensorValueNew (*class in vent.common.message*), 7
SensorValues (*class in vent.common.message*), 7
set_alarm() (*vent.gui.widgets.monitor.Monitor method*), 31
set_control() (*vent.controller.control_module.ControlModuleBase method*), 15
set_control() (*vent.coordinator.coordinator.CoordinatorBase method*), 19
set_control() (*vent.coordinator.coordinator.CoordinatorLocal method*), 20
set_control() (*vent.coordinator.coordinator.CoordinatorRemote method*), 21
set_dark_palette() (*in module vent.gui.styles*), 38
set_duration() (*vent.gui.widgets.plot.Plot method*), 32
set_flow_in() (*vent.controller.control_module.Balloon_Simulator method*), 16
set_flow_out() (*vent.controller.control_module.Balloon_Simulator method*), 16
set_indicator() (*vent.gui.widgets.status_bar.HeartBeat method*), 34
set_plot_duration() (*vent.gui.main.Vent_Gui method*), 29
set_safe_limits() (*vent.gui.widgets.plot.Plot method*), 32
set_value() (*vent.gui.main.Vent_Gui method*), 29
setColor() (*vent.gui.widgets.components.QHLine method*), 38
setDecimals() (*vent.gui.widgets.components.DoubleSlider method*), 35
setHigh() (*vent.gui.widgets.components.RangeSlider method*), 37
setLabelEditableAction() (*vent.gui.widgets.components.EditableLabel method*), 38
setLow() (*vent.gui.widgets.components.RangeSlider method*), 37
setMaximum() (*vent.gui.widgets.components.DoubleSlider method*), 35
setMinimum() (*vent.gui.widgets.components.DoubleSlider method*), 35
setSingleStep() (*vent.gui.widgets.components.DoubleSlider method*), 35
setText() (*vent.gui.widgets.components.EditableLabel method*), 38
setValue() (*vent.gui.widgets.components.DoubleSlider method*), 35
setValue() (*vent.gui.widgets.components.RangeSlider method*), 37
singleStep() (*vent.gui.widgets.components.DoubleSlider method*), 35
start() (*vent.controller.control_module.ControlModuleBase method*), 15
start() (*vent.coordinator.coordinator.CoordinatorBase method*), 20
start() (*vent.coordinator.coordinator.CoordinatorLocal method*), 20
start() (*vent.coordinator.coordinator.CoordinatorRemote method*), 21
start_process() (*vent.coordinator.process_manager.ProcessManager method*), 21
start_time (*vent.gui.main.Vent_Gui attribute*), 27, 29
start_timer() (*vent.gui.widgets.status_bar.HeartBeat method*), 34
staticMetaObject (*vent.gui.alarm_manager.AlarmManager attribute*), 39
staticMetaObject (*vent.gui.main.Vent_Gui attribute*), 29
staticMetaObject (*vent.gui.widgets.components.DoubleSlider attribute*), 36
staticMetaObject (*vent.gui.widgets.components.EditableLabel*

attribute), 38
staticMetaObject (*vent.gui.widgets.components.KeyPressHandler attribute*), 37
staticMetaObject (*vent.gui.widgets.components.QHLine attribute*), 38
staticMetaObject (*vent.gui.widgets.components.RangeSlider attribute*), 37
staticMetaObject (*vent.gui.widgets.control.Control attribute*), 30
staticMetaObject (*vent.gui.widgets.monitor.Monitor attribute*), 31
staticMetaObject (*vent.gui.widgets.plot.Plot attribute*), 32
staticMetaObject (*vent.gui.widgets.status_bar.HeartBeat attribute*), 34
staticMetaObject (*vent.gui.widgets.status_bar.Message_Display attribute*), 33
staticMetaObject (*vent.gui.widgets.status_bar.Power_Button attribute*), 34
staticMetaObject (*vent.gui.widgets.status_bar.Status_Bar attribute*), 33
Status_Bar (*class in vent.gui.widgets.status_bar*), 33
status_height (*vent.gui.main.Vent_Gui attribute*), 28
stop() (*vent.controller.control_module.ControlModuleBase method*), 15
stop() (*vent.coordinator.coordinator.CoordinatorBase method*), 20
stop() (*vent.coordinator.coordinator.CoordinatorLocal method*), 20
stop() (*vent.coordinator.coordinator.CoordinatorRemote method*), 21
stop_timer() (*vent.gui.widgets.status_bar.HeartBeat method*), 34

T

TEMP (*vent.common.values.ValueName attribute*), 10
text() (*vent.gui.widgets.components.EditableLabel method*), 37
textChanged (*vent.gui.widgets.components.EditableLabel attribute*), 37
timed_update() (*vent.gui.widgets.monitor.Monitor method*), 31
timeout (*vent.gui.widgets.status_bar.HeartBeat attribute*), 34
to_dict() (*vent.common.message.SensorValues method*), 7
to_dict() (*vent.common.values.Value method*), 11
toggle_alarm() (*vent.gui.widgets.monitor.Monitor method*), 31
toggle_control() (*vent.gui.widgets.control.Control method*), 30
toggle_control() (*vent.gui.widgets.monitor.Monitor method*), 31
total_height (*vent.gui.main.Vent_Gui attribute*), 28
total_width (*vent.gui.main.Vent_Gui attribute*), 28
try_stop_process() (*vent.coordinator.process_manager.ProcessManager method*), 22

U

update() (*vent.controller.control_module.Balloon_Simulator method*), 16
update_alarms() (*vent.gui.alarm_manager.AlarmManager method*), 39
update_boxes() (*vent.gui.widgets.monitor.Monitor method*), 31
update_gui() (*vent.gui.main.Vent_Gui method*), 29
update_limits() (*vent.gui.widgets.monitor.Monitor method*), 31
update_message() (*vent.gui.widgets.status_bar.Message_Display method*), 33
update_period (*vent.gui.main.Vent_Gui attribute*), 27, 29
update_period() (*vent.gui.main.Vent_Gui property*), 29
update_value() (*vent.gui.main.Vent_Gui method*), 29
update_value() (*vent.gui.widgets.control.Control method*), 30
update_value() (*vent.gui.widgets.monitor.Monitor method*), 31
update_value() (*vent.gui.widgets.plot.Plot method*), 32

V

Value (*class in vent.common.values*), 10
value() (*vent.gui.widgets.components.DoubleSlider method*), 35
value() (*vent.gui.widgets.components.RangeSlider method*), 37
value_changed (*vent.gui.widgets.control.Control attribute*), 30
valueChanged (*vent.gui.widgets.components.RangeSlider attribute*), 36
ValueName (*class in vent.common.values*), 9
vent.common.message
    module, 7
vent.common.values
    module, 9
vent.controller.control_module
    module, 13
vent.coordinator.coordinator
    module, 19
vent.coordinator.process_manager
    module, 21
vent.gui.alarm_manager
    module, 39

vent.gui.main
    module, 26
vent.gui.styles
    module, 38
vent.gui.widgets.components
    module, 35
vent.gui.widgets.control
    module, 30
vent.gui.widgets.monitor
    module, 30
vent.gui.widgets.plot
    module, 32
vent.gui.widgets.status_bar
    module, 32
vent.io
    module, 41
vent.io.devices
    module, 41
Vent_Gui (*class in vent.gui.main*), 26
VTE (*vent.common.values.ValueName attribute*), 10

Y

YELLOW (*vent.common.message.AlarmSeverity attribute*), 8